university of groningen / faculty of science and engineering

Machine Learning Solutions for Patient Monitoring during Anesthesia

Tomasz Maciąg
S3177211

February 2019

Master Thesis

Human-Machine Communication
Dept. of Artificial Intelligence
University of Groningen, The Netherlands

Internal supervisor:
Dr. Fokie Cnossen (Artificial Intelligence, University of Groningen)

External supervisors:
Kai van Amsterdam (Anesthesiology, University Medical Center Groningen)
Dr. Albertus Ballast (Anesthesiology, University Medical Center Groningen)

Abstract

In an operating room, one of the roles of an anesthesiologist is to track the patient's vital parameters displayed on two monitors. These have certain thresholds for each measured value which when surpassed trigger an alarm. Such a simplistic approach often leads to situations where most of the warnings are false, hence the credibility of the whole alarming system is low. Moreover, several potentially dangerous complications consist of compound interactions between different variables in prolonged periods of time; therefore, a solution capable of capturing a higher underlying complexity is needed.

This study aims to investigate how Machine Learning techniques can be applied to improve the quality of alarms during anesthesia. Two distinct approaches were tested – Complication Detection and Anomaly Detection. In the former, several algorithms were trained to generate improved warnings for hypotension (low blood pressure), while in the latter, more general approach, different methods of detecting anomalous health states were evaluated. We showed that Complication Detection, based on supervised Neural Networks, was the best performing method; however, we did not rule out Anomaly Detection, as it was in overall a more flexible approach that gives more possibilities for future improvements. Finally, we stated that Machine Learning offers promising solutions for patient monitoring, nevertheless due to a phenomenon called the precision-recall tradeoff, it might be difficult to reduce the number of false alarms without a small decrease in the number of detected complications, regardless of the applied technique. The outcome of this study leads to a number of guidelines for future research in this area, including appropriate preprocessing of medical time-series data, comparisons between simple algorithms and more intricate deep learning architectures.

Acknowledgments

Completing this project would not be possible without the involvement of the following people:

Dr. Fokie Cnossen who with her exceptional supervision skills kept the whole project on-track. Thank you for the weekly Cognitive Engineering group meetings which not only were highly motivational but also gave everyone an opportunity to discuss the most random topics one could imagine without feeling bad about it.

Dr. Bert Ballast, thank you for providing your expertise at every step of this project. Without your passion (I think I can publish a book from your elaborate emails) and your rule-based health monitoring system this study would never come to life.

Kai van Amsterdam, thank you for being always available for consulting any issues I came across. Along with your colleague and wizard-scientist Dr. Douglas Eleveld, you created a friendly and inspiring work atmosphere in the office.

My parents, friends and family who supported me all along my studies. Thank you.

HPC Peregrine Cluster (a.k.a. the Supercomputer), thank you for being there when I needed your 180 GB of RAM, GPUs, and 24 cores the most.

Table of Contents

1. Introduction

One of the tasks of an anesthesiologist in an *operating room* (OR) is to control the vital parameters of a sedated patient. This is done by observing multiple measurements displayed on two monitors. These monitors also have alarm functions, which provide a warning when one of the parameters surpasses a preset threshold. However, at present, an abundance of false alarms is generated with this method; hence the reliability of the whole system is severely compromised. This study aims to answer the question: *"How can modern-day Machine Learning algorithms be applied to patient monitoring under anesthesia in order to improve the quality of the alarms?"*. Two distinct approaches were investigated – a more specialized method, called Complication Detection, where the task was to detect a specific medical complication; and more general one, namely Anomaly Detection, sensitive to any abnormalities in the patient's state. Before exploring these two ideas in-depth, a short philosophical background will be introduced to situate the given problem in an appropriate scientific context.

The two main disciplines at the core of this text are *anesthesiology* and *artificial intelligence (AI)*. While anesthesiology is primarily concerned with being able to safely control the state of the patient, AI tries to make complicated reasoning processes more automated and less dependent on human supervision. Therefore, applying algorithms which imitate human thought processes to monitoring patient's health during operations could relieve anesthesiologists from a cognitively and perceptually demanding task. A practical framework for developing intelligent health monitoring systems comes from the research domain called *cognitive engineering* – an applied sub-field of cognitive science, which approaches a problem by making an in-depth analysis of the task from the perspective of a human worker. Its goal is to provide design guidance for the implementation stage or deliver the full product when possible (Norman, 1986; Wilson, Helton, & Wiggins, 2013; Woods & Roth, 1988). For simple systems, such a meticulous approach may be received as an exaggeration; therefore, cognitive engineering methods are most often used in high-risk domains, like aviation, military, or most importantly medicine, where cognitive factors play a major role in the general safety of the final product. Although in this study a full cognitive engineering process will not be presented, the overarching philosophy of creating a user-centric system will be present. More specifically, a problem regarding the human inability to track all changes in the patient's vital parameters will be addressed by testing several machine learning architectures, which could possibly offload some cognitive effort of an anesthesiologist.

Another way of thinking about designing an efficient human-machine system is in terms of an *extended mind* (Clark & Chalmers, 1998; Rowlands, 2010) – a hypothesis according to which mental processes can happen beyond human bodies by including external objects, systems or interactions. Relating this statement to anesthesiology, one might think of an AI-based monitoring system as an extension of the anesthesiologist's mind, acting as his/her outsourced attention and pattern recognition mechanism, which informs the higher executive

structures in case of an emergency. In an even more futuristic perspective, known as *human-extended machine cognition*, the AI system would include the cognitive states of the human, allowing the machine to adjust its behavior to the current cognitive load of the worker, ask for expertise or take suggestions into account (Smart, 2018).

With the grand scheme of an ideal human-AI interaction in mind, the rest of this text will be constrained to the subject matter. The following chapter will present a more detailed introduction to the problem of monitoring during anesthesia, a review of approaches already investigated in the literature and a proposal of two modern-day solutions based on machine learning techniques. Next, the used dataset will be described followed by explanations and tests of multiple algorithms. Lastly, a discussion containing a meta-evaluation of the tested approaches and suggestions for future studies will be presented.

2. Theoretical Background

The term *anesthesia* originates from the Greek "without sensations" and describes an induced state of temporary loss of sensations or awareness. For the needs of this study, the point of interest will be *general anesthesia*, which refers to a patient being in a medically induced coma. Such a state, as opposed to sedation or local anesthesia, requires a ventilator machine to maintain patient's respiratory processes and supervision of an anesthesiologist to make sure that other vital parameters are intact since first-person reports are not available.

2.1 False Alarms

During operation, the anesthesiologist relies on measurements displayed on the ventilator and patient monitors. These monitors have built-in alarming functionalities which produce warnings when a preset threshold is surpassed. Such an over-simplistic design of the mechanism can lead to situations where most of the alarms are false positives. The rate of false alarms varies according to the type of surgery, settings of the monitors chosen by the performing anesthesiologist and the amount of monitoring equipment used. Nevertheless, every position in the literature on this subject shows an abundance of meaningless warnings. A study in which the researchers evaluated 38 moderate-risk operations indicated that 64% of the alarms were clinically irrelevant, while only 5% required immediate action (de Man, Greuters, Boer, Veerman, & Loer, 2013). Another group researched 25 high-risk cardiac operations and found that 80% out of almost 9000 alarms were useless (Schmid et al., 2011).

A common source of false alarms is related to artifacts that have a non-physiological origin. Most modern monitors include only simple linear filters to tackle these issues, which results in a modest success rate; therefore, more advanced signal-processing or artificial intelligence techniques are needed (Takla, Petre, Doyle, Horibe, & Gopakumaran, 2006). Although several promising attempts to refine alarms in anesthetic monitors were made, manufacturers of medical equipment are still reluctant towards implementing any major innovations in their products due to legal regulations which impose the usage of the most sensitive alarming settings to prevent any undetected critical events (Imhoff, Kuhls, Gather & Fried, 2009).

2.2 Consequences

It is important to realize what consequences arise from the high rate of false alarms. In human-factors research one may often encounter the *Crying-Wolf Phenomenon* (Breznitz, 2013). It is inspired by the tale about a young shepherd who called for help when there was no threat, but when a wolf eventually came to eat the sheep, no one from the village would help him – they assumed it is just another one of his old tricks.

Following this analogy, it was empirically shown that there is a correlation between the reliability of an alarm and how often and how fast people respond to it (Bliss & Dunn, 2000; Westenskow, Orr, Simon, Bender, & Frankenberger, 1992). Moreover, there is a dependency regarding the workload of a primary task and the attention paid to alarms, suggesting that if

an anesthesiologist is occupied with another activity, he or she performs worse at interpreting the relevance of an alarm, hence the warnings should be less ambiguous in order to prevent unnoticing true positive alarms. Another negative consequence is the annoyance induced by the high rate of false alarms which resulted in the anesthesiologists changing the thresholds to more liberal values or turning them off entirely – over 70% of anesthesiologist admitted in a survey that they turn off the alarming systems due to an excess of false warnings (Block, Nuutinen, & Ballast, 1999). In critical care units (ICU), the mood and stress levels of nurses suffer from the noise generated by the redundant alerts leading to an early burnout syndrome (Topf & Dillon, 1988). It was suggested that the stress induced by such alarms might account for up to half of the total stress experienced by the nurses in the ICU.

2.3 Basic Solutions

Extensive research in the field of human factors concerning the design of alarming systems resulted in clear guidelines (Stanton, 1994), which can easily be transferred to the medical domain (McIntyre, 1985). Nonetheless, such rules are mostly based in ergonomics, thus they refer to the physical properties of the visual and auditory alarms, while, as can be deduced from the previous paragraphs, the main source of the discussed problem lays in the quality of the alert generating mechanism.

Schmid, Goepfert, & Reuter (2013) in their overview of the current state of alarming systems in the OR and ICU outline several approaches to tackling the problem of false alarms. They suggest that it is useful to introduce different solutions according to different phases of the operation since many alarms do not require action in certain contexts (Seagull & Sanderson, 2001). The most common division of a medical operation consists of three periods: *induction* (patient is put into a coma), *maintenance/perioperative phase* (the actual operation that starts with the first incision) and *emergence* (waking up). Another simple solution is based on time delays which do not trigger an alarm immediately after a threshold has been violated but aggregate more measurements for a given period of time. It was shown that a 14-second time delay minimizes the rate of false alarms by 50% (Görges, Markewitz, & Westenskow, 2009). On the other hand, it also means that some complications, which require immediate action, are not reported adequately. Therefore, a more adaptive approach is needed. Slightly more complex solutions are based on statistical methods, such as autoregressive models, Kalman filters, statistical process control or median filters (for an overview see: Imhoff & Kuhls, 2006 and Imhoff et al., 2009). These methods proved to be successful for artifact reduction; however, since they are often univariate approaches, they cannot substitute intricate decision-making processes.

2.4 Artificial Intelligence

2.4.1 Rule-Based Approaches

Methods from the domain of artificial intelligence (AI) are theoretically more fit to support higher-level cognitive processing than just filtering the signal. *Rule-based systems* are built on expert knowledge which is reapplied in a new context (Ballast, 1992; Sukuvaara, Koski, Mäkivirta, & Kari, 1993). These systems yield promising results in detecting a large number of critical states but have not been used in practice, because their optimization is cumbersome and they often fail to fully embrace the complexity underlying anesthesiology (Alexander & Joshi, 2017). There were approaches to expand rule-based systems with machine learning and statistical techniques; however, their goal was to support the choice of the correct intervention, rather than to provide reliable alarms (Bates & Young, 2003; Morik, Imboff, Brockhausen, Joachims, & Gather, 2000; Müller, Hasman, & Blom, 1997). Other notable AI solutions to health monitoring found in the literature involve systems based on fuzzy logic which are designed similarly to rule-based systems but operate on fuzzy sets rather than on binary rules (Becker et al., 1997; Oberli et al., 1999).

2.4.2 Machine Learning

A nowadays popular subfield of AI is *machine learning (ML)*, which instead of relying on predefined sets of rules, makes computer systems learn the dependencies in the data by being exposed to it in a particular way. In other words, most ML algorithms can be defined as a complex statistical function $y = f(X|\theta)$, where *f* is the model and θ are its parameters. The program tries to optimize θ so that the difference between the output value *y* and its actual value y_{targ} is minimized for a set of attributes *X*. This definition is especially true for *supervised learning* algorithms which are trained on *labeled datasets* that for each sample of attributes *X* contain a corresponding target variable y_{targ}. Supervised learning can be either a *classification* or a *regression problem* where in the former y_{targ} is a binary (0/1) value that indicates whether an input *X* is an entity of a given class, for example, does a car with attributes *X = (fast, red, expensive)* belong to the class of racing cars – the answer must be no/yes (0/1). On the other hand, a regression model maps the *X* to a continuous *y* and could be applied to predict the price of a car with attributes *X = (fast, red, expensive)* where *y* would be drawn from a continuous distribution.

In an *unlabeled dataset* – one that does not contain the y_{targ} variables, *unsupervised* ML techniques can be used. These seek for reoccurring patterns in the data and provide an output based on the presence or absence of such commonalities in *X* – similarly to density estimation in statistics. Unsupervised techniques are often used for *clustering* where an algorithm is given an unlabeled dataset with the task to group the samples. For example, online stores make use of such an approach for customer segmentation in order to provide custom offers for groups of clients with different shopping habits (Daoud, Amine, Bouikhalene, & Lbibb, 2015).

In the medical domain, an interesting ML-based approach was developed by Rejab, Nouira, & Trabelsi (2014) who used a k-Prototypes Clustering method to find patients with similar diseases and trained so-called incremental Support Vector Machines to model the vital parameters for different patient groups. When a new patient was added to the system, he or she was assigned to a group based on certain attributes (age, surgery type, respiration rate etc.) and then monitored using the classification algorithm of their group. The results of this study are promising, as the false alarms were reduced almost completely (by 99,8%) and a warning was produced in 97% of expected situations. However, the used dataset was small. Another ML technique, namely logistic regression, was successfully used to predict hypotension (low blood pressure) from early alterations in high-frequency arterial pressure waveforms (Hatib et al., 2018). The program was able to produce a warning 15 minutes before the event with a 95% chance of a correct detection. A comparative study by Kendale, Kulkarni, Rosenberg, & Wang (2018) investigated how different ML classification algorithms perform at predicting the occurrence of hypotension during an operation based on the patient's pre-operative state. Their results suggest that Linear Discriminant Analysis, Gradient Boosting Machines, Neural Networks, and Random Forests are the best performing methods in predicting the chance of low blood pressure during perioperative anesthesia.

2.4.3 Neural Networks

Artificial Neural Networks (ANNs or *NNs)* are ML algorithms, loosely inspired by the physiology of the human brain, which often combine thousands of artificial neurons organized in multiple layers similarly to the cortical architecture. Such learning systems are capable of capturing the complexity of intricate non-linear processes that occur in the real world, which explains their high popularity in the last two decades (for more details see: 4.3 Basic principles of Neural Networks). ANNs have been used in several early studies to detect complications in the breathing circuit (Orr & Westenskow, 1994), systolic blood pressure (Tsien, 2000), and acute myocardial infarction (Baxt & Skora, 1996). Their efficacy is difficult to assess due to the specifics of the datasets and the not necessarily objective success rate indicators used, although the accuracy of the ANNs in these studies ranged from 87% to 99%. More recently, *Recurrent Neural Networks* (*RNNs*), which have the ability to process time-depended changes, were successfully used by Choi, Schuetz, Stewart, & Sun (2017) to diagnose heart failure based on 12 and 18-month observation windows from Electronic Health Records. Predictions made by their NN were based on long-term measurements of vital parameters, prescribed drugs, past illnesses, diseases, and other annotations. This study, together with others (Jagannatha & Yu, 2016; Luo, 2017), leads to the conclusions that RNNs are capable of modeling times-series data and that processing clinical notes at a word level can provide useful support for the general practitioners in diagnosing health complications.

2.4.4 Explainable Artificial Intelligence

Deep Learning, a sub-field of ML focused on exploring multilayer NNs, is gaining popularity in healthcare thanks to its ability to find patterns in large, multivariate, irregular, and noisy

datasets (LeCun, Bengio, & Hinton, 2015; Ravì et al., 2017). However, it must be noted that the steps that a (multilayer) ANN takes to produce a result are not easily interpretable. For example, an ANN may yield a result that the patient has a heart failure, but it is impossible to obtain from the algorithm explicit declarative knowledge about the reasoning behind the decision. Therefore, such programs might be of little use to the medical staff who need to understand the underlying mechanisms of a diagnosis to fully trust it and react to it properly. Moreover, the recently updated European General Data Protection Regulation (GDPR 2016/679 and ISO/IEC 27001) makes it much more difficult to use black-box approaches in practice if sufficient explanation of a decision process is not provided by a system. On the other hand, this limitation does not mean that Deep Learning should be completely banished from health informatics, as its predictive strengths might complement well the human abilities to foresee certain complications. A novel approach to AI, known as *Explainable-AI*, is becoming an important part of developing modern software, as it implies the need to make the decision process retraceable and interpretable (Holzinger, Biemann, Pattichis, & Kell, 2017). In their study, Choi et al. (2016) developed an algorithm based on an RNN which predicts heart failure from Electronic Health Records. More interestingly, they implemented a two-level attention mechanism in their network which determined the most crucial visits and the most influential variables within a visit. As a result, a clinician received a medical diagnosis backed by a graph depicting the contribution of different measurements on a timeline of visits. The cited study proves that it is possible to apply AI solutions to medical problems without compromising the transparency of such systems.

2.5 Literature Review Conclusions

The literature review indicates that the problem of false alarms in health monitoring is a major issue that has a strong negative impact on the reliability of the alarming systems. Moreover, the way in which anesthesiologists interact with such warnings (or more precisely, disregard them) is inconsistent with the modern view on ideal interaction between humans and machines. In order to achieve a situation in which the anesthesiologist can trust the alarming system, it seems necessary to investigate the use of techniques that can perform the task of detecting complications in a more intelligent or human-like way. Studies mentioned earlier in this chapter suggest that the domain of ML offers methods that could be used for that purpose. However, to our best knowledge, there are no studies in which ML would be used for health monitoring during routine operations. In other words, we described studies that managed to successfully use ML methods in specialized cases: generating reliable alarms for predefined subjects and operation types (Rejab et al., 2014), detecting a particular complication during an operation from high-fidelity recordings (Hatib et al., 2018), and estimating the chance of a complication before the operation (Kendale et al., 2018). We conclude that the next step is a search for a more universal approach – one that could reduce the current high rate of false alarms regardless of the operation type or patient demographics. Our belief is that an alarming system based on an ML technique could improve the presently

used threshold-based warnings and eventually become the default approach for health monitoring during anesthesia.

2.6 Present Study

2.6.1 Goal of the Study

In order to test our beliefs and presumptions expressed above, we made an initial step towards the creation of a general approach to health monitoring. In our study, we aimed to provide more insights about applying ML techniques to data collected in the OR without putting strict constraints on the operation type or patient's demographics. More specifically, multiple algorithms, with most focus put on NNs, were developed and tested with the goal of minimizing the rate of false alarms. Considering the explorative character of this study, it was important to maintain a relatively broad scope of the research; hence, we intended to answer the question: "*How can modern ML techniques can be applied to monitoring patients under anesthesia to improve the medical relevance of the alarms*?". Because of its open-ended formulation, it was necessary to investigate the whole process of developing an ML-based product, starting with a choice of a proper approach to detecting medical complications, followed by data preparation, determining the best-suited ML method, generating meaningful results, and interpreting them. The idea of an alarming system was constrained to generating alarms regarding the current state of the patient; hence the algorithms performed the task of *detection*. In more technically-oriented literature such a task is also referred to as *prediction* (e.g. an ML classifier outputs a prediction about the current state of the patient), which should not be confused with *forecasting* – an act of making predictions about future states (e.g. an ML method makes forecasts about the patient's state in 2 minutes). The main assumption of this study is that medical complications span prolonged periods of time. This implicates that the tested ML techniques must be capable of processing sequences in some form.

There are two main approaches to detecting events in time-series data which we tested:

1) *Complication Detection* where the algorithm was trained to detect a particular medical complication. For the needs of this study, hypotension (low blood pressure) was selected since, depending on its exact definition, it occurs in 41%-93% of all operations (Bijker et al., 2007) and posits a serious danger if overlooked. Moreover, there are other studies, mentioned earlier in the text, which indicate that ML techniques are capable of modeling hypotension (Hatib et al., 2018; Kendale et al., 2018).

2) *Anomaly Detection* in which the algorithms were trained to model normal changes in the vital parameters and classify any new abnormal patterns as anomalous states.

Multiple neural architectures were tested and compared to simpler ML techniques and non-ML baselines. The choice of ML algorithms was made with regards to their previous success in

obtaining good results on time-series data (reasoning behind the selection of each technique will be given along with their descriptions).

2.6.2 Performance Metrics

Having established the goal of our study, it is important to understand how we approached the operationalization of our research question. The choice of a proper metric is crucial, as it determines which aspects of an alarming system are valued the most or, to be more specific, which components of the confusion matrix reflect the needs of a target user in a given setting. In the case of medical diagnosis, the priority is to detect all occurrences of hazardous events given by the ratio of true positives, known also as *recall*:

$$Recall = \frac{true\ positives}{true\ postives + false\ negatives}$$

Intuitively, it describes the probability of detection – a recall score of 0.83 on hypotension classification would mean that 83% of hypotensive states that occurred during operation have been detected.

In the current study, it is also important to register the number of correct alarms in relation to all generated alarms which is measured by the *precision* metric:

$$Precision = \frac{true\ positives}{true\ postives + false\ postives}$$

A precision score of 0.9 would mean that 90% of all alarms were clinically significant. Together, recall and precision describe the completeness and exactness of an alarming system which is often expressed in a single metric of F_1 score – a harmonic mean between the two:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

F_1 score is not directly interpretable but allows us to easily compare models with regards to precision and recall without being biased by a high score of one of these two metrics. For example, a model with precision and recall of 0.6 and 0.8 will have an F_1 score of 0.685, while a model with 0.65 and 0.75 will have an F_1 equal to 0.696 – favoring the latter model, which achieves more balanced precision and recall scores.

Since precision and recall formulas do not include true negatives, which are not the key interest in medical diagnosis systems, they are more robust to class imbalance in the dataset (Davis & Goadrich, 2006). Therefore, their use in this study seems well motivated. Nevertheless, *Area Under the ROC Curve (AUC)* will be reported, as it is a popular measure in the medical literature and allows for more direct comparisons with other studies. AUC indicates the probability that a new observation will be correctly classified, hence 0.5 is a completely random guess, 1 would be achieved by a perfect classifier and scores below 0.5 mean that the classifier is reciprocating the result. Similarly to F_1, AUC is also based on recall;

however, instead of precision, it makes use of the false positive rate. This means that the AUC scores are more optimistic when the classes are imbalanced and more specifically when the negative class (no-complication) dominates the positive one (complication) as it was in our dataset. It must also be noted that the models were optimized using the F_1 score and not AUC; therefore, the scores of the latter should only be treated as an additional measure that gives a more general overview of the performance of an alarming system.

3. Dataset

Data for this project was provided by the Department of Anesthesiology of University Medical Center Groningen (UMCG). Each file in the datasets was a typical example of a multivariate time-series that contained various measurements of the patient's vital state while being under general anesthesia. The data was sampled every 15 seconds by the patient monitor and the ventilator monitor. From a larger dataset of roughly 70 000 operations performed between 2014 and 2017, two subsets were selected:

1. *Hypotension set:* contained operations during which an anesthesiologist reported hypotension. After preprocessing (described in section 3.2) 83 of such operations were selected. They consisted of over 46 000 timesteps (or almost 200 hours), with 21% of them being labeled as hypotensive by a rule-based system (see section 3.2.5).

2. *Stable set:* contained operations without any registered complications which lasted at least 2 hours and the patient's age was between 20 and 60 years. These restrictions were applied to reject operations with perioperative phases that were too short for the learning algorithms to extract enough meaningful information and operations in which the age of the patient could significantly impact the changes in the vital parameters or the routine procedures in the OR. After preprocessing this set consisted of 632 operations with roughly 500 000 timesteps (2 000 hours) in total.

3.1 Variables

The number of recorded vital parameters differed between operations. Twelve variables were chosen basing on expert recommendations, their explanatory power, and the number of operations in which they were recorded. The following list contains the 12 selected variables and 2 additional features computed in the preprocessing phase.

1. **Heart Rate (HR)** is the number of heart beats per minute. It is a crucial variable for anesthesiologists, as it is sensitive to the body's reaction to anesthetics, pain and respiratory problems amongst others. Heart rate has usually the same or a very similar value as pulse rate.

2. **End-Tidal Carbon Dioxide ($EtCO_2$)** is a measure of exhaled carbon dioxide (CO_2). It can be expressed as partial pressure in millimeters of mercury (mm Hg) or as the percentage of CO_2 in expired air. The latter option is the one found in our dataset. Monitoring of $EtCO_2$, called capnography, is often used for tracking important changes in the cardiorespiratory system or for early detection of respiratory complications, such as hypoventilation.

3. **Fraction of inspired Oxygen (FiO_2)** indicates the percentage of oxygen (O_2) in the inspired air. Natural air includes 21% of O_2 and is the equivalent of $FiO_2 = .21$. However, patients are often given Oxygen-enriched air in which case FiO_2 is between .21 and 1.

4. **Tidal Volume Expired (TVE)** is the volume of expired air in milliliters (ml) which differs according to the weight of the patient. During mechanical ventilation, TVE is used for monitoring if the volume of air pumped into the lungs is equal to the expired volume. It is an important measure which helps to prevent causing trauma to the lungs with inadequate ventilator settings.

5. **Respiratory Rate** is the rate of breathing per time measure, in this case, per minute. Under general anesthesia, the respiratory rate is controlled by the ventilator machine.

6. **Oxygen Saturation (SpO_2)** measures the percentage of oxygen-containing hemoglobin in the total hemoglobin. To function properly the body needs the SpO_2 value to be between 95%-100%. A score below 90% is considered as hypoxemia – often caused by respiratory malfunctions resulting in an improper supply of O_2 from the inhaled air to the vascular system.

7. **Peak Inspiratory Pressure (P_{max})** is the highest level of pressure applied by the ventilator machine during inhalation measured in cmH_2O.

8. **Positive End-Expiratory Pressure (PEEP)** is the positive pressure that remains in the lungs at the end of an exhale and is expressed in cmH_2O. In more descriptive words, during mechanical ventilation, the air is pumped into the lungs by the ventilator machine and passively exhaled by the lungs back to the machine. However, a small positive pressure (in relation to the atmospheric pressure) is applied to the lungs to preserve them from closing completely.

9. **Temperature**, measured in °C, often drops during anesthesia due to the suppression of thermoregulatory mechanisms, administration of cold fluids, or redistribution of warmth to the extremities at the expense of the core temperature. It has to be controlled to prevent hypothermia.

10. **Systolic Blood Pressure (BP_{sys})** is the maximum pressure of blood on the walls of blood vessels during a heartbeat, or more specifically, during a cardiac contraction. Blood pressure, given in mmHg, can be measured non-invasively with an inflatable cuff or invasively through an arterial line. The latter option allows tracking changes at higher fidelity.

11. **Diastolic Blood Pressure (BP_{dia})** is the lowest blood pressure measured in between two heartbeats. BP_{sys} and BP_{dia} are often displayed together (e.g. 120/70).

12. **Mean Arterial Pressure (MAP)** is the average blood pressure during a cardiac cycle. With invasive methods it can be measured directly, while with non-invasive it has to be estimated with BP_{sys} and BP_{dia} as:

$$MAP_{mn} = \frac{1}{3}BP_{sys} + \frac{2}{3}BP_{dia}$$

13. **Dynamic Lung Compliance (C_{dyn})** tells how much a lung can stretch during air movement and indicates whether the lungs are stiff (low C_{dyn}) or flexible (high C_{dyn}). It is not measured directly during an operation, but was computed post-hoc by applying the following formula:

$$C_{dyn} = \frac{TVE}{P_{max} - PEEP}$$

14. **Body weight** gives grounds for a better understanding of the magnitude of changes in other variables, which is especially important when monitoring newborns or obese patients. In files where body weight was not registered, it was roughly approximated with patient' age, basing on equations used in emergency situations (Tinning & Acworth, 2007):

    a. For patients of age over 14 years use average weight in the Netherlands, which for females is 70kg and for males 84kg. If the patient's sex is not available use the average weight of 77kg.
    b. For children between the age of 5 and 14 years: *weight = age in years* $\times$ *4*.
    c. For children between the age of 1 and 5 years: *weight = (age in years + 5)* $\times$ *2*.
    d. For newborns under 12 months: *weight =* (*age in months + 9) / 2*

3.2 Preprocessing

Data registered during operations contains numerous faults, such as extended periods of missing data, artifacts of a non-physiological origin, or vital parameters that were not recorded at all. This setting is far from the perfect datasets used in number-cracking research of ML algorithm; therefore, an extensive preprocessing phase backed by appropriate testing was necessary. Moreover, the performance of some ML algorithms can be boosted by taking extra steps in data preparation. The following section will describe the general preprocessing applied to the dataset.

3.2.1 Perioperative Phase Selection

Even though the dataset included annotated events, such as the time of intubation or the first incision, a more robust method of selecting the perioperative phase was necessary, since anesthesiologist often forget to add appropriate notes or place them incorrectly in time. Our approach was based solely on an analysis of the vital measurements, thus was independent of the hand-made notes.

The perioperative phase was selected by computing a threshold which had to be surpassed by the tidal volume (TVE) variable, as it indicated that the patient had been intubated and mechanical ventilation was engaged. Since patient's TVE can be estimated with *TVE = 7ml* $\times$

weight in kg and the lungs do not need to be filled completely with air, the threshold was set at 30% of patient's TVE and at 15% for newborns. If the level of TVE was maintained over the threshold for at least 5 minutes, this was selected as the starting point of the actual operation; a shorter period was regarded as a testing phase of the ventilator machine. The end of the operation was determined as the time when TVE was below the patient's threshold for at least 10 minutes (in order to disregard any malfunctions of the ventilator). In cases where the patient remained intubated also in the postoperative phase, the end of the operation was determined by the moment when blood pressure measurements were turned off.

3.2.2 Blood Pressure Measurement Selection

Invasive measurement of blood pressure was selected as a default in most cases, as it is more precise and is registered at every update frequency. When it was not available, the non-invasive measure was used instead. However, its values had to be interpolated since it is sampled only every 5 minutes (20 timesteps).

3.2.3 Missing Values

Data from the OR is rarely complete – not all measurements are continuously recorded, equipment does not always work and has to be restarted during the operation, or the staff keeps stepping on cables. However, almost all ML algorithms cannot be trained on data with missing values. For this reason, missing data points had to be imputed. When possible, *akima* interpolation (Akima, 1970) was used to fill gaps of missing data, as it provides smooth transitions between two data points. If interpolation was not possible, the value from the nearest timestep was copied. For situations where a variable was not recorded at all during the whole span of operation, its values were set to 0.

3.2.4 Standardization

As with other ML techniques, NNs perform better when their input data is centered around 0 and scaled to unit variance. This enables the backpropagation algorithm to update the weights of the network more effectively, which results in shorter training time (LeCun, Bottou, Orr, & Müller, 2012). RobustScaler from Scikit-learn library (Pedregosa et al., 2011) was selected as the method of standardization. It uses the median as a central tendency measure and scales the data according to the quantile range, where 50% of data points are within one unit from the median. In other words, the values in a standardized variable are shifted so that the median value of this variable becomes a 0 and are scaled to fit 50% of the observations in a corridor between -1 and 1.

Having selected the standardization technique, it was important to apply it in a way that would preserve the relevant differences between patients. Each variable in every operation had to be first centered to its own median value and afterward, the variables were scaled to the quantile range calculated across all operations. Such an approach removes initial differences in the vital parameters (a 0 can be considered as a baseline for every variable of every patient)

but makes deviations from 0 still comparable across patients. For example, if Patient A generally has high blood pressure (140/100) and Patient B (95/60) has low blood pressure, these initial differences become irrelevant (median blood pressure is set to 0/0 for both of them); however, any changes with regards to the central value can be compared on a new, but common, scale.

One exception from this is the weight variable which had to be centered to the median weight of all operations to prevent it from being always equal to 0. Hence, an individual's weight was represented as the deviation from the median weight of all patients. For example, if the median weight of all patients was 75kg and Patient A weighted 62kg, his/her weight was denoted as -13 and scaled down according to the quantile range of all patients.

This approach to standardizing the dataset gave considerably better results than any other tested combination. We compared the RobustScaler to other techniques available in the Scikit-learn package, such as the popular StandardScaler which makes use of mean and standard deviation instead of the median and percentile range. RobustScaler gave better results than other techniques, as it is less susceptible to outliers which were present in the OR dataset.

3.2.5 Labeling the Data

For most algorithms, a target label was needed regardless of whether it would be used to train a supervised architecture to detect a specific complication or for evaluating unsupervised algorithms. Such a label should indicate if at any given timestep the vital parameters have values that can be considered as a complication. Unfortunately, the only annotations in our dataset concerning complications are made in severe cases, and even then, they do not have a timestamp.

For the needs of this study, a rule-based system, which is a yet unpublished continuation of Ballast's (1992) work, was used to generate the labels. Its algorithm is built on relatively simple mathematical expressions based on expert knowledge. In short, for every time step, the program computes complication indicators by considering multiple factors such as moving averages of past observations measured in different time spans, the time for which a threshold was surpassed by some vital parameter and relationships between the parameters. Since hypotension was the complication to which most attention was given in the current study, it is worth understanding how its labels were generated by the rule-based system.

The calculations performed to produce clinically significant alarms of hypotension are based on arterial systolic blood pressure (BP_{sys}) and focus on two main aspects of the signal – its absolute value and change over time. More specifically, an equation checks whether BP_{sys} is within a safe range of 70 and 140mmHg by outputting a -1 or +1 if the measured value crosses the lower or upper bound respectively. The second set of equations is responsible for monitoring the trend of consecutive measurements. This is done by first establishing a threshold for an acceptable rate of change in BP_{sys} expressed in percentages. This threshold

has a gradually decreasing tolerance to changes depending on the proximity of BP_{sys} to the upper and lower bounds of the safe range. It has, for example, a 20% change tolerance in the mid-range, but a 10% change tolerance for the more dangerous ranges closer to 70 and 140mmHg. Next, for every incoming measurement of BP_{sys}, moving averages are calculated with time constants of 20, 200 and 2000 seconds. This allows creating change indicators which show whether the blood pressure is rising or falling at a fast, medium or slow speed. If any of these change indicators surpass the mentioned change threshold, -1 is outputted for a falling trend and +1 if the trend is raising. To sum up, in the end, there are two indicators for blood pressure related complications with safe output values between -1 and +1: the first indicating if the BP_{sys} is within the safe range between 80 and 140 mmHg and the second saying whether there is a significant falling or raising trend. If at least one of these indicators has a value lower or equal to -1, hypotension is detected.

3.2.6 Data Windowing

In ML, it is useful to think about time-series data in terms of time windows. This is an especially important concept in our study, as we assumed that health complications span across prolonged periods of time. Time windows are simply constructed by applying a sliding window that slices a time-series into smaller equally-sized overlapping intervals (Figure 1). A time-series in the windowed form is described by its three dimensions. *Lookback* is the length of window (equal to the length of the sliding window); *timesteps* is the number of windows or number of steps that need to be taken by the sliding window to cover the initial time-series; *features* indicates how many variables/features are in the time-series – in our dataset it was the 14 vital parameters. Moreover, in the "Hypotension set", for each step of the window X_i, we had a corresponding hypotension label y_i generated by the rule-based system.

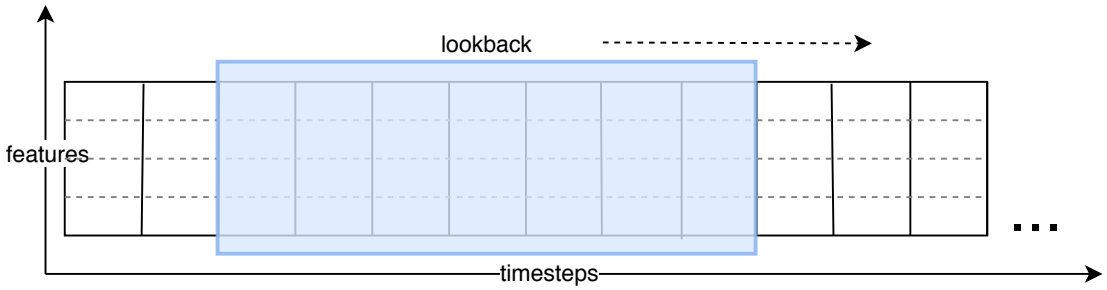


Figure 1. Sliding window that will create time windows with lookback=5, timesteps=6, and features=4.

To give the ML techniques the capability to base their decisions on multiple timesteps, we needed to establish the optimal lookback value. In other words, we needed to choose what interval of past events would be most appropriate for detecting medical complications. After consultations with an anesthesiologist and preliminary tests, 10-minute windows (40 timesteps) were selected. We considered that time-period a good balance between having enough information about the recent vital measurement and not overloading the system with too old and irrelevant events.

4. Complication Detection Approach

The first half of this study focused on investigating how different ML techniques would perform in detecting a single predefined complication, rather than monitoring the patient's health for any possible abnormality (Anomaly Detection is presented in chapter 5). Our Complication Detection approach belongs to the family of supervised learning methods – for every tested ML technique, a model was trained on a dataset with labels that indicated whether a complication was present at a given timepoint; later, in the testing phase, the model was set to generate such complication indicators on data from previously unseen operations.

For the use of this study, the complication that we selected was *hypotension* – often specified in the medical literature as a low level of blood pressure, i.e. $BP_{sys} < 90mmHg$ or $BP_{dia} < 60mmHg$. Hypotension, defined as such, occurs in 64% of all operations (Bijker et al., 2007). It is considered a serious complication, as it can lead to a shock caused by a lack of oxygen and nutrients in the brain or other vital organs if it is overlooked. Moreover, hypotension is among the most often encountered causes of death during anesthesia (Lienhart et al., 2006)

The dataset used for training and testing was the "Hypotension set" that had binary labels generated by the rule-based system (section 3.2.5). It is crucial to note, that the models could not simply learn the hypotension thresholds from the above definition ($BP_{sys} < 90mmHg$) which would be a trivial task. Instead, the more intricate understanding of hypotension implemented in the rule-based system forced the models to reproduce the reasoning about trend changes, similarly to how an anesthesiologist would approach monitoring the blood pressure measurements.

The "Hypotension set" was randomly assigned into three subsets: training set (59 operations, 35 830 timesteps), validation set (12 ops., 4815 ts.) and test set (12 ops., 5633 ts.). The validation set was used for hyperparameters' tuning and was later merged with the training set for final testing.

In the Complication Detection approach, we can distinguish two types of algorithms, these that can only work on specially engineered features/variables and these that operate directly on the windowed time-series data. Below we describe each of the tested ML techniques, starting with the first type that required additional preprocessing.

4.1 Machine Learning on Engineered Features

Since most classic ML techniques cannot process time-dependent relations, a common approach is to extract additional time-based features from the dataset. This reconceptualization of the data was done with a Python package called TSFRESH, which stands for "Time Series FeatuRe Extraction based on Scalable Hypothesis tests" (Christ, Braun, Neuffer, & Kempa-Liehr, 2018; Christ, Kempa-Liehr, & Feindt, 2016). Generally speaking, TSFRESH reduces the time dimension (lookback) by transforming time windows (3-

dimensional data) into single-value features (2-dimensional data). More precisely, TSFRESH first extracts features (e.g. mean, entropy, number of peaks etc.) from the windowed time-series (Figure 2.1); then, performs statistical tests on these new features to see if any of them carry relevant information (Figure 2.2) that could be later used by an ML technique to make decisions about the dependent variable (hypotension label in our study); if so, the new features are kept (Figure 2.3).

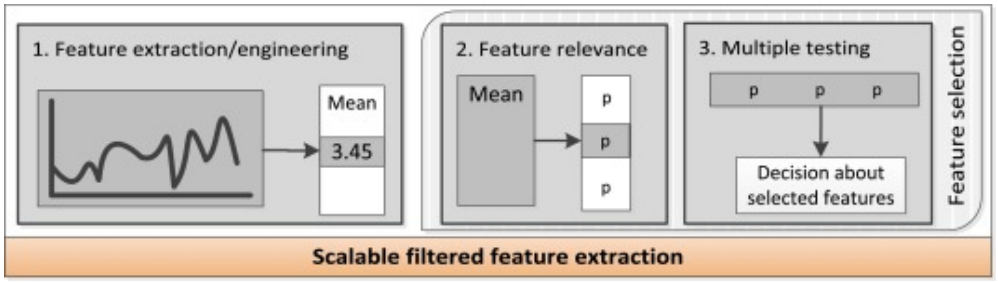


Figure 2. Three steps of feature extraction in TSFRESH package (adapted from Christ et al., 2018).

4.1.1 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a conceptually basic, but mathematically quite complex ML algorithm. It can be described intuitively by saying that it finds a linear combination on the training set that maximizes the distance between class averages and minimizes within-class variance, which results in an optimal separation between two classes. Such dimensionality reduction is helpful for datasets which have a large number of features – it lowers the chance of overfitting and reduces the computational costs. Moreover, it helps to avoid the "curse of dimensionality" – a situation in which a large number of dimensions results in a rapidly increasing problem space making the available data sparsely distributed, that in turn, raises problems regarding the statistical significance of the sample or grouping of similar objects (Friedman, 1997). Even though LDA assumes that both classes have normally distributed instances and identical covariance matrices, it has been proven to be reasonably robust to violations of these rules (Li, Zhu, & Ogihara, 2006). LDA is often compared with another basic and widely used ML technique, namely Logistic Regression, which does not require the two mentioned assumption to be met (Pohar, Blas, & Turk, 2004). Nevertheless, the first one was selected as it gave better results in a study by Kendale et al. (2018), which predicted the chance of a perioperative hypotension from a medical dataset.

4.1.2 Decision Tree & Random Forest

One of the simplest ML techniques are *Decision Trees*, which are often used in an ensemble called a *Random Forest*. A Decision Tree is constructed out of true/false nodes which ask questions about the data until the samples from the training set have been split in a way that the last nodes, called leaves, are pure – they contain samples of only one class (Figure 3). In the training stage, which is performed in a typical supervised setting, a measure known as *Gini impurity* is calculated to indicate the probability of falsely assigning a label to a sample in a node. Afterward, the measure of *information gain* is used to find the best split for a node. In

other words, it determines what question should be asked about which feature to disentangle the classes in the best possible way. More formally, in an iterative process information gain describes the change in impurity between parent and child nodes for all possible splits. As a result, the split that has the largest information gain is selected and the process is repeated for the next nodes until the impurity is equal to 0.

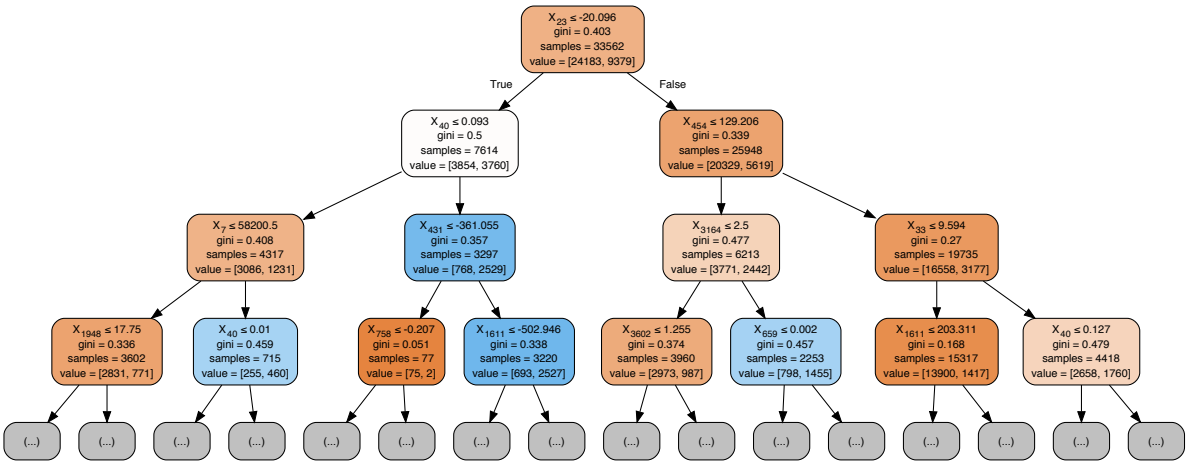


Figure 3. The first 4 levels of a Decision Tree. "Samples" are the number of training samples that fall into a node and "value" is the resulting split on the training set also expressed by the color shading (dark orange indicates that most samples in a node are the 0 or no-hypotension class; the nodes become bluer when instances labeled as 1 or hypotensionsive events are the majority in a node).

One of the biggest disadvantages of a Decision Tree classifier is the ease of overfitting the training set – such a model has a low bias (it is flexible) but a high variance (tries to model random noise during training). Intuitively, it is quite likely that in a deep tree, a new sample will not meet one of the learned conditions and will follow a completely different path in the tree resulting in misclassification. Rather than tuning the classifier to stop overfitting, it is a common practice to train multiple Decision Trees (a Forest) and take their averaged prediction. Every tree in a Random Forest is trained on a subset drawn with replacement from the complete training dataset. Additionally, the best split is no longer optimal across all input features, but it is the best split on a random subset of features. This randomness results in an increased bias of a single Decision Tree that helps to generalize better over the data. On the other hand, it drops the overall performance, which in turn, is compensated by the reduced variance from averaging results from multiple trees, making the Random Forest a better classifier. The largest disadvantage of a Random Forest is its decreased interpretability in comparison to Decision Tree which is an easy-to-trace-back glass-box model.

4.1.3 k-Nearest Neighbors

Another basic ML algorithm is *k-Nearest Neighbors (k-NN)* where in the training phase feature vectors are saved with their labels in a multidimensional space and in the classification phase a new instance is mapped onto that space by computing distances to the training instances.

Then, the program performs a majority vote for the class of the test instance by checking the classes of *k* training instances with the smallest Euclidean distance (Figure 4), given by the formula:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

where *d* is the distance between two points $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$ in *n*-dimensional space, is used as the measure of distance, since it gives comparable results to other metrics and is simple to compute (Hu, Huang, Ke, & Tsai, 2016). One of the drawbacks of k-NN is the fact that it belongs to a family of lazy learning algorithms, which means that the heaviest computations, namely comparing the testing and training instance, are not performed in the training phase, but while predicting the classes. This can posit a serious problem when a big time-series dataset is used, as finding the k-nearest points might take longer than the update rate of new timepoints (e.g. 15 seconds) which would result in jamming the alarming system.

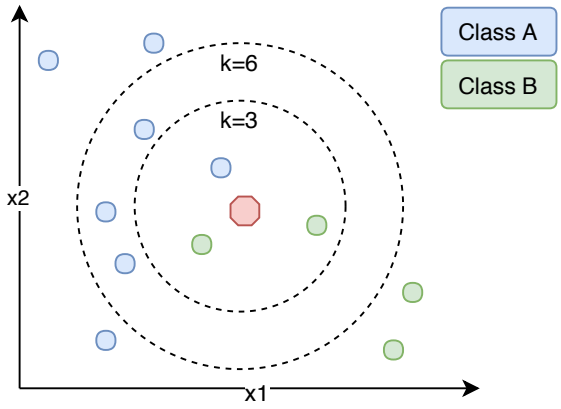


Figure 4. k-NN algorithm in a two-dimensional space on a two-class problem. With k=3 the new sample will be assigned class to B, but k=6 class A has the majority of nearest neighbors.

4.2 1-Nearest Neighbor with DTW

K-Nearest Neighbors is a unique method – not only can it be trained on the engineered features, as shown in the previous section, but after a few adjustments it is also capable of processing sequential data. To achieve the best results on a windowed time-series, multiple studies recommend setting the number of neighbors to *k=1* and using *Dynamic Time Warping (DTW)* as the method of calculating the distance to the nearest neighbor. The resulting *1-Nearest Neighbor with DTW* (*1-NN + DTW)* often outperforms other more complex approaches (Ding, Trajcevski, Scheuermann, Wang, & Keogh, 2008; Mitsa, 2010; Sakoe & Chiba, 1978). By introducing non-linear transformations in the time dimension, DTW is a distance well-suited for measuring the similarity of two time-series, even if they are out of phase or their speed of change varies (Figure 5). For example, two series $X = \langle a, a, b, a \rangle$ and

$Y = \langle a, b, a, a \rangle$ would be considered different according to Euclidean distance, even though there clearly is a similar trend – this aspect would be captured by DTW. In order to investigate whether Complication Detection would benefit from using DTW, we compared 1-NN + DTW with 1-NN + Euclidean in our study.

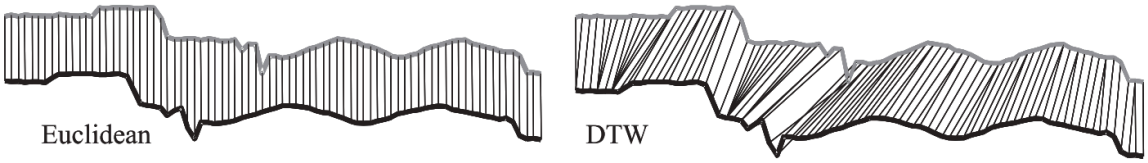


Figure 5. Comparison of Euclidean distance and DTW on two similar, but shifted in time, sequences. (adapted from Keogh & Ratanamahatana, 2005)

4.3 Basic principles of Neural Networks

Neural Networks are another family of ML techniques that can process sequential datasets without the need of additional feature engineering. Because of their high popularity in the recent years and multiple architectural variants, we provide an in-depth description of their workings in combination with time-series datasets.

4.3.1 Artificial Neurons

Artificial Neural Networks are constructed from simple units which work together similarly to the way in which neurons in the human brain hierarchically propagate electrical potential through the nervous system (Figure 6A). An artificial neuron is a function $y = a(w \cdot x + b)$, where *a* is an activation function, $w \cdot x$ is the dot product of weights and inputs of the neuron, and *b* is the added bias. In more graphical terms, the general shape of the neuron's function is determined by *a* which can be, for example, a sigmoid function (Figure 6B). The steepness of the function is determined by *w* and its horizontal shift by *b* (for more details see: Nielsen, 2015).

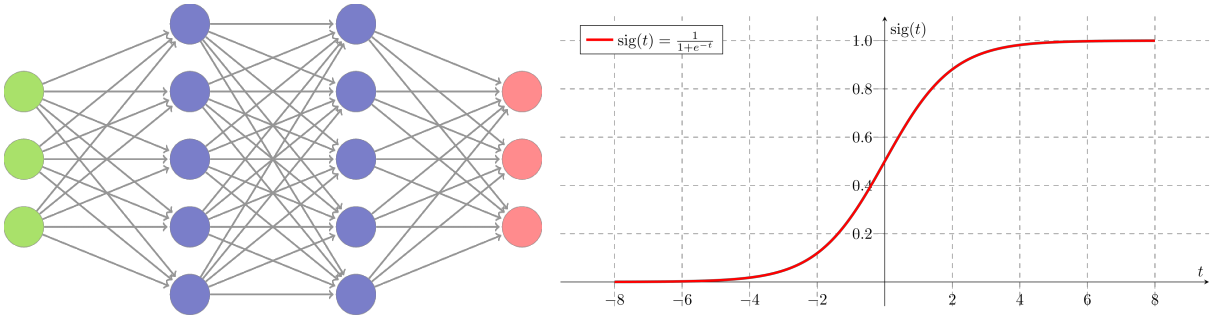


Figure 6. A. Deep Feedforward Neural Network consisting of three input neurons, two hidden layers with five neurons per each layer and three output nodes (adapted from Holzinger, Malle, et al., 2017).
B. Sigmoid function

4.3.2 Learning in Neural Networks

Two important properties of a Neural Network are the *loss function* and *optimizer*. The loss function is a measure of the network's predictive capability. It is calculated between the output generated by the network, which is a class probability *p*, and the expected output *y'* given by a corresponding label. The loss function used in this study was *cross-entropy*, also known as *log-loss* in case of binary classification:

$$c = -(y' \log(p) + (1 - y) \log(1 - p))$$

For example, for a positive class *y'*=1 (hypotension is present) and a prediction *p*=0.37, the right-hand side of the addition sign is 0; therefore, the loss is $c = -\log(p)$, so $c = -\log(.37) = 0.43$. Binary cross-entropy is simply a vertical reflection of a logarithmic function, which penalizes more values deviating largely from the target (Figure 7).

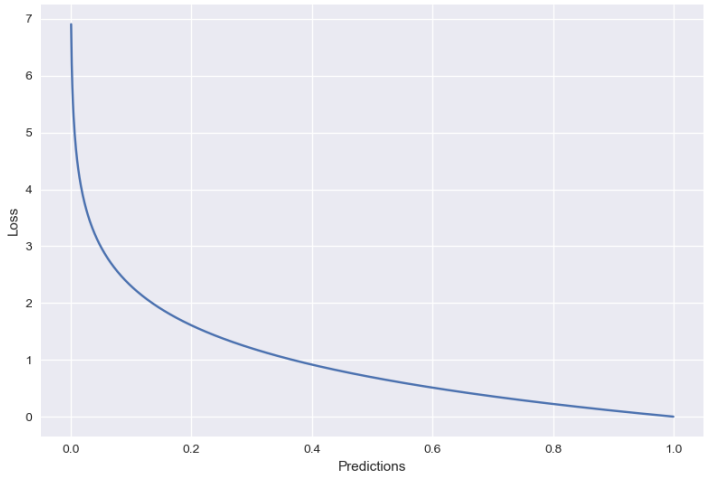


Figure 7. The log-loss function for a positive class.

While the loss function is an internal measure that allows the network to evaluate its performance on the training data, the optimization function, and more precisely *gradient-based optimization,* aims to minimize this loss by adjusting the network's parameters (weights and biases). This is done by applying a *Backpropagation* algorithm to calculate the gradient of the loss function with regards to the parameters. The *gradient* is a multidimensional generalization of a derivative which indicates the direction of the greatest increase of the loss function. In other words, Backpropagation tells how a small change in each parameter impacts the loss function. Hence, by combining these small changes, it is possible to determine the gradient – a compound adjustment in the weights and biases that will result in the biggest increase of the loss function. Thus, making an adjustment in the parameters in the opposite direction from the gradient (known as *gradient descent*, see Figure 8) results in minimizing the loss and an increase in the performance of the network. How small this adjustment or step towards the function's minimum is depends on an *optimizer* – different optimizers take under consideration different properties to adjust the steps towards the global minimum of the loss

function, such as previous gradients or previous weight updates. A good optimizer should be able to escape local minima and find the global minimum within a reasonable number of steps.

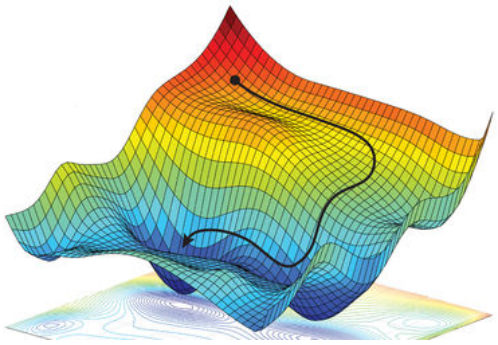


Figure 8. Gradient descent in a three-dimensional space (adapted from sciencemag.org).

4.3.3 Fully-Connected Feed-Forward Neural Network

Before testing more sophisticated architectures, it is worth establishing a baseline with a *Fully-Connected Feed-Forward Neural Network* which uses classic neurons described in a previous section (4.3.1). All its units in one layer are connected to all units in the contiguous layers (Figure 6A). Most importantly, such an architecture does not process the time domain; therefore, its input must be passed through a *Flatten* layer which reduces the lookback and feature dimensions into a single vector of attributes. A common practice is to apply *Dropout* to Fully-Connected layers – a regularization technique that randomly deactivates a certain proportion of units in a layer (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). For example, if a layer has a dropout rate of 0.5 applied, it means that it will operate using only half of its units during an update cycle. This method is only used during the training phase to prevent co-adaptation of neurons, or in other words, to increase the independence of each neuron which results in a network that overfits less the training data. Another, often encountered in the literature, practice is to apply a *Rectified Linear Unit (ReLU)* activation function to Fully-Connected layers in order to speed up the training of a network (Ramachandran, Zoph, & Le, 2017). ReLU's biggest advantage is its simplicity which can be seen in its formula $f(x) = \max(0, x)$. Because the gradient of this activation function can only be equal to 0 or 1, its calculation is easier than in the case of the previously mentioned sigmoid function.

4.3.4 Recurrent Neural Network (LSTM)

A *Recurrent Neural Network (RNN)*, as opposed to a Feedforward Network, has the ability to remember the previous inputs by maintaining a *state*. It is useful to imagine that when a time window is passed to an RNN, it creates a copy of itself at every timestep. Such a copy produces an output, but also passes its state to the next copy (Figure 9). Thus, at timestep *t* the hidden state h_t of RNN is:

$$h_t = a(Ux_t + Vh_{t-1})$$

with a being an activation function – usually a hyperbolic tangent function (tanh), U the weight for input x_t and V the weight for the previous hidden state h_{t-1}. The output o_t at each timestep t is defined as:

$$o_t = W h_t$$

where W is the weight matrix for the current hidden state h_t. It is often the case that only the output o_t from the last of t iterations is of interest for a classifier, as it conveys information from all the previous timesteps; however, if multiple RNN layers are stacked, the whole output sequence $o_t, o_{t-1}, \ldots$ is used as input for the next recurrent layer.

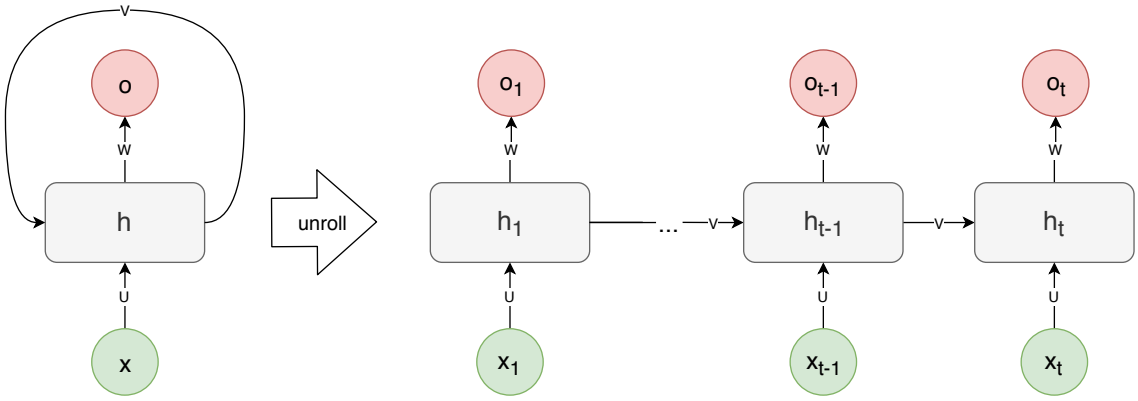


Figure 9. Recurrent Neural Network, with input x_t and output h_t unrolled in time.

RNNs, as defined above, suffer from *the vanishing gradient problem* which arises when the number of timesteps is too large and the network cannot learn dependencies separated by big time intervals (Bengio, Simard, & Frasconi, 1994). Therefore, in theory an RNN should be able to learn that output o_{35} is linked to the input x_4 and x_5, but in practice in *Backpropagation Through Time* the gradients in the early timesteps become so small, that these units are untrainable.

Fortunately, the concept of an RNN was enhanced by *Long Short-Term Memory (LSTM)* networks (Hochreiter & Schmidhuber, 1997) which are capable of capturing long-term dependencies (Figure 10). There are several intriguing innovations in the LSTM worth mentioning with the most important being the *cell state* c_t, also called the *carry*, which runs from one copy to another – it is useful to think about it as of a "conveyor belt" that transports information across timesteps. The information in the cell state can be modified with specialized gates (f_t, i_t & o_t in Figure 10) which are additional layers with sigmoid activation functions followed by pointwise operations – addition and multiplication (the latter is called the Hadamard product – denoted as "$\times$" further on).

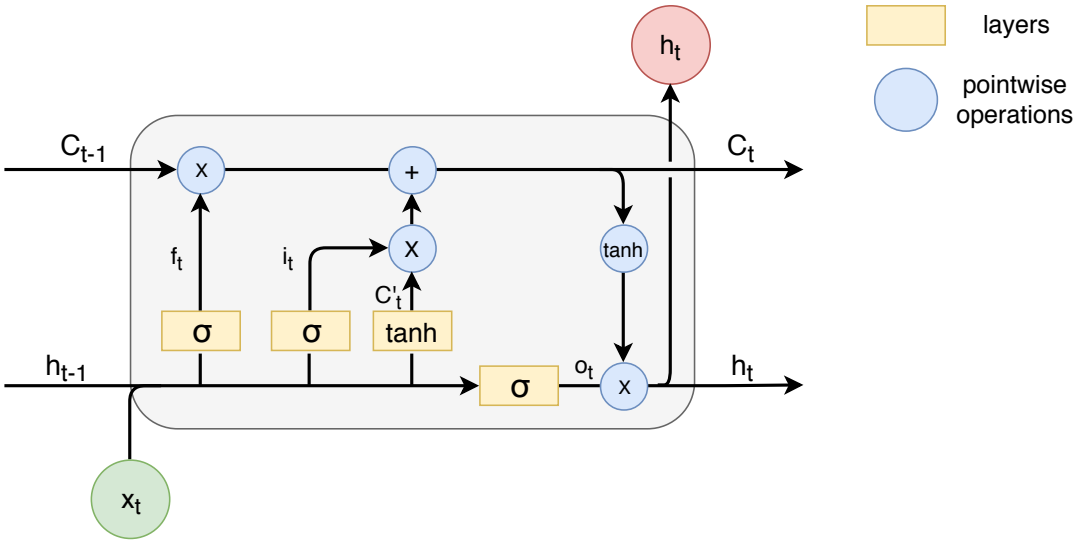


Figure 10. LSTM unit, where f_t, i_t & o_t are the gates c'_t is the candidate values.

The activation functions in the three gates are sigmoid functions; hence, their outputs are within the 0 to 1 range (Figure 6B). That allows the first *forget gate* f_t to decide how much of which information from the *cell state* c_{t-1} should be dropped or let through, based on the current input x_t and the previous hidden state h_{t-1}. Since the sigmoid forget layer is multiplied pointwise with c_{t-1}, a 0 is equivalent to completely erasing a carried value and 1 would indicate to leave it unchanged. The mathematical formulation of the forget gate f_t, where W and U are the weight matrices for current input x_t and previous hidden state h_{t-1} respectively, and b is the bias, would be:

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right)$$

Next, the *input gate layer* i_t selects which values in the cell state c_{t-1} will be updated. It applies the same sigmoid layer and pointwise multiplication mechanism as the forget gate; however, this time it is not multiplied directly with the cell state, but with *candidate values* c'_t from a tanh layer which outputs the current input x_t and previous hidden state h_{t-1} squished into a range between -1 and 1.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$c'_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

Hence, after the input gate layer i_t has decided which of the candidate values c'_t should be passed to what extent, a new cell state c_t is calculated by adding new values to c_{t-1} already prepared by the forget gate f_t.

$$c_t = f_t \times c_{t-1} + i_t \times c'_t$$

The final operation is selecting the new output h_t, again with a sigmoid layer, called the *output gate* o_t, weighting relevant values from the current input x_t and the previous hidden state

h_{t-1}. However, this time they are multiplied by the new cell state c_t passed through a tanh layer, so that the output has values within the range from -1 to 1, rather than just from 0 to 1.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

4.3.5 1-D Convolutional Neural Network

Convolutional Neural Networks (CNNs) are most often associated with image recognition since by applying special filters that slide across images they can detect features – from basic ones in the early layers, to more complex patterns in the deeper layers. These convolutions use 2-dimensional kernels, although in the recent years, applying 1-dimensional convolutions to time-series data is becoming more popular due to the fact that CNNs have fewer parameters that have to be trained than Fully-Connected Networks (Zhao, Lu, Chen, Liu, & Wu, 2017; Zheng, Liu, Chen, Ge, & Zhao, 2014). Moreover, 1-D CNNs should be more sensitive than an LSTM to specific patterns that occur even at an early stage of a sequence.

A *1-D Convolutional Layer* consists of multiple filters that slide on the input sequence. All filters have the same width, meaning that they can consider only a limited number of timesteps at a time. However, every filter is sensitive to a different pattern in the data. Therefore, a Convolution Layer determines whether a pattern has been detected at each step of the filter (Figure 11, top).

A common practice is to add a *Pooling Layer* after a Convolutional layer to further reduce the dimensions of the data which helps in preventing overfitting the training data. These layers are simply constructed by sliding a pooling window of a given width (usually 2 or 3) on the Convolutional Layer and taking the largest number in the window or their average – depending if the pooling operation is MaxPooling or AveragePooling (Figure 11, middle).

A stack of multiple Convolutional and Pooling Layers, which is a good strategy for obtaining higher-level features, has to be flattened into a single vector that can be inputted into a Fully-Connected layer in order to generate class predictions (Figure 11, bottom). The Fully-Connected layer might also be substituted with a recurrent layer, such as an LSTM layer. This should theoretically help the network with not only basing its predictions on pattern presence but also on the time dependency of these patterns. A CNN version with and without a LSTM layer was tested in this study.

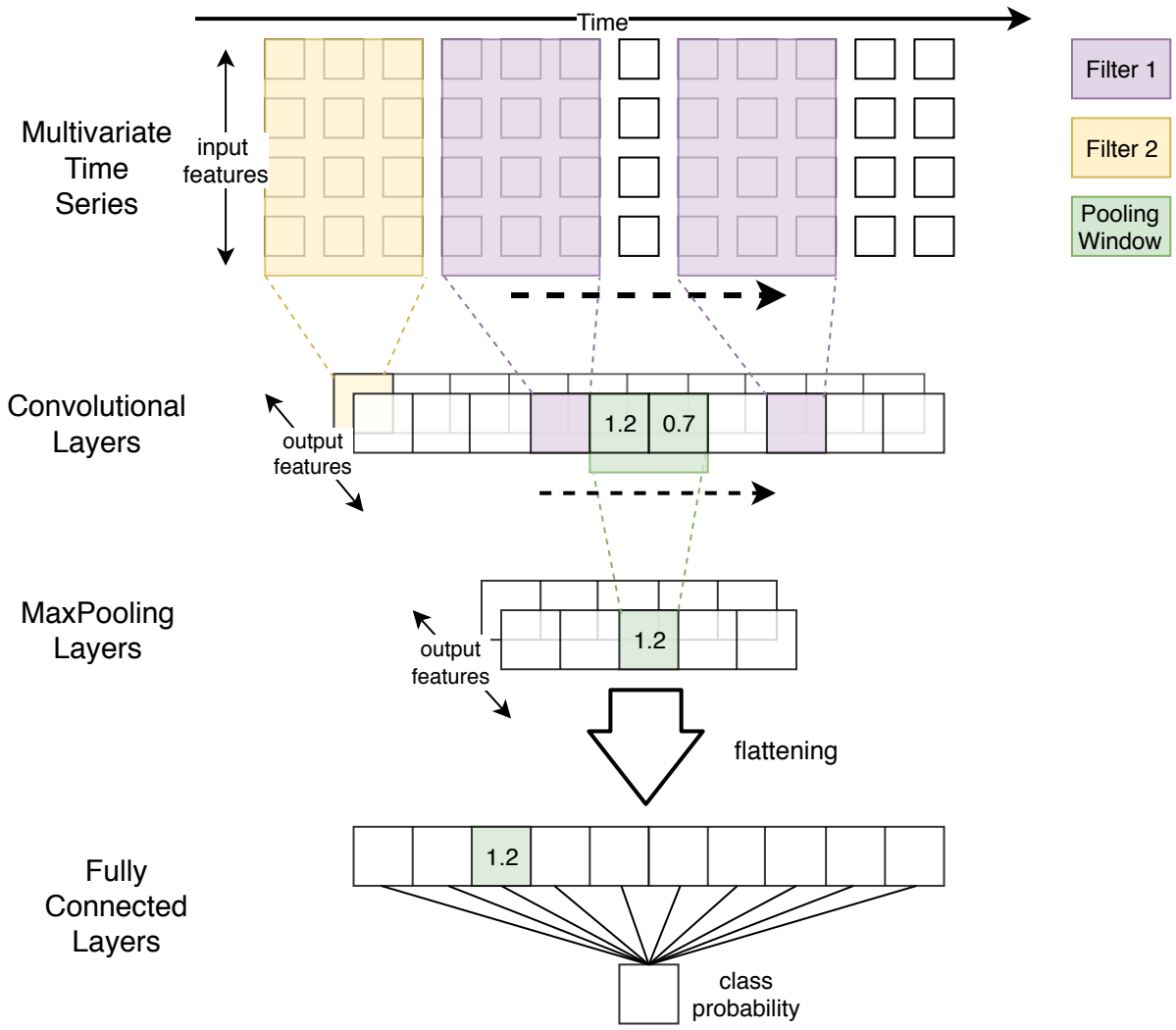


Figure 11. 1-D Convolutional Layers constructed by sliding two filters of size 3 on a time-series with 4 features, MaxPooling Layers with window size of 2 and a Fully-Connected Layer. The dimensions of the resulting data have been reduced from 12×4 to 2×5 and flattened to 1×10.

4.3.6 Experimental Setup for Neural Networks

K-Fold Cross Validation is a testing method that allows obtaining stable results, which are less dependent on random weigh initialization in the early training phase and helps to partly compensate for the bias of a small dataset. Using k-Fold Cross Validation means that a NN is trained *k* times on a dataset constructed by merging together the training and validation sets. Each of these *k* independent experiments is evaluated on $1/k^{th}$ of the expanded set (Figure 12). The selected *k*-value should provide an optimal trade-off between acceptable computational costs and non-biased performance scores. Setting *k* to a too high value results in unfeasibly high training times, while a lower *k* makes the final scores more dependent on the random weight initialization.

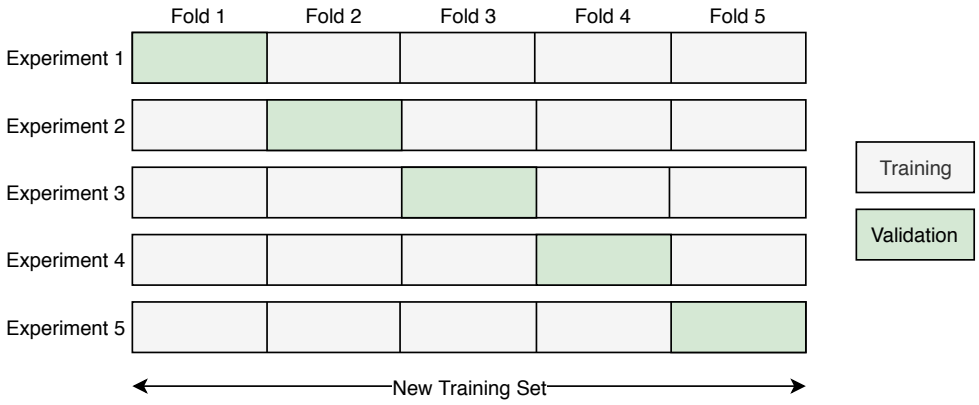


Figure 12. k-Fold Cross Validation, with k=5.

Evaluating a model using k-Fold Cross Validation means that the scores obtained in each of the *k* runs must be combined together to provide a final indicator of the network's performance. It is often recommended that the final evaluation metrics (in our case: precision, recall, and F1-score – see Chapter 2.6.2) are computed using a micro-averaging technique, in which the resulting confusion matrices from each of the *k* experiments are summed together at the end of the whole k-Fold Cross Validation process and only then the final metrics are calculated (Forman & Scholz, 2010). Such an approach allows avoiding averaging the results across the *k* experiments which is known to introduce bias to the final measures.

4.4 Results

4.4.1 Implementation

Linear Discriminant Analysis, Decision Tree, Random Forest, and k-Nearest Neighbors were implemented using *Scikit-learn* Python library (Pedregosa et al., 2011) and their hyperparameters were left to their default values, unless stated otherwise in Table 1. Since these techniques required the engineered features as an input, the feature extraction procedure was performed with use of the TSFRESH library, which resulted in 3604 new features instead of the 14 standard sequential variables.

1-NN with DTW and Euclidean distances was implemented in *tslearn* Python package and, due to significant computational costs, had to be run on the Peregrine HPC cluster owned by the University of Groningen.

All Neural Networks were implemented using *Keras* Python library with *tensorflow* backend and run on a Nvidia k40 GPU from the Peregrine HPC cluster. Table 1 presents the best-performing architectures layer-by-layer. Tuning of the hyperparameters was done in the k-Fold-Cross-Validation experimental setup with *k* (the number of folds and experiments) set to 5. All networks in this study were trained using the *Adam* optimizer (Ting et al., 2010) which, after comparisons with other simpler optimizers available in the Keras library, proved to support the most optimal convergence to the minimum of the loss function.

Table 1. Hyperparameters of the best performing complication detection methods.
Disclaimer: when no information is provided, the hyperparameters were left to their default values.

Features	*Algorithm*	Hyperparameters
Engineered	*Linear Discriminant Analysis*	-
	Decision Tree	-
	Random Forest	number of estimators/trees = 75 (see Appendix 1 for a comparison with other values)
	k-Nearest Neighbors	k=1 (see Appendix 2 for a comparison with other values)
Standard (sequential)	*1-NN + Euclidean*	-
	1-NN + DTW	-
	Fully-Connected Network	1. **Flatten layer** – 560 units, dropout=0.2 2. **Hidden layer** – 700 units, activation=ReLU, dropout=0.5, 3. **Output layer** – units=1, activation=sigmoid
	LSTM Network	1. **LSTM layer** – units=16, dropout=0.2, recurrent_dropout=0.5 2. **Fully-Connected layer** – units=32, activation=ReLU, dropout=0.5 3. **Fully-Connected output layer** – units=1, activation=sigmoid
	Convolutional Network	1. **1-D Convolutional layer** – filters=16, filter width=2, activation=ReLU 2. **Max Pooling layer** – window size=3 3. **1-D Convolutional layer** – filters=32, filter width=2, activation=ReLU 4. **Max Pooling layer** – window size=3 5. **Flatten layer** – dropout=0.4 6. **Fully-Connected layer** – units=32, activation=ReLU, dropout=0.4 7. **Fully-Connected output layer** – units=1, activation=sigmoid
	Convolutional LSTM Network	1. **1-D Convolutional layer** – filters=16, filter width=2, activation=ReLU 2. **Max Pooling layer** – window size=3 3. **1-D Convolutional layer** – filters=32, filter width=2, activation=ReLU 4. **Max Pooling layer** – window size=3 5. **LSTM layer** – units=8, dropout=0.5, recurrent_dropout=0.5 6. **Fully-connected output layer** – units=1, activation=sigmoid

4.4.2 Final Scores

The scores achieved by the ML techniques were compared to a non-ML baseline (first section of Table 2). Several naïve classifiers were tested for this purpose (Majority Guess, Positive Guess, and Random Guess). Positive Guess achieved the highest F_1 score out of these methods ($F_1 = 0.35$); hence, it was used as the baseline that had to be surpassed by an ML technique in order to state that it learns meaningful patterns from the data. To provide objective final scores, that were not biased by any random initialization processes, we trained each model 5 times and tested each of these iterations on the test set. Scores from these 5 runs were then micro-averaged to obtain the final results presented in Table 2.

The final F_1 scores, as presented in the middle section of Table 2, show that out of the algorithms trained on the engineered features, Linear Discriminant Analysis and k-Nearest Neighbors performed below the baseline ($F_1 = 0.2$ and $F_1 = 0.18$ respectively). Moreover, Linear Discriminant Analysis generated a warning concerning collinearity between variables which compromises its performance. The Decision Tree scored slightly above the baseline ($F_1 = 0.38$) and the Random Forest was clearly the best performing ML technique on engineered features ($F_1 = 0.49$). The resulting Random Forest model was additionally inspected by checking which features contributed most to the final outcome. The most important were those that contained information about Systolic Blood Pressure, Diastolic Blood Pressure and Mean Blood Pressure. More precisely, the most contributing feature was a mean of consecutive changes inside a quantile corridor set between 1 and 0 of the BP_{sys} variable.

The last section of Table 2 contains results from the ML techniques that operate on sequential input. Both 1-Nearest Neighbors, regardless of the distance measure (Euclidean or DTW), performed below the baseline ($F_1 = 0.31$ for both). In contrast, all tested Neural Networks scored above the baseline, with the Fully-Connected architecture performing the best ($F_1 = 0.56$). LSTM Network and Convolutional LSTM obtained similar, yet lower scores ($F_1 = 0.52$ and $F_1 = 0.51$ respectively). CNN was the worst of the Neural Networks with a F_1 score of 0.48. Neural architectures with a convolutional base (CNN and Convolutional LSTM) achieved higher recall scores in comparison to the Fully-Connected and LSTM Networks, but lacked in precision.

Table 2. Results of Hypotension Detection algorithms.

Features	*Algorithm*	Precision	Recall	F_1	AUC
None (baseline)	*Positive Guess*	0.21	1	0.35	0.50
Engineered	*Linear Discriminant Analysis*	0.17	0.23	0.20	0.50
	Decision Tree	0.28	0.57	0.38	0.64
	Random Forest	0.37	0.73	0.49	0.73
	k-Nearest Neighbors	0.14	0.23	0.18	0.47
Standard (sequential)	*1-NN + Euclidean*	0.28	0.33	0.31	0.58
	1-NN + DTW	0.28	0.35	0.31	0.58
	Fully-Connected Network	0.51	0.63	0.56	0.82
	LSTM Network	0.45	0.61	0.52	0.81
	Convolutional Network	0.37	0.67	0.48	0.79
	Convolutional LSTM Network	0.40	0.70	0.51	0.82

Additionally, Figure 13 displays a precision-recall trade-off for the best performing technique – the Fully-Connected Network. In this plot, precision and recall vary according to a threshold which defines whether the outputted class probability is an indicator of a hypotensive event. In all NNs this threshold was set to 0.5 by default, so when the neuron in the output layer produced a value below 0.5 hypotension was not present; if the score was equal or above 0.5 hypotension was detected. The plot depicts how recall and precision would change if that threshold was altered. It can be observed that in order for the Fully-Connected Network to obtain a recall score of 1, precision would have to decrease to around 0.21. This means that the classifier turns basically into the Positive Guess baseline when the threshold is set to a low value.

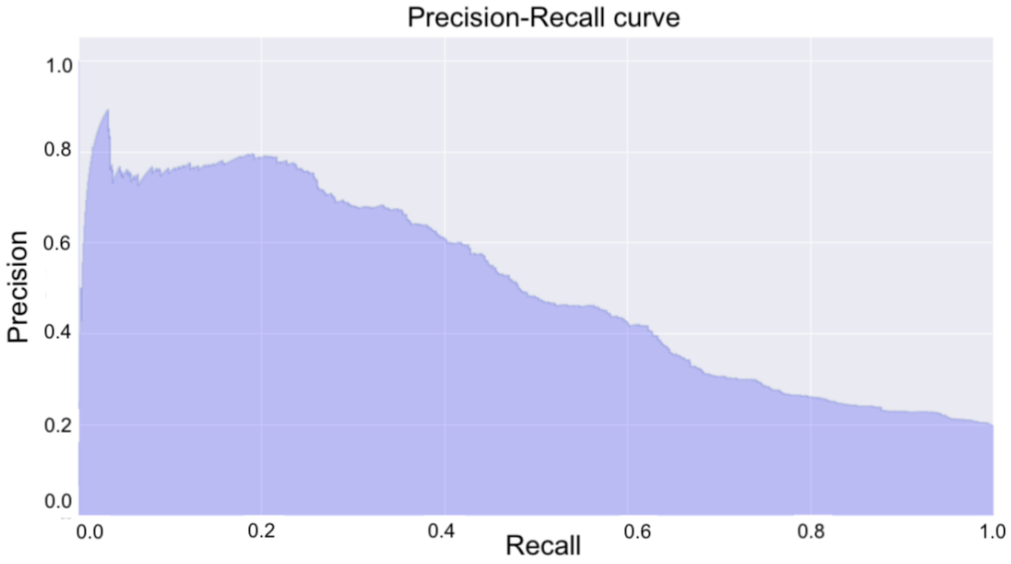


Figure 13. Precision-Recall curve of the Fully-Connected Neural Network.

4.5 Discussion

4.5.1 Basic ML Solutions

The above results clearly show that the simplest ML techniques are not appropriate for capturing the dynamics of the patient's blood pressure measurements. It can be argued that Linear Discriminant Analysis did not perform well due to the fact that the decision space of the given problem cannot be simplified from hundreds of features into a single dimension. Moreover, the fact that collinearity exists between some of the vital parameters is an inherent characteristic of such physiological measurements; hence, Linear Discriminant Analysis is simply not an adequate approach to the discussed problem.

Another tested technique that seems to suffer from a large number of engineered features is the k-Nearest Neighbors. It seems reasonable to assume that this simple algorithm, based on calculating the distance between two data points, failed since by design it does not prioritize features that could carry relevant information for detecting hypotension. Additionally, this approach is generally susceptible to noise that is only being amplified by feature engineering.

It seems that these issues can be overcome by the Decision Tree and the Random Forest. These methods were more successful because during the training period they searched for a feature that provided an optimal class split at every step. Moreover, inspecting the most relevant features showed that the classification performed by the Random Forest was mostly based on engineered features of the blood pressure measurements and more specifically the mean change in the 10-minute window. This indicates that the Random Forest was able to successfully extract the relevant information that represented the general intuition of how an anesthesiologist would think about monitoring blood pressure in terms of trend changes. It seems reasonable to assume that fine-tuning the hyperparameters of this method would

slightly increase its performance. At the same time, its decent performance with the default settings suggests that it would be interesting for future studies to follow the lead of Decision Trees working in an ensemble and to test Gradient Boosting Machines which were proved to perform even better than Random Forests on medical datasets (Kendale et al., 2018). Briefly speaking, Gradient Boosting Machines are simply a different approach to optimizing an ensemble of Decision Trees that is more cumbersome to tune; therefore, was not included in this broad study, but seems a good choice for a more narrowed research project.

Lastly, it should be noted that these basic ML techniques that work on engineered features would likely perform better in general if the feature extraction process was less automated and more tailored for this particular problem. On the other hand, preparing such features is a labor-intensive task, as it requires close cooperation between experts from several domains – in this case, anesthesiology and statistics. Hence using an automated tool (the TSFRESH library) seems a justifiable simplification, especially when taking under consideration the wide scope of our study.

4.5.2 1-Nearest Neighbor

Moving to ML techniques trained on sequential input, it seems that the fact that 1-Nearest Neighbor with Euclidean distance scored higher than the same algorithm on engineered features proves again the point that the k-Nearest Neighbor method is not capable of dealing with a large number of variables after the feature extraction procedure. Nevertheless, 1-NN + DTW and 1-NN + Euclidean did not score above the naïve baseline, so most likely the k-Nearest Neighbor technique, regardless of the input format or distance measure, is too simple to capture the relevant differences between hypotensive and non-hypotensive periods. Moreover, the fact that 1-NN + DTW scored very similarly to the version with the Euclidean metric leads to a conclusion that such a poor performance of the 1-Nearest Neighbor is not caused by how the distance between two sequences is calculated, but rather by the general idea performing such a comparison. At the same time, it cannot be assumed that DTW is not a better distance measure for sequences than the Euclidean metric because the underlying algorithm, with which these metrics were meant to be compared, turned out to be a flawed approach on this particular dataset. Therefore, the k-Nearest Neighbor in any form approach should be rejected as a potential ML solution for detecting complications in data from the OR, even though its implementations with the DTW distance measure were documented to perform exceptionally on other time-series datasets (Ding et al., 2008).

4.5.3 Neural Architectures

The most promising results were obtained by the Neural Networks. All of the four tested architectures performed similarly. The Fully-Connected network performed best on the test set, as it did not overfit the training set as much as the other neural architectures did (concluded by inspecting the loss function values on the validation and training sets). Therefore, it is not a surprise that the networks with a convolutional base scored lower – it is

often observed that these architectures require a large training dataset to prevent them from overfitting. Contrary to initial beliefs, networks with LSTM units did not outperform the simpler Fully-Connected design, even though it was expected that the latter would struggle to capture time-dependent changes in the vital parameters. Although such outcome is difficult to fully explain, it can be hypothesized that changes in the vital parameters that are most relevant for classifying a time window as hypotensive happen usually in the most recent measurements, and the LSTM's extended ability to "remember" previous timesteps is excessive for this particular problem. On the other hand, decreasing the size of the time window from 10 minutes to 5 minutes resulted in generally lower scores which would suggest that the earlier measurements do also contribute to the final class prediction.

According to the AUC measure, the best performing ML methods were the Fully-Connected Network (AUC=0.82) and Convolutional LSTM (AUC=0.82). The fact that the latter architecture had a very good AUC score, while its F_1 score was the third-best, was most likely the result of the technique generating a large number of false positives, which were compensated by an abundance of true negatives while computing the false negatives rate for the AUC ($false\ positive\ rate = false\ positives/(false\ postives + false\ negatives)$).
Moreover, the AUC was boosted, in this case, by a high recall score, which shows that the AUC might be a biased indicator of model's performance in an unbalanced dataset.

A deeper comparison of the alarms generated by the four neural architectures does not indicate that any of them has more potential than the others in monitoring patient's health, as they all react well to fluctuations in the measured variables. Hence, it can be assumed that the differences in the final scores reflect how different types of neural networks perform on a small training time-series dataset, rather than which architecture actually works best for complication detection. Surely, the best-suited neural architecture could be determined by running these tests on a larger dataset. At the same time, the conducted experiments proved that NNs are sensitive to different characteristics of changes that occur in the vital parameters over time, such as the rate and magnitude of a change. Moreover, additional tests were run in which the networks were passed extra variables that contained moving averages of blood pressure measurements; however, they did not impact the results in any way. This suggest that the networks do not need engineered features, as during the training phase they learn to extract such features without any external help.

A visual inspection of the probabilistic outputs of the networks clearly shows that detecting hypotension conceptualized in an expert-like manner is generally an easy task for such ML techniques. It can be clearly observed that operations which contain less noise are modelled almost perfectly by the NNs and the resulting alarms are of high medical relevance (Appendix 3 & 4). More precisely, the operations that did not posit difficulties for the networks were ones where the blood-pressure measurements had been taken from an arterial line, fewer data points were missing and artifacts with a non-physiological background were not dominating the actual relevant measurements. One can imagine that if deep learning alarming

systems were to be used during actual operations, the staff would have to ensure that the recorded data is reasonably uncorrupted and sampled with a sufficient update frequency. On the other hand, this is not always possible since placing an arterial line instead of a non-invasive measurement introduces a risk of other medical complications or may be the cause of incorrect results when the line becomes partially blocked by a blood clot. Therefore, in operations where patient's safety posits certain constraints on the quality of the recorded data, we have to accept the fact that more false alarms will be generated.

It should be noted that in order to reduce the influence of the missing data points that have been filled-in during the preprocessing phase, additional experiments were made. The regular dataset with the 14 vital parameters was extended with 14 binary features which indicated whether a value had been measured during an operation or was inputted post-hoc. Nevertheless, these potential improvements failed to impact the classification scores of the networks in any way, which indicates that this method of dealing with missing data is not effective.

4.5.4 Complication Detection Conclusions

Summing up the complication detection section, it seems reasonable to state that two families of ML techniques, namely ones based on Decision Trees and Neural Networks, should be of particular interest for patient monitoring during anesthesia. Both of them are well-suited for the task since most of the computationally-expensive processing is done during the training phase meaning that the class predictions are generated almost instantly even when run on a basic CPU. Out of these two ML techniques, NNs seems to be more flexible since they do not need to rely on specially engineered features.

The biggest limitation of this section was the relatively small number of training instances which resulted in problems with overfitting and made it impossible to indicate the most promising neural architecture. Despite this issue, it was shown that approaching patient monitoring with a supervised learning methodology has the potential to succeed if basic measures are taken to provide the ML technique an input of a decent quality.

5. Anomaly Detection Approach

In the previous Complication Detection section, we showed that monitoring patient's health in search of a specific complication is an effective but difficult task, mainly because a large labeled dataset is needed to train the supervised ML algorithms. Due to this limitation, expanding such a system with more complications that do not occur as frequently as, for example, hypotension might be unfeasible. Therefore, we decided to investigate another, more universal, approach that should be able to overcome the issues related to the necessity of having a labeled dataset for traditional supervised learning. This second approach, namely Anomaly Detection, is based on an idea that an ML technique that was trained on a dataset without any complications, should be able to model a standard non-anomalous behavior of the vital parameters. In other words, such an ML model should be capable of explaining the normal changes in the patient's measurements during anesthesia. Following this assumption, any measured value that does not match the predictions or expectations of the model is considered an anomaly. An anomaly in our understanding is simply another term for a medical complication, but in order to distinguish any unspecified health abnormalities from a specific well-defined complication, such as hypotension, we decided to use the word "anomaly" in this section. This also means, that we expected the techniques from the Anomaly Detection approach to be able to detect all complications (so hypotension among others). However, we did not assume that such techniques would be capable of explicitly naming the detected anomaly since they operate on a more general level, where the incoming time-series patterns can be classified only as normal or anomalous.

In order to learn the normal behavior of the vital parameters, the ML algorithms from the Anomaly Detection approach were trained on the "stable" dataset. It must be however noted, that this dataset certainly contained some short periods during which the patient was not perfectly stable, because even in routine operations minor complications that require an intervention from the anesthesiologist are present. However, we believe that such imperfections would be averaged out by an abundance of periods with stable/normal behavior. Evaluation of the Anomaly Detection systems was performed on the "stable" dataset and the "hypotension dataset" since we expected the algorithms to provide meaningful alerts in both types of operations – routine and the more demanding ones. The "stable" dataset was split into a training set (70% of the total 632 operations), a validation set (15%) and a test set (15%). For additional tuning of the anomaly detection algorithms, 10 operations from the validation set were randomly selected and manually inspected by an anesthesiologist in search of any abnormal periods during which an expert would expect an alert. When testing the algorithms, an alarm was considered as correct if it was within a 5-minute range from the timestep marked by the expert – that decision was motivated by the fact that it is difficult to determine precisely the beginning of an anomalous event since such changes usually happen over a prolonged period of time. The same expert-tagging procedure was repeated for the test set.

5.1 k-Means Clustering

The most renowned unsupervised learning algorithm is arguably *k-Means Clustering* which takes a set of data points as an input and a value of *k* in order to cluster the points into *k* sets. This is done by first placing *k centroids* (also called cluster means or cluster centers) in the problem space and iterating between two steps:

1. Assign each point x_i from the training set to the nearest centroid c_j $(j = 1, \dots k)$ by applying the rule $\arg\min_j D(x_i, c_j)$, where *D* is a distance measure (Figure 12.1 & 12.3).
2. For each cluster $j = 1, \dots k$ calculate a new centroid c_j which is the mean of all points in cluster *j*, so that $c_j(v) = \frac{1}{n_j}\sum_{x_i \in c_j} x_i(v)$ where *v* is the number of dimensions or variables in the dataset (Figure 14.2 & Figure 14.4).

These two steps are repeated until an end condition is met, such as the points no longer change clusters or a certain number of iterations has been reached. A crucial part of applying k-Means Clustering to time series data is the choice of a proper distance measure. Once again, *Dynamic Time Wrapping* (*DTW*, see 4.2) was selected. However, it should be noted that some studies point out that k-Means Clustering does not perform well when using DTW for averaging time-series, as k-Means Clustering is designed to minimize the squared Euclidean distance (Niennattrakul & Ratanamahatana, 2007). For this reason, it is recommended to use a global averaging method called *DTW Barycenter Averaging (DBA)* that improves the estimation of the centroids when clustering a time-series (Petitjean, Ketterlin, & Gancarski, 2011).

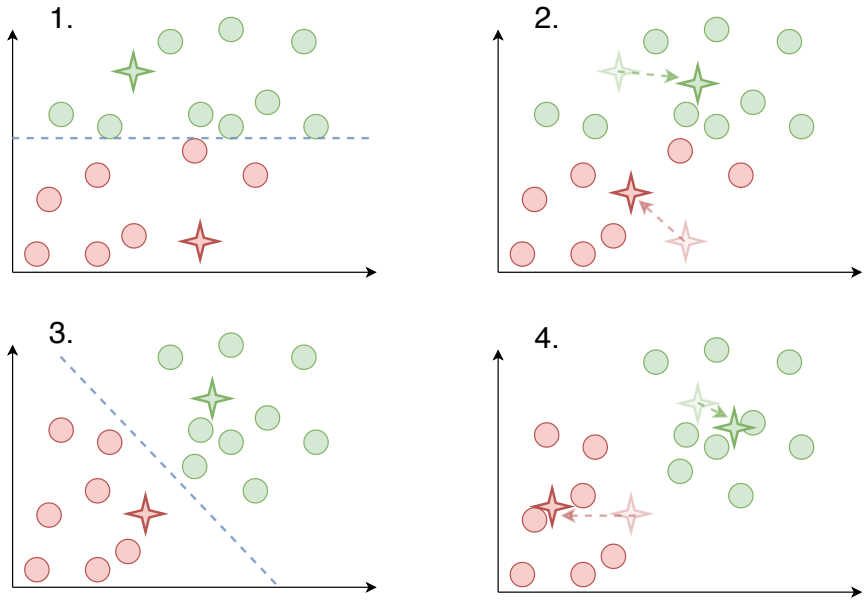


Figure 14. Two iterations of k-Means Clustering on a 2-D problem, where k=2.
1. Cluster centers are randomly placed in the problem space and the points are assigned to them.
2. The cluster centers are recomputed to be the cluster means.
3. Points are assigned to the new cluster centers.
4. Cluster centers are computed once again.

The core idea behind applying k-Means Clustering to Anomaly Detection is that the resulting centroids act as averaged representations of different patterns that occur in the "stable" dataset. Therefore, after fitting the algorithm on the training set, when a new sample x_i is observed, it is mapped onto the problem space and assigned to the nearest cluster, so that $\arg\min_j DTW(x_i, c_j)$. If the resulting distance is within a certain pre-defined range, it is considered as non-anomalous, otherwise, it is an anomaly. Ideally, the threshold for classifying this distance from the cluster center to the new sample as anomalous or non-anomalous should be based on the furthest point p_i in the cluster c, $\arg max_{p_i \in c} DTW(p_i, c)$. However, it needs to be considered that this "stable" dataset includes periods when the patient was unstable. Therefore, the threshold was defined by finding a point in the cluster which is the 50th percentile, so a distance below which 50% of observations can be found. Finding the most optimal percentile value and the number of clusters was done by maximizing the F_1 score on the 10 expert-tagged operations from the validation set.

5.2 Forecasting Neural Networks

The next concept for Anomaly Detection is based on using Neural Networks to forecast the future timesteps and checking if the forecasted state matches the actual data incoming from the patient – if it does, one can assume that there are no complications at the moment since the network was trained to forecast only non-anomalous behavior. Even though the networks described in this section were based on the previously discussed concepts of an LSTM unit and convolutions, their architectures differed. This is caused by the fact that forecasting is a regression problem, as opposed to classification, in which the outputs are probability values. One could also argue that predicting multiple future timesteps is a more complex task than predicting class probability; hence, the models also needed to be more advanced.

5.2.1 Anomaly Detection Method

The exact method that each network applies to generate forecasts differs; however, the way in which these predictions were used for Anomaly Detection was constant across architectures to make them comparable. The general idea was loosely inspired by the work of Malhotra, Vig, Shroff, & Agarwal (2015) but some changes had to be introduced due to the characteristics of the OR data. Each tested network was set to forecast *l=16* (4 minutes) future timesteps for each variable. This value was obtained by establishing a compromise between a number that is not too large to be feasible from the point of view of a forecasting ML algorithm and a number of forecasts that is large enough to escape any biases caused by local fluctuations in the vital parameters These future predictions were accumulated, so that every timestep *t* was forecasted 16 times at *t-16, t-15, …, t-1*. Then, a prediction error was computed where e_{ij}^t is the absolute difference between x_i^t the actual value of the variable *i* measured at time *t* and its predicted value at time *t-j*. Malhotra et al. (2015) used the errors vectors $e^t = [e_{1,1}^t, \ldots, e_{1,l}^t, \ldots, e_{d,1}^t, \ldots, e_{d,l}^t]$, where *l* is the number of future predictions, *d* is the number of dimensions or variables in the dataset, to fit a multivariate normal distribution and use the

probability density function to estimate the likelihood of observing a new error vector e^t. This likelihood was then compared to a threshold in order to classify e^t as anomalous or normal. Such an approach was a good choice in the cited study, as the LSTM network had to forecast repetitive patterns in an ECG signal or power consumption measurements, rather than a less stationary time-series. In case of the OR dataset, an alarming system based on the probability density would generate satisfactory results only for operations where the errors were generally very low but would produce mostly false alerts when an operation was not as trivial to model.

Eventually, a more successful approach was to average the error vectors across the *l* and *d* dimensions, so that for every timestep there would be only a single $\bar{e}^t$ *grand mean error value*. Then, in order to provide a context for every timestep t, a median value was calculated from the last 8 steps *median*($m^{t-8} \dots m^{t-1}$) and summed with a constant threshold τ. Using a running median was motivated by the fact that some operations have periods which a NN could not model with sufficient accuracy and had a generally higher prediction error $\bar{e}^t$. Also using a simple grand mean $\bar{e}^t$, rather than tracking each variable separately, provided more consistent results. Hence, anomaly detection at time *t* was done by averaging all prediction errors $e_{d,l}^t$ into $\bar{e}^t$ and checking whether the following condition was violated $\bar{e} < median(\bar{e}^{t-8} \dots \bar{e}^{t-1}) + \tau$. Finding the optimal threshold τ value was done by maximizing the F_1 score on the 10 expert-tagged operations from the validation set. Additionally, such an approach to Anomaly Detection allows retrieving the source of abnormal changes. This can be done by simply checking in which of the vital parameters the error was the largest. Therefore, an alarming system based on the described anomaly detection approach can not only alert the anesthesiologist that there is generally something abnormal happening but can provide information with which vital parameter(s) the observed complication is associated.

5.2.2 Loss Function

The previously mentioned cross-entropy loss function is only suitable for classification problems; hence, more appropriate loss function for Anomaly Detection is *logcosh*. It was selected instead of the popular *Mean Absolute Error (MAE or L1 loss)* and *Mean Squared Error (MSE or L2 loss)* as it applies less penalization to very incorrect predictions which, in case of the OR data, would be the anomalies that the network should not learn to model (Figure 15). Closer to the minimum Logcosh increases its precision, so the gradient of the loss function becomes smaller at the end of the training phase, which in theory facilitates network optimization with gradient descent. Hence, for a prediction *y* and its true value y_p, $logcosh = \log\left(\cosh\left(y_p - y\right)\right)$ is approximately equal for large errors to $MAE = \left|y_p - y\right|$ and for small errors to $MSE = \left(y_p - y\right)^2$.

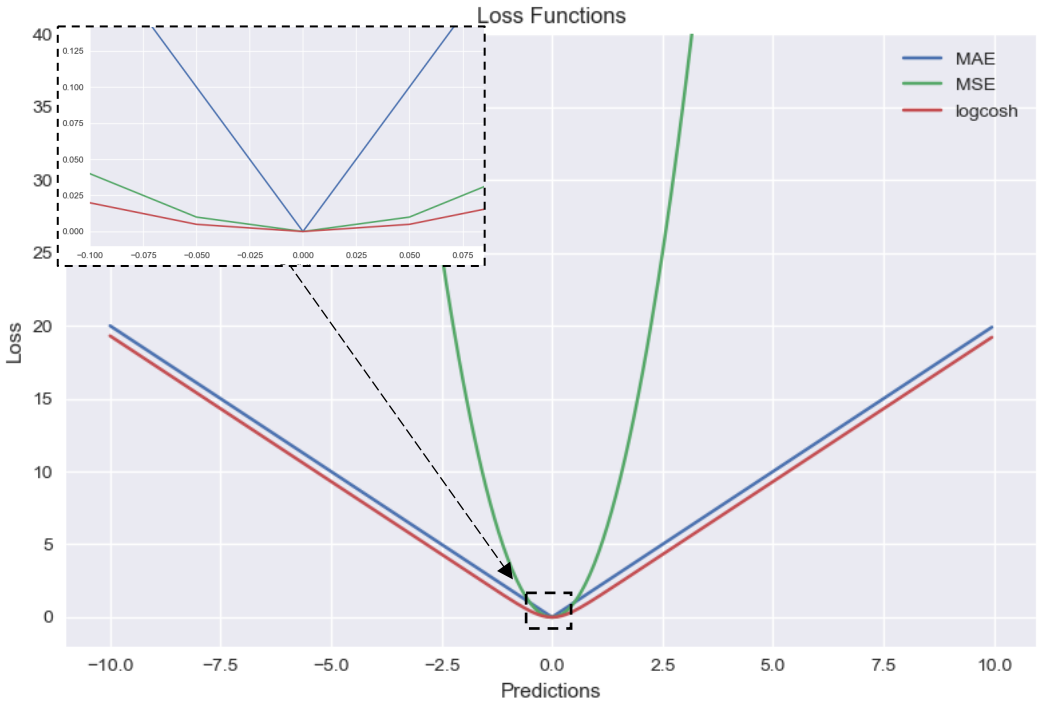


Figure 15. Comparison of loss functions on small and large prediction error, a perfect prediction is at x = 0.

5.2.3 Simple LSTM

Recurrent Neural Networks, especially LSTM networks, are the most popular ML approach to predicting sequences because of their ability to model the time domain (Section 4.3.5). The basic method for training an LSTM to generate predictions about the future is to use a sliding window $X = (x_{t-n}, \dots x_{t-1,} x_t)$ and a preceding target window $Y = (y_{t+1}, y_{t+2}, \dots y_{t+m})$ with y_t being the vector of variables where x_t and y_t are vectors of variables measured at time t (Figure 16).

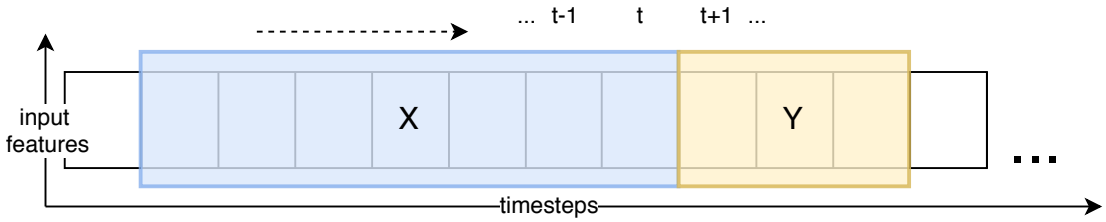


Figure 16. Sliding window with X of size 7 and target Y of size 3.

Such a forecasting network does not differ much in its architecture from a typical NN classifier besides the fact that it must have an output layer with a linear activation function and the number of units in this layer has to be equal to the number of input variables. Such an architecture is often over-simplistic since it will most likely minimize the loss function by copying the input *t* as the next timesteps t*+1, t+2…t+m*. Therefore, rather than using a simple network that is overwhelmed by the intricate task of predicting multiple future timesteps at once, a better idea is to forecast only a single step $p(x_{t+1}|x_1, x_2, \dots, x_t)$ which is then fed back

as an input for the next iteration $p(x_{t+2}|x_2, x_3, \dots, x_{t+1})$ and the process is repeated until all of the predictions about the target window *y* are generated. This *Simple Sequence-to-Sequence LSTM* model was evaluated in order to obtain a baseline for more advanced architectures.

5.2.4 Encoder-Decoder LSTM

A more sophisticated approach to forecasting time-series data originates from the domain of machine translation and is known as *Sequence-to-Sequence Modelling* (Graves, 2013; Sutskever, Vinyals, & Le, 2014). Once again, an LSTM-based network was implemented. However, its architecture differed quite significantly in comparison to its simplified version from the previous section. This more advanced Sequence-to-Sequence network consisted of an *Encoder* and a *Decoder* (Figure 17), hence it will be referred to as the *Encoder-Decoder LSTM*. The Encoder consists of LSTM units which transform the typical time-window input into two vectors where the first is the last hidden state h_t and the second is the last cell state c_t. Intuitively, these two states provide context about the encoded sequence as they contain historical information from the recurrent connections. Then, the Decoder with its LSTMs generates *n* new sequences basing on the Encoder's state vectors *s* and the most recent input. The Decoder unrolls a new sequence by making *n* iterative predictions *p* at time *t* so that: $p_1(x_{t+1}|s_t,\ x_t)$, $p_2(x_{t+2}|s_{t+1},\ x_{t+1})$,… , $p_n(x_{t+n}|s_{t+n-1}, x_{t+n-1})$ and updating its internal states timestep-by-timestep. On top of the LSTM layer, the Decoder has a Fully-Connected linear layer with one unit for every predicted variable (so 13 units in total because there was no point in forecasting the weight variable). While implementing such an architecture, the Decoder is trained using a *teacher forcing* technique. In a normal training procedure, at time *t* the model forecasts x_{t+1} and this forecast is used as input for forecasting x_{t+2}. However, in teacher forcing training the model is passed the actual value x^{true}_{t+1} instead of the forecasted x_{t+1} in order to make the Decoder learn the dependencies between values that actually appear in the dataset. Therefore, the predictions *p* at time *t* during training with teacher forcing are as follows $p_1(x_{t+1}|s_t,\ x_t)$, $p_2(x_{t+2}|s_{t+1}, x^{true}_{t+1})$ etc.

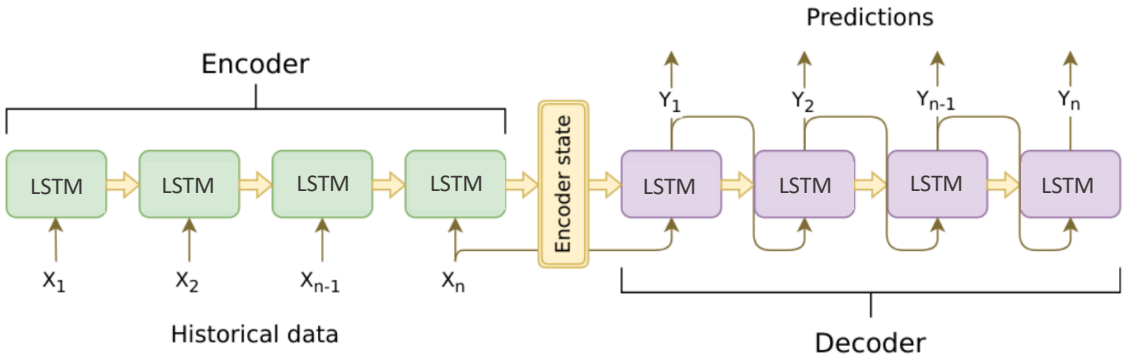


Figure 17. Sequent-to-Sequence Recurrent Neural Network with LSTM cells.

Britz, Goldie, Luong, & Le (2017), suggest in their study to use a Bidirectional LSTM layer for the Encoder part. A Bidirectional LSTM simply consists of regular LSTM units paired with LSTM units that process the input moving back in time. Therefore, a Bidirectional LSTM has the stream of information moving not only from t_1 to t_n but also in the opposite direction, so from t_n to t_1.

5.2.5 Dilated Causal Convolutional Neural Network

Another method of predicting future timesteps is based on specially designed Convolutional Neural Networks. More specifically, this architecture was inspired by Deep Minds' WaveNet (Oord et al., 2016) originally developed for audio forecasting but later proved to perform well on more conventional time-series data (Borovykh, Bohte, & Oosterlee, 2017). The core of this approach is the concept of a *dilated causal convolution.* First, *causal convolution* is one where the connections between the layers are structured such that future timesteps do not influence the past, so that a prediction $p(x_{t+1}|x_1, x_2, \dots, x_t)$ at timestep t cannot depend on $x_{t+1}, x_{t+2}, \dots$ (Figure 18).

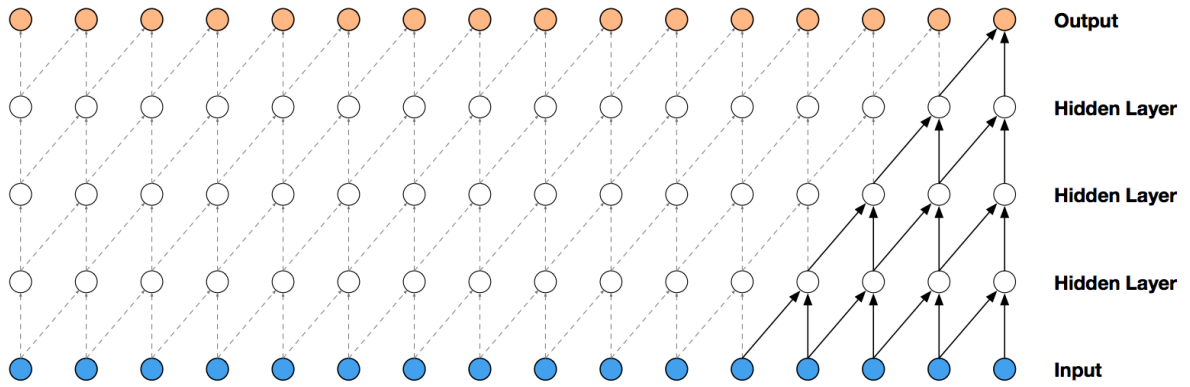


Figure 18. Causal convolution layers with filter width of 2, where future timesteps do not influence the past. The receptive field is equal to 5 timesteps (adapted from Oord et al., 2016).

In such a setting, the *receptive field* – the number of time steps in the input having an impact on the output – is limited to the number of layers. This configuration is not efficient because for longer sequences the depth of the network would have to be increased too much, making it computationally infeasible. Introducing *dilation* to causal convolutions solves this issue by applying the convolution filter to a larger area than its width by skipping a number of steps expressed by the dilation rate (Figure 19). Applying an exponentially increasing dilation rate allows collecting information from a large receptive field with just a few layers which makes dilated causal convolutions computationally more efficient than other CNNs or RNNs on time-series data.

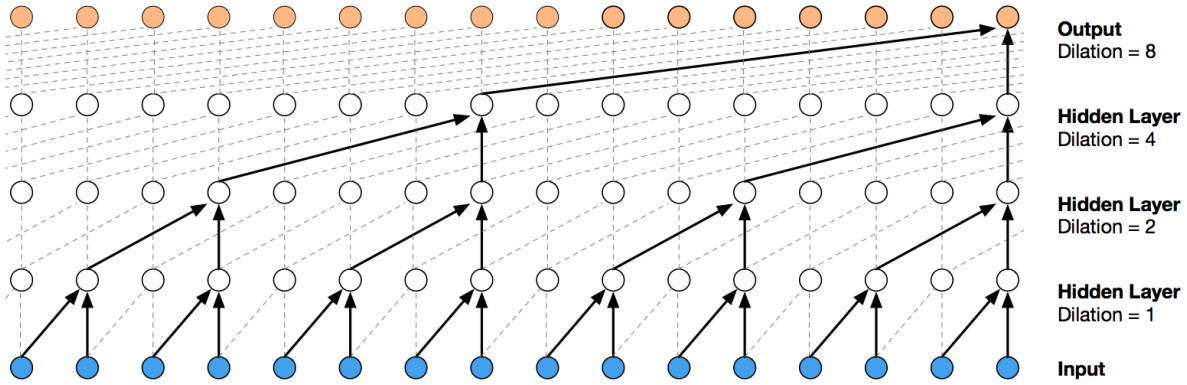


Figure 19. Dilated causal convolutions (adapted from Oord et al., 2016).

The original WaveNet architecture includes more innovative approaches to sequence-to-sequence forecasting; however, for the purpose of this paper, only the idea of dilated causal convolutions was used.

5.3 Autoencoders

The last approach to detecting anomalies is based on a self-supervised ML technique, namely an *Autoencoder*, which is constructed out of a Decoder and Encoder. The key concept in an Autoencoder is that it predicts its own input which might appear as a trivial task, but because such architecture includes a "bottleneck" – at least one hidden layer that has smaller dimensions than the input (Figure 20) – it is not trivial. The Encoder performs the task of dimensionality reduction or compression of the input and the Decoder uses this embedded representation to reconstruct the input as closely as possible. In principal, an Autoencoder applied to time-series data works similarly to the previous forcasting NNs, with the only difference being that instead of using *X* and *Y* time windows, as shown in Figure 16, it needs only the *X* window, which serves as both – input and target.

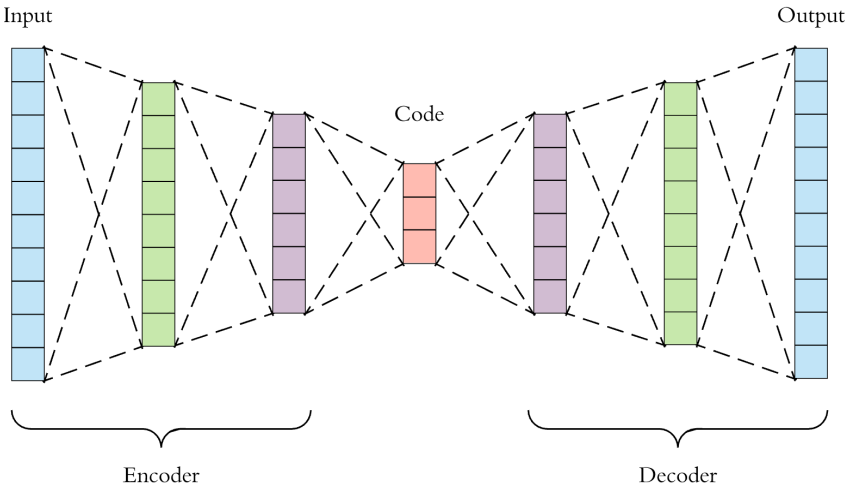


Figure 20. Autoencoder architecture (adapted from towardsdatascience.com).

Autoencoders were trained to reconstruct their input on normal/stable data; hence, if the input contained any abnormalities, the reconstruction would not be successful indicating that an anomaly is present. In other words, Anomaly Detection using an Autoencoder was based on the same method as the in previously described forecasting networks, with the only difference being that rather than using accumulated prediction errors, the errors were computed by calculating MAE between the reconstructed sequence and the original sequence. The remaining steps of the Anomaly Detection process were exactly the same as described in 5.2.1 – the mean reconstruction errors were compared with their running median in the 8 previous timesteps to check for abnormalities in the present time step.

5.3.1 Simple Autoencoder

A basic approach to implementing a sequence-to-sequence Autoencoder is applying one or more LSTM layers in the Encoder part of the network, with the last layer returning the output generated only by the last iteration of the LSTM unit at time *t.* In other words, the processing performed by the Encoder results in a dimension reduction from the 2-D matrix of *lookback* $\times$ *features* to a single vector of size *features* which is the "bottleneck". Next, the Decoder with LSTM units expands this compressed representation of the time-series into its original dimensions of size *lookback* $\times$ *features*. This simplistic architecture was evaluated in order to establish a baseline for a more advanced Autoencoder.

5.3.2 Advanced Autoencoder

A more sophisticated Autoencoder architecture took advantage of the cell state and the hidden state of the LSTM unit. The general design of the network was the same as described in chapter 5.2.4 Encoder-Decoder LSTM with the only difference being that it was not meant to forecast future timesteps but to replicate its input. A similar approach, where an Encoder-Decoder architecture would reconstruct its input sequence for the purpose anomaly detection, was tested by Malhotra et al. (2016) and provided promising results.

5.4 Results

5.4.1 Implementation

K-Means Clustering was implemented in *tslearn* Python library. The optimal number of clusters equal to 100 was found by running the algorithm with different *k* values. The tslearn implementation uses the before-mentioned *DBA* fix by default. Moreover, the training set had to be downsampled by a factor of 10 because the used k-Means Clustering implementation was computationally too expensive. However, we expected that DTW, due to its flexibility in comparing time series, would be able to compensate for the stripped-down training set.

All Neural Networks were implemented in *Keras* Python library and run on a Nvidia k40 GPU from the Peregrine HPC cluster. The selected optimizer was again Adam and the loss function was logcosh. The used combinations of hyperparameters are shown in Table 3.

Table 3. Hyperparameters of the best performing anomaly detection methods.

Technique	**Hyperparameters**
k-Means Clustering	k =100
Simple LSTM	1. **LSTM layer** – 128 units, dropout = 0.2, recurrent_dropout = 0.5 2. **LSTM layer** – 64 units, dropout = 0.5, recurrent_dropout = 0.5 3. **Fully-Connected output layer** – 14 units, linear activation
Encoder-Decoder LSTM	1. **Encoder** – 64 Bidirectional LSTM units (128 units in total), dropout=0.2, recurrent_dropout=0.2 2. **Decoder** – 128 LSTM units, dropout=0.2, recurrent_dropout=0.2 3. **Fully-Connected output layer** – 13 units, linear activation
Dilated Causal CNN	1. **Causal Dilated Convolutional layer** – 32 filters, filter_width=2, dilation_rate=1 2. **Causal Dilated Convolutional layer** – 32 filters, filter_width=2, dilation_rate=2 3. **Causal Dilated Convolutional layer** – 32 filters, filter_width=2, dilation_rate=4 4. **Causal Dilated Convolutional layer** – 32 filters, filter_width=2, dilation_rate=8 5. **Causal Dilated Convolutional layer** – 32 filters, filter_width=2, dilation_rate=16 6. **Fully-Connected layer** – 256 units, activation=ReLU, dropout=0.2 7. **Fully-Connected output layer** – 14 units, linear activation
Simple Autoencoder	1. **Encoder** – 128 LSTM units, dropout=0.2, recurrent_dropout=0.2 2. **Decoder** – 14 LSTM units
Advanced Autoencoder	1. **Encoder** – 64 LSTM units (processing time in reversed order), dropout=0.2, recurrent_dropout=0.2 2. **Decoder** – 16 LSTM units, dropout=0.2, recurrent_dropout=0.2 3. **Fully-Connected output layer** – 14 units, linear activation

5.4.2 Final Scores

The forecasting Anomaly Detection ML techniques were first tested with regards to their predictive performance on the test set. Table 4 shows mean absolute errors (MAE) between the values of the vital parameters of 16 future timesteps and their predicted values generated by the forecasting algorithms. Two naïve baselines were created where Mean Prediction simply returned a 0 for each future timestep and Future Duplication took the most recent measurements and copied them 16 times. The latter resulted in a score of 0.37 that was not surpassed by any of the forecasting models (the smaller the MAE is, the better the predictions

are). Encoder-Decoder LSTM performed better (MAE = 0.44) than Simple LSTM and Dilated Causal CNN (MAE = 0.5 for both) but still worse than the Future Duplication baseline.

Table 4. Prediction errors from the Sequence-to-Sequence models in forecasting 16 timesteps to the future.

Technique	**Mean Absolute Error**
Mean Prediction	0.80
Future Duplication	0.37
Simple LSTM	0.50
Encoder-Decoder LSTM	0.44
Dilated Causal CNN	0.50

The same tests were made for the two Autoencoders (Table 5) – MAE was calculated between the reconstructed input and the actual input. Reconstructive Duplication, in which the most recent measurements were copied 40 times, was the best scoring baseline (MAE = 0.46) but the Advanced Autoencoder managed to generate a slightly better result (MAE = 0.42), as opposed to the Simple Autoencoder (MAE = 0.72) which scored significantly worse than the Reconstructive Duplication baseline.

Table 5. Reconstruction errors.

Technique	**Mean Absolute Error**
Mean prediction	0.81
Reconstructive Duplication	0.46
Simple Autoencoder	0.72
Advanced Autoencoder	0.42

Next, all techniques were evaluated on the 10 expert-tagged operations from the “stable” test set. First, two baselines were established that made use of the Duplication method which provided the best scores in the two previous tests, as shown in Tables 4 & 5. Secondly, each of the ML Anomaly Detection techniques was tested. Table 6 suggests that all methods performed on a similar level to their respective baselines.

Table 6. Anomaly detection on "stable" dataset.

Technique	**Precision**	**Recall**	**F1**
Baseline (Future Duplication)	0.22	0.81	0.34
Baseline (Reconstructive Duplication)	0.31	0.31	0.31
k-Means Clustering	0.28	0.43	0.34
Simple LSTM	0.20	0.72	0.31
Encoder-Decoder LSTM	0.20	0.83	0.32
Dilated Causal CNN	0.18	0.85	0.30
Simple Autoencoder	0.20	0.78	0.31
Advanced Autoencoder	0.19	0.75	0.30

Lastly, all techniques were tested on the test set of the “hypotension” dataset. It is important to note that the labeling of the hypotensive events in this dataset had to be adjusted because the Anomaly Detection algorithms by design generated an alarm only in the first timestep when an abnormality was encountered. Such behavior is a result of the running median approach and is different from the Hypotension Detection alarms discussed in the previous chapter that did output an alarm during the whole period of hypotension. Hence, the hypotension test set had to be modified to contain only the first timestep of a hypotensive period. Moreover, an alarm was considered as correct only if the highest prediction/reconstruction error was found in one of the variables that described the blood pressure. For these reasons, even though the Hypotension Detection approaches and Anomaly Detection methods were tested on the hypotension set, their results should not be compared directly, as this test set was formatted differently in both cases. Nevertheless, the results Table 7 can be used to state that, out of all tested Anomaly Detection approaches, the best were: Simple LSTM ($F_1 = 0.25$), Encoder-Decoder LSTM ($F_1 = 0.23$), and Dilated Causal CNN ($F_1 = 0.29$), which performed similarly to the Future Duplication baseline score ($F_1 = 0.26$).

Table 7. Anomaly detection on the adjusted "hypotension" set.

Technique	**Precision**	**Recall**	**F1**
Baseline (Future Duplication)	0.24	0.29	0.26
Baseline (Reconstructive Duplication)	0.47	0.17	0.25
k-Means Clustering	0	0	0
Simple LSTM	0.30	0.22	0.25
Encoder-Decoder LSTM	0.24	0.22	0.23
Dilated Causal CNN	0.31	0.27	0.29
Simple Autoencoder	0.05	0.10	0.06
Advance Autoencoder	0.11	0.22	0.15

5.5 Discussion

5.5.1 Forecasting Neural Networks

First, investigating the approaches based on forecasting algorithms (Simple LSTM, Encoder-Decoder LSTM, Dilated Causal CNN) shows that their predictions of the future are below the naïve Future Duplication baseline (Table 4). A visual inspection of the forecasts suggested that these three approaches managed to infer at least some patterns from the data, as they do not simply copy the last measurement or converge to the mean immediately. Especially the early predictions seem to have interesting variability in them. Nevertheless, it is impossible to determine what changes or patterns in the vital measurements influence these predictions and to what extent. What can be said, is that the Encoder-Decoder LSTM architecture outperformed the Simple LSTM suggesting that the task of modeling the vital parameters benefits from a more advanced approach. In order to further increase the complexity of this Encoder-Decoder LSTM network, it would be recommended to test how an Attention mechanism (Bahdanau, Cho, & Bengio, 2014) impacts its forcasting capabilities. Several studies showed that similar architectures significantly benefited from such addition (Britz et al., 2017). Moreover, the Attention layer can provide extra insights about what elements of the input influence the output at any given moment, which addresses well the need of transparency in decision making in medicine (Jagannatha & Yu, 2016). Evaluating this additional element would perhaps also help in confirming an implicit assumption made in this study which states that a lower prediction MAE improves the quality of an alarming system. This hypothesis, in which the final scores depend on the MAE, was partly confirmed by the fact that the Future Duplication baseline with a lower MAE performed generally better than any of the networks; however, the differences in the resulting F_1 scores were too small to fully legitimate it.

Second, it is worth exploring how the network with causal dilated convolutions performed in comparison to the two architectures based on LSTM units since they differed considerably in how they processed the data. The CNN scored very similarly to the Simple LSTM in terms of the MAE of its forecasts and in detecting the anomalies on the "stable" dataset, which does not raise any interesting points for discussion or give any reasons to explore it further. On the other hand, the fact that on the "hypotension" test set the CNN outperformed both LSTM architectures and the baseline suggests that it should not be immediately rejected. Moreover, the implementation of causal dilated convolutions available in Keras was not computationally optimal. Therefore, expanding this architecture to its full form, as proposed by the original WaveNet article (Oord et al., 2016) would increase its complexity, which in turn, could improve the quality of the Anomaly Detection system. Hence, at this point, it cannot be stated with certainty which approach is better – the LSTM-based Encoder-Decoder or the Dilated Causal CNN.

Next, it can be argued that setting the number of future timepoints predicted at each step to 16 (an equivalent of 4 minutes) was an arbitrary choice. Therefore, more explanations are needed on this topic. The selected value seemed like a fair balance between a larger number, that would be impossible to accurately model, and a number that would be too small to give a broader perspective to the current input. In other words, a short forecasting period, such as 4 timesteps (1 minute), would mean that the alarming system would base its decision only on the 4 most recent errors. These errors would most likely be small since the changes in the vital parameters happen slowly over time (several minutes). Therefore, aggregating a larger number of errors from more distant timesteps should provide more objective errors which are less biased by the aspect of recency. Moreover, reducing the number of forecasted timepoints from 16 to 8 did not improve the MAE significantly and increasing it was computationally inviable. Nevertheless, more testing should be done in the future in order to precisely optimize the number forecasted timepoints with regards to the distribution of the prediction errors and its impact on Anomaly Detection overall.

Lastly, it should be noted that an attempt was made to boost the LSTM Encoder-Decoder by adding extra variables with information about the administered medication during operation. Two methods of encoding such information were investigated. In the first method, the initial 14 variables were expanded with 12 additional ones which contained binary data about the presence a particular drug (Adrenaline, Dobutamine, Dopamine, Ephedrine, Noradrenaline, Etomidate, Midazolam, Propofol, Fentanyl, Morphine, Piritramide, Sufentanil). The list of medication was prepared by an anesthesiologist and contained the most often used drugs which have the most impact on the patient's state. In the second method, each of the 12 medications was assigned to one of the 3 groups: opioids, sedatives, and drugs that raise heart rate and blood pressure. These 3 new binary variables that held the reconceptualized high-level information about the administered medications were then appended to the initial 14 variables. Both of these attempts failed to improve the MAE of the forecasts, suggesting that the networks were not developed enough to infer the meaningful relationships between the

medications and the vital parameters. This also leads us to a conclusion that it is crucial to first have a well-functioning base algorithm that can embrace the physiological changes before adding more features that increase the complexity.

5.5.2 Autoencoders

The results obtained from testing the two Autoencoder models do not lead to any breakthroughs but give additional support to the statements raised in the previous paragraphs. The Advanced model had a significantly smaller reconstruction MAE in comparison to the Simple Autoencoder and the Reconstructive Duplication baseline. The poor performance of the Simple model was to be expected, as the information loss in its "bottleneck" was too large, leading to the reconstructions being basically duplications of the last measurement. This observation seems to be also backed by the fact that this architecture scored very similarly to the Reconstructive Duplication baseline on the "stable" dataset – both methods reconstructed the input almost identically. Contrary to our expectations, the more promising Advanced Autoencoder did not outperform the baseline or the Simple model on the "stable" dataset which means that either Autoencoders are not the best technique for the Anomaly Detection approach or this particular method of anomaly detection does not utilize their potential correctly.

Another possible issue is that the selected size of the time window (or lookback value) was not optimal. Generally, this study trained all models on the most recent 40 observations in order to make the results more comparable and the optimization less cumbersome. Even though this value seemed to be a good choice for most techniques, it might not be for Autoencoders, as in their case it not only defines the input size but also the output size. Hence, the number of timesteps predicted by such a model is significantly larger than for the algorithms that forecasted 16 steps. Therefore, on the one hand, not optimizing the time window's size for Autoencoders is a limitation of the present study, which prevents us from clearly stating if this architecture is a good choice for patient monitoring. On the other hand, the fact that the input and output sizes must be the same can be viewed as a limitation of Autoencoders themselves. Moreover, shrinking the window size would mean that we violate our initial assumption – a heuristic according to which medical complications happen during a prolonged time period of 10 minutes. Optimizing this number is necessary if one wishes to use such architectures in similar setting; however, our results suggest that the forecasting models are more flexible because the number of timesteps they predict is not dependent on the size of the input window.

5.5.3 k-Means Clustering

At first sight, the results from k-Means Clustering might be promising since it performed the best out of all ML techniques on the "stable" dataset. However, the tests on the "hypotension" dataset revealed that it is quite limited, as it was unable to detect any hypotensive events, suggesting that this technique is more sensitive to pronounced artifacts and complications

than to subtle physiological abnormalities. Moreover, the design of the algorithm which is based on performing numerous comparisons between data points with the DTW distance makes the whole alarming system computationally inefficient, not only during the training phase but more importantly while running tests. It is also quite difficult to suggest how the k-Means Clustering approach could be improved since adding more training data would only slow down the system which already had to be trained on only 10% of the "stable" training set. Perhaps a more efficient implementation of DTW could be used but to our best knowledge, no such package, capable of processing a multivariate time-series, is currently available. These limitations of the k-Means Clustering method do not encourage one to use it in the future but at the same time, the relatively high scores on the "stable" set support the conclusion that the results from the neural architectures are at least mediocre if such simple approach managed to perform better.

5.5.4 Anomaly Detection Conclusions

In conclusion, the Anomaly Detection method based on the prediction/reconstruction error seems to be a promising approach. However, the predictive/reconstructive capabilities of the tested ML techniques are not satisfactory – at least in their current setup. It seems that increasing the complexity of the networks could be a potential solution to this problem. Overfitting is not a threat since an abundance of data is available; therefore, there are no major contraindications against approaching the challenge of modeling the vital parameters with more advanced neural architectures. On the other hand, future studies might find themselves in a situation where they reached the limit of exploiting the available dataset without achieving substantial progress in modeling it. It is likely that without high-fidelity recordings it is impossible to surpass a certain performance threshold because by nature some physiological changes are simply not explainable in a lower time-resolution. Out of the tested approaches, the neural architectures that predict the future timesteps seem to be most promising, due to their flexibility in terms of the independence between the dimensionalities of the input and output, as well as the availability of potential improvements.

6. Conclusions

Patient monitoring during anesthesia requires abilities and skills that are often associated with human reasoning, rather than traditional computer processing. Hence, the currently used threshold-based alarming systems produce an excess number of false alarms, which compromise the reliability of such warnings. Fortunately, AI offers nowadays techniques that, to some extent, are capable of mimicking the decision-making processes of a human. The present study aimed to investigate whether such novel solutions could potentially replace the over-simplistic alarm-generating methods that are currently in use. More specifically, we tried to answer the question: *"How can modern-day Machine Learning algorithms be applied to patient monitoring under anesthesia in order to improve the quality of the alarms?"*.

From a general perspective, this study managed to provide new insights on the topic of applying ML solutions to monitoring the patient's health during operations. Two distinct approaches, Complication Detection and Anomaly Detection, were thoroughly tested; it can be stated that they both have their advantages and disadvantages which depend on the specific use-case and the available training dataset. Complication Detection algorithms proved to produce higher quality alarms. Especially when applied to operations which contained less noise, their performance was close to ideal. However, it is unlikely that a complete alarming system could be based solely on this approach, because obtaining a dataset that would contain a sufficient number of labeled complications would be a major obstacle. This problem becomes even more prominent if one wishes to train such models to detect less frequently occurring complications. Perhaps a more realistic scenario would be to combine this ML learning approach with a rule-based system to obtain a product that offers the adaptiveness and sensitivity of the ML algorithms on the most often encountered complications, expanded by a broad range of expert-defined complications.

The Anomaly Detection approach proved to be a more flexible approach than had initially been expected. The option of retrieving prediction/reconstruction errors per vital parameter opens new possibilities for further development of such systems and most importantly follows the concept of Explainable Artificial Intelligence. Moreover, the fact that Anomaly Detection is based on methods that do not require a labeled training dataset should theoretically make it more generalizable across different patients and operation types in comparison to Complication Detection methods that must rely on a more limited training dataset. Anomaly Detection could also serve as a base for an alarming system by being responsible for the initial recognition of an abnormality in the incoming data stream. Such abnormality would be then processed by a rule-based system or any other classifier that could interpret the abnormality based either on the raw values of the vital parameters or/and the prediction errors. Additionally, the forecasts generated by an Anomaly Detection system could be used to fill in missing data in real-time. However, these speculations about the future applications are only valid if the quality of the forecasts improves. As stated before, the most effective solution for boosting the performance of any detection algorithm is to control the quality of the data

registered during the operations. Moreover, increasing the sampling frequency of the recording would presumably further improve the results.

All the above becomes irrelevant if the pragmatic approach to the role of an alarming system in an operating room remains unchanged. Expecting of such systems to achieve a perfect recall score while maintaining a small percentage of false alarms is implausible. The precision-recall tradeoff seems to be acutely relevant in the context of health monitoring with the unpredictable nature of the medical events only accentuating the significance of the concept. In order to introduce any impactful improvements in the domain of alarming systems, it is crucial to first investigate the practical implications of a small reduction in recall and a big boost in precision. Since anesthesiologists already tend to disregard the alerts, it can be expected that the proposed changes will not impact the patient's safety negatively and should increase the reliability of the whole system in the eyes of the medical staff. Moreover, we emphasize the need to reassess the way in which alarming systems are viewed from a human-computer interaction perspective. We propose a shift from alert-generating devices to support systems that act as an additional staff member, but with a different type of intelligence that complements limitations of human cognition. If such conceptual change was to be regarded as plausible, the methods offered by ML, especially neural architectures, should provide the best solutions for generating reliable, generalizable, and meaningful alarms.

References

Akima, H. (1970). A new method of interpolation and smooth curve fitting based on local procedures. *Journal of the ACM (JACM)*, *17*(4), 589–602.

Alexander, J., & Joshi, G. P. (2017). Anesthesiology, automation, and artificial intelligence. *Baylor University Medical Center Proceedings*, *31*, 1–3. https://doi.org/10.1080/08998280.2017.1391036

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *ArXiv:1409.0473 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1409.0473

Ballast, A. (1992). *Warning Systems in Anesthesia* (Doctoral Thesis). University of Groningen.

Bates, J. H. T., & Young, M. P. (2003). Applying Fuzzy Logic to Medical Decision Making in the Intensive Care Unit. *American Journal of Respiratory and Critical Care Medicine*, *167*(7), 948–952. https://doi.org/10.1164/rccm.200207-777CP

Baxt, W. G., & Skora, J. (1996). Prospective validation of artificial neural network trained to identify acute myocardial infarction. *The Lancet*, *347*(8993), 12–15. https://doi.org/10.1016/S0140-6736(96)91555-X

Becker, K., Thull, B., Käsmacher-Leidinger, H., Johannes Stemmer, Rau, G., Kalff, G., & Zimmermann, H.-J. (1997). Design and validation of an intelligent patient monitoring and alarm system based on a fuzzy logic process model. *Artificial Intelligence in Medicine*, *11*(1), 33–53. https://doi.org/10.1016/S0933-3657(97)00020-1

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Bijker, J. B., Klei, W. A. van, Kappen, T. H., Wolfswinkel, L. van, Moons, K. G. M., & Kalkman, C. J. (2007). Incidence of Intraoperative Hypotension as a Function of the Chosen DefinitionLiterature Definitions Applied to a Retrospective Cohort Using Automated Data Collection. *Anesthesiology: The Journal of the American Society of Anesthesiologists*, *107*(2), 213–220. https://doi.org/10.1097/01.anes.0000270724.40897.8e

Bliss, J. P., & Dunn, M. C. (2000). Behavioural implications of alarm mistrust as a function of task workload. *Ergonomics*, *43*(9), 1283–1300. https://doi.org/001401300421743

Block, F. E., Nuutinen, L., & Ballast, B. (1999). Optimization of Alarms: A Study on Alarm Limits, Alarm Sounds, and False Alarms, Intended to Reduce Annoyance. *Journal of Clinical Monitoring and Computing*, *15*(2), 75–83. https://doi.org/10.1023/A:1009992830942

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional Time Series Forecasting with Convolutional Neural Networks. Retrieved from https://arxiv.org/abs/1703.04691

Breznitz, S. (2013). *Cry wolf: The psychology of false alarms*. Psychology Press.

Britz, D., Goldie, A., Luong, M.-T., & Le, Q. (2017). Massive Exploration of Neural Machine Translation Architectures. *ArXiv:1703.03906 [Cs]*. Retrieved from http://arxiv.org/abs/1703.03906

Choi, E., Bahadori, M. T., Kulas, J. A., Schuetz, A., Stewart, W. F., & Sun, J. (2016). RETAIN: An Interpretable Predictive Model for Healthcare using Reverse Time Attention Mechanism. *ArXiv:1608.05745 [Cs]*. Retrieved from http://arxiv.org/abs/1608.05745

Choi, E., Schuetz, A., Stewart, W. F., & Sun, J. (2017). Using recurrent neural network models for early detection of heart failure onset. *Journal of the American Medical Informatics Association*, *24*(2), 361–370. https://doi.org/10.1093/jamia/ocw112

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh–A Python package). *Neurocomputing*.

Christ, M., Kempa-Liehr, A. W., & Feindt, M. (2016). Distributed and parallel time series feature extraction for industrial big data applications. *ArXiv:1610.07717 [Cs]*. Retrieved from http://arxiv.org/abs/1610.07717

Clark, A., & Chalmers, D. (1998). The extended mind. *Analysis*, 7–19.

Daoud, R. A., Amine, A., Bouikhalene, B., & Lbibb, R. (2015). Customer Segmentation Model in E-commerce Using Clustering Techniques and LRFM Model : The Case of Online Stores in Morocco.

Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning - ICML '06* (pp. 233–240). Pittsburgh, Pennsylvania: ACM Press. https://doi.org/10.1145/1143844.1143874

de Man, F. R., Greuters, S., Boer, C., Veerman, D. P., & Loer, S. A. (2013). Intra-operative monitoring--many alarms with minor impact. *Anaesthesia*, *68*(8), 804–810. https://doi.org/10.1111/anae.12289

Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., & Keogh, E. (2008). Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1542–1552.

Forman, G., & Scholz, M. (2010). Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter*, *12*(1), 49–57. https://doi.org/10.1145/1882471.1882479

Friedman, J. H. (1997). On Bias, Variance, 0/1—Loss, and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery*, *1*(1), 55–77. https://doi.org/10.1023/A:1009778005914

Görges, M., Markewitz, B. A., & Westenskow, D. R. (2009). Improving Alarm Performance in the Medical Intensive Care Unit Using Delays and Clinical Context. *Anesthesia & Analgesia*, *108*(5), 1546–1552. https://doi.org/10.1213/ane.0b013e31819bdfbb

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *ArXiv:1308.0850 [Cs]*. Retrieved from http://arxiv.org/abs/1308.0850

Hatib, F., Jian, Z., Buddi, S., Lee, C., Settels, J., Sibert, K., … Cannesson, M. (2018). Machine-learning Algorithm to Predict Hypotension Based on High-fidelity Arterial Pressure Waveform Analysis: *Anesthesiology*, 1. https://doi.org/10.1097/ALN.0000000000002300

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Holzinger, A., Biemann, C., Pattichis, C. S., & Kell, D. B. (2017). What do we need to build explainable AI systems for the medical domain? *ArXiv:1712.09923 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1712.09923

Hu, L.-Y., Huang, M.-W., Ke, S.-W., & Tsai, C.-F. (2016). The distance function effect on k-nearest neighbor classification for medical datasets. *SpringerPlus*, *5*(1). https://doi.org/10.1186/s40064-016-2941-7

Imhoff, M., & Kuhls, S. (2006). Alarm Algorithms in Critical Care Monitoring. *Anesthesia & Analgesia*, *102*(5), 1525–1537. https://doi.org/10.1213/01.ane.0000204385.01983.61

Imhoff, M., Kuhls, S., Gather, U., & Fried, R. (2009). Smart alarms from medical devices in the OR and ICU. *Best Practice & Research. Clinical Anaesthesiology*, *23*(1), 39–50. https://doi.org/10.1016/j.bpa.2008.07.008

Jagannatha, A. N., & Yu, H. (2016). Bidirectional RNN for Medical Event Detection in Electronic Health Records. *Proceedings of the Conference. Association for Computational Linguistics. North American Chapter. Meeting*, *2016*, 473–482.

Kendale, S., Kulkarni, P., Rosenberg, A. D., & Wang, J. (2018). Supervised Machine Learning Predictive Analytics for Prediction of Postinduction Hypotension: *Anesthesiology*, 1. https://doi.org/10.1097/ALN.0000000000002374

LeCun, Y. A., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade* (pp. 9–48). Springer.

Li, T., Zhu, S., & Ogihara, M. (2006). Using discriminant analysis for multi-class classification: an experimental investigation. *Knowledge and Information Systems*, *10*(4), 453–472. https://doi.org/10.1007/s10115-006-0013-y

Lienhart, A., Auroy, Y., Péquignot, F., Benhamou, D., Warszawski, J., Bovet, M., & Jougla, E. (2006). Survey of anesthesia-related mortality in France. *Anesthesiology*, *105*(6), 1087–1097.

Luo, Y. (2017). Recurrent neural networks for classifying relations in clinical notes. *Journal of Biomedical Informatics*, *72*, 85–95. https://doi.org/10.1016/j.jbi.2017.07.006

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. *ArXiv:1607.00148 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1607.00148

Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *Proceedings* (p. 89). Presses universitaires de Louvain.

McIntyre, J. W. R. (1985). Ergonomics: Anaesthetists' use of auditory alarms in the operating room. *International Journal of Clinical Monitoring and Computing*, *2*(1), 47–55. https://doi.org/10.1007/BF02915873

Mitsa, T. (2010). *Temporal Data Mining*. Chapman and Hall/CRC. https://doi.org/10.1201/9781420089776

Morik, K., Imboff, M., Brockhausen, P., Joachims, T., & Gather, U. (2000). Knowledge discovery and knowledge validation in intensive care. *Artificial Intelligence in Medicine*, *19*(3), 225–249. https://doi.org/10.1016/S0933-3657(00)00047-6

Müller, B., Hasman, A., & Blom, J. A. (1997). Evaluation of automatically learned intelligent alarm systems. *Computer Methods and Programs in Biomedicine*, *54*(3), 209–226. https://doi.org/10.1016/S0169-2607(97)00045-X

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press. Retrieved from http://neuralnetworksanddeeplearning.com

Niennattrakul, V., & Ratanamahatana, C. A. (2007). On Clustering Multimedia Time Series Data Using K-Means and Dynamic Time Warping. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)* (pp. 733–738). https://doi.org/10.1109/MUE.2007.165

Norman, D. A. (1986). Cognitive engineering. *User Centered System Design*, *31*, 61.

Oberli, C., Urzua, J., Saez, C., Guarini, M., Ciprianio, A., Garayar, B., … Irarrazaval, M. (1999). An expert system for monitor alarm integration. *Journal of Clinical Monitoring and Computing*, *15*(1), 29–35.

Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., … Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. Retrieved from https://arxiv.org/abs/1609.03499

Orr, J. A., & Westenskow, D. R. (1994). A breathing circuit alarm system based on neural networks. *Journal of Clinical Monitoring*, *10*(2), 101–109. https://doi.org/10.1007/BF02886822

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

Petitjean, F., Ketterlin, A., & Gancarski, P. (2011). A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, *44*(3), 678–693. https://doi.org/10.1016/j.patcog.2010.09.013

Pohar, M., Blas, M., & Turk, S. (2004). Comparison of logistic regression and linear discriminant analysis: a simulation study. *Metodoloski Zvezki*, *1*(1), 143.

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for Activation Functions. *ArXiv:1710.05941 [Cs]*. Retrieved from http://arxiv.org/abs/1710.05941

Ravì, D., Wong, C., Deligianni, F., Berthelot, M., Andreu-Perez, J., Lo, B., & Yang, G. (2017). Deep Learning for Health Informatics. *IEEE Journal of Biomedical and Health Informatics*, *21*(1), 4–21. https://doi.org/10.1109/JBHI.2016.2636665

Rejab, F. B., Nouira, K., & Trabelsi, A. (2014). Health Monitoring Systems Using Machine Learning Techniques. In *Intelligent Systems for Science and Information* (pp. 423–440). Springer, Cham. https://doi.org/10.1007/978-3-319-04702-7_24

Rowlands, M. (2010). *The New Science of the Mind: From Extended Mind to Embodied Phenomenology*. MIT Press.

Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *26*(1), 43–49. https://doi.org/10.1109/TASSP.1978.1163055

Schmid, F., Goepfert, M. S., Kuhnt, D., Eichhorn, V., Diedrichs, S., Reichenspurner, H., … Reuter, D. A. (2011). The Wolf Is Crying in the Operating Room: Patient Monitor and Anesthesia Workstation Alarming Patterns During Cardiac Surgery. *Anesthesia & Analgesia*, *112*(1), 78. https://doi.org/10.1213/ANE.0b013e3181fcc504

Schmid, F., Goepfert, M. S., & Reuter, D. A. (2013). Patient monitoring alarms in the ICU and in the operating room. *Critical Care*, *17*(2), 216. https://doi.org/10.1186/cc12525

Seagull, F. J., & Sanderson, P. M. (2001). Anesthesia Alarms in Context: An Observational Study. *Human Factors*, *43*(1), 66–78. https://doi.org/10.1518/001872001775992453

Smart, P. R. (2018). Human-extended machine cognition. *Cognitive Systems Research*, *49*, 9–23. https://doi.org/10.1016/j.cogsys.2017.11.001

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Stanton, N. A. (1994). *Human Factors in Alarm Design*. CRC Press.

Sukuvaara, T., Koski, E. M. J., Mäkivirta, A., & Kari, A. (1993). A knowledge-based alarm system for monitoring cardiac operated patients — technical construction and

evaluation. *International Journal of Clinical Monitoring and Computing*, *10*(2), 117. https://doi.org/10.1007/BF01142282

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Takla, G., Petre, J. H., Doyle, D. J., Horibe, M., & Gopakumaran, B. (2006). The problem of artifacts in patient monitor data during surgery: a clinical and methodological review. *Anesthesia and Analgesia*, *103*(5), 1196–1204. https://doi.org/10.1213/01.ane.0000247964.47706.5d

Ting, C.-H., Mahfouf, M., Nassef, A., Linkens, D. A., Panoutsos, G., Nickel, P., … Hockey, G. R. J. (2010). Real-time adaptive automation system based on identification of operator functional state in simulated process control operations. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *40*(2), 251–262.

Tinning, K., & Acworth, J. (2007). Make your Best Guess: An updated method for paediatric weight estimation in emergencies. *Emergency Medicine Australasia : EMA*, *19*, 528–534. https://doi.org/10.1111/j.1742-6723.2007.01026.x

Topf, M., & Dillon, E. (1988). Noise-induced stress as a predictor of burnout in critical care nurses. *Heart & Lung: The Journal of Critical Care*, *17*(5), 567–574.

Tsien, C. L. (2000). Event discovery in medical time-series data. *Proceedings of the AMIA Symposium*, 858–862.

Westenskow, D. R., Orr, J. A., Simon, F. H., Bender, H. J., & Frankenberger, H. (1992). Intelligent alarms reduce anesthesiologist's response time to critical faults. *Anesthesiology*, *77*(6), 1074–1079.

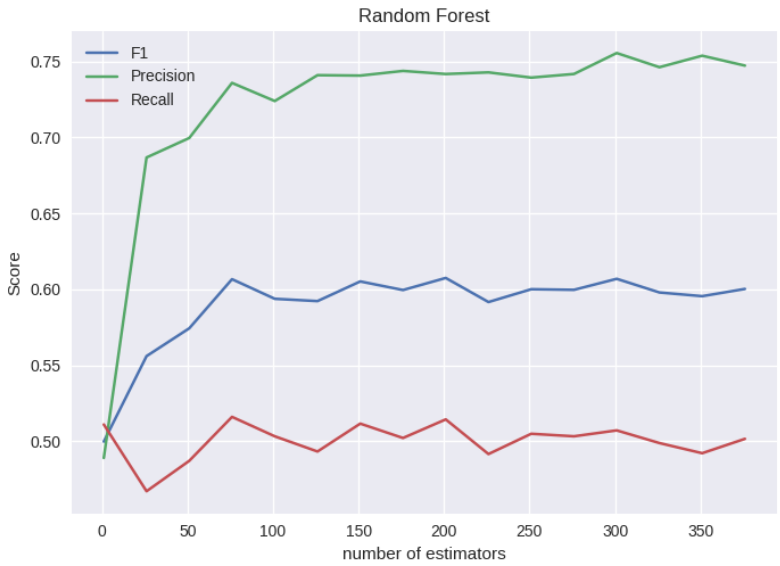
Wilson, K. M., Helton, W. S., & Wiggins, M. W. (2013). Cognitive engineering. *Wiley Interdisciplinary Reviews. Cognitive Science*, *4*(1), 17–31. https://doi.org/10.1002/wcs.1204

Woods, D. D., & Roth, E. M. (1988). Cognitive Engineering: Human Problem Solving with Tools. *Human Factors*, *30*(4), 415–430. https://doi.org/10.1177/001872088803000404

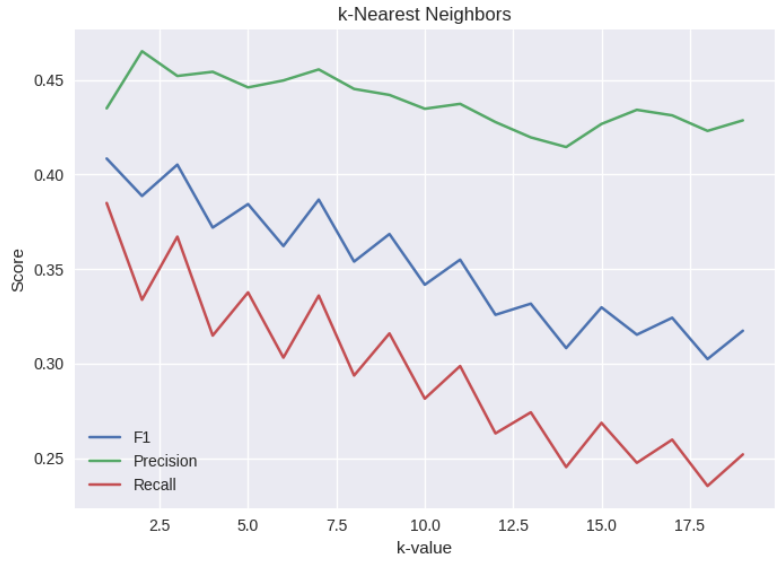
Zhao, B., Lu, H., Chen, S., Liu, J., & Wu, D. (2017). Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, *28*(1), 162–169. https://doi.org/10.21629/JSEE.2017.01.18

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014). Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management* (pp. 298–310). Springer.

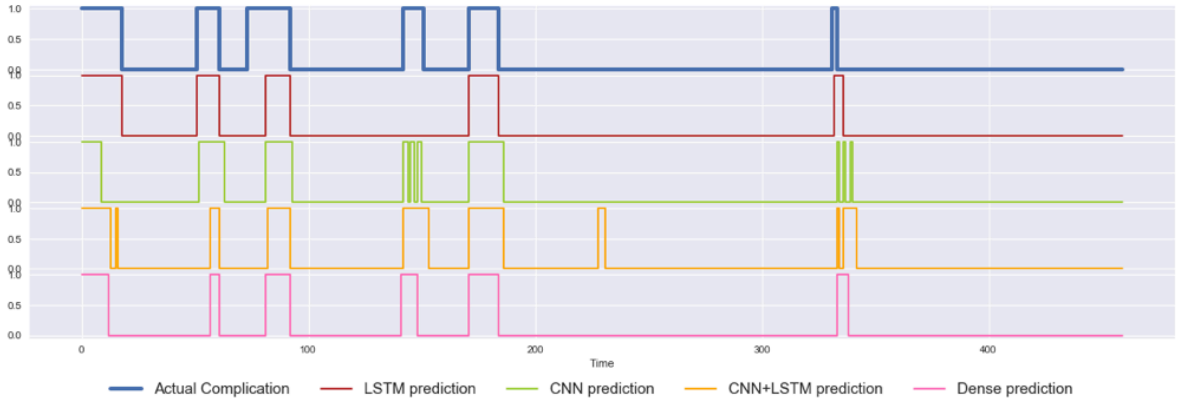
Appendices



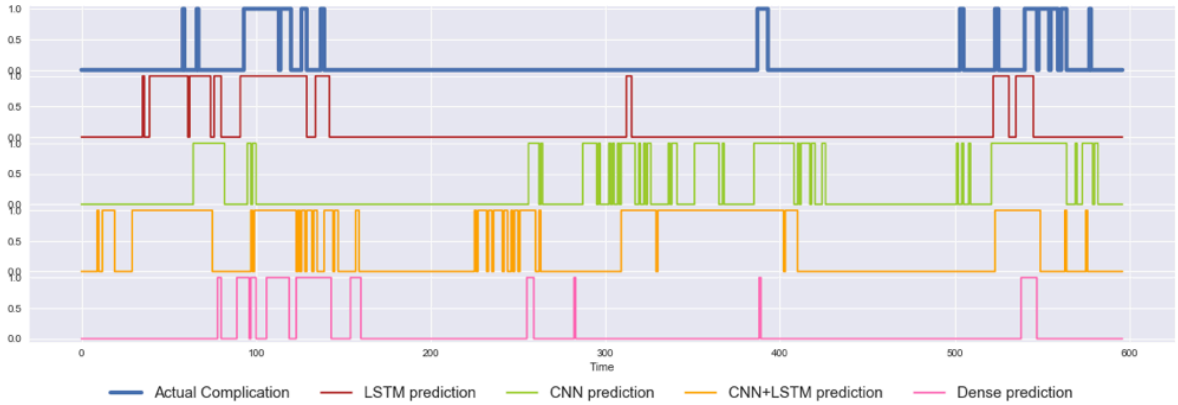
Appendix 1. Random Forest on the validation set for varying number of trees (number of estimators).



Appendix 2. k-NN scores on the validation set for varying number of neighbors (k-value).



Appendix 3. Hypotension detection on operation with good quality data.



Appendix 4. Hypotension detection on operation with noisy data.