UNIVERSITAT DE BARCELONA

Facultat de Medicina
i Ciències de la Salut

BIOMEDICAL ENGINEERING DEGREE

FINAL DEGREE PROJECT

Machine learning prediction of adverse events in anaesthesia after laryngeal mask airway insertion

Joan Altés Depares

Supervised by

Dr. Pedro Luis Gambús

Hospital Clínic de Barcelona
SPEC-M Research Group

Barcelona, 15 June 2020

Declaration

I hereby certify that this report, which I now submit for assessment on the Final Degree Project leading to the award of Biomedical Engineering Degree, is entirely my own work and, where the work of the others has been used, it is fully acknowledged in the text and in captions to table illustrations. No portion of the work contained in this project has been submitted in support of an application for another degree or qualification to this or any other institution.

Joan Altés Depares
15 June 2020

Abstract

Background: During general anesthesia, monitoring systems are used to evaluate the hypnotic, analgesic and immobility effects of the patient. These monitoring systems acquire and analyze in real time physiological signals of the patient such that the anesthesiologist can assess the current state of the patient, but they lack predictive power. For surgeries that are expected short, LMA devices are used rather than tracheal intubation, as they are associated with a lower incidence of complications. However, adverse events can still occur after LMA insertion, and its prediction could be valuable in clinical set up. In this project data driven prediction models are train to predict adverse events after LMA insertion.

Methodology: A dataset containing variables from many monitoring systems during several surgical procedures is used to train predictive models. Adverse events to be predicted are movement, light hypnosis (BIS>60%) and hypotension (mean NIBP<60mmHg). The 45s preceding LMA insertion are used as model input and output is set according to the presence or absence of adverse event in the 45s following LMA insertion. Tested models are RF, kNN, SVC, NN and LSTM. Models are tuned, trained and validated retrospectively.

Results: For light hypnosis prediction, best performing model is LSTM with a ROC AUC of 0.944 and a specificity of 0.784 for a sensitivity of 1. For hypotension prediction, best performing model is SVC with a ROC AUC of 0.913 although it is the LSTM that reaches a specificity of 0.667 for a sensitivity of 1. Movement prediction remains somewhat elusive, the best model being RF with a ROC AUC of 0.703.

Conclusion: Light hypnosis and hypotension after LMA insertion can be accurately predicted, while movement prediction needs further studies. LSTM had the most solid performance for both conditions, hinting that the time series nature of the data must be taken into account. Prospective validation is required before this can be implemented. Integrating this solution within a larger scheme of prediction models would improve personalized anesthesia management.

Keywords: *General anesthesia*, *Anesthesia monitoring*, *Hypnosis*, *Movement*, *Hypotension*, *Predictive model*, *Machine Learning*, *Deep Learning*.

Acknowledgements

First of all, I would like to express my gratitude to my supervisor, Dr. Pedro Luis Gambús, for giving me the opportunity to conduct this project in such an enjoyable field and for his guidance in all moments.

In addition, I would like to thank everyone in SPEC-M Research Group for the pleasant environment and for helping me whenever I reached out to them. Special thanks to Dr. Joan Fernández, Sebastián Jaramillo and Andrew Chen for the knowledge provided and for showing me how to apply it on real life medical applications.

Finally, I would like to show appreciation to all the team in box 4 of Hospital Clínic CMA for the friendly and direct contact and for helping me understand the insights of the clinical practice.

List of Figures

List of Tables

List of Abbreviations

ACh	AcetylCholine
AI	Artificial intelligence
ANI	Analgesia/nociception index
AUC	Area under curve
BIS	Bispectral index
BP	Blood pressure
BS	Burst suppression
CMA	Major ambulatory surgery
CNN	Convolutional neural networks
CV	Cross validation
DA	Difficult airway
DL	Deep learning
EEG	Electroencephalogram
EHR	Electronic health record
EMG	Electromyogram
FN	False negative
FP	False positive
GABA	Gamma-aminobutyric acid
HR	Heart rate
ICD	International classification of diseases
kNN	*k*-Nearest neighbor
LMA	Laryngeal mask airway

LSTM	Long short term memory
ML	Machine learning
NIBP	Non invasive arterial blood pressure
NN	Neural network
PC	Principal component
PCA	Principal component analysis
RF	Random forest
ROC	Receiver operating characteristic
SaMD	Software as medical device
SHAP	Shapley additive explanations
SVC	Support vector classifier
SWOT	Strengths, weaknesses, opportunities and threats
TCI	Target controlled infusion
TIVA	Total intravenous anesthesia
TN	True negative
TP	True positive

Contents

1 Background

1.1 General Anesthesia

General anesthesia is a drug induced, reversible condition that includes specific traits both behavioral and physiological, namely, unconsciousness, amnesia, analgesia and akinesia. Considering that coma is a state of profound unresponsiveness even to painful stimuli, it can be stated that general anesthesia is a reversible drug induced coma [1]. This assertion is backed by the similitude on the electroencephalogram (EEG) patterns, one of the many monitoring tools used during anesthesia.

General anesthesia must be regarded as a spectrum of separate pharmacological actions selected to achieve multiple goals [2]. Clinical outcome will depend on the drugs chosen and its dose, as well as the interaction between them. The key concept is that providing more anesthetic than necessary is best avoided as there are side effects, yet there is no rule of thumb on how little anesthetic is enough. Monitoring systems can help give a better idea by measuring anesthesia related quantities, but there are no objective standards for quantifying the entire set of therapeutic goals and clinical outcomes [3]. Such outcomes include changes in the level of consciousness, protective reflexes of the organism and alteration of heart rate and blood pressure.

When a patient undergoes general anesthesia there is no homeostatic control, unlike when sleeping [4]. It is the anesthesiologist job to administer the correct drugs in the correct dosage to maintain homeostatic equilibrium of the patient, mainly hemodynamic, respiratory and autonomous systems. Monitoring of body signals is key in order to provide personalized anaesthetic management, that is, to best administer drugs such that pain for the patient is avoided.

Motivation for this project is using data from several monitoring systems to build, train and validate predictive models capable of anticipating clinical outcomes of the patient in order to provide a tool to the anesthesiologist and improve personalized anaesthetic managements. Before moving to that it is necessary to understand anesthesia states, the drugs used to achieve them and the periods that general

anesthesia course is divided into.

1.1.1 Anesthesia states

Hypnotic effect

Hypnosis commonly refers to drug induced impairment of cognitive functions required for responding adequately to environmental stimuli, including attention and perception [5]. There are several levels of hypnosis: drowsy, sedated and unconscious, from mild to severe. In clinical settings patients are assumed to be unconscious when there is a loss of verbal response [6].

Unconsciousness induction is reached by the disconnection of the cerebral cortex. Gamma-aminobutyric acid (GABA) is the chief inhibitory neurotransmitter of the brain, acting at inhibitory synapses [7]. Most anesthetics act by selectively modulating $GABA_A$ receptors (ligand gated ion channels) to increase channel opening and inhibit synapses [8]. Enhanced $GABA_A$ inhibition spreads along neurons thus efficiently inactivating large regions of the brain including the brainstem and causing unconsciousness [9].

Hypnotic effect can be measured through EEG derived parameters.

Analgesic effect

Analgesia is the absence of sensibility to pain without loss of consciousness in response to a noxious stimuli, that is, a damaging or potentially damaging stimuli [10]. This state is achieved by selective blocking of the pathways responsible for afferent transmission starting at primary afferent nociceptors [11].

Opiods block nociceptive transmission by binding to μ, δ and κ opioid receptors, which are G protein coupled receptors, thus leading to membrane hyperpolarization and inhibiting synapses in the pain modulating circuit [11, 12].

Analgesic effect is not measurable and can only be indirectly monitored via hemodynamic changes and EEG changes.

Immobility

Muscular relaxation is performed so that the patient does not move during the surgical procedure. This state is achieved by blocking the AcetylCholine (ACh) receptor action and therefore inhibiting neuron transmission to muscle [13].

Neuromuscular blocking agents can be either depolarizing or non depolarazing. Depolarazing blocking agents act as ACh receptor agonists, stimulating an initial opening of the ion channel and causing muscular contraction followed by prolonged relaxation as they bind to the receptor longer than ACh. On the other hand, non depolarazing blocking agents act as ACh receptor antagonists and do not cause initial muscular contraction [14].

Immobility effect can be measured through electromyography (EMG) response to electric impulses to a specific nerve.

1.1.2 Anesthesia drugs

Anesthesia drugs can either be inhalational or intravenous. Total intravenous anesthesia (TIVA) refers to the administration of anesthetic agents exclusively via intravenous route. Compared to inhalational anesthesia, TIVA shows several benefits including better hemodynamic stability and reduced post-operative incidence of nausea and vomiting [15]. Table 1 summarizes most used intravenous anesthetics classified by effect.

Intravenous drugs can be administered either manually as a bolus or using target controlled infusion systems (TCI), which use integrated algorithms based on pharmacokinetic and pharmacodynamic models [17] for calculating the amount of drug to be delivered every ten seconds to achieve, maintain and rapidly change a desired level of anesthetic effect in response to demands, in order to compensate for elimination and transfer to other tissues. TCI systems offer many benefits, among them reduced movement incidence at the introduction of the laryngoscope [18].

Most frequently used groups of drugs in TIVA include hypnotics and opioids for hypnotic and analgesic effect. During major ambulatory surgery (CMA) in Hospital Clínic a synergy of propofol (hypnotic) and remifentanil (opioid) is used and oc-

Table 1. Commonly used intravenous anesthetics
Most commonly used intravenous anesthetics classified according to effect. Adapted from [16].

Hypnotics	
Propofol	
Thiopental	
Etomidate	
Ketamine	
Opioids	
Fentanyl	
Sufentanil	
Remifentanil	
Morphine	
Muscle relaxants	Class
Cisatracurium	Non depolarizing
Vecuronium	Non depolarizing
Pancuronium	Non depolarizing
Rocuronium	Non depolarizing
Succinylcholine	Depolarizing

casionally combined with other drugs when needed, including rocuronium (muscle relaxant). This mixture of drugs is widely accepted since propofol and remifentanil complement each other pharmacodynamic profiles resulting in a reduction of dosage requirements for both drugs [19].

Propofol

Propofol is an intravenous sedative hypnotic agent which rapidly and reliably causes loss of consciousness, and it is associated with quick recovery [20]. It acts mainly in the cerebral cortex as a positive allosteric modulator of the $GABA_A$ receptor, inhibiting synapses and leading to a loss of consciousness.

Propofol acts rapidly given that it is highly lipophilic and rapidly crosses the blood-brain barrier resulting in a rapid onset of action. Emergence from sedation is also rapid because of a fast redistribution into peripheral tissues and metabolic clear-

ance. Pharmacokinetic properties of propofol can be well characterized by a three compartment model [21], that is, a model grouping organs according to vascularization into three compartments: a central one responsible for drug administration and elimination and two peripheral compartments responsible for distribution.

Aside from neurological effects, propofol causes a decrease in blood pressure as well as a decrease in heart rate, but these effects can be partially controlled keeping continuous infusion at low doses [21]. Injection of propofol produces stinging like pain [22].

Remifentanil

Remifentanil is an opioid analgesic agent that shows a rapid clearance caused by its unique extrahepatic metabolic pathway (among other opioids) [23]. It acts mainly as a selective μ receptor agonist.

Remifentanil, like propofol, is highly lipophilic, passing easily through the blood brain barrier. Remifentanil is rapidly and extensively metabolised by nonspecific blood and tissue esterases, resulting in rapid clearance [24].

Aside from analgesic effect, at hemodynamic level remifentanil causes bradycardia and hypotension, but the incidence of these adverse events is low if administered by infusion [25].

Rocuronium

Rocuronium is an aminosteroid non depolarizing neuromuscular blocker or muscle relaxant used in modern anaesthesia to facilitate tracheal intubation by providing muscle relaxation [26]. In propofol-remifentanil anesthesia, tracheal intubation without muscle relaxant is acceptable but at low doses might provoke hypotension and severe bradycardia, effects that are significantly reduced while using rocuronium [27].

In CMA Hospital Clínic, rocuronium is used before tracheal intubation, but not before laryngeal mask airway (LMA) introduction (see section 1.1.3 for details) since LMA provides lower incidence of complications [28]. However, unpleasant effects

can still occur with LMA and they can be mitigated by low dose rocuronium administration [29]. If a validated predictive algorithm for adverse events related to LMA like the one proposed in this project was available, such events could be avoided by administering rocuronium or other anesthetic agents.

1.1.3 Anesthesia periods

The course of general anesthesia is divided into three periods: induction, maintenance and emergence.

Induction

After administration of a hypnotic drug such as propofol the patient goes from active to drowsy, sedated, and finally loses consciousness. At this point there is loss of verbal response and corneal reflexes are lost. Analgesic drugs such as remifentanil are also administered during this period to prevent the patient from painful stimuli.

As more hypnotic agent is administered respiratory activity decays leading to apnea, at which point bag mask ventilation must be initiated [1]. Either tracheal intubation or LMA are performed typically at the end of induction to ensure patient ventilation. LMA are single use supraglottic airway devices that may be used as a temporary method to maintain an open airway. Even if LMA provides lower adverse event incidence it has some contraindications mainly related to pulmonary disease and it is not advised for long interventions [30].

Maintenance

Anesthesia effects are maintained the necessary time to successfully perform surgery. In order to achieve it the drugs used during induction period are administered as bolus or via TCI, as well as any other anesthetic agent required.

The level of general anesthesia is monitored using several non invasive devices (see section 1.2) in order to get an idea of anesthesia states.

Emergence

Once the surgical procedure has ended it is convenient that the patient quickly recovers from anesthesia. Drug administration is stopped yet emergence is a passive process depending on several factors including drugs used, duration and amount of administration and patients physiological conditions. Once the patient is fully awake and able to express discomfort extubation can be performed [1].

1.2 Anesthesia Monitoring

During anesthesia it is necessary to be able to evaluate, attend and control anesthesia effects to ensure protection of the patient. Anesthesia monitoring consists of evaluating indicators in real time to assess how the patient is reacting to surgery and anesthetic drugs, which can produce rapid changes in vital functions. This includes tracking basic reflectory activity of the patient such as eyelash and corneal reflexes during induction period and using engineering solutions to obtain and process physiological signals. Monitoring systems can be classified into basic monitoring systems and advanced monitoring systems.

1.2.1 Basic monitoring systems

Electrocardiogram

Electrocardiogram is a graph of voltage against time representing the electrical activity of the heart obtained by placing electrodes on the skin. Several parameters can be derived from it including heart rate (HR) and ST segment. Changes in HR can have several causes related to hypnotic effect and analgesic effect [31]. It is desirable to keep HR between 40 and 100 beats per minute.

Arterial blood pressure

Arterial blood pressure (BP) is the pressure exerted by circulating blood on arteries. It can be measured invasively by using intra arterial catheters leading to continuous monitoring or non invasively by using cuffs. Invasive arterial BP monitoring is only used under specific indications such as previous medical condition or complicated surgery. Non invasive arterial BP (NIBP) monitoring relies on automated cuffs and

the oscillometry principle [32]. It is desirable to keep mean NIBP above 60mmHg and systolic NIBP and diastolic NIBP below 140 and 90mmHg, respectively, and there are drugs to increase or decrease NIBP.

Pulse oximetry

Pulse oximetry is a non invasive method for monitoring oxigen saturation by placing a sensor device at the patient, typically at index fingertip. The device passes two wavelengths through the finger to a detector and the change in absorbance of each wavelength is measured. From this measurement oxigen saturation in blood, hemoglobin levels and pulse rate can be determined, among others.

Capnography

Capnography is the monitoring of the concentration or partial pressure of carbon dioxide (CO_2) in the respiratory gases. Since during anesthesia the patient is intubated, capnography is a direct monitor of the elimination of CO_2 by the lungs to the anesthesia device. Indirectly, it monitors the production of CO_2 by tissues and the circulatory transport to the lungs.

1.2.2 Advanced monitoring systems

Electroencephalogram

EEG is a graph of voltage against time representing the electrical activity produced in the cortical area of the brain, obtained by placing electrodes on the forehead and processing the signal obtained. By applying a Fourier transform algorithm to the EEG the signal can be expressed as a sequence of waves of different frequency [33]. Table 2 shows EEG wave categorization by frequency as well as its expected brain activity.

EEG patterns during general anesthesia are altered depending on drugs, dose and duration of infusion, with significant variance within subjects. Most commonly, general anesthesia produces a progressive increase in low frequency, high amplitude activity as the level of unconsciousness rises [1].

Table 2. EEG waves
Electroencephalogram (EEG) wave categorization by frequency. Adapted from [34].

Wave name	Frequency (Hz)	Brain activiy
β - Beta	13 - 30	Awake with mental activity
α - Alpha	8 - 13	Awake and resting
θ - Theta	4 - 7	Sleeping
δ - Delta	<4	Deep Sleep

Additionally, some patients show during certain moments of anesthesia a specific EEG pattern called burst suppression (BS). BS consists of alternative periods of slow waves of high amplitude (the burst) and periods of silenced cortical activity and isoelectric EEG line, that is, flat EEG (the suppression) [35]. It is a sign of unnecessarily high level of unconsciousness and its pattern may vary depending on drugs used [36].

Raw EEG is hard to evaluate and extracting the information it contains requires a high level of expertise. In order to overcome this, EEG monitors use advanced signal processing algorithms to compute parameters and indexes that simplify EEG data into a single number called index, representative of the state of the patient.

One of the most popular EEG derived parameters is the Bispectral Index (BIS), which measures hypnotic effects of anesthesia. It was developed using EEG bispectral analysis of a large number of patients. The algorithm, which is proprietary and confidential, produces a dimensionless number from 0 to 100, representative of the level of unconsciousness of the patient, 100 being fully awake. It is considered that a BIS between 40 and 60 represents an appropriate level of hypnotic effect. BIS guided anesthesia reduces the risk of awareness, reduces the requirement for propofol and improves post operative recovery, which has security and economic advantages [37–39]. Nonetheless, it is necessary that the practitioner is familiar with BIS in order to be able to recognize artifacts, identify inaccurate readings and rapidly troubleshoot any problems pointed by BIS [40].

Anaesthetic state monitoring

Besides monitoring physiological signals it is also important to keep track of administered drugs and its dose during the procedure, rather than just the current value. TCI devices register throughout the intervention infusion rate, total volume, plasmatic concentration and concentration at the site of effect for each drug. The latter two are based on estimations from pharmacokinetic and pharmacodynamic models.

Anesthesia monitoring covers a wide range of variables of different nature, hence there is a necessity for increasingly integrated environments that cover clinical demands. Anesthesia workstations receive different inputs from monitors and administration devices in order to, ideally, integrate all relevant indicators to control anesthesia effects and ensure protection of the patient. If this is achieved, future behavior can, ideally, be predicted for the sake of patient safety. However, anesthesia workstations show a high number of false alarms which hinder their reliability [41].

Therefore, it is equally necessary to define what adverse events to predict, that is, relevant events for patient safety and intervention success, and to develop precise predictive models that help provide personalized anesthesia. At this point it is important to highlight that ideal integration is not currently available (see section 2.1).

1.3 Predictive Models

Predictive models use data and statistics to predict outcomes. While several approaches to predictive modeling are possible, predictive modeling is often referred to as Machine Learning (ML), which is a subset of Artificial Intelligence (AI). AI is a broad term and many definitions are available, but it mainly refers to building systems or machines (computers) to accomplish tasks that typically require human intelligence, such as making decisions. In medical field, AI is particularly relevant due to the vast amount of data available from electronic health records (EHR), clinical data and wearable devices [42]. Yet, AI is struggling to fulfill its potential in the medical field mainly because medical data is of high complexity, which makes it difficult to obtain valid algorithms since they require a lot of work from trained professionals with skills both in the medical field and in data science [43]. The success of the model completely depends on its training: only clinically meaningful data will

yield clinically meaningful results and avoid a so called Garbage In, Garbage Out model [44]. Additionally, poorly trained models can lead to overfitting, namely, the production of models that have good results on training data but do not generalise well on new data.

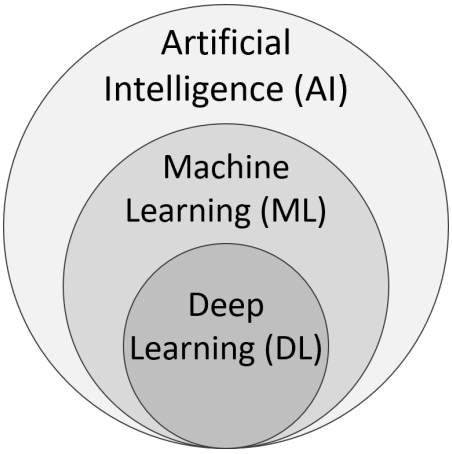

Figure 1. AI Venn diagram
Venn diagram displaying the relationship between AI, ML and DL.

Especially relevant to medical field is ML, a subset of AI algorithms that improve automatically when trained with data. ML algorithms use a data driven mathematical model in order to make predictions without being explicitly programmed to do so. In order to learn from data, ML algorithms update the parameters of the model using past experience [45]. Again, several approaches and classifications of ML are possible, but, overall, three broad categories can be drawn: supervised, unsupervised and reinforcement learning. The main difference between supervised and unsupervised learning is that the first one is achieved using ground truth, that is, there is prior knowledge of what the output values for the data should be. On the other hand, reinforcement learning is based on a reward and punishment approach for performing actions on a dynamic environment. Depending on the application, one approach or the other will be used, or a combination of them [46].

Typical ML applications use unsupervised learning for dimensionality reduction and for identifying clusters on the data, that is, subsets with observations that behave in a similar way, and supervised learning for classification or regression problems, that is, making predictions on a categorical output or a continuous output, respectively.

Most medical AI applications use supervised learning for classification problems that improve clinical decision making [43]. A great deal of ML classification models based on different principles are available and there is no rule of thumb on which one is superior to the other, the best approach will depend on each application and the only way to find out is trying many of them.

Deep Learning (DL) algorithms are a subset of ML (see figure 1) based on artificial Neural Networks (NN) that are gaining popularity among ML models as they are very useful for unstructured data and complex non linear data. Nonetheless, DL approaches require to train on a large amount of data to improve the results obtained using more cost efficient approaches [47].

In this project, five ML models for classification are tried including two DL models. Models were chosen according to recommendations from supervisor and his team.

1.3.1 Machine learning models

Random forest classifier

Random forest (RF) classifier is an ensemble learning method for classification problems. Ensemble learning methods or multiple classifier systems use the combination of several algorithms that put together improve the performance of any of the constituent algorithms alone, by reducing the variance and therefore improving accuracy [48]. RF are built by constructing many decision trees, training them and averaging their output, thus obtaining the mean prediction.

Decision trees are predictive models that only use conditional control statements to classify the input. In other words, decision trees apply conditions on features, called nodes, to predict the output class label, called leaves, to a certain depth of nodes defined by the number of layers (see figure 2).

After training many decision trees, RF obtain the mean prediction by averaging the prediction for all individual trees on the input data, which intuitively gives the fraction of trees that predicted a positive output as displayed in equation 1, where x stands for the input, B for the number of trees and p_b for the prediction of each tree, while $\hat{p}$ must be regarded as the predicted probability of x belonging to positive class.

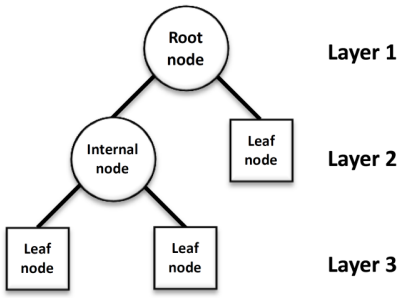

Figure 2. Decision tree schematic
Basic organization of a decision tree displaying nodes, leaves and layers. Retrieved from [49].

$$\hat{p} = \frac{1}{B} \sum_{n=1}^{B} p_b(x) \tag{1}$$

The main parameter to take into account when building a RF is the number of trees, but there are other important parameters such as the maximum number of features that decision trees are allowed to try and the minimum leaf size, which determines the minimum number of samples necessary for constituting a leaf node. The latter two are specially important to prevent overfitting.

RF also allow to determine feature importance, which is a great attribute that most ML models do not share. Feature importance represents the weighted relevance of each variable in input data for determining the output, by counting the number of observations that reach each node, divided by the total number of samples. The higher the value, the more important the feature, thus providing an excellent tool for understanding the insights of the model.

k-Nearest neighbor classifier

k-Nearest neighbor (kNN) classifier is a prediction model built under the assumption that same class observations exist in close feature proximity, that it to say, that observations that have similar feature value will belong to the same class.

Given any input data, the model must first find the k nearest observations using a distance function. Typically, euclidian distance is used but other specialized al-

gorithms for determining the distance between two points within a dataset are available. The output is then computed as the fraction of positive class in the nearest observations as displayed on equation 2, where $p(x_i)$ is the class of the nearest neighbors, k is the number of neighbors and $\hat{p}$ must be regarded as the predicted probability of input data belonging to positive class. Moreover, kNN models can be weighted such that observations that are closer to input data have higher influence on the prediction as displayed on equation 3, where w_i is the weight that depends on distance with sum of all weights equal to one.

$$\hat{p} = \frac{1}{k} \sum_{n=1}^{k} p(x_i) \tag{2}$$

$$\hat{p} = \sum_{n=1}^{k} w_i p(x_i) \tag{3}$$

The optimal value of k must be determined through trial an error method. Other important parameters include weight function, distance function and the algorithm used to find the nearest neighbors. These parameters can have a strong effect on model accuracy [50].

Support vector classifier

Support vector classifier (SVC) is a prediction model based on support vector machines. Support vector machine algorithms find an $N - 1$ dimensional hyperplane that best classifies the observations, N being the number of features in the dataset. Support vectors are the observations that are closest to the hyperplane and determine its position and orientation.

To find such hyperplane it is equally important that most observations are correctly classified and that there is a wide margin between both categories so that the model is not overfitted, in other words, most of the times small changes in input data should not completely change the output. The balance between these two criteria is mediated by a regularization parameter, which intuitively tells how much missclassification should be penalised. Again, the trade offs are unclear and the only way to find

out is trying many values.

Although SVC output directly the predicted class for a given input data depending on its side of the hyperplane, it is possible to obtain probability estimates by training several SVC models, each time dropping a partition of the training data leading to different support vectors producing different hyperplanes. This method is called cross validation (CV) and the number of partitions is known as folds. CV is a great tool to prevent overfitting [51]. Once the models are trained, the probability of an input belonging to a certain class can be estimated by averaging the output of the models, in a similar way than in equation 1, where B would now stand for the number of folds and p_b for the prediction of each model.

Parameters to take into account when training a SVC are the regularization parameter, gamma, which tells how many support vectors should be used, and kernel function. Kernel functions are transformations in the data that define new coordinates in which it might be easier to define a hyperplane.

1.3.2 Deep learning models

Deep neural network

Artificial NN are models inspired by the human brain, built by a connection of nodes called neurons. Neurons are organized in multiple layers, where a neuron from one layer is connected to each neuron in the following layer (see figure 3A) and outputs for a layer serve as inputs for the next layer. Neurons work by applying an activation function to the weighted sum of input values (see figure 3B). A deep NN is a NN that has more than one hidden layer.

NN start at random weights and learn by adjusting the values of the weights when trained, such that performance is improved. This is done by defining a measurement of error on how bad is the predicted value compared to actual class, known as loss function. A method called backpropagation is used, which divides the prediction error among weights and updates them to reduce the error. When training, observations are not passed through the NN all together but rather in batches of a certain size, and it is after a batch has passed through the model that weights are adjusted, until all training data has passed through the model. This process of

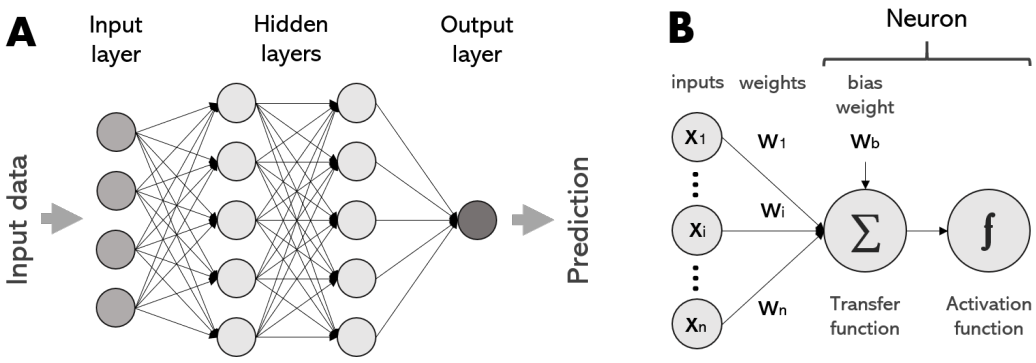

Figure 3. NN schematic
Basic neural network implementation. A) Layer distribution and neuron connection between layers. B) Internal functioning of a neuron: transfer function computes the weighted sum of input values plus the bias weight and activation function applies a function to obtain the output for that neuron.

passing all the training data through the NN is done iteratively, where each iteration is called an epoch.

Deep NN have many parameters to take into account when building a model including the number of neurons and the number of hidden layers, the number of epochs, the batch size, the activation function and the loss function and the learning rate, which determines how fast the model moves towards a minimum of loss function by adjusting the weights smoothly or abruptly. Dropout rate tells what fraction of neurons are randomly inactivated each epoch, a method to prevent overfitting.

Training a single NN requires a vast amount of data and the process is computationally expensive, compared to traditional, non NN models. However, DL based models match or surpass state of the art in many medical field problems [52].

Long short term memory

Long short term memory (LSTM) is an artificial recurrent NN architecture that uses sequences of data as input rather than individual observations, hence LSTM models are well suited to make predictions on time series. In addition to everything explained about deep NN, recurrent NN also remember prior inputs and its predictions are influenced on it, in other words, prediction at time t depends as well on

prediction at time $t-1$, which depends on prediction at time $t-2$, and so on. This is accomplished by including hidden state weights that are updated according to model output each time an input is passed though the model, thus influencing the next input (see figure 4).

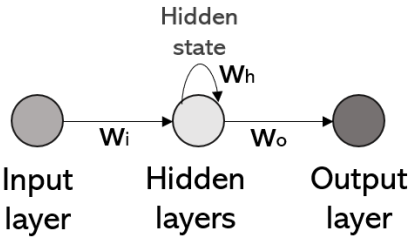

Figure 4. Recurrent NN schematic
Basic representation of a recurrent NN layer, where w_h stands for hidden state weights. w_h is updated each time input data is passed through the model and therefore influence the following prediction.

LSTM are improved recurrent NN that include additional layers called cell structure gates. These gates are in fact mathematical functions that add or remove information from the hidden state, such that the context is kept or reset. When training models with multiple sequences of data this becomes crucial since memory should be kept within a sequence but removed between sequences, so that two time series that do not have continuity do not influence each other outputs.

Parameters to take into account to train a LSTM do not change respect to deep NN, yet LSTM requires a higher number of epochs to train. LSTM models for clinical data show good performance while minimizing the need for data processing and feature extraction, but again are computationally expensive. [53].

1.3.3 Model performance

There are several metrics that can be used to determine the performance of a binary classification model. Given that a prediction can either match or not the actual label, there are four combinations between actual label and predicted label, which can be arranged in a confusion matrix (see table 3).

From a confusion matrix, several ratios can be obtained to evaluate model perform-

Table 3. Confusion matrix
Basic confusion matrix layout depicting combination between actual label and predicted label.

	Label Positive	**Label Negative**
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

ance. The most simple one is accuracy which displays the proportion of correct classifications, as in equation 4. However, accuracy by itself is not too reliable as it does not differentiate between positive and negative labels, which might lead to misjudgement, for instance, if a model yields many TP and none TN, since accuracy might be good even if the model is useless.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{4}$$

Instead, sensitivity and specificity are widely used in medicine [54]. Sensitivity refers to the ability of the model to correctly identify true positive patients, as in equation 5, while specificity refers to the ability of the model to correctly identify true negative patients, as in equation 6. This solves the problem accuracy introduced.

$$Sensitivity = \frac{TP}{TP + FN} \tag{5}$$

$$Specificity = \frac{TN}{TN + FP} \tag{6}$$

Moreover, proposed predictive models do not output the predicted label but rather a probability, and then a threshold must be set so that predicted probability values above the threshold are considered to be positive label. Obviously, different threshold values will yield different sensitivity and specificity so it is desirable to assess the performance of the model over different threshold values. This is achieved by the receiver operating characteristic (ROC) curve, which is a graphical plot that illustrates the relationship between sensitivity and 1 - specificity for all possible prob-

ability thresholds [55].

One effective way to summarize the overall performance of the test is by assessing the area under ROC curve (AUC). ROC AUC takes values from 1 to 0 with 1 being a perfectly accurate model, that is, sensitivity equals 1 even when specificity equals 1, and 0.5 representing half of the predictions right, that is, random probability which is illustrated by a straight line from origin to top right corner in ROC (see figure 5). ROC AUC values between 0.7 and 0.8 are considered acceptable, 0.8 to 0.9 is considered excellent and above 0.9 is considered outstanding [55]. Obviously, the latter assertion will depend on application, but it serves as a general idea.

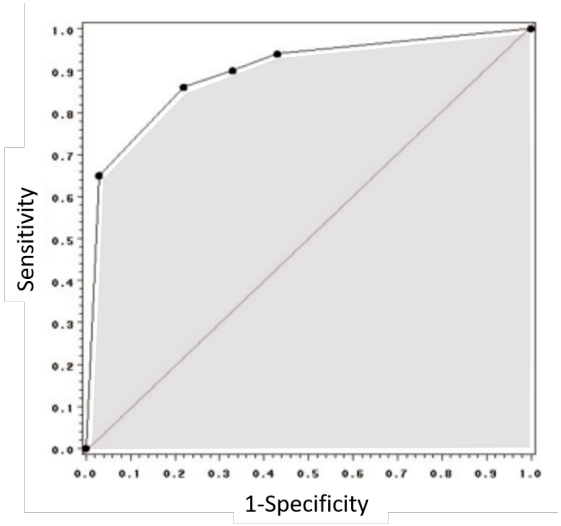

Figure 5. Receiver operating characteristic curve example
Example of a ROC curve as well as its area under curve (shaded region) and random probability (straight line from origion to top right corner). ROC AUC is higher than 0.5. Adapted from [55].

2 Objective

Motivation for this project is using data from several monitoring systems to build, train and validate predictive models capable of anticipating adverse clinical outcomes of the patient such that the anesthesiologist can administer the correct drugs in advance and prevent potential damage to the patient, improving personalized anaesthetic management.

2.1 State of the Art

The difficult airway (DA) has been defined as the clinical situation in which a trained anesthesiologist experiences difficulty with face mask ventilation of the upper airway, difficulty with tracheal intubation, or both [56]. DA can lead to many complications and therefore its prediction is key. Although there has been a major improvement over the last few years thanks to improvement in monitoring systems, complications related to DA continue to occur at an arguably high rate [57]. Traditionally, prediction of the DA involves previous clinical examination including imaging techniques such as ultrasound, computed tomography and nasendoscopy, among others [58]. These techniques allow to determine certain parameters which are known to predict DA. For LMA, predictors include reduced mouth opening and spine deformity [59]. However, prediction scores are rather poor [60], several methods have been explored with values for ROC AUC ranging from 0.71 to 0.81 [61–65].

Nonetheless, in this project the goal is not to predict DA but rather adverse events related to LMA insertion. In this matter, monitoring systems offer a wide variety of indices meant to represent the anesthetic effects of the patient. BIS accounts for hypnotic effect of the patient, but there are other indices displayed by many monitors. The Analgesia/Nociception Index (ANI), which is another confidential and proprietary algorithm, accounts for the analgesic effect. However, the issue with ANI is that even if it correctly displays the current anesthetic state of the patient, it fails to anticipate the state of the patient after a noxious stimuli [66]. In other words, it detects that the patient is suffering pain when the patient is already suffering pain, but it cannot predict it in advance. ANI and analogous indices do predict reason-

ably well changes in blood pressure and heart rate reaching ROC AUC values as high as 0.9 [67], but prediction of movement after noxious stimuli is poor with P_k values between 0.41 and 0.62 [68]. P_k is equivalent to AUC ROC, with 0.5 indicating random probability.

2.2 Market Analysis

Many monitors and indices are available which allow personalized monitoring of hypnotic effect and immobility [66] yet there is no evidence of an objective measure of analgesic effect [69]. There are several analgesia indices but all have drawbacks and limitations. For instance, pupillometry (measurement of pupil size and reactivity) based indices require extra effort and can only be used intermittently, while beat to beat analysis of BP and HR based indices need further validation, both approaches showing promising results [66].

Arguably, future of the field is closed loop TCI systems. Currently, TCI systems rely on statistical data (weight, height) and pharmacokinetic and pharmacodynamic models to determine the dose to be administered, but this approach is limited as it is based on population statistics. Automated closed loop TCI systems could integrate the data from the monitors to maintain anesthesia states on target enabling a personalized approach. So far, there is not any closed loop TCI system on the market, mainly due to the lack of good indices for the analgesic effect [66].

2.3 Concept Generation

In clinical set up it is hard to determine whether or not the patient will suffer from adverse events after LMA insertion, which are best avoided, since monitors and indices currently lack predictive power, as discussed on section 2.1. If adverse events could be predicted, a personalized management of drug administration would be provided leading to higher quality anesthetic management.

First approach that came to mind was turning this into a time series forecasting problem, that is, trying to predict future values of a continuous variable based on past and present data. Regression models can do this and it would make up for the lack of predictive power of indices. However, the goal is not to predict the values

of the variables at any time, the focus is at the specific moment of LMA insertion and the model must somehow account for that. One way to do this is including in the model a LMA variable made of zeros that turns one at the moment of LMA insertion, but this would require the practitioner to manually input that LMA is about to be inserted before finding out the prediction. This solution does seem to solve the problem but it is not quite straightforward, mainly because it requires an extra step for the anesthesiologist. Moreover, the output would be a set of predictions for desired continuous variables that would need to be analysed on top of present values for same variables, making it a tedious work.

Producing a simple solution is key, so it can be actually used in clinical set up and ultimately provide benefits for the patient without much extra work for the anesthesiologist. In this way, making it a classification problem is much simpler than regression, as output is a label indicating the presence or absence of adverse events, rather than a continuous value. This is a multi label classification problem, that is, more than one adverse events are possible within the same patient. This can be done either by training a single multi label model or by training one binary model for each adverse event with two possible outputs, namely, positive or negative for presence or absence of adverse event, respectively. This latter approach is chosen since different kinds of adverse events might be better predicted by different models.

Along the same direction, instead of manually inputting that LMA is about to be inserted, the model will be trained with data prior to LMA insertion as input and data following LMA insertion as output. This way, the model will account for it without requiring the practitioner to do any extra steps. The only point to be taken into account is that the anesthesiologist must know that the predictions are only valid at the moment of LMA insertion, as the model has been trained using data at that point and not another, but this should not be an issue.

Summing up, the proposed solution is to develop a classification model that accurately predicts for each patient the presence or absence of each adverse event following LMA insertion, without needing any extra preliminary work.

3 Solution Implementation

3.1 Overview

First step to develop a ML model is adequately defining the problem and desired outputs, which has already been covered in section 2.3. Next step is to gather data. In here, data acquisition has not been performed specifically for this project, but rather it is an ongoing process that is systematically carried out by the supervisor and his team, so it is more a matter of taking advantage of the existing dataset.

At this point it is important to define a measure of success so that the model can be evaluated once it is ready, otherwise there is no objective aim and it is not possible to work towards fulfilling it. Ultimately, the solution will be evaluated using ROC AUC, but accuracy will be used in intermediate steps for simplicity.

Once the data is ready and the goal is clear, the actual steps specific for this project can be defined (see figure 6). As in any ML project, preliminary steps include data processing and data exploration to get rid of artifacts and get an idea on how the data looks like. Then, the model development process includes tuning for each model the parameters that were mentioned in section 1.3. Unfortunately, this operation can only be done through a trial and error process by trying many parameters and finding out which yield better results, so this step is iterative.

When optimal values are known the final models can be trained and finally validated. Validation includes model testing and model explanation. Model testing accounts for how well the model generalises on data that has not seen before. Retrospective validation is done with a subset of data that has been separated from training data so the model can be evaluated, while prospective validation is done with data acquired after model development. Ideally, both retrospective and prospective validation should be done to ensure model success. In this case, prospective validation will not be possible since data acquisition is in standby due to the ongoing COVID19 situation. Model explanation is key in order to avoid a distrust situation, where humans might feel like there is not enough control on predictions [66]. To overcome this, it is important both to get an idea on how the model is making its predictions

and to identify any possible source of bias [70].

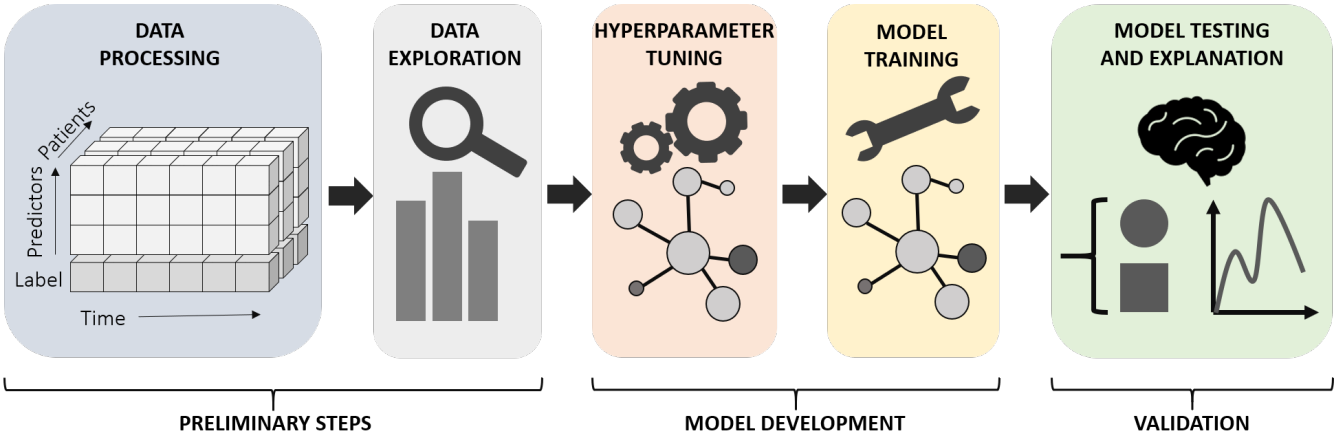

Figure 6. Project workflow
Illustration of the necessary steps followed to complete the project. Data processing section additionally illustrates the shape of the data.

3.2 Detailed Engineering

3.2.1 The data

The dataset used for this study belongs to the SPEC-M research group in Hospital Clínic of Barcelona. Data was recorded under the authorization of the Ethics and Clinical Research Committee (CEIC) of Hospital CLINIC de Barcelona (Ref nº 2013/ 8356). Data collected was completely anonymized from the start of the data collection process.

Data collection is extended to all patients undergoing TIVA procedure at operating room 4 of ambulatory surgery in Hospital Clínic, which is mostly dedicated to gynecologic procedures (therefore the dataset contains mainly female patients). Each patient is monitored with different monitoring systems as well as the TCI device. Data is collected with high resolution techniques at a rate of 1 entry per second, except for BP which is monitored intermittently. After the procedure, this data is stored in a computer and subsequently synchronized in order to obtain one record per patient. Synchronization is done through system time of the computer, which is also written down for each entry from each monitoring system.

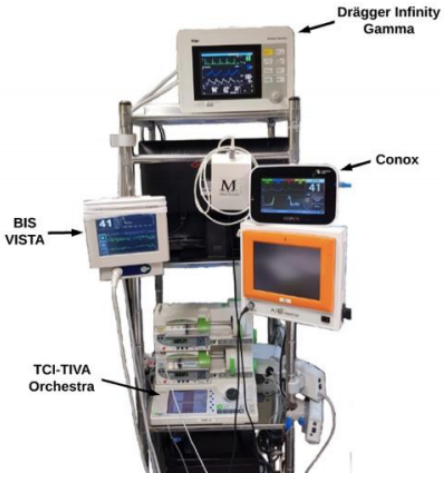

Figure 7. Data acquisition set
Front view of the data acquisition set up of SPEC-M research group in Hospital Clínic including a TCI system and three monitoring devices connected to a central computer.

Data acquisition set includes three monitoring systems and a TCI (see figure 7):

- BIS VISTA® Bilateral Monitoring System (Aspect Medical Systems, Inc, Norwood, MA): provides patient BIS by analysing EEG signal, obtained through electrodes placed on the forehead of the patient.
- Conox® (Quantium Medical, Mataró, Spain): provides both an index related to hypnotic effect and an index for analgesic effect. EEG signal is obtained by placing another set of electrodes on the forehead of the patient.
- Dräger Infinity® Gamma (Dräger, Lübeck, Germany): provides HR, BP and oxigen saturation signals by placing three electrodes on the chest, a cuff on the arm and a pulse oximeter on the index finger of the patient.
- TCI-TIVA Orchestra® Base Primea and two Module DPS (Fresenius Kabi AG, Homburg, Germany): TCI system that records infusion and concentrations of administered drugs, as well as demographic data.

Aside from Conox® all data is written down through RugloopII® software, so synchronization is actually between Conox® and RugloopII® files. Table 4 summarises all the variables that are stored in the resulting patient record. Events column stands

for several events that occur throughout surgery that are manually written down on RugloopII® software, such as the presence of movement, LMA insertion, adminitration of other drugs, and so on.

Table 4. Patient record variables
Variables at resulting patient records. Propofol variables are also available for remifentanil. Redundant variables have not been considered, such as those derived from weight and height.

Variable	Monitor	Description
ID	-	Record number (used as patient ID)
Time(s)	-	Time since start of recording
BIS	BIS VISTA®	Hypnotic effect index
BSBIS	BIS VISTA®	Burst suppression index
EMGBIS	BIS VISTA®	Electromiographic signal intensity index
SQI09	BIS VISTA®	Quality index to assess BIS reliability
qCON	Conox®	Hypnotic effect index
qCONBS	Conox®	Burst suppression index
qCONEMG	Conox®	Electromiographic signal intensity index
qCONSQI	Conox®	Quality index to assess qCON reliability
qCONqNOX	Conox®	Analgesic effect index
nHz	Conox®	EEG spectral analysis (n=0-127Hz)
HR	Dräger Infinity®	Heart rate (beats per minute)
NIBPsys	Dräger Infinity®	Systolic BP (mmHg)
NIBPdia	Dräger Infinity®	Diastolic BP (mmHg)
SPO2	Dräger Infinity®	Percentage oxyhemoglobin in blood
RespiRate	Dräger Infinity®	Respiratory rate (breaths per minute)
CpPROPO	TCI Orchestra®	Propofol plasmatic concetration
CePROPO	TCI Orchestra®	Propofol effect site concetration
InfRatePROPO	TCI Orchestra®	Propofol infusion rate
InfVolPROPO	TCI Orchestra®	Propofol infused volume
Age	TCI Orchestra®	Patient age
Height	TCI Orchestra®	Patient height
Weight	TCI Orchestra®	Patient weight
Events	RugloopII®	Surgery related events

3.2.2 Data processing

The data is modified using Python software, specifically using `pandas` module and occasionally `numpy` module.

The data has two main issues that need to be covered in the first place. First issue is regarding missing values since there are many, for instance, qCON reading goes to 128 whenever there is an artifact in the signal. To make up for this, missing values are first set to `NaN` (not a number) and interpolated afterwards using a linear method, that is, a straight line between available entries is fit to estimate the value of the missing ones in between them, by using the function `interpolate()`. Second issue is that Events column is chaotic as events are written down manually, which leads to different entries meaning the same events. To make up for this events are manually classified and then a column is created in each patient record for each event. For instance, LMA variable will be a column with zeros and a one at LMA insertion time. This is done both for LMA insertion and for movement.

After the missing values and the events issues are handled, records that are incomplete have to be excluded from the analysis, and so do records where intubation was used instead of LMA. Then, records must be classified according to adverse events (see figure 8A). As suggested by supervisor, adverse events to be predicted in this project are movement, light hypnosis (BIS>60%) and hypotension (mean BP<60mmHg).

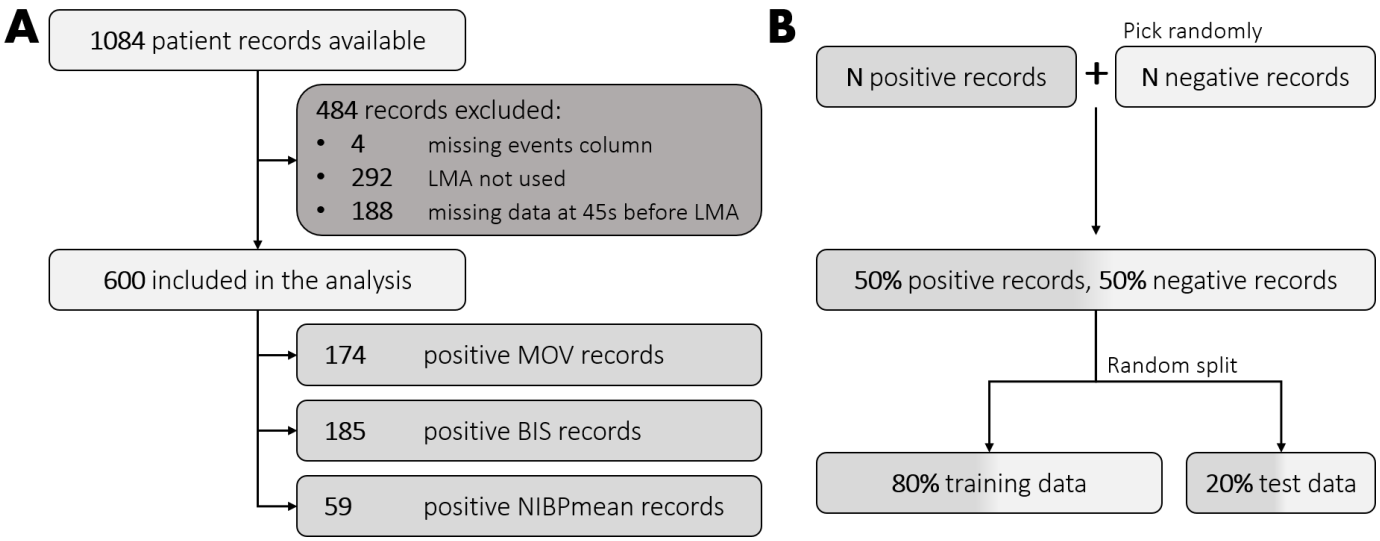

Figure 8. Data inclusion and split
A) Inclusion critera and classification. B) Data split into train and test procedure, in order to keep a balance between positive and negative records.

Even if 600 patient records are included in the analysis, only about one fourth of them are positive for movement or for light hypnosis, and about one sixth of them for hypotension. This is an issue because ML models are biased towards the overrepresented group in imbalanced datasets [71]. To overcome this, not all negative records will be used, but rather a random sample of them will be picked to match positive records (see figure 8B). Then, 80% of patient records will be randomly chosen as training data, and the rest will be used as validation data.

Finally, the data needs to be prepared to train the models by normalizing it and selecting the region of interest, namely, the moment of LMA insertion. Data normalization is necessary since the range for different features varies widely and feature contributions on the final model will depend on it. The data is manually scaled between 0 and 1 using a min max normalization method, as in equation 7 where min and max stand for minimum and maximum values for each feature.

$$scaled = \frac{value - min}{max - min} \tag{7}$$

The moment on LMA insertion is spotted and the 45s preceding it are selected as input data. To determine output data, that is, the label for each adverse event, the 45s after LMA insertion are analyzed (see figure 9). For each patient, if at least one output entry fulfills the adverse event condition, the whole output is set to 1 (positive) for that adverse event, otherwise, it is set to 0 (negative). The 45s boundary is chosen as a trade off between two criteria. On the one hand, input data must be accountable for patient state at the right moment that precedes LMA insertion but not earlier, since that is the moment where the anesthesiologist decides that patient hypnotic, analgesic and immobility states are deep enough. On the other hand, output data must be accountable for any adverse events caused by LMA insertion, which might take up to few minutes. Obviously, input data and output data must have the same length. 45s is chosen as it does not go much back in time, and most movement events after LMA insertion (about 85%) happen before 45s.

Variables included in the analysis are in table 4, but frequencies are restricted from 0 to 30Hz to match waves displayed in table 2, leading to a total of 59 variables to be

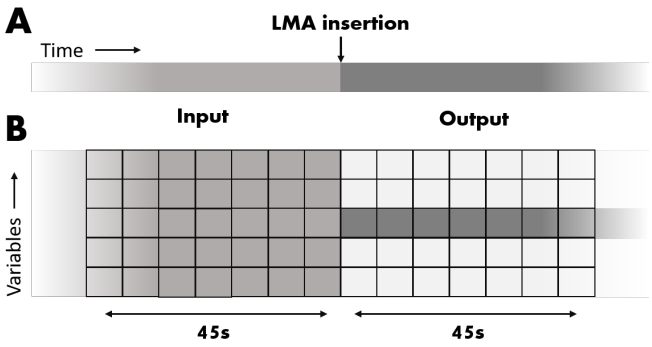

Figure 9. Input and output definition
A) Input definition at 45s preceding LMA insertion and output at following 45s. B) Input data is based on all variables, while output is restricted to one variable, representative of the adverse event.

used as predictors. This process is reproduced three times, one for light hypnosis, one for movement and one for hypotension.

Therefore, the resulting subsets have three dimensions, namely, number of patients, number of variables (59 predictors and the label) and time (45s per patient) (see figure 6). However, only LSTM model accounts for time and the rest of the models do not. Therefore, data is flattened to two dimensions such that columns represent variables and rows represent all entries from all patients.

3.2.3 Exploratory analysis

Data is ready to start model development but prior to that some exploratory data analysis is performed just in case there are some anomalies. At this point, data is divided into three subsets corresponding to each adverse event group. Each subset contains half and half positive (labeled 1) and negative (labeled 0) records and is divided into training (80%) and testing set (20%).

There are multiple useful tools for exploratory data analysis. In here, principal component analysis (PCA) is used. PCA is an unsupervised ML technique that creates new variables, called principal components (PC), by doing a linear combination of the initial variables in the data in such a way that these PCs are uncorrelated, that is, orthogonal, and most variability is captured within first PC, then the second PC

captures most of the remaining variability, and so on. At the end, there are as much PCs as initial variables, but most variability can be explained through first PCs.

PCA is performed on the data using R software, specifically using the function `prcomp()` from the `stats` library

In our data, there are 59 variables and it is expected that some of them are strongly correlated, for example, systolic, diastolic and mean BP or weight and height. Therefore, it is expected that few PCs will explain most variability. However, it takes up to 30 PCs to cover for 90% of variability (see figure 10A), which is quite a lot. PC1 only explains about 12% of variability, and the 5 variables that most contribute to it are EEG derived indicators (see figure 10B), PC1 is therefore an EEG summary.

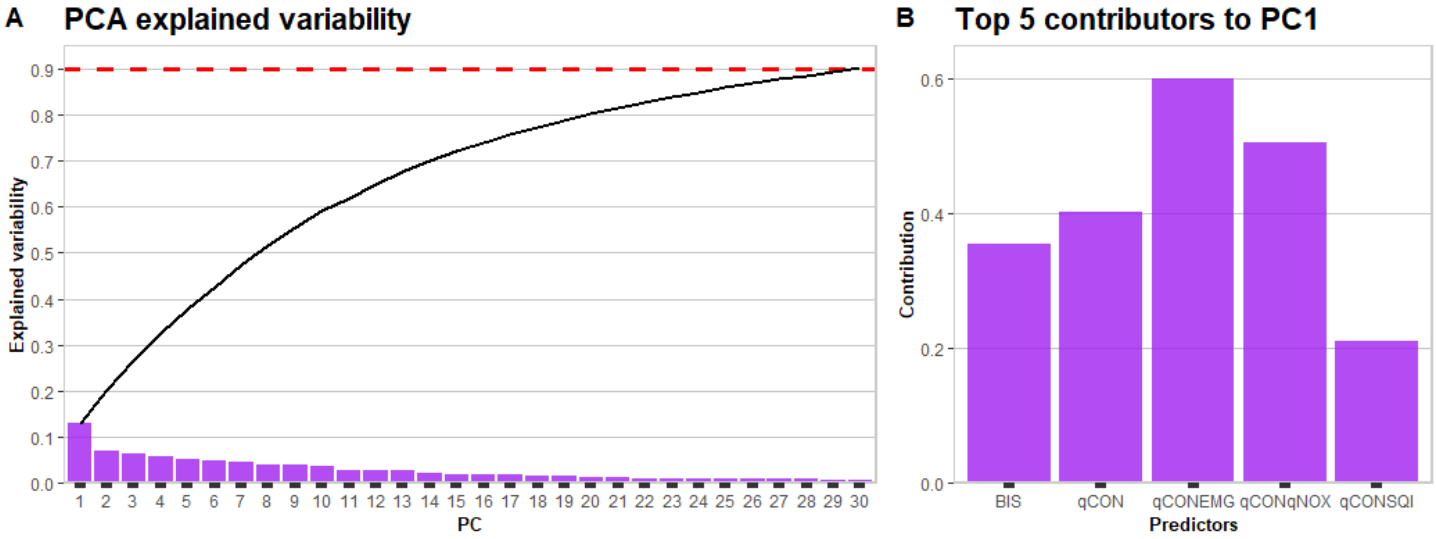

Figure 10. Principal component analysis results
A) Relative (column) and cumulative (line) variability explained by each PC. B) Contribution to the first PC of the 5 variables that most contribute to it.

PCA results show that data normalization has been properly done, since there seems to be no range effect on PCs. This is confirmed by making scatter plots of the first few PCs and seeing that they do not go beyond 1. These scatter plots are included in annex A, and they also show that there is some difference in positive and negative groups within three first PCs for light hypnosis and hypotension events, but not for movement. This fact already gives an idea that movement will be harder to predict, but it is not determinant since three first PCs do not explain much variability.

In light of these results, the possibility of making some feature engineering to reduce data dimensionality arises, since the models could be trained with PCs and require less inputs while gathering a similar amount of information. However, this would mean adding an extra computational step when applying the solution in clinical practice, as data would have to be transformed before it could be used as an input to the model, and it would not provide any benefit other than less storage capacity required. Since storage is not an issue, this option is rejected. Nevertheless, other feature engineering techniques will be discussed in section 4.6.

3.2.4 Model hyperparameter tuning

Hyperparameter tuning refers to the problem of choosing the best hyperparameters for a predictive model. Hyperparameters are parameters whose value controls the learning process, as opposed to parameters that are learned during the learning process, such as weights. In section 1.3, relevant hyperparameters for each model were covered (the term hyperparameter had not been introduced yet).

As previously explained, there is no rule of thumb on which hyperparameter values generalise better for a given problem. To optimize model hyperparameter values, the only way is to try many values and see which combination performs better by using a certain model performance metric. Of course, this cannot be done by hand, if there are 4 hyperparameters and 4 values are tested for each of them this already yields a total of 256 (4^4) different combinations and the number of combination grows exponentially, so it is just not manageable.

Instead, there are some Python modules that automate this routine in a simplified way. As inputs, they take the baseline model (see section 3.2.5) and a parameter grid, that is, a list with the values to be tested for each hyperparameter. As output, they provide a matrix where each entry is a possible combination of hyperparameter values as well as model performance when trained with it. Therefore, a model performance metric must be defined. For simplicity, accuracy will be used rather than ROC AUC. This should not be an issue because the data is balanced and accuracy and ROC AUC should correlate. Of course, accuracy is evaluated on test data, otherwise a good performance might be due to overfitting.

Training hundreds of models and evaluating them is computationally expensive so it takes some time. This is relevant because it is nor feasible nor efficient to test a large amount of values. Instead, a randomised search can be done to test some combinations rather than all of them. This was done as a preliminary step to narrow down the spectrum of possible hyperparameter values before moving to the actual testing of all combinations. That being said, a distinction between DL models and the rest of ML must be done, since hyperparameter tuning is worked out differently for them.

Machine learning models

Hyperparameter tuning of ML models was done using the module `sklearn` which covers a wide variety of tools for different problems.

For ML models cross validation (CV) was used in order to make sure that the chosen combination of hyperparameter values is the optimal one. Even if initial data split to train and test was random, it is still possible that it got a lucky partition leading to a model that apparently has a good performance but does not generalise well on new data. Four fold CV was performed, such that every parameter combination was tested four times, with different test data each time. Of course, this supposes four time as computational efforts and therefore computation time, but ML models are not that computationally expensive to train and CV is manageable. The function `GridSearchCV()` was used for this purpose.

For random forest (RF) the tested hyperparameters were the number of trees (`n_estimators`), the minimum leaf size (`min_samples_leaf`), the minimum node size (`min_samples_split`) and the maximum depth (`max_depth`). For *k*-nearest neighbor (kNN) the tested hyperparameters were the number of neighbors (`n_neighbors`), the weight function (`weights`) to be chosen between uniform or weighted, the leaf size (`leaf_size`) and the algorithm used (`algorithm`) to be chosen between automatic (the function chooses an algorithm according to data) or ball tree (fast approach). For support vector classifier (SVC) the tested hyperparameters were the regularization parameter (`C`), the kernel function (`kernel`) to be chosen between linear, polynomial, sigmoid or radial basis function (RBF) kernel and the kernel coef-

ficient (gamma) to be chosen between automatic or scaled.

All tested values and the best combination for the prediction of each adverse event are displayed in table 5.

Deep learning models

Hyperparameter tuning of DL models was done using the module talos. The great thing about this module is that baseline models are built using Keras and TensorFlow (see section 3.2.5) so no further language needs to be learnt.

The deal with DL models is that training them is much more computationally expensive. Moreover, compared to the rest of ML models, there are more hyperparameters to take into account and therefore more combinations. Because of this, CV is no longer feasible as it would take too much time. Just to illustrate the point, tuning a long short term memory (LSTM) model took about seven hours (without CV) while tuning a RF model took about one hour (with 4 folds CV), and this process was repeated three times, one for each event to be predicted. Therefore, CV was not done for DL models and instead they were tuned with the initial random split in train and test data. Naturally, this comes at a cost, since if the initial partition was lucky performance might be overestimated and generalisation on new data might not be as good. The Scan() function was used.

For neural network (NN) the tested hyperparameters were the learning rate (lr), the number of neurons (num_neurons), the number of layers (hidden_layers), the number of epochs (epochs), the batch size (batch_size) and the dropout rate (dropout). For LSTM the tested hyperparameters were the same, but using LSTM layers (LSTM_layers) instead of hidden layers.

As explained in section 3.2.2 the data was flattened for simplicity, so it needs to be reshaped from two dimensions to three, that is, number of patients, number of variables and time, so the LSTM model can be tuned, as it is important that memory is kept within a patient but not between patients.

All tested values and the best combination for the prediction of each adverse event are displayed in table 5.

Table 5. Hyperparameter tuning
Tested values for each hyperparameter and each model, along with the combination that yields better validation results for each one of the three adverse events to be predicted (BIS >60%, movement, mean BP >60mmHg). In the last columns parameters are abbreviated by the first letter.

	Tested parameters	**BIS**	**MOV**	**NIBP**
RF				
max_depth	10, 30	10	30	10
min_samples_leaf	2, 4	4	2	2
min_samples_split	5, 10	5	10	5
n_estimators	100 - 2000 by 100	200	400	100
kNN				
n_neighbors	100 - 1000 by 20	740	100	100
weights	uniform, distance	d	u	d
leaf_size	5, 30	5	5	5
algorithm	auto, ball_tree	a	a	a
SVC				
C	0.1 - 1000 by x10	10	0.1	10
gamma	auto, scale	a	s	a
kernel	linear, rbf, poly, sigmoid	r	r	r
NN				
learning_rate	3.5, 5	5	3.5	3.5
num_neurons	50 to 100 by 10	80	100	90
hidden_layers	2, 3	3	2	3
batch_size	16, 32, 64	16	16	16
epochs	10, 50	10	50	10
dropout	0.1, 0.2	0.2	0.1	0.2
LSTM				
learning_rate	3.5, 5	5	3.5	3.5
num_neurons	25 to 75 by 10	35	75	75
LSTM_layers	1, 2	1	1	1
batch_size	16, 32, 64	64	64	16
epochs	200, 300	300	200	200
dropout	0.1, 0.2	0.1	0.2	0.1

3.2.5 Model training

For each one of the three adverse conditions to be predicted, all models are trained using the hyperparameter values displayed in table 5, that is, the combination that was found to be optimal for that adverse event. Again, DL models are built differently from the rest of ML models.

Machine learning models

ML models were built using the module `sklearn`. This module includes several packages in which many ML classifying models can be found. The way they work is simple, first the model is initialised with the desired set of hyperparameters and then the model is trained by calling function `fit()` on it and providing the training data separated by input (predictors) and output (label). Once the model is trained, predictions can be made by calling the function `predict_proba()` on the model and providing test data (only the input). This way, the model outputs the predicted probability of each entry belonging to the positive class, which can be later compared to actual label (see section 3.2.6).

The RF models were built using the function `RandomForestClassifier()` from the `ensemble` package, the kNN models were built using the `KNeighborsClassifier()` from the `neigbors` package and the SVC models were built using the function `SVC()` from the `svm` package. Something odd about SVC is that it does not predict a probability but rather a label directly, and probability can only be obtained by using CV to train many models and then averaging their predictions. This can be done by setting the `probability` argument to `True`.

Deep learning models

DL models were built using the `Keras` module, which is a high level DL API that runs on top of `TensorFlow` module, which is a lower level library for ML applications. Since DL models can have many architectures, building the models might require few more operations compared to the rest of the ML models, but the main steps are the same. The model is initialised and trained by calling the function `fit()` on it and providing the training data. When training the model, the batch size and the number

of epochs must be specified. Once the model is trained, predictions can be made by calling the function `predict()` on it and providing test data.

Models were built using the `Sequential()` function, which allows to create a model and subsequently add layers one by one using the `add()` method, starting at the input layer for which the input shape must be specified. In the case of the NN, hidden layers are added using the function `Dense()`, its arguments being the number of neurons, the dropout rate and input shape, which is the number of variables. On the other hand, LSTM layers are added using the function `LSTM()`, for which number of neurons and dropout rate also have to be specified, but input shape is two dimensional, that is, the number of variables and the number of seconds per record (so memory is not kept between patients). Finally, the output layer is a `Dense()` layer with only one neuron. Once all layers have been defined, the function `compile` is called to define the loss function, the performance metric and the optimizer function.

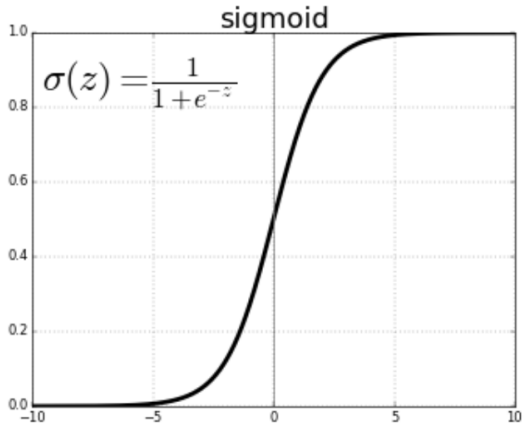

Figure 11. Sigmoid function
Activition function used for DL models, such that the output for each neuron is a value between 0 and 1.

Sigmoid function is used as activation function for all layers. The sigmoid function yields values between 0 and 1 (see figure 11) as output, which is relevant because we want the output layer (which only has one neuron) to be a value between 0 and 1 that represents the predicted probability of an input belonging to positive class. Other neurons could use other functions but sigmoid is used such that output values

are normalized, just like data is.

Since this is a binary classification problem, the loss function used is binary cross entropy, which is a mathematical function that tries to minimize missclassifications between two groups. Similarly, the optimizer function defines in which way weights are updated after each epoch. In here, the Adam optimizer [72] is used, since it is a popular and widely accepted method.

Once the model is built, the `summary()` method can be used to display model architecture as well as the number of trainable parameters (weights) that will be updated when training the model. LSTM layers have way more trainable parameters than dense layers even with less neurons (see figure 12), which makes sense since each LSTM cell contains many gates as well as hidden state weights.

A

```
Model: "LSTM model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 35)                13300
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 36
=================================================================
Total params: 13,336
Trainable params: 13,336
Non-trainable params: 0
_________________________________________________________________
```

B

```
Model: "NN model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 80)                4800
_________________________________________________________________
dropout_1 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_2 (Dense)              (None, 80)                6480
_________________________________________________________________
dropout_2 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_3 (Dense)              (None, 80)                6480
_________________________________________________________________
dropout_3 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 81
=================================================================
Total params: 17,841
Trainable params: 17,841
Non-trainable params: 0
_________________________________________________________________
```

Figure 12. DL models summary
Model architecture and number of trainable parameters (weights). A) LSTM model for one LSTM layer with 35 neurons. B) NN model for three hidden layers with 80 neurons each.

3.2.6 Model validation

Validation includes model testing and explanation. As discussed, model testing will be assessed using ROC curves as well as its AUC. Something to take into account is that models take as input 45 entries per patient (one each second), so each pa-

tient gets 45 predictions (one output per entry), except for the LSTM model which inputs the whole 45s sequence and therefore each patient gets only one prediction. This is the reason why ROC curves cannot be built straight away, since comparison between LSTM and rest of the models could not be done as they would be displaying different things. To make up for this, model predictions are summarised into one prediction per patient such that if at least one of the 45 predictions for a patient is positive, the patient is said to be predicted as positive. This method might look troublesome at first sight since if a patient gets one positive and 44 negative predictions it is still set to positive, but bear in mind that most entries for a patient are almost equal so different predictions within a patient are not likely to happen. With this issue solved ROC curves can be compared between all models (see figure 13).

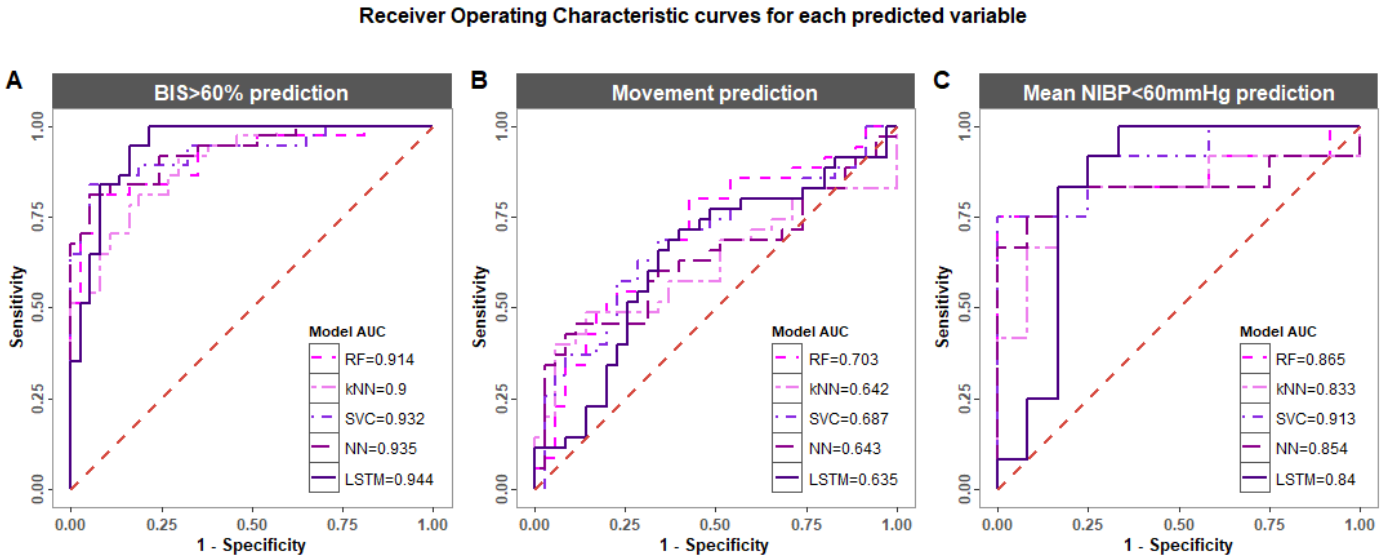

Figure 13. ROC curve results
ROC curves for each model as well as its AUC (legend) for adverse event prediction following LMA insertion. A) Light hypnosis prediction (BIS above 60%). B) Movement prediction. C) Hypotension prediction (mean BP below 60 mmHg).

By looking at the results, the first conclusion that can be drawn is that predictions for movement are way worse than predictions for light hypnosis and hypotension. This is consistent with what has been previously reported on the literature and with expectations from PCA analysis. Nonetheless, RF movement prediction does get a

ROC AUC above 0.7 (see figure 13B) which indicates that there might be some room for improvement leading to more accurate predictions, by making some changes on how the model is built. In here, any movement was considered a positive label, from minor finger movement to body convulsions, which might be inconvenient since the difference between slight movement and no movement might be subtle (or none at all). One possibility is that trying to predict severe movement (instead of any movement) might yield better results as there might be more variability in the data. Obviously, previous studies should be carried out on what features are more correlated to different intensities of movement and perform some feature engineering to come out with reliable predictors.

On the other hand, hypotension and light hypnosis predictions are quite solid. In the case of light hypnosis prediction, the best performing model is the LSTM with a ROC AUC of 0.944. Moreover, it reaches a sensitivity value of 1 with a specificity of 0.784, which means that it can correctly predict 78.4% of the light hypnosis cases without yielding a single false positive, in other words, light hypnosis condition after LMA insertion can potentially be reduced by 78.4% by increasing drug dose whenever a prediction is positive, without administering too much anesthetic drugs to patients that do not need it. In the case of hypotension prediction, the best performing model is the SVC with a ROC AUC of 0.913, but it is again the LSTM model that yields the highest specificity value (0.667) for a sensitivity of 1.

It is important to remember that hyperparameter tuning of the LSTM and the NN models was not performed using CV, so there is a higher chance that they are biased compared to the rest of the models. This is precisely why a prospective validation is required and until it is done the claims regarding ROC AUC, sensitivity and specificity are not well grounded even though they should not be too far from reality. Another issue that has not been covered is that ROC AUC metric must be assessed with a 95% confidance interval such that potential bias effects on ROC AUC can be appreciated [73, 74], so confidence intervals should be included in a prospective validation. Moreover, prospective validation data should be processed differently such that zero order interpolation is used instead of linear method, because linear interpolation requires knowledge of future values in order to estimate current ones and such knowledge is not available in clinical practice.

The next step towards model validation is to explain what components are influencing the most on model predictions, such that the practitioner understands what factors are driving the model to make a prediction and the black box nature of ML models is somehow debugged, leading to an increase in trust. There are many ways of building explainable models. For simplicity the feature importance attribute of the RF models is used.

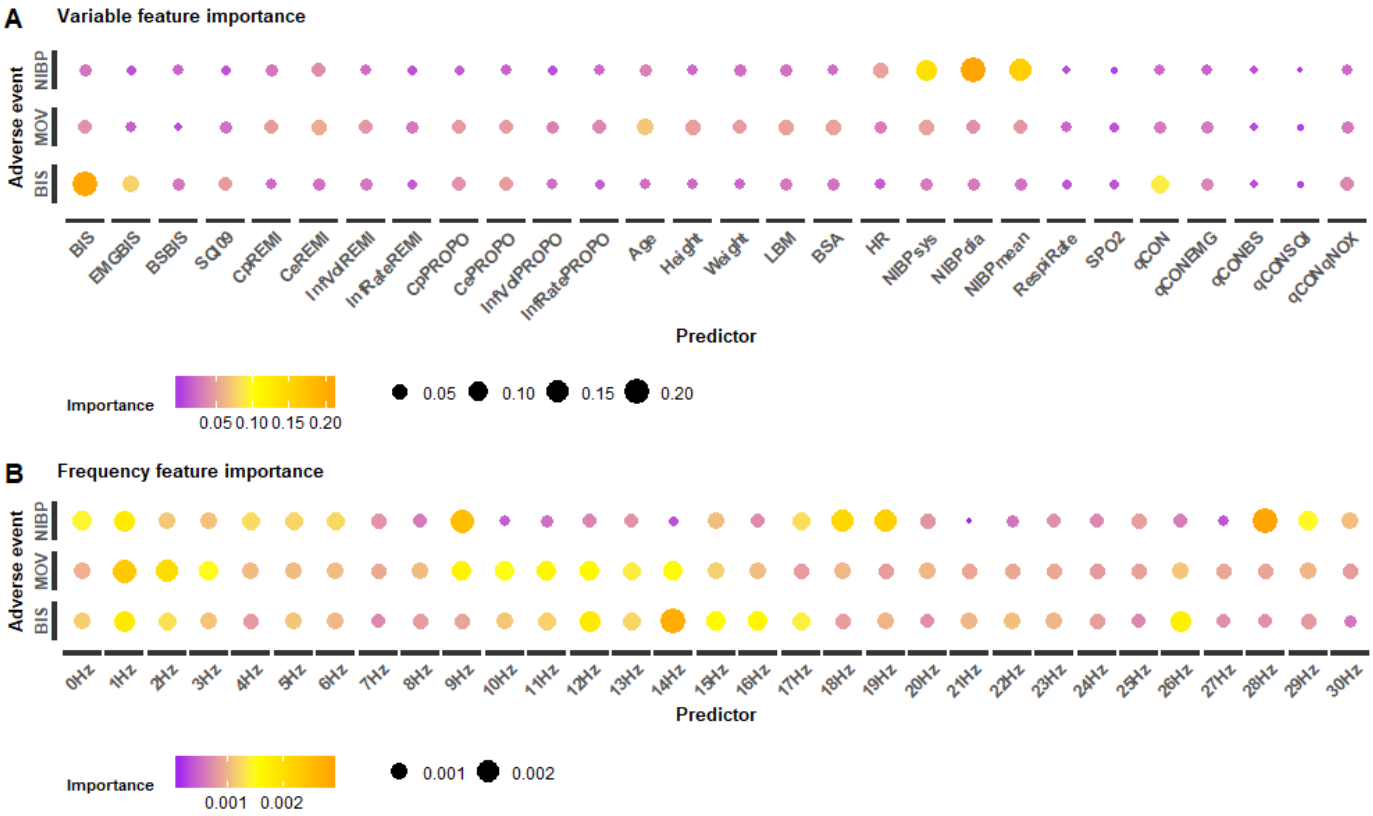

Figure 14. Feature importance
Feature importance for each one of the adverse events prediction on RF model. BIS adverse event refers to light hypnosis (BIS>60%), NIBP to hypotension (mean BP<60mmHg) and MOV to movement. A) Indices and signals from monitoring systems. B) Frequency intensity from EEG spectral analysis. Color and size scales are not the same on both plots.

Figure 14 shows feature importance on RF model predictions for each of the predicted events. First conclusion that can be drawn is that frequencies from EEG spectral analysis influence way less than the rest of the predictors. Putting the focus

on indices and signals from monitoring systems (see figure 14A), some interesting aspects of the model can be appreciated. For light hypnosis prediction, EEG derived parameters both from BIS VISTA® and from Conox® have the highest influence, which makes sense as they account for the hypnotic effect of the patient. Linking it to the fact that these parameters hold most of data variability, as seen with PCA, this explains why light hypnosis predictions are better than predictions for other adverse events. Moreover, propofol concentration parameters also have quite high importance, which is reasonable given that propofol is a hypnotic agent. For hypotension prediction, BP parameters have the most influence, which again makes sense. On top of that, HR also holds quite high relevance followed by remifentanil concentrations, which comes in no surprise as hypotension is one of remifentanil side effects. Movement prediction is less clear, BP parameters and EEG derived parameters seem equally important, demographic data plays a bigger role than with other models and both remifetanil and propofol concentrations are significant.

Even though they are not as important, by looking at the predictors from EEG spectral analysis (see figure 14B) it can still be seen that some frequencies influence more than others. For all three adverse events frequencies below 4Hz, corresponding to δ waves, are quite important. For both light hypnosis and movement models frequencies between 8 at 15Hz, corresponding to α waves, are important, but not for hypotension model. For hypotension model, some frequencies have higher importance but there does not seem to be a pattern so it might just be due to overfitting.

This analysis only holds for the explainability of RF models and it cannot be stated that the rest of the models operate this way. However, it can be assumed that important features will not be that different between models. Still, if some of the models other than RF are chosen after prospective validation, a model explainability experiment should be performed to understand what features drive the predictions, since interpretability is key in clinical practice. Many algorithms for this goal have been proposed, but for consistency and standardization SHAP (SHapley Additive exPlanations) methodology should be applied [75].

The last necessary step towards model validation and explanation is assessing any possible bias [70]. It is not that biases are bad in itself since personalized medicine

implies that each patient is considered differently, but it is important to identify undesirable biases such as skewed performance depending on race or gender, and estimate risks to ultimately ensure patient wellbeing [76, 77]. AI models might inherit such biases from an unbalanced training dataset such that performance in the majority group is put before performance on the underrepresented group. This problem must be identified and, whenever possible, solved by balancing the training data [78].

In this project, the dataset used is clearly unbalanced towards female gender since data is mostly from gynecologic surgery. Moreover, data is from Hospital Clínic in Barcelona where population is diverse but there is a higher percentage of white people and it is likely that other ethnicities and races are underrepresented in the trainig data. At this point, there is no way to figure out whether or not model performance is biased based on race or gender, but there is a chance it is and further studies are required on this matter.

4 Discussion and Future Outlook

4.1 Implications

The fact that LSTM models are the ones that have the most solid performance indicates that the dataset is best analysed as time series, otherwise there is information that is lost, linked to the temporal component in the data.

Preliminary results suggest an increase in ROC AUC compared to state of the art (see section 2.1) obtained by integrating many different signals from several monitors, but further validation is required. More interestingly, light hypnosis and hypotension prediction models show a specificity of 0.784 and 0.667 for a sensitivity of 1, respectively, suggesting that these occurrences could be reduced by 78.4% and 66.7% at no cost (other than the drugs required to avoid them), that is, no patient would be administered a higher than required drug dose based on a false positive prediction.

Movement prediction models reach a ROC AUC of 0.703 which can be considered acceptable, hinting that there is room for anticipating movement after LMA insertion and that some methodology changes, such as performing feature engineering or classifying movement intensities, might result into reliable predictive models.

The proposed solution must be regarded as a tool to aid the anesthesiologist in clinical practice, but it could also be integrated into an automated closed loop TCI system such that predictions would be analysed automatically and drug concentrations would be adjusted accordingly. All the same, the proposed solution enables a personalized approach for anesthesia administration.

4.2 Limitations

Considering that the proposed solution is data driven most limitations will arise from training data. Even if the dataset used for this project is of very high quality, taking into account the amount of physiological signals captured, the number of patient records it contains is not that large compared to similar studies, for instance, a LSTM model for hypoxemia prediction during anesthesia was trained using only one vari-

able (oxygen saturation), but it had data from more than 50 thousand records [79]. Something quite revealing about hyperparameter tuning results (see table 5) is that LSTM models performed better with just one LSTM layer in all three adverse conditions, which might be because there is not enough data to train a higher complexity model without overfitting it. The lack of high quality large scale datasets is the main limitation in most DL applications in medical field [52].

Other limitations regarding the data might come to light due to the imbalance of the dataset, which contains mostly white European female patients. This needs further study.

Since the models were trained on data from 45s before LMA insertion its predictions are only valid at that specific time. Using these models at any other time during surgery will likely result in erroneous predictions. Moreover, these models assume that the anesthesiologist has correctly done all the necessary steps for safe LMA insertion before model predictions are made, that is, drug administration is stable and verbal response and eyelash reflexes are lost, amongst others.

Finally, it is important to point out that these models will only work if all monitoring systems used to obtained the dataset, namely, BIS VISTA®, Conox®, Dräger Infinity®, TCI-TIVA Orchestra® and RugloopII®, are working and its indices can signals can be extracted and processed in real time.

4.3 Technical Viability

The goal for this project is to develop prediction models to anticipate adverse events after LMA insertion. At this point it is necessary to assess the feasibility of the proposed solution. One way to evaluate technical viability of the project is through strengths, weaknesses, opportunities and threats (SWOT) analysis, which is a tool to determine the position of the solution with regards of it internal attributes (characteristics of the project) and external attributes (characteristics of the environment) considering both helpful and harmful aspects, leading to the four categories responsible for its name.

Table 6 displays SWOT analysis of the proposed solution. Overall, it seems like

strengths and opportunities are satisfactory and that the solution could make it in the market. However, weaknesses play an important role mainly because the solution is only useful for LMA insertion and it might not be worth the trouble for anesthesiologists. Probably, this solution should be integrated within a larger scheme of prediction models for different anesthesia problems to ensure that implementation is worth the effort.

Table 6. SWOT analysis
Strengths,acquisitio weaknesses, opportunities and threats of the proposed solution.

	POSITIVE	NEGATIVE
INTERNAL	• Prediction will either be positive or negative → Easy interpretation • No added effort • Decrease in adverse event occurrence	• Only useful for LMA insertion • Requires having many active monitoring systems • Movement prediction is still not too accurate
EXTERNAL	• Lack of predictive ability from state of the art monitoring systems • AI not yet commonly applied in anesthesia	• There are many monitoring systems available • Clinical practice remains unchanged for many years → Reluctance to change

4.4 Economic Viability

The budget for this project is divided into the expenses required for data acquisition, the hardware and software for data processing and model development and finally human resources.

Data acquisition was not performed specifically for this project and the required monitors would need to be used either way by the anesthesiologist if data was not recorded. Nonetheless, it would not be necessary to use both Conox® and BIS VISTA®, one of them would be enough. Table 7 provides an estimate of the costs of having both monitors on for 600 surgery procedures. Again, this is not a direct cost of this project but rather an estimate for the costs of obtaining the database (counting only the records included in the analysis and not counting the TCI and the hemodynamic monitor), so the subtotal for data acquisition should not be regarded as a cost of the project.

Table 7. Data acquisition costs
Estimated costs of obtaining a database of 600 patient records, like the one used in this project. One of the monitors should have been used either way.

Concept	Units	Price/unit	Total (EUR)
Conox® monitor	1	2000	2000
BIS VISTA® monitor	1	11100	11100
Conox® electrodes	600	13.15	7890
BIS VISTA® electrodes	600	21.15	12690
Subtotal			33680

Data processing and model development was performed using open source software mostly. Still, the process is somewhat computationally expensive so a powerful enough computer is required. Moreover, dataset is stored in `.xlsx` format and Microsoft Excel® is used to access the patient records. Table 8 gives an estimate for the costs of software and hardware used for the project.

Finally, human resources refers to the working hours of people for project development. Table 9 provides an estimate of the costs of developing the project according to working hours from undergraduate and senior supervisor.

What makes this project feasible is the database that the supervisor and his team are continuously enlarging allowing to obtain multiple data driven solutions. Making use of this database, this project is cost effective since no significant additional expenses are required.

Table 8. Data processing costs
Estimated costs of software and hardware required for analysing and processing the data as well as model development.

Concept	Units	Price/unit	Total (EUR)
Python software	1	Open source	0
R software	1	Open source	0
Microsoft Office® 365	1	69	69
Lenovo ideapad 330	1	600	600
Subtotal			669

Table 9. Human resources costs
Estimated costs of working hours from people that has worked on this project.

Concept	Hours	Price/hour	Total (EUR)
Undergraduate	300	8	2400
Senior supervisor	20	30	600
Subtotal			3000

4.5 Legal Issues

The proposed solution lies inside the category of software as medical device (SaMD). Some AI solutions within the same category have been previously approved by the Food and Drug Administration (FDA) [80], which is the regulatory agency responsible for medical devices market approval in the United States. As a matter of fact, the FDA has proposed a regulatory framework specifically for AI and ML based SaMD [81].

Like any medical device eligible to be marketed, the FDA framework requires a risk categorization according to the International Medical Device Regulators Forum (IMDRF), that is, according to the significance of the SaMD for the healthcare environment (intended use of the SaMD) and according to the state of the healthcare condition. In here, the proposed solution is intended to inform clinical management and potentially to drive clinical management, while the healthcare condition (anesthesia) ranges from non serious to serious. According to this, The risks of the proposed solution lie within categories I and II (on a scale from I to IV where I in-

dicates the lowest risk). On top of that, it requires an evaluation of performance, used inputs and intended use, elements that have been covered in this project. FDA also proposes good machine learning practices (GMLP), specific for SaMD that use ML, that include evaluating the relevance of the data used to train the model for the specific application and an appropriate level of transparency of generated outputs towards the users. These items haven covered even though the explainability issue has been only partially evaluated.

On the other hand, in Europe there are no laws or standards specific for SaMD that use AI. However, these devices must match some categories within the Medical Device Regulation (MDR), which is the regulatory framework within the European Union. For instance, it is mandatory to demonstrate the benefits of using the SaMD by evaluating sensitivity and specificity. In addition, ML applications must be developed in a way that ensures repeatability. Overall, the requirements in the MDR match those proposed by the FDA.

Once the model is prospectively validated and possible biases are identified the proposed solution should not struggle to get market approval both in the European Union and in the United States after filling in all the papers. However, as stated in section 4.3 the proposed solution by itself might not be ready to hit the market and integrating it into a larger prediction scheme is suggested.

4.6 Recommendations

Subsequent studies should start by making a prospective validation of the model such that well grounded assertions regarding model performance can be made. Results must be given using ROC AUC as well as 95% confidence intervals.

Biases should also be studied in case model performance varies across population groups and there is any risk associated to them, in which case it should be explained and, if possible, mitigated.

A larger number of adverse events could be analysed following the methodology used for this project. Prediction models for hypertension, bradycardia and tachycardia, burst suppression and excessive hypnosis would be valuable.

Concerning the data used to train the model, variables came exclusively from TCI and monitoring systems. Integrating more data regarding patient pre existing conditions and patient daily habits might improve model performance. Such information can be found at the patient file that is provided to the anesthesiologist before surgery, so it must be first coded into a standardized framework like the International Classification of Diseases (ICD) before it can be included in training data. Data integration is one of the main challenges for DL in medical field [52] but in this case it would not take that much effort since preliminary work on coding patient files to ICD framework has already been done by the supervisor and his team.

Another matter that should be considered is performing feature engineering to extract new variables that ultimately improve model performance. There are many techniques for feature engineering such as using windows of a certain length and calculating some metrics within these windows like mean, minimum and maximum, number of peaks, and lots more. These routines can be automatized and there are Python packages to do it [82].

Since the data acquisition is an ongoing process, whenever there is a larger amount of data available more complex DL models should be evaluated. For instance, overlaying convolutional neural networks (CNN) with LSTM layers has shown great results in time series applications with great potential in medical field [83]. CNN apply a convolution filter which intuitively adds each element of the time series to its preceding and following entries, weighted by a kernel whose weights are updated throughout the learning process. CNN and LSTM complement each other since CNN works as a feature extractor within short temporal regions and LSTM can pick up on that to make its predictions. Moreover, LSTM model must be trained on time series, but the dataset used for this project contains static variables (age, weight and height) and it would make sense to input these variables to the model on a separate layer so that they are not treated as time series. After more data is gathered, evaluating a CNN LSTM model that treats static data and time series separately (see figure 15) is highly recommended since this approach has already been successful in other medical field domains [84]. This model can be built using `Keras` functional API.

After prospective validation, the best performing model should be chosen and de-

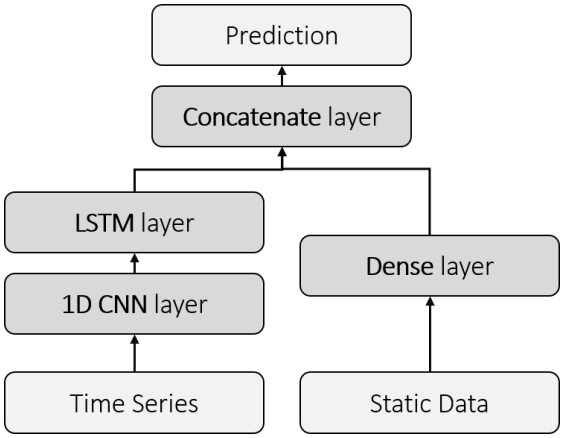

Figure 15. CNN LSTM model
Recommended model to evaluate when more data is gathered, overlaying convolutional with long short term memory layers and separating time series from static data.

ployed so it can be used in clinical set up. An exhaustive model explanation using SHAP methodology should be conducted on the chosen model. Moreover, a program will be needed to retrieve the variables in real time from the monitoring systems and make the predictions.

Conclusions

In clinical set up there is not any objective measure to determine if the patient will suffer from adverse events after LMA insertion, since currently monitors and indices lack predictive power. ML and DL models might help overcome this by making use of the data offered by the monitors.

A high quality dataset containing many variables relevant to anesthesia throughout many surgical procedures can be used to train ML and DL models to predict light hypnosis and hypotension after LMA insertion with solid performance. Movement prediction remains somewhat elusive but an acceptable performance is achieved, hinting that there might be room for improvement.

The LSTM model is the best performing one for light hypnosis and hypotension prediction, pointing that the time series nature of anesthesia data is key for accurate predictions. Knowing this, more complex models suitable for time series such as CNN LSTM should be evaluated.

This solution by itself might not be appealing enough for the market since it only solves a specific problem. Integrating these models within a larger scheme of properly validated prediction models for anesthesia is more likely to be embraced by anesthesiologists, as long as decisions can be properly explained. Ultimately, this solution could be integrated within a closed loop TCI system such that predictions are automatically analyzed and drug administration is adjusted according to them.

More advanced stages of the proposed solution would suppose an improvement in personalized management of drug administration. If the solution is implemented, it would be a valuable tool for the anesthesiologist since adverse events after LMA insertion could be predicted in advance and preventive actions could be taken to avoid these adverse events from happening. Moreover, this solution does not add any extra steps or inconveniences to the clinical practice, so it would not suppose any additional effort to the anesthesiologist.

References

[1] Emery N Brown, Ralph Lydic and Nicholas D Schiff. 'General anesthesia, sleep, and coma'. In: *New England Journal of Medicine* 363.27 (2010), pp. 2638–2650.

[2] Igor Kissin. 'General anesthetic action: an obsolete notion?' In: *ANESTHESIA AND ANALGESIA-CLEVELAND-* 76 (1993), pp. 215–215.

[3] BW Urban and M Bleckwenn. 'Concepts and correlations relevant to general anaesthesia'. In: *British journal of anaesthesia* 89.1 (2002), pp. 3–16.

[4] Avery Tung and Wallace B Mendelson. 'Anesthesia and sleep'. In: *Sleep medicine reviews* 8.3 (2004), pp. 213–225.

[5] Jason A Campagna, Keith W Miller and Stuart A Forman. 'Mechanisms of actions of inhaled anesthetics'. In: *New England Journal of Medicine* 348.21 (2003), pp. 2110–2124.

[6] Uwe Rudolph and Bernd Antkowiak. 'Molecular and neuronal substrates for general anaesthetics'. In: *Nature Reviews Neuroscience* 5.9 (2004), pp. 709–720.

[7] Barry J Shelp, Alan W Bown and Michael D McLean. 'Metabolism and functions of gamma-aminobutyric acid'. In: *Trends in plant science* 4.11 (1999), pp. 446–452.

[8] Hugh C Hemmings Jr et al. 'Emerging molecular mechanisms of general anesthetic action'. In: *Trends in pharmacological sciences* 26.10 (2005), pp. 503–510.

[9] Gordon M Shepherd. *The synaptic organization of the brain*. Oxford university press, 2004.

[10] Adrienne E Dubin and Ardem Patapoutian. 'Nociceptors: the sensors of the pain pathway'. In: *The Journal of clinical investigation* 120.11 (2010), pp. 3760–3772.

[11] Howard Fields. 'State-dependent opioid control of pain'. In: *Nature Reviews Neuroscience* 5.7 (2004), pp. 565–575.

[12] Jie Zhang et al. 'Role for G protein-coupled receptor kinase in agonist-specific regulation of μ-opioid receptor responsiveness'. In: *Proceedings of the National Academy of Sciences* 95.12 (1998), pp. 7157–7162.

[13] Thandla Raghavendra. 'Neuromuscular blocking drugs: discovery and development'. In: *Journal of the Royal Society of Medicine* 95.7 (2002), pp. 363–367.

[14] Julia M Adam et al. 'Cyclodextrin-Derived Host Molecules as Reversal Agents for the Neuromuscular Blocker Rocuronium Bromide: Synthesis and Structure- Activity Relationships'. In: *Journal of Medicinal Chemistry* 45.9 (2002), pp. 1806–1816.

[15] Bhavna Gupta and Lalit Gupta. 'Total Intravenous Anesthesia (Tiva)- A Brief Review. Pharma Sci Analytical Res J. 2018; 1(1.):180002.' In: (Oct. 2018).

[16] *Anesthesia key, introduction of anesthesia*. https://aneskey.com/induction-of-anesthesia/. Accessed: 19-05-2020.

[17] Kenichi Masui et al. 'The performance of compartmental and physiologically based recirculatory pharmacokinetic models for propofol: a comparison using bolus, continuous, and target-controlled infusion data'. In: *Anesthesia & Analgesia* 111.2 (2010), pp. 368–379.

[18] Sylvie Passot et al. 'Target-controlled versus manually-controlled infusion of propofol for direct laryngoscopy and bronchoscopy'. In: *Anesthesia & Analgesia* 94.5 (2002), pp. 1212–1216.

[19] Martijn J Mertens et al. 'Propofol Reduces Perioperative Remifentanil Requirements in a Synergistic MannerResponse Surface Modeling of Perioperative Remifentanil–Propofol Interactions'. In: *Anesthesiology: The Journal of the American Society of Anesthesiologists* 99.2 (2003), pp. 347–359.

[20] Harriet M Bryson, Bret R Fulton and Diana Faulds. 'Propofol'. In: *Drugs* 50.3 (1995), pp. 513–559.

[21] Kate McKeage and Caroline M Perry. 'Propofol'. In: *CNS drugs* 17.4 (2003), pp. 235–272.

[22] Pascale Picard and Martin R Tramèr. 'Prevention of pain on injection with propofol: a quantitative systematic review'. In: *Anesthesia & Analgesia* 90.4 (2000), pp. 963–969.

[23] Talmage D Egan. 'Remifentanil pharmacokinetics and pharmacodynamics'. In: *Clinical pharmacokinetics* 29.2 (1995), pp. 80–94.

[24] Lesley J Scott and Caroline M Perry. 'Remifentanil'. In: *Drugs* 65.13 (2005), pp. 1793–1823.

[25] Richard Beers and Enrico Camporesi. 'Remifentanil update'. In: *CNS drugs* 18.15 (2004), pp. 1085–1104.

[26] Diem TT Tran et al. 'Rocuronium versus succinylcholine for rapid sequence induction intubation'. In: *Cochrane database of systematic reviews* 10 (2015).

[27] Thomas Mencke et al. 'Intubating conditions and side effects of propofol, remifentanil and sevoflurane compared with propofol, remifentanil and rocuronium: a randomised, prospective, clinical trial'. In: *BMC anesthesiology* 14.1 (2014), p. 39.

[28] H Yu Seung and O Ross Beirne. 'Laryngeal mask airways have a lower risk of airway complications compared with endotracheal intubation: a systematic review'. In: *Journal of oral and maxillofacial surgery* 68.10 (2010), pp. 2359–2376.

[29] M Naguib and AH Samarkandi. 'The use of low-dose rocuronium to facilitate laryngeal mask airway insertion.' In: *Middle East journal of anaesthesiology* 16.1 (2001), pp. 41–54.

[30] Leslie V Simon, Muhammad F Hashmi and Klaus D Torp. 'Laryngeal Mask Airway'. In: (2019).

[31] Mathieu Jeanne et al. 'Heart rate variability during total intravenous anesthesia: effects of nociception and analgesia'. In: *Autonomic Neuroscience* 147.1-2 (2009), pp. 91–96.

[32] Elena Chung et al. ‘Non-invasive continuous blood pressure monitoring: a review of current applications’. In: *Frontiers of medicine* 7.1 (2013), pp. 91–101.

[33] Maan M Shaker. ‘EEG waves classifier using wavelet transform and Fourier transform’. In: *brain* 2.3 (2006).

[34] Priyanka A Abhang and Bharti W Gawali. ‘Correlation of EEG images and speech signals for emotion analysis’. In: *British Journal of Applied Science & Technology* 10.5 (2015), pp. 1–13.

[35] Florin Amzica. ‘Basic physiology of burst-suppression’. In: *Epilepsia* 50 (2009), pp. 38–39.

[36] Mika Särkelä et al. ‘Automatic analysis and monitoring of burst suppression in anesthesia’. In: *Journal of clinical monitoring and computing* 17.2 (2002), pp. 125–134.

[37] Carl Rosow and Paul J Manberg. ‘Bispectral index monitoring’. In: *Anesthesiology Clinics of North America* 19.4 (2001), pp. 947–966.

[38] Paul S Myles et al. ‘Bispectral index monitoring to prevent awareness during anaesthesia: the B-Aware randomised controlled trial’. In: *The lancet* 363.9423 (2004), pp. 1757–1763.

[39] Yodying Punjasawadwong, Aram Phongchiewboon and Nutchanart Bunchungmongkol. ‘Bispectral index for improving anaesthetic delivery and postoperative recovery’. In: *Cochrane database of systematic reviews* 6 (2014).

[40] Jay W Johansen. ‘Update on bispectral index monitoring’. In: *Best practice & research Clinical anaesthesiology* 20.1 (2006), pp. 81–99.

[41] Felix Schmid et al. ‘The wolf is crying in the operating room: patient monitor and anesthesia workstation alarming patterns during cardiac surgery’. In: *Anesthesia & Analgesia* 112.1 (2011), pp. 78–83.

[42] Pavel Hamet and Johanne Tremblay. ‘Artificial intelligence in medicine’. In: *Metabolism* 69 (2017), S36–S40.

[43] Jeremy Goecks et al. ‘How Machine Learning Will Transform Biomedicine’. In: *Cell* 181.1 (2020), pp. 92–101.

[44] Olaf RP Bininda-Emonds et al. ‘Garbage in, garbage out’. In: *Phylogenetic supertrees*. Springer, 2004, pp. 267–280.

[45] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[46] Xiaojin Zhu and Andrew B Goldberg. ‘Introduction to semi-supervised learning’. In: *Synthesis lectures on artificial intelligence and machine learning* 3.1 (2009), pp. 1–130.

[47] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. ‘Deep learning’. In: *nature* 521.7553 (2015), pp. 436–444.

[48] Robi Polikar. ‘Ensemble learning’. In: *Ensemble machine learning*. Springer, 2012, pp. 1–34.

[49] Mei-Hung Chiu et al. ‘The use of facial micro-expression state and Tree-Forest Model for predicting conceptual-conflict based conceptual change’. In: *Chapter Title & Authors Page* 184 (2016).

[50] Li-Yu Hu et al. ‘The distance function effect on k-nearest neighbor classification for medical datasets’. In: *SpringerPlus* 5.1 (2016), pp. 1–9.

[51] Gavin C Cawley and Nicola LC Talbot. ‘On over-fitting in model selection and subsequent selection bias in performance evaluation’. In: *Journal of Machine Learning Research* 11.Jul (2010), pp. 2079–2107.

[52] Travers Ching et al. ‘Opportunities and obstacles for deep learning in biology and medicine’. In: *Journal of The Royal Society Interface* 15.141 (2018), p. 20170387.

[53] Zachary C Lipton et al. ‘Learning to diagnose with LSTM recurrent neural networks’. In: *arXiv preprint arXiv:1511.03677* (2015).

[54] Abdul Ghaaliq Lalkhen and Anthony McCluskey. ‘Clinical tests: sensitivity and specificity’. In: *Continuing Education in Anaesthesia Critical Care & Pain* 8.6 (2008), pp. 221–223.

[55] Jayawant N Mandrekar. ‘Receiver operating characteristic curve in diagnostic test assessment’. In: *Journal of Thoracic Oncology* 5.9 (2010), pp. 1315–1316.

[56] Jeffrey L Apfelbaum et al. ‘Practice guidelines for management of the difficult airwayan updated report by the American Society of Anesthesiologists task force on management of the difficult airway’. In: *Anesthesiology: The Journal of the American Society of Anesthesiologists* 118.2 (2013), pp. 251–270.

[57] TM Cook. ‘Strategies for the prevention of airway complications–a narrative review’. In: *Anaesthesia* 73.1 (2018), pp. 93–111.

[58] SM Crawley and AJ Dalton. ‘Predicting the difficult airway’. In: *Bja Education* 15.5 (2015), pp. 253–257.

[59] J Adam Law et al. ‘The difficult airway with recommendations for management–part 2–the anticipated difficult airway’. In: *Canadian Journal of Anesthesia/Journal canadien d’anesthésie* 60.11 (2013), pp. 1119–1138.

[60] TM Cook and SR MacDougall-Davis. ‘Complications and failure of airway management’. In: *British journal of anaesthesia* 109.suppl_1 (2012), pp. i68–i85.

[61] Arunotai Siriussawakul et al. ‘Predictive performance of a multivariable difficult intubation model for obese patients’. In: *PloS one* 13.8 (2018).

[62] Ruggero M Corso et al. ‘Post analysis simulated correlation of the El-Ganzouri airway difficulty score with difficult airway’. In: *Brazilian Journal of Anesthesiology (English Edition)* 66.3 (2016), pp. 298–303.

[63] T Saito et al. ‘A proposal for a new scoring system to predict difficult ventilation through a supraglottic airway’. In: *BJA: British Journal of Anaesthesia* 117.suppl_1 (2016), pp. i83–i86.

[64] Winchana Srivilaithon et al. ‘Predicting difficult intubation in emergency department by intubation assessment score’. In: *Journal of clinical medicine research* 10.3 (2018), p. 247.

[65] Gabriel Louis Cuendet et al. ‘Facial image analysis for fully automatic prediction of difficult endotracheal intubation’. In: *IEEE Transactions on Biomedical Engineering* 63.2 (2015), pp. 328–339.

[66] Pedro L Gambús and Jan FA Hendrickx. *Personalized Anaesthesia: Targeting Physiological Systems for Optimal Effect*. Cambridge University Press, 2020.

[67] E Boselli et al. ‘Prediction of hemodynamic reactivity using dynamic variations of Analgesia/Nociception Index (ANI)’. In: *Journal of clinical monitoring and computing* 30.6 (2016), pp. 977–984.

[68] M Gruenewald et al. ‘Influence of nociceptive stimulation on analgesia nociception index (ANI) during propofol–remifentanil anaesthesia’. In: *British journal of anaesthesia* 110.6 (2013), pp. 1024–1030.

[69] Ruth Cowen et al. ‘Assessing pain objectively: the use of physiological markers’. In: *Anaesthesia* 70.7 (2015), pp. 828–847.

[70] Karel GM Moons et al. ‘Transparent Reporting of a multivariable prediction model for Individual Prognosis or Diagnosis (TRIPOD): explanation and elaboration’. In: *Annals of internal medicine* 162.1 (2015), W1–W73.

[71] Max Schubach et al. ‘Variant relevance prediction in extremely imbalanced training sets’. In: *F1000Research* 6 (2017), p. 1392.

[72] Diederik P Kingma and Jimmy Ba. ‘Adam: A method for stochastic optimization’. In: *arXiv preprint arXiv:1412.6980* (2014).

[73] Corinna Cortes and Mehryar Mohri. ‘Confidence intervals for the area under the ROC curve’. In: *Advances in neural information processing systems*. 2005, pp. 305–312.

[74] Martina Kottas, Oliver Kuss and Antonia Zapf. ‘A modified Wald interval for the area under the ROC curve (AUC) in diagnostic case-control studies’. In: *BMC medical research methodology* 14.1 (2014), p. 26.

[75] Scott M Lundberg and Su-In Lee. ‘A unified approach to interpreting model predictions’. In: *Advances in neural information processing systems*. 2017, pp. 4765–4774.

[76] Davide Cirillo et al. ‘Sex and gender differences and biases in artificial intelligence for biomedicine and healthcare’. In: *npj Digital Medicine* 3.1 (2020), pp. 1–11.

[77] Sam Corbett-Davies and Sharad Goel. ‘The measure and mismeasure of fairness: A critical review of fair machine learning’. In: *arXiv preprint arXiv:1808.00023* (2018).

[78] James Zou and Londa Schiebinger. *AI can be sexist and racist—it's time to make it fair*. 2018.

[79] Gabriel Erion et al. 'Anesthesiologist-level forecasting of hypoxemia with only SpO2 data using deep learning'. In: *arXiv preprint arXiv:1712.00563* (2017).

[80] Sara Gerke et al. 'The need for a system view to regulate artificial intelligence/machine learning-based software as medical device'. In: *NPJ Digital Medicine* 3.1 (2020), pp. 1–4.

[81] US Food, Drug Administration et al. 'Proposed Regulatory Framework for Modifications to Artificial Intelligence'. In: *Machine Learning (AI/ML)-based Software as a Medical Device (SaMD). Discussion Paper and Request for Feedback. Available at: https://www. fda. gov/-media/122535/download. Accessed June* 12 (2019).

[82] Maximilian Christ et al. 'Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)'. In: *Neurocomputing* 307 (2018), pp. 72–77.

[83] Shruti Kaushik et al. 'Ensemble of multi-headed machine learning architectures for time-series forecasting of healthcare expenditures'. In: *Applications of Machine Learning*. Springer, 2020, pp. 199–216.

[84] Chen Lin et al. 'Early diagnosis and prediction of sepsis shock by combining static and dynamic information using convolutional-LSTM'. In: *2018 IEEE International Conference on Healthcare Informatics (ICHI)*. IEEE. 2018, pp. 219–228.

Appendices

Annex A: PCA scatter plots

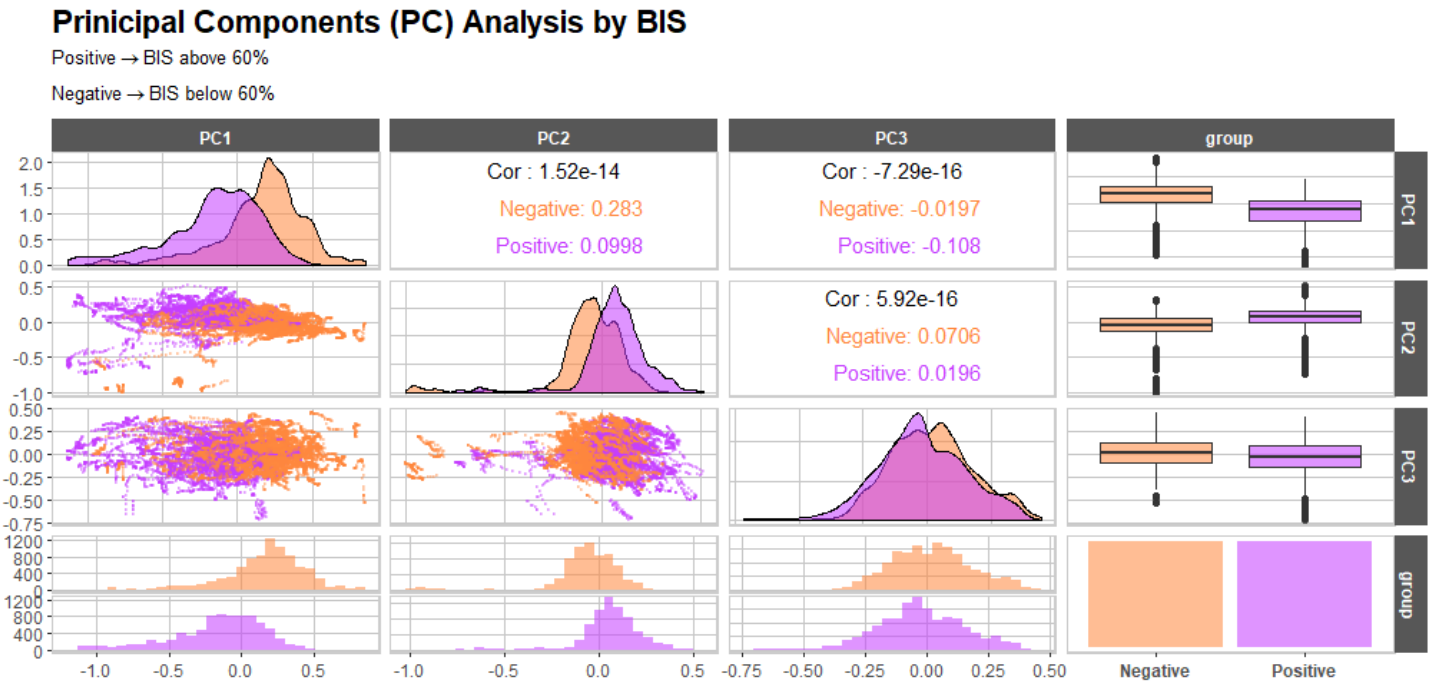

Figure 16. PCA scatter plot for BIS
Scatter plot of the three first PCs colored according to BIS label, as well as histograms and boxplots. There seems to be a different trend between positive and negative groups at first two PCs.

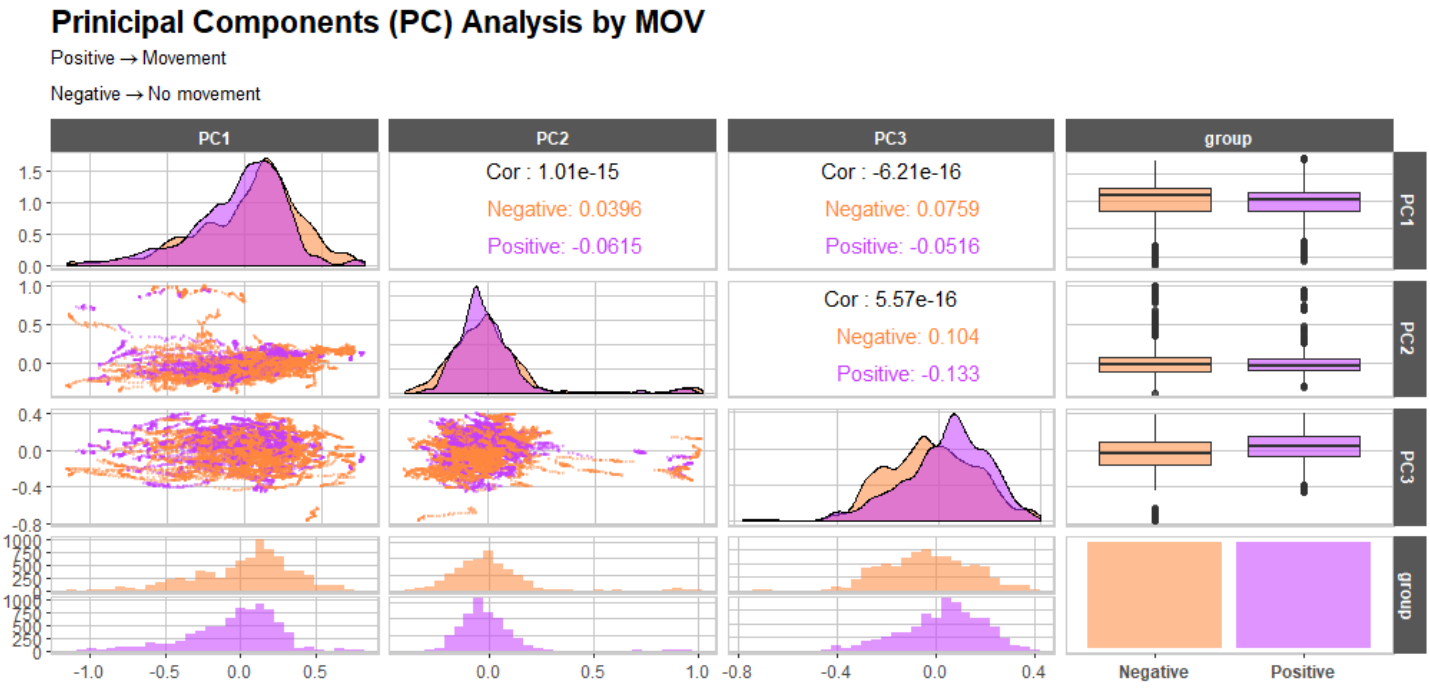

Figure 17. PCA scatter plot for MOV
Scatter plot of the three first PCs colored according to MOV label, as well as histograms and boxplots. A slight difference in PC3 between the two groups can be appreciated.

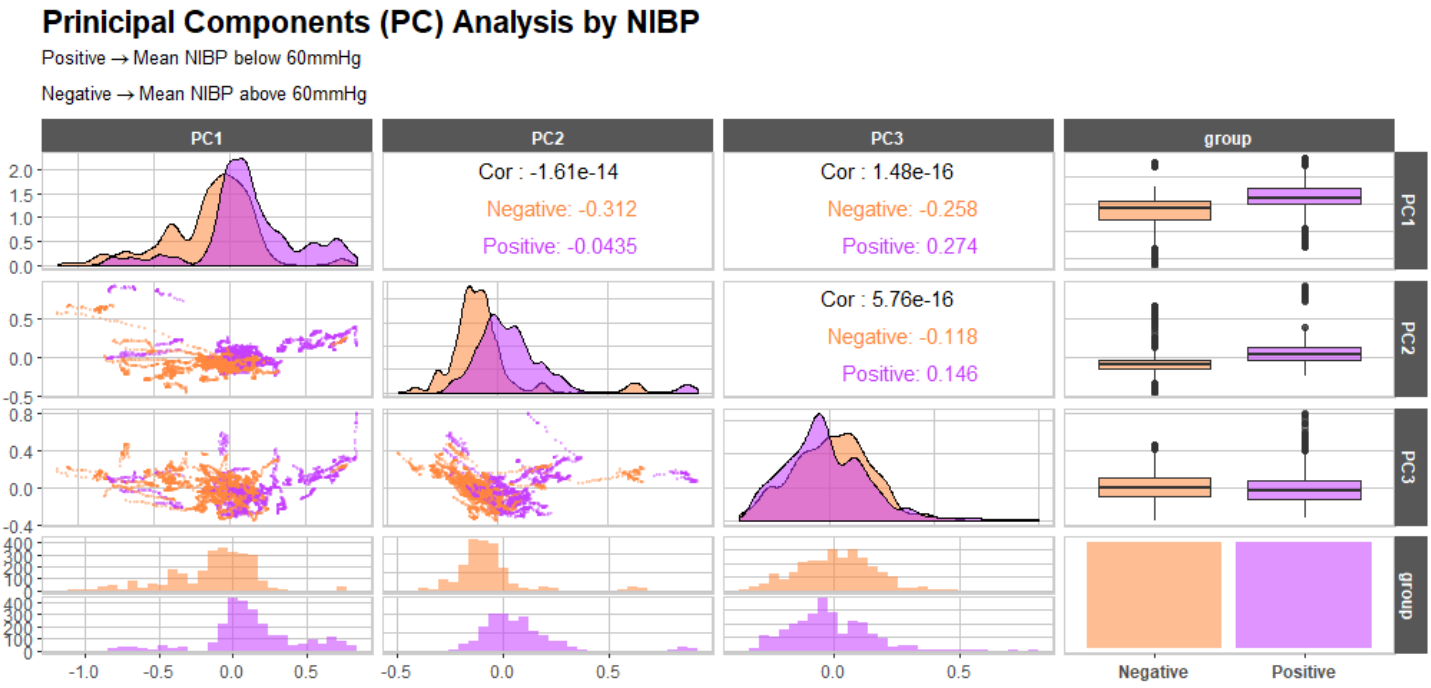

Figure 18. PCA scatter plot for NIBP
Scatter plot of the three first PCs colored according to NIBP label, as well as histograms and boxplots. There seems to be a different trend between positive and negative groups at first two PCs.

Annex B: Data Processing Script

Import modules

[ ]:
```
import os
import numpy as np
import pandas as pd
import re
import time
```

Event filtration

Events column is a bit chaotic as events are written manually and for many registers it is not standarized. In here I write a list with all events written in there, from which I will manually select those meaning `MOV` (movement), `LMA` (larynge mask), `INT` (intubation) and drugs, by searching if certain characters are in the event. Ideally we should use natural lenguage processing for this.

[ ]:
```
events=[]
no_events=[]

#list with all registers
my_path=r'C:\Users\joana\OneDrive\Escriptori\TFG\data\Datos_Clinic'
file_names=os.listdir(my_path)

#Iterate over each register
for i in range(len(file_names)):

    data=pd.read_excel('Datos_Clinic/'+file_names[i],'Hoja1')

    #Check if column Events exists
    if 'Events' in data.columns:

        #Iterate over column Events
        for event in data['Events']:

            #Append all events to a list except if it is already there
            if event not in events:
                events.append(event)

    #If column Events is missing append file name to a list
    else:
        no_events.append(file_names[i])


#Save lists
pd.DataFrame(no_events).to_csv('events//no_events.
 ↪csv',index=False,header=False,encoding='latin1')
```

```
#Avoid saving NaNs by selecting only strings
events=[event for event in events if type(event)==str]
pd.DataFrame(events).to_csv('events\events.
 ↪csv',index=False,header=False,encoding='latin1')
```

Data wrangling

For each register, `MOV`, `INT`, `LMA` and drugs columns are created and filled. Values of `128` in conox variables are set as NaNs, as they mean the machine is recalibrating. Missing values are then interpolated linearly. registers are cut before `CpPROPO` or `CpREMI` reaches 1, as it means anaesthesia is starting. Time column is reindexed so it starts at 1. For registers lacking frequencies columns they are created and set to NaN. Finally, columns of interest are selected and saved.

[ ]:
```
start_time=time.time()

#lists of events meaning LMA, MOV and INT
lma=pd.read_csv('events/lma.
 ↪csv',encoding='latin1',header=None,index_col=False)[0].tolist()
movs=pd.read_csv('events/movs.
 ↪csv',encoding='latin1',header=None,index_col=False)[0].tolist()
tub=pd.read_csv('events/tub.
 ↪csv',encoding='latin1',header=None,index_col=False)[0].tolist()

#list that contains registers without events in it (therefore useless)
no_events=pd.read_csv('events/no_events.
 ↪csv',encoding='latin1',header=None,index_col=False)[0].tolist()

#list with all registers with events
my_path=r'C:\Users\joana\OneDrive\Escriptori\TFG\Datos_Clinic'
file_names=[elem for elem in os.listdir(my_path) if elem not in no_events]

#Extract float or integer numbers in a string (dose)
def extract_dose(event):

    #create empty list
    nums=[]

    #find float number (comma and dot) and integers and append first number to␣
 ↪appear (so that it doesn't add 5 in 1.5)
    nums.append(float(re.findall("[+-]?\d+\.\d+",event)[0]) if re.findall("[+-]?
 ↪\d+\.\d+",event) else 0)
    nums.append(float(re.findall("[+-]?\d+\,\d+",event)[0].replace(',','.')) if␣
 ↪re.findall("[+-]?\d+\,\d+",event) else 0)
    nums.append(float(re.findall("[+-]?\d+",event)[0]) if re.findall("[+-]?
 ↪\d+",event) else 0)
```

```
    #return highest so that it neglects integers (1.5>1)
    return(max(nums))


#iterate over each raw register
for i in range(len(file_names)):

    #open data
    data=pd.read_excel('Datos_Clinic/'+file_names[i],'Hoja1')

    #initialize new separated events columns
    for col in␣
↪['MOV','LMA','INT','ROCURONIUM','EFEDRINE','ATROPINE','METHADONE','DROPERIDOL']:
↪
        data[col]=0


    #iterate over each register
    for j in range(len(data)):

        #select event and check if it is string (otherwise nan -> no event)
        event=data['Events'][j]
        if type(event)!=str:
            continue

        #fill the new columns (MOV -> 1 or 0, drugs -> dose)
        if event in movs:
            data.at[j,'MOV']=1
        if 'rocu' in event.lower() and 'min' not in event.lower():
            data.at[j,'ROCURONIUM']=int(re.findall(r"\d+",event.
↪replace('1O','10').replace('330','30'))[0]) if re.findall(r"\d+",event) else 0
        if 'efed' in event.lower() or 'ephe' in event.lower() and 'min' not in␣
↪event.lower():
            data.at[j,'EFEDRINE']=int(re.findall(r"\d+",event)[0]) if re.
↪findall(r"\d+",event) else 0
        if 'meta' in event.lower() or 'meth' in event.lower():
            if event in ['0.75mg droperidol 3 mg metadona','0.8mg droperidol 3mg␣
↪metadona','.75 droperidol, 3 metadona']:
                num=3
            elif event in ['abans de la metadona, 2.5 neoxinina i 0.5␣
↪atropina','10 metadona puesta a 10:28:14']:
                num=0
            else:
                num=int((re.findall("[+-]?\d+",event)[0]) if re.findall("[+-]?
↪\d+",event) else 0)
```

```
            data.at[j,'METHADONE']=num
        if 'atro' in event.lower() and 'min' not in event.lower():
            num=extract_dose(event.replace('0.06','0.6').replace('4,7','0,7'))
            if event=='abans de la metadona, 2.5 neoxinina i 0.5 atropina':
                num=0.5
            elif event=='4 mg prostignina + 1mg atropina':
                num=1
            data.at[j,'ATROPINE']=num
        if 'dro' in event.lower():
            num=extract_dose(event)
            num=0.75 if num>1.5 else num
            data.at[j,'DROPERIDOL']=num

    #iterate over each register
    for j in range(len(data)):

        #fill LMA or INT columns (cannot be both, cannot be more than once)
        if data['Events'][j] in lma:
            data.at[j,'LMA']=1
            break
        elif data['Events'][j] in tub:
            data.at[j,'INT']=1
            break

    #The data acquisition machine will recalibrate itself and the values of 128␣
↪will appear in qCON and qCONqNOX which does
    #is not feasible as they cannot go above 100% -> converted to NaNs
    for col in ['qCON', 'qCONEMG', 'qCONBS', 'qCONSQI', 'qCONqNOX',␣
↪'qCONZneg','qCONZref', 'qCONZpos']:
        data.loc[data[col]==128, col] = np.nan

    #find first value where drug concentration is higher that 1 -> surgery start
    k=0
    for k in range(len(data)):
        if data['CpPROPO'][k]>1 or data['CpREMI'][k]>1:
            break

    #interpolate missing values, even if not perfect it is the best possible␣
↪approach
    data_int=data.interpolate(method='linear')

    #cut all values before surgery start, reset index and reset Time(s)
    data_cut=data_int[k:]
    data_cut.reset_index(drop=True, inplace=True)
    data_cut.at[:,'Time(s)']=range(1,1+len(data_cut))

    #list containing names for all desired frequency columns (0Hz to 45Hz)
```

```
    freqs=[str(i)+'Hz' for i in range(46)]

    #there are a few registers without frequencies, for which empty frequency␣
 ↪columns are created
    if ~data_cut.columns.isin(freqs).any():
        for col in freqs:
            data_cut[col]=np.nan

    #select & order columns to be exported
    data_out=data_cut[['ID', 'Time(s)', 'ApproxRAppTime', 'SystTimeBIS', 'BIS',␣
 ↪'EMGBIS',
       'BSBIS', 'SQI09', 'CpREMI', 'CeREMI', 'InfVolREMI', 'InfRateREMI',
       'CpPROPO', 'CePROPO', 'InfVolPROPO', 'InfRatePROPO', 'Age',
       'Height', 'Weight', 'Gender', 'LBM', 'BSA', 'HR', 'NIBPsys',
       'NIBPdia', 'NIBPmean', 'RespiRate', 'SPO2',  'LMA','INT',
       'MOV',  'ROCURONIUM', 'ATROPINE',
       'EFEDRINE', 'METHADONE',  'Events',
       'qCON', 'qCONEMG', 'qCONBS', 'qCONSQI', 'qCONqNOX', 'qCONZneg',
       'qCONZref', 'qCONZpos']+freqs]


    data_out.
 ↪to_excel('Datos_Procesados\processed_data_'+str(int(data_out['ID'][0]))+'.
 ↪xlsx',sheet_name='Hoja1',index=False)


print('took ',round((time.time()-start_time)/60),' mins')
```

Missing columns & Mask separation

As we will focus on LMA registers, we want to create a list with these registers.

There are some registers which are missing columns. In here, I search for LMA registers that are missing columns in the region of interest (when the mask is introduced) in order to exclude them.

[ ]:
```
start_time=time.time()

#List of registers folowing inclusion criteria
my_path=r'C:\Users\joana\OneDrive\Escriptori\TFG\data\Datos_Procesados'
file_names=os.listdir(my_path)

#Initialize empty list for registers having empty columns
missing_cols=[]
lma_registers=[]

#Columns of interest
predictors=['BIS', 'EMGBIS',
```

```
'BSBIS', 'SQI09', 'CpREMI', 'CeREMI', 'InfVolREMI', 'InfRateREMI',
'CpPROPO', 'CePROPO', 'InfVolPROPO', 'InfRatePROPO', 'Age',
'Height', 'Weight', 'Gender', 'LBM', 'BSA', 'HR', 'NIBPsys',
'NIBPdia', 'NIBPmean', 'RespiRate', 'SPO2',
'qCON', 'qCONEMG', 'qCONBS', 'qCONSQI', 'qCONqNOX']+[str(i)+'Hz' for i in␣
 ↪range(31)]

#Iterate over each register
for reg in file_names:

    #Open file and select columns
    file_name='data\Datos_Procesados/'+reg
    df=pd.read_excel(file_name,'Hoja1')
    df_select=df[predictors]

    #initialize max and min columns
    if reg==file_names[0]:
        max_values=pd.DataFrame(df_select.min())
        min_values=pd.DataFrame(df_select.min())

    #update max and mon columns
    min_values=pd.concat([min_values,pd.DataFrame(df_select.min())],axis=1).
 ↪min(axis=1)
    max_values=pd.concat([max_values,pd.DataFrame(df_select.min())],axis=1).
 ↪max(axis=1)

    #check if it is a LMA register
    if df['LMA'].sum()==1:

        #append to lma list
        lma_registers.append(reg)

        #Fin region of interest (LMA) and cut dataframe
        k=0
        for k in range(len(df)):
            if df['LMA'][k]==1:
                break

        #select 45s before LMA and remove nans by rows
        df_start=df_select.iloc[(k-45):k,:]
        df_start=df_start[~np.isnan(df_start).any(axis=1)]

        #check if there are missing rows
        if len(df_start)!=45:
            missing_cols.append(reg)
```

```
#save list
pd.DataFrame(missing_cols).to_csv('events\\missing_columns.
 ↪csv',index=False,header=False)
pd.DataFrame(lma_registers).to_csv('events\\reg_lma.
 ↪csv',index=False,header=False)
maxmin=pd.concat([min_values,max_values],axis=1)
maxmin.columns=['min','max']
pd.DataFrame(maxmin).to_csv('events\\maxmin.csv')

print('took ',round((time.time()-start_time)/60),' mins')
```

Classify registers

Classify registers according to output

[ ]:
```
#function to shift the desired variable column by a desired number
def series_to_supervised(data,var,n_out=1):


    #turn data do pandas dataframe
    df=pd.DataFrame(data)
    cols=[df]


    #create shifted column
    shift_col=df[var].shift(-n_out)


    #add shifted column to dataframe
    cols.append(shift_col)
    df_shift=pd.concat(cols,axis=1)


    #rename column
    df_shift.columns=np.append(df.columns.values,var+' shift='+str(n_out))


    #return dataframe
    return df_shift
```

[ ]:
```
#open LMA registers that follow inclusion criteria
registers=pd.read_csv('events/reg_lma.csv',header=None,index_col=False)[0].
 ↪tolist()
missing_col=pd.read_csv('events/missing_columns.
 ↪csv',header=None,index_col=False)[0].tolist()
registers=[reg for reg in registers if reg not in missing_col]
```

```
#initialise lists
reg_mov=[]
reg_nibp=[]
reg_bis=[]

#Columns of interest
predictors=['BIS', 'EMGBIS',
'BSBIS', 'SQI09', 'CpREMI', 'CeREMI', 'InfVolREMI', 'InfRateREMI',
'CpPROPO', 'CePROPO', 'InfVolPROPO', 'InfRatePROPO', 'Age',
'Height', 'Weight', 'Gender', 'LBM', 'BSA', 'HR', 'NIBPsys',
'NIBPdia', 'NIBPmean', 'RespiRate', 'SPO2',
'qCON', 'qCONEMG', 'qCONBS', 'qCONSQI', 'qCONqNOX','MOV']+[str(i)+'Hz' for i in␣
 ↪range(31)]

#Iterate over each register
for reg in registers:

    #Open file and select columns
    file_name='data\Datos_Procesados/'+reg
    df=pd.read_excel(file_name,'Hoja1')
    df_select=df[predictors]

    #Reframe for supervised learning (shift=45s) + remove MOV col (not really a␣
 ↪predictor)
    df_reframed=series_to_supervised(df_select,'BIS',45)
    df_reframed=series_to_supervised(df_reframed,'MOV',45)
    df_reframed=series_to_supervised(df_reframed,'NIBPmean',45)

    #Fin region of interest (LMA) and cut dataframe
    k=0
    for k in range(len(df)):
        if df['LMA'][k]==1:
            break

    #classify registers
    df_start=df_reframed.iloc[(k-45):k,:]

    if (df_start.iloc[:,-3]>60).any():
        reg_bis.append(reg)

    if (df_start.iloc[:,-2]==1).any():
        reg_mov.append(reg)

    if (df_start.iloc[:,-1]<60).any():
        reg_nibp.append(reg)

#save list
```

```
pd.DataFrame(reg_bis).to_csv('events\\reg_bis.csv',index=False,header=False)
pd.DataFrame(reg_mov).to_csv('events\\reg_mov.csv',index=False,header=False)
pd.DataFrame(reg_nibp).to_csv('events\\reg_nibp.csv',index=False,header=False)
```

Balance data and define train and test

First, data is balanced so that positive and negative group have the same amount of registers. To do so, for each group, negative registers are randomly selected to match the number of positive registers.

Then, randomly split the registers that fulfill the inclusion criteria (LMA and no missing columns) in train (80%) and test (20%) set.

[ ]:
```
#Open list with register names
registers=pd.read_csv('events/reg_lma.csv',header=None,index_col=False)[0].
 ↪tolist()
missing_col=pd.read_csv('events/missing_columns.
 ↪csv',header=None,index_col=False)[0].tolist()
registers=[reg for reg in registers if reg not in missing_col]

reg_bis=pd.read_csv('events/reg_bis.csv',header=None,index_col=False)[0].tolist()
reg_mov=pd.read_csv('events/reg_mov.csv',header=None,index_col=False)[0].tolist()
reg_nibps=pd.read_csv('events/reg_nibp.csv',header=None,index_col=False)[0].
 ↪tolist()

#BIS

#radnomly choose registers with bis lower than 60
reg_bis_neg=np.random.choice([reg for reg in registers if reg not in␣
 ↪reg_bis],len(reg_bis),replace=False)

#create a mask
msk_bis = np.full(len(reg_bis),False)
inds=np.random.choice(np.arange(len(reg_bis)),size=round(len(reg_bis)*.
 ↪8),replace=False)
msk_bis[inds]=True

#create lists with indexes
reg_train_bis=[reg for reg,tr in zip(reg_bis,msk_bis) if tr] + [reg for reg,tr␣
 ↪in zip(reg_bis_neg,msk_bis) if tr]
reg_test_bis=[reg for reg,tr in zip(reg_bis,msk_bis) if not tr] + [reg for␣
 ↪reg,tr in zip(reg_bis_neg,msk_bis) if not tr]

#Save sets
pd.DataFrame(reg_train_bis).to_csv('events\\reg_train_bis.
 ↪csv',index=False,header=False)
pd.DataFrame(reg_test_bis).to_csv('events\\reg_test_bis.
 ↪csv',index=False,header=False)
```

```
#MOV

#radnomly choose registers with bis lower than 60
reg_mov_neg=np.random.choice([reg for reg in registers if reg not in␣
 ↪reg_mov],len(reg_mov),replace=False)

#create a mask
msk_mov = np.full(len(reg_mov),False)
inds=np.random.choice(np.arange(len(reg_mov)),size=round(len(reg_mov)*.
 ↪8),replace=False)
msk_mov[inds]=True

#create lists with indexes
reg_train_mov=[reg for reg,tr in zip(reg_mov,msk_mov) if tr] + [reg for reg,tr␣
 ↪in zip(reg_mov_neg,msk_mov) if tr]
reg_test_mov=[reg for reg,tr in zip(reg_mov,msk_mov) if not tr] + [reg for␣
 ↪reg,tr in zip(reg_mov_neg,msk_mov) if not tr]

#Save sets
pd.DataFrame(reg_train_mov).to_csv('events\\reg_train_mov.
 ↪csv',index=False,header=False)
pd.DataFrame(reg_test_mov).to_csv('events\\reg_test_mov.
 ↪csv',index=False,header=False)

#NIBP

#radnomly choose registers with bis lower than 60
reg_nibp_neg=np.random.choice([reg for reg in registers if reg not in␣
 ↪reg_nibp],len(reg_nibp),replace=False)

#create a mask
msk_nibp = np.full(len(reg_nibp),False)
inds=np.random.choice(np.arange(len(reg_nibp)),size=round(len(reg_nibp)*.
 ↪8),replace=False)
msk_nibp[inds]=True

#create lists with indexes
reg_train_nibp=[reg for reg,tr in zip(reg_nibp,msk_nibp) if tr] + [reg for␣
 ↪reg,tr in zip(reg_nibp_neg,msk_nibp) if tr]
reg_test_nibp=[reg for reg,tr in zip(reg_nibp,msk_nibp) if not tr] + [reg for␣
 ↪reg,tr in zip(reg_nibp_neg,msk_nibp) if not tr]

#Save sets
pd.DataFrame(reg_train_nibp).to_csv('events\\reg_train_nibp.
 ↪csv',index=False,header=False)
```

```
pd.DataFrame(reg_test_nibp).to_csv('events\\reg_test_nibp.
 ↪csv',index=False,header=False)
```

Prepare data to train the models

Data must be prepared to train a model such that it matches between registers. To do this data will be normalized between 0 and 1 using $scaled = \frac{value-min}{max-min}$ and the same moment of the operation will be used, that is, the 45s before LMA as input and the 45s after LMA as output. Output will be categorical, either 0 (good outcome) or 1 (bad outcome) for three different variables: * MOV: whether the patient moves * BIS: whether there is a BIS above 60% (high state of consciousness) * Mean arterial pressure: whether there is mean arterial pressure below 60mmHg (bad tissue perfusion)

If one of these variables for the patient is positive even if it is for just a second, we will consider positive for all 45 points (seconds).

It is important to bear in mind that values are scaled so thresholds also have to be scaled. For BIS it is as simple as dividing by 100 (as it ranges from 0 to 100), for NIBP we must use the formula above.

[ ]:
```
#Load test and train
reg_train_bis=pd.read_csv('events/reg_train_bis.
 ↪csv',header=None,index_col=False)[0].tolist()
reg_test_bis=pd.read_csv('events/reg_test_bis.
 ↪csv',header=None,index_col=False)[0].tolist()

reg_train_mov=pd.read_csv('events/reg_train_mov.
 ↪csv',header=None,index_col=False)[0].tolist()
reg_test_mov=pd.read_csv('events/reg_test_mov.
 ↪csv',header=None,index_col=False)[0].tolist()

reg_train_nibp=pd.read_csv('events/reg_train_nibp.
 ↪csv',header=None,index_col=False)[0].tolist()
reg_test_nibp=pd.read_csv('events/reg_test_nibp.
 ↪csv',header=None,index_col=False)[0].tolist()
```

[ ]:
```
#function to turn output to categorical
def to_categorical(y_data,model):


    #turn data to pandas dataframe
    y_data=pd.DataFrame(y_data)


    #for BIS positive means above 60%
    if model=='BIS':


        #iterate over each patient (45s/patient chosen)
```

```
        for i in range(int(len(y_data)/45)):

            #if any point is above 60%BIS patient is set to 1
            if (y_data.iloc[i*45:(i+1)*45]>0.614).any()[0]:
                y_data.iloc[i*45:(i+1)*45]=1
            else:
                y_data.iloc[i*45:(i+1)*45]=0


    #for MOV positive means at least 1
    elif model=='MOV':


        #iterate over each patient (45s/patient chosen)
        for i in range(int(len(y_data)/45)):

            #if any point is mov 1 patient is set to 1
            if (y_data.iloc[i*45:(i+1)*45]==1).any()[0]:
                y_data.iloc[i*45:(i+1)*45]=1
            else:
                y_data.iloc[i*45:(i+1)*45]=0


    #for NIBP positive means below 60
    elif model=='NIBPmean':


        #iterate over each patient (45s/patient chosen)
        for i in range(int(len(y_data)/45)):

            #if any point is above 60%BIS patient is set to 1
            if (y_data.iloc[i*45:(i+1)*45]<0.21348).any()[0]:
                y_data.iloc[i*45:(i+1)*45]=1
            else:
                y_data.iloc[i*45:(i+1)*45]=0


    #return data as array
    y_data=np.asarray(y_data)
    y_data=y_data.astype(int)

    return y_data
```

[ ]:
```
def data_processing(registers,variable):


    #list of predictors
```

```
   predictors=['BIS', 'EMGBIS',
   'BSBIS', 'SQI09', 'CpREMI', 'CeREMI', 'InfVolREMI', 'InfRateREMI',
   'CpPROPO', 'CePROPO', 'InfVolPROPO', 'InfRatePROPO', 'Age',
   'Height', 'Weight', 'LBM', 'BSA', 'HR', 'NIBPsys',
   'NIBPdia', 'NIBPmean', 'RespiRate', 'SPO2',
   'qCON', 'qCONEMG', 'qCONBS', 'qCONSQI', 'qCONqNOX','MOV']+[str(i)+'Hz' for i␣
↪in range(31)]


   #Open up maxmin from each column in order to manually scale from 0 to 1
   maxmin=pd.read_csv('events/max_min.csv',index_col=0)
   maxmin=maxmin.transpose(copy=True)[predictors]


   #For each patient
   for reg in registers:


       #Open up file and select desired columns
       file_name='data/Datos_Procesados/'+reg
       df=pd.read_excel(file_name,'Hoja1')
       df_select=df[predictors]


       #Normalize
       df_scaled=(df_select-maxmin.loc['min'])/(maxmin.loc['max']-maxmin.
↪loc['min'])
       df_scaled=df_scaled[predictors]


       #Reframe for supervised learning (shift=45s) + remove MOV col (not␣
↪really a predictor)
       df_reframed=series_to_supervised(df_scaled,variable,45)
       df_reframed.drop('MOV',axis=1, inplace=True)


       #Find laryngeal mask introduction point
       k=0
       for k in range(len(df)):
           if df['LMA'][k]==1:
               break


       #select 45s before intubation (so input is the 45s before and output is␣
↪the 45s after)
       df_start=df_reframed.iloc[(k-45):k,:]
```

```
        #split X and y (first initialise then append)
        if reg==registers[0]:
            inp=df_start.values[:,:-1]
            out=df_start.values[:,-1]
        else:
            inp=np.concatenate((inp,df_start.values[:,:-1]))
            out=np.concatenate((out,df_start.values[:,-1]))


    #turn y to categorical
    out=to_categorical(out,variable)


    #return X and y
    return inp,out
```

[ ]:
```
start_time=time.time()


#process X and y bis
train_X_bis,train_y_bis=data_processing(reg_train_bis,'BIS')
test_X_bis,test_y_bis=data_processing(reg_test_bis,'BIS')

#process X and y mov
train_X_mov,train_y_mov=data_processing(reg_train_mov,'MOV')
test_X_mov,test_y_mov=data_processing(reg_test_mov,'MOV')

#process X and y nibp
train_X_nibp,train_y_nibp=data_processing(reg_train_nibp,'NIBPmean')
test_X_nibp,test_y_nibp=data_processing(reg_test_nibp,'NIBPmean')


print("--- %s minutes ---" % round((time.time() - start_time)/60))
```

[ ]:
```
#save processed data
pd.DataFrame(train_X_bis).to_csv('data\Data_model\\train_X_BIS.
 ↪csv',index=False,header=False)
pd.DataFrame(train_X_mov).to_csv('data\Data_model\\train_X_MOV.
 ↪csv',index=False,header=False)
pd.DataFrame(train_X_nibp).to_csv('data\Data_model\\train_X_NIBP.
 ↪csv',index=False,header=False)

pd.DataFrame(train_y_bis).to_csv('data\Data_model\\train_y_BIS.
 ↪csv',index=False,header=False)
```

```
pd.DataFrame(train_y_mov).to_csv('data\Data_model\\train_y_MOV.
 ↪csv',index=False,header=False)
pd.DataFrame(train_y_nibp).to_csv('data\Data_model\\train_y_NIBP.
 ↪csv',index=False,header=False)

pd.DataFrame(test_X_bis).to_csv('data\Data_model\\test_X_BIS.
 ↪csv',index=False,header=False)
pd.DataFrame(test_X_mov).to_csv('data\Data_model\\test_X_MOV.
 ↪csv',index=False,header=False)
pd.DataFrame(test_X_nibp).to_csv('data\Data_model\\test_X_NIBP.
 ↪csv',index=False,header=False)

pd.DataFrame(test_y_bis).to_csv('data\Data_model\\test_y_BIS.
 ↪csv',index=False,header=False)
pd.DataFrame(test_y_mov).to_csv('data\Data_model\\test_y_MOV.
 ↪csv',index=False,header=False)
pd.DataFrame(test_y_nibp).to_csv('data\Data_model\\test_y_NIBP.
 ↪csv',index=False,header=False)
```

Anex C: Hyperparameter Tuning Script

[ ]:
```
#Basic libraries
import os
import time
import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


#Deep Learning libraries
import keras
from keras.layers import Dense, Activation,Dropout, BatchNormalization,LSTM
from keras.models import Sequential
from keras.optimizers import Adam
from keras.activations import sigmoid
from keras.losses import binary_crossentropy
import talos as ta


#Machine Learning libraries
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

Define a function to tune each model

For each model, a function will be defined to perform hyperparameter tunning. Parameter grid will be within the function even though it could also be an input if we wanted to modify it for each variable. Inputs will be the data and the name of the variable used for model output. For Machine Learning models `GridSearchCV` will be used, which check every parameter combination on the list using cross-validation. For Deep Learning models `Talos` will be used, which does the same thing but without cross-validation. The function saved the results including variable name so it can be distinguished later.

Random Forest

For Machine Learning models we just need to initialise the model and then use `GridSearchCV` from `sklearn` with the desired grid. `X` and `y` data are not separated by test and train as cross-validation will be performed.

[ ]:
```
def tune_rf(x,y,variable):

```

```
    # Create the parameter grid
    rf_grid = {
        'bootstrap': [True],
        'max_depth': [10,30],
        'max_features': ['auto'],
        'min_samples_leaf': [2, 4],
        'min_samples_split': [5,10],
        'n_estimators': list(range(100,2001,100))
    }


    # Create a base model
    rf = RandomForestClassifier(random_state = 42)


    # Instantiate the grid search model (cross-validation=4)
    rf_search = GridSearchCV(rf, rf_grid, cv = 4, n_jobs = -1, verbose = 2)


    # Fit the grid search to the data
    rf_search.fit(x,y)


    # Save results
    rf_results=pd.DataFrame(rf_search.cv_results_)
    rf_results.drop(['params'],axis=1,inplace=True)
    rf_results.to_csv('hyperparameter\\' + variable +'\\RF_results.
↪csv',index=False)
```

k Nearest Neighbors

```
[ ]: def tune_knn(x,y,variable):


    # Create the parameter grid
    knn_grid = {'n_neighbors': list(range(100,1001,20)),
           'weights': ['uniform', 'distance'],
           'leaf_size':[5,30],
           'algorithm':['auto', 'ball_tree']}


    # Create a base model
    knn = KNeighborsClassifier()
```

```
    # Instantiate the grid search model (cross-validation=4)
    knn_search = GridSearchCV(knn, knn_grid,  cv = 4, verbose=2, n_jobs = -1)


    # Fit the grid search to the data
    knn_search.fit(x,y)


    # Save results
    knn_results=pd.DataFrame(knn_search.cv_results_)
    knn_results.drop(['params'],axis=1,inplace=True)
    knn_results.to_csv('hyperparameter\\' + variable + '\\kNN_results.
↪csv',index=False)
```

Support Vector Classifier

[ ]:
```
def tune_svc(x,y,variable):


    # Create the parameter grid
    svc_grid = {'C': [0.1,1, 10, 100, 1000],
                'gamma': ['auto','scale'],
                'kernel': ['linear','rbf', 'poly', 'sigmoid']}


    # Create a base model
    svc= SVC()


    # Instantiate the grid search model (cross-validation=4)
    svc_search = GridSearchCV(svc,svc_grid,cv=4,verbose=2,n_jobs=-1)


    # Fit the grid search to the data
    svc_search.fit(x,y)


    # Save results
    svc_results=pd.DataFrame(svc_search.cv_results_)
    svc_results.drop(['params'],axis=1,inplace=True)
    svc_results.to_csv('hyperparameter\\' + variable + '\\SVC_results.
↪csv',index=False)
```

Neural Network

For Deep Learning models `Talos` works quite different as we have to manually define the model according to the parameters. After the experiment is run we save the model using the `Deploy` method, which saved the results as well as the best model and its weights in `h5` format, which can be later used.

[ ]:
```
#define model to be optimized
def NN_model(x_train, y_train, x_val, y_val, params):


    #initialise model
    model = Sequential()


    #add hidden layers (dense + dropout)
    for i in range(params['hidden_layers']):


        #first layer must have input dimension according to data
        if i==0:
            model.add(Dense(params['num_neurons'], input_dim=x_train.shape[1],
                            activation=sigmoid,
                            kernel_initializer='normal'))

            model.add(Dropout(params['dropout']))


        #rest of hidden layers
        else:
            model.add(Dense(params['num_neurons'],
                            activation=sigmoid,
                            kernel_initializer='normal'))

            model.add(Dropout(params['dropout']))


    #output layer
    model.add(Dense(1, activation=sigmoid,
                    kernel_initializer='normal'))


    #compile model
    model.compile(loss=binary_crossentropy,
                  #add a regulizer normalization function from Talos
                  optimizer=Adam(lr=ta.utils.lr_normalizer(params['lr'],Adam)),
                  metrics=['acc'])
```

```
    #fit model
    history = model.fit(x_train, y_train,
                        validation_data=[x_val, y_val],
                        batch_size=params['batch_size'],
                        epochs=params['epochs'],
                        verbose=0)

    #output history and model
    return history, model
```

[ ]:
```
def tune_nn(train_X,train_y,test_X,test_y,variable):


    #create parameter grid
    nn_grid = {'lr': [3.5,5],
         'num_neurons':[50,60,70,80,90,100],
         'hidden_layers':[2,3],
         'batch_size': [16,32,64],
         'epochs': [10,50],
         'dropout': [0.1,0.2]}


    #create scan object and search (no cross validation)
    nn_search = ta.Scan(x=train_X,
                y=train_y,
                model=NN_model,
                x_val=test_X,
                y_val=test_y,
                params=nn_grid,
                experiment_name='hyper_parameter_NN_' + variable)


    #save results
    ta.Deploy(scan_object=nn_search, model_name= variable + '_NN_results',␣
↪metric='val_acc')
```

LSTM

For LSTM data must be reshaped as for a 45s input sequence there will be only one output. The function `to_LSTM` does that by grouping X data so that each register has 45points (1 patient), and then reshaping it so that it is (number of patients x 45s x 77 variables) while simplifying output to 1 or 0.

[ ]:
```
#fuction to reshape data to match LSTM requirements
def to_LSTM(x_data, y_data):


    #initialise X and y
```

```
    out=[]
    inp=[]


    #iterate over patients (45s/patient)
    for i in range(int(x_data.shape[0]/45)):


        #reshape X (per patient)
        X=(pd.DataFrame(x_data).iloc[45*i:45*(i+1),]).values
        inp.append(X.reshape((X.shape[0], 1,X.shape[1])))


        #turn y to 1 point per patient
        if (pd.DataFrame(y_data).iloc[i*45:(i+1)*45]==1).any()[0]:
            out.append(1)
        else:
            out.append(0)


    #reshape X (overall)
    inp=np.array(inp)
    inp=inp.reshape((inp.shape[0],inp.shape[1],inp.shape[3]))


    #output X and y
    return inp,out
```

[ ]:
```
#define model to be optimized
def LSTM_model(x_train, y_train, x_val, y_val, params):


    #initialise model
    model = Sequential()


    #add lstm layer if there must be only one (no return_sequences)
    if params['LSTM_layers']==1:
        model.add(LSTM(params['num_neurons'], input_shape=(x_train.shape[1],␣
 ↪x_train.shape[2]),
                       ␣
 ↪activation=sigmoid,kernel_initializer='normal',dropout=params['dropout']))


    #if there are many layers (return_sequences needed)
    else:
```

```
      for i in range(params['LSTM_layers']):


          #if it is the first layer add input shape
          if i==0:
              model.add(LSTM(params['num_neurons'], input_shape=(x_train.
↪shape[1], x_train.shape[2]),
                             ␣
↪activation=sigmoid,kernel_initializer='normal',dropout=params['dropout'],
                                 return_sequences=True))


          #if it is the last layer return_sequences is not needed
          elif (i+1)==params['LSTM_layers']:
              model.
↪add(LSTM(params['num_neurons'],activation=sigmoid,kernel_initializer='normal',
                             dropout=params['dropout']))


          #any other layer
          else:
              model.
↪add(LSTM(params['num_neurons'],activation=sigmoid,kernel_initializer='normal',
                             dropout=params['dropout'],return_sequences=True))


  #add output layer
  model.add(Dense(1, activation=sigmoid,
                  kernel_initializer='normal'))


  #compile model
  model.compile(loss=binary_crossentropy,
                #add a regulizer normalization function from Talos
                optimizer=Adam(lr=ta.utils.lr_normalizer(params['lr'],Adam)),
                metrics=['acc'])


  #fit model and save history
  history = model.fit(x_train, y_train,
                      validation_data=[x_val, y_val],
                      batch_size=params['batch_size'],
                      epochs=params['epochs'],
                      verbose=0)


  #output history and model
```

```
    return history, model
```

[ ]:
```
def tune_lstm(train_X_LSTM,train_y_LSTM,test_X_LSTM,test_y_LSTM,variable):


    #create parameter grid
    lstm_grid = {'lr': [3.5,5],
         'num_neurons':[25,35,45,55,65,75],
         'LSTM_layers':[1,2],
         'batch_size': [16,32,64],
         'epochs': [200,300],
         'dropout': [0.1,0.2]}


    #create scan object and search (no cross validation)
    lstm_search = ta.Scan(x=train_X_LSTM,
                y=train_y_LSTM,
                model=LSTM_model,
                x_val=test_X_LSTM,
                y_val=test_y_LSTM,
                params=lstm_grid,
                experiment_name='hyper_parameter_LSTM_'+variable)


    #save results
    ta.Deploy(scan_object=lstm_search, model_name= variable + '_LSTM_results',␣
↪metric='val_acc')
```

Define a function to tune all models

This function works as a `Main()`, it uses the data and variable name as inputs and passes it through tunning functions for each model. It basically tunes every model for given data and stores the results.

[ ]:
```
def tune_all(train_X,train_y,test_X,test_y,variable):


    #merge train and test as for ML methods, cross validation will be used
    x=np.vstack((train_X,test_X))
    y=np.concatenate((train_y,test_y))


    #ML models
    tune_rf(x,y,variable)
    tune_knn(x,y,variable)
    tune_svc(x,y,variable)
```

```
    #rehsape X and y data for LSTM
    train_X_LSTM,train_y_LSTM=to_LSTM(train_X,train_y)
    test_X_LSTM,test_y_LSTM=to_LSTM(test_X,test_y)


    #DL models
    tune_nn(train_X,train_y,test_X,test_y,variable)
    tune_lstm(train_X_LSTM,train_y_LSTM,test_X_LSTM,test_y_LSTM,variable)
```

Tune all models for each variable

We can use `tune_all` to perform hyperparameter tunning for each variable. First we load the data, the use it as input to run the experiment.

[ ]:
```
#train data
train_X_bis=pd.read_csv('data\Data_model\\train_X_BIS.
 ↪csv',header=None,index_col=False).values
train_X_mov=pd.read_csv('data\Data_model\\train_X_MOV.
 ↪csv',header=None,index_col=False).values
train_X_nibp=pd.read_csv('data\Data_model\\train_X_NIBP.
 ↪csv',header=None,index_col=False).values

train_y_bis=pd.read_csv('data\Data_model\\train_y_BIS.
 ↪csv',header=None,index_col=False)[0].tolist()
train_y_mov=pd.read_csv('data\Data_model\\train_y_MOV.
 ↪csv',header=None,index_col=False)[0].tolist()
train_y_nibp=pd.read_csv('data\Data_model\\train_y_NIBP.
 ↪csv',header=None,index_col=False)[0].tolist()


#test data
test_X_bis=pd.read_csv('data\Data_model\\test_X_BIS.
 ↪csv',header=None,index_col=False).values
test_X_mov=pd.read_csv('data\Data_model\\test_X_MOV.
 ↪csv',header=None,index_col=False).values
test_X_nibp=pd.read_csv('data\Data_model\\test_X_NIBP.
 ↪csv',header=None,index_col=False).values

test_y_bis=pd.read_csv('data\Data_model\\test_y_BIS.
 ↪csv',header=None,index_col=False)[0].tolist()
test_y_mov=pd.read_csv('data\Data_model\\test_y_MOV.
 ↪csv',header=None,index_col=False)[0].tolist()
test_y_nibp=pd.read_csv('data\Data_model\\test_y_NIBP.
 ↪csv',header=None,index_col=False)[0].tolist()
```

[ ]:
```
start_time=time.time()

#BIS
tune_all(train_X_bis,train_y_bis,test_X_bis,test_y_bis,'BIS')


#MOV
tune_all(train_X_mov,train_y_mov,test_X_mov,test_y_mov,'MOV')


#NIBP
tune_all(train_X_nibp,train_y_nibp,test_X_nibp,test_y_nibp,'NIBP')

print("--- %s minutes ---" % round((time.time() - start_time)/60))
```

Annex D: Model Training Script

[ ]:
```
#Basic libraries
import os
import time
import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


#Deep Learning libraries
import keras
from keras.layers import Dense, Activation,Dropout, BatchNormalization,LSTM
from keras.models import Sequential
from keras.optimizers import Adam
from keras.activations import sigmoid
from keras.losses import binary_crossentropy
import talos as ta


#Machine Learning libraries
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

Create function to create ROC curves per patient

In order to correctly compare the models we want to see how many patients do they succesfully predict. To do that predictions are grouped each 45s (45 points), and increasing thresholds are used to see how do they perform. `any()` method is used as if the model predicts even for 1s that output is positive we take that it predicted positive patient. It is worthy to say that if `all()` method is used instead results are pretty much the same.

[ ]:
```
#inputs are data (predicted probability) and group (actual class)
def per_patient(data,group):


    #initialise lists for tpr and fpr
    tpr=[]
    fpr=[]
```

```
    #iterare over 101 thresholds (less would be enough but it is just to make␣
↪sure)
    for j in range(101):


        #initialise prediction lists (to be filled with True or False)
        out1=[]
        out2=[]


        #iterate over each patient
        for i in range(int(len(data)/45)):


            #for first iteration (threshold=0) greater or equal condition is␣
↪needed
            if j==0:


                #apend to out1 predicted class and to out2 actual class
                out1.append((data[i*45:45*(i+1)]>=0).any())
                out2.append((np.asarray(group[i*45:45*(i+1)])==1).any())


            #all other iterations greater condition is used
            else:
                out1.append((data[i*45:45*(i+1)]>j/100).any())
                out2.append((np.asarray(group[i*45:45*(i+1)])==1).any())


        #create confusion matrix and append tpr and fpr to lists
        cf=pd.DataFrame(confusion_matrix(pd.DataFrame(out2),pd.DataFrame(out1)))
        tpr.append(cf.iloc[0,0]/(cf.iloc[0,0]+cf.iloc[0,1]))
        fpr.append(cf.iloc[1,0]/(cf.iloc[1,0]+cf.iloc[1,1]))


    #return tpr and fpr values for increasing thresholds
    return tpr,fpr
```

Load data

[ ]:
```
#train data
train_X_bis=pd.read_csv('data\Data_model\\train_X_BIS.
↪csv',header=None,index_col=False).values
train_X_mov=pd.read_csv('data\Data_model\\train_X_MOV.
↪csv',header=None,index_col=False).values
```

```
train_X_nibp=pd.read_csv('data\Data_model\\train_X_NIBP.
 ↪csv',header=None,index_col=False).values

train_y_bis=pd.read_csv('data\Data_model\\train_y_BIS.
 ↪csv',header=None,index_col=False)[0].tolist()
train_y_mov=pd.read_csv('data\Data_model\\train_y_MOV.
 ↪csv',header=None,index_col=False)[0].tolist()
train_y_nibp=pd.read_csv('data\Data_model\\train_y_NIBP.
 ↪csv',header=None,index_col=False)[0].tolist()


#test data
test_X_bis=pd.read_csv('data\Data_model\\test_X_BIS.
 ↪csv',header=None,index_col=False).values
test_X_mov=pd.read_csv('data\Data_model\\test_X_MOV.
 ↪csv',header=None,index_col=False).values
test_X_nibp=pd.read_csv('data\Data_model\\test_X_NIBP.
 ↪csv',header=None,index_col=False).values

test_y_bis=pd.read_csv('data\Data_model\\test_y_BIS.
 ↪csv',header=None,index_col=False)[0].tolist()
test_y_mov=pd.read_csv('data\Data_model\\test_y_MOV.
 ↪csv',header=None,index_col=False)[0].tolist()
test_y_nibp=pd.read_csv('data\Data_model\\test_y_NIBP.
 ↪csv',header=None,index_col=False)[0].tolist()
```

Reshape LSTM data

[ ]:
```
#fuction to reshape data to match LSTM requirements
def to_LSTM(x_data, y_data):


    #initialise X and y
    out=[]
    inp=[]


    #iterate over patients (45s/patient)
    for i in range(int(x_data.shape[0]/45)):


        #reshape X (per patient)
        X=(pd.DataFrame(x_data).iloc[45*i:45*(i+1),]).values
        inp.append(X.reshape((X.shape[0], 1,X.shape[1])))
```

```
        #turn y to 1 point per patient
        if (pd.DataFrame(y_data).iloc[i*45:(i+1)*45]==1).any()[0]:
            out.append(1)
        else:
            out.append(0)


    #reshape X (overall)
    inp=np.array(inp)
    inp=inp.reshape((inp.shape[0],inp.shape[1],inp.shape[3]))


    #output X and y
    return inp, out
```

[ ]:
```
#rehsape X and y data
train_X_LSTM_bis,train_y_LSTM_bis = to_LSTM(train_X_bis,train_y_bis)
test_X_LSTM_bis,test_y_LSTM_bis = to_LSTM(test_X_bis,test_y_bis)

train_X_LSTM_mov,train_y_LSTM_mov = to_LSTM(train_X_mov,train_y_mov)
test_X_LSTM_mov,test_y_LSTM_mov = to_LSTM(test_X_mov,test_y_mov)

train_X_LSTM_nibp,train_y_LSTM_nibp = to_LSTM(train_X_nibp,train_y_nibp)
test_X_LSTM_nibp,test_y_LSTM_nibp = to_LSTM(test_X_nibp,test_y_nibp)
```

Prediction

Models are created according to best parameters and trained. Then predictions for test set are made and its ROC curves calculated as well as area under curve (AUC).

BIS

[ ]:
```
#RF
model_rf_bis =␣
 ↪RandomForestClassifier(n_estimators=200,min_samples_split=5,min_samples_leaf=4,
                                    max_depth=10,n_jobs=-1,random_state=42)
model_rf_bis.fit(train_X_bis, train_y_bis)
pred_rf_bis = model_rf_bis.predict_proba(test_X_bis)
tpr_rf_bis,fpr_rf_bis=per_patient(pred_rf_bis[:,1],test_y_bis)
auc_rf_bis = auc(fpr_rf_bis, tpr_rf_bis)


#kNN
model_knn_bis =␣
 ↪KNeighborsClassifier(n_neighbors=740,weights='distance',algorithm='auto',
                                   leaf_size=5,n_jobs=-1)
```

```
model_knn_bis.fit(train_X_bis, train_y_bis)
pred_knn_bis = model_knn_bis.predict_proba(test_X_bis)
tpr_knn_bis,fpr_knn_bis=per_patient(pred_knn_bis[:,1],test_y_bis)
auc_knn_bis = auc(fpr_knn_bis, tpr_knn_bis)


#SVC
model_svc_bis =␣
 ↪SVC(C=10,kernel='rbf',gamma='auto',probability=True,random_state=42)
model_svc_bis.fit(train_X_bis, train_y_bis)
pred_svc_bis = model_svc_bis.predict_proba(test_X_bis)
tpr_svc_bis,fpr_svc_bis=per_patient(pred_svc_bis[:,1],test_y_bis)
auc_svc_bis = auc(fpr_svc_bis, tpr_svc_bis)
```

[ ]:
```
#NN
num_neurons_bis=80
model_nn_bis=Sequential()
model_nn_bis.add(Dense(num_neurons_bis,input_dim=59,activation='sigmoid'))␣
 ↪#INPUT DIM
model_nn_bis.add(Dropout(0.2)) #avoid overfitting
model_nn_bis.add(Dense(num_neurons_bis,activation='sigmoid'))
model_nn_bis.add(Dropout(0.2))
model_nn_bis.add(Dense(num_neurons_bis,activation='sigmoid'))
model_nn_bis.add(Dropout(0.2))
model_nn_bis.add(Dense(1,activation='sigmoid',kernel_initializer='normal'))
model_nn_bis.
 ↪compile(optimizer=Adam(),loss='binary_crossentropy',metrics=['acc']) #lr=ta.
 ↪utils.lr_normalizer(5,Adam)
model_nn_bis.fit(train_X_bis, train_y_bis, epochs=10, batch_size=16,
                        validation_data=(test_X_bis, test_y_bis), verbose=0,␣
 ↪shuffle=True)
pred_nn_bis=model_nn_bis.predict(test_X_bis)
tpr_nn_bis,fpr_nn_bis=per_patient(pred_nn_bis,test_y_bis)
auc_nn_bis = auc(fpr_nn_bis, tpr_nn_bis)
```

[ ]:
```
#LSTM
num_neurons_lstm_bis=35
model_lstm_bis=Sequential()
model_lstm_bis.add(LSTM(num_neurons_lstm_bis, input_shape=(train_X_LSTM_bis.
 ↪shape[1], train_X_LSTM_bis.shape[2]),
                            ␣
 ↪activation=sigmoid,kernel_initializer='normal',dropout=0.1))
model_lstm_bis.add(Dense(1, activation=sigmoid,kernel_initializer='normal'))
model_lstm_bis.
 ↪compile(loss=binary_crossentropy,optimizer=Adam(),metrics=['acc']) #lr=ta.
 ↪utils.lr_normalizer(3.5,Adam)
```

```
model_lstm_bis.fit(train_X_LSTM_bis, train_y_LSTM_bis, epochs=300, batch_size=64,
                        validation_data=(test_X_LSTM_bis, test_y_LSTM_bis),␣
 ↪verbose=0, shuffle=True)
pred_bis=model_lstm_bis.predict(test_X_LSTM_bis)
pred_lstm_bis=[l[0] for l in pred_bis]
fpr_lstm_bis, tpr_lstm_bis, threshold = roc_curve(test_y_LSTM_bis, pred_lstm_bis)
auc_lstm_bis = auc(fpr_lstm_bis, tpr_lstm_bis)
```

[ ]:
```
#store BIS results
pd.DataFrame.from_dict({'tpr':
 ↪tpr_rf_bis+tpr_knn_bis+tpr_svc_bis+tpr_nn_bis+list(tpr_lstm_bis),
             'fpr':
 ↪fpr_rf_bis+fpr_knn_bis+fpr_svc_bis+fpr_nn_bis+list(fpr_lstm_bis),
             'model':
 ↪['RF']*len(tpr_rf_bis)+['kNN']*len(tpr_knn_bis)+['SVC']*len(tpr_svc_bis)
                        +['NN']*len(tpr_nn_bis)+['LSTM']*len(tpr_lstm_bis)}).
 ↪to_csv(
'results\BIS_roc.csv',index=False)
pd.DataFrame.from_dict({'auc':
 ↪[auc_rf_bis,auc_knn_bis,auc_svc_bis,auc_nn_bis,auc_lstm_bis],
                        'model':['RF','kNN','SVC','NN','LSTM']}).to_csv(
'results\BIS_auc.csv',index=False)
```

MOV

[ ]:
```
#RF
model_rf_mov =␣
 ↪RandomForestClassifier(n_estimators=400,max_depth=30,min_samples_split=10,
                                                 ␣
 ↪min_samples_leaf=2,n_jobs=-1,random_state=42)
model_rf_mov.fit(train_X_mov, train_y_mov)
pred_rf_mov = model_rf_mov.predict_proba(test_X_mov)
tpr_rf_mov,fpr_rf_mov=per_patient(pred_rf_mov[:,1],test_y_mov)
auc_rf_mov = auc(fpr_rf_mov, tpr_rf_mov)


#kNN
model_knn_mov =␣
 ↪KNeighborsClassifier(n_neighbors=100,weights='uniform',algorithm='auto',
                                      leaf_size=5,n_jobs=-1)
model_knn_mov.fit(train_X_mov, train_y_mov)
pred_knn_mov = model_knn_mov.predict_proba(test_X_mov)
tpr_knn_mov,fpr_knn_mov=per_patient(pred_knn_mov[:,1],test_y_mov)
auc_knn_mov = auc(fpr_knn_mov, tpr_knn_mov)
```

```
#SVC
model_svc_mov = SVC(C=0.
 ↪1,kernel='rbf',gamma='scale',probability=True,random_state=42)
model_svc_mov.fit(train_X_mov, train_y_mov)
pred_svc_mov = model_svc_mov.predict_proba(test_X_mov)
tpr_svc_mov,fpr_svc_mov=per_patient(pred_svc_mov[:,1],test_y_mov)
auc_svc_mov = auc(fpr_svc_mov, tpr_svc_mov)
```

[ ]:
```
#NN
num_neurons_mov=100
model_nn_mov=Sequential()
model_nn_mov.add(Dense(num_neurons_mov,input_dim=59,activation='sigmoid'))
model_nn_mov.add(Dropout(0.1)) #avoid overfitting
model_nn_mov.add(Dense(num_neurons_mov,activation='sigmoid'))
model_nn_mov.add(Dropout(0.1))
model_nn_mov.add(Dense(1,activation='sigmoid',kernel_initializer='normal'))
model_nn_mov.
 ↪compile(optimizer=Adam(),loss='binary_crossentropy',metrics=['acc']) #lr=ta.
 ↪utils.lr_normalizer(3.5,Adam)
model_nn_mov.fit(train_X_mov, train_y_mov, epochs=50, batch_size=16,
                         validation_data=(test_X_mov, test_y_mov), verbose=0,␣
 ↪shuffle=True)
pred_nn_mov=model_nn_mov.predict(test_X_mov)
tpr_nn_mov,fpr_nn_mov=per_patient(pred_nn_mov,test_y_mov)
auc_nn_mov = auc(fpr_nn_mov, tpr_nn_mov)
```

[ ]:
```
#LSTM
num_neurons_lstm_mov=75
model_lstm_mov=Sequential()
model_lstm_mov.add(LSTM(num_neurons_lstm_mov, input_shape=(train_X_LSTM_mov.
 ↪shape[1], train_X_LSTM_mov.shape[2]),
                                ␣
 ↪activation=sigmoid,kernel_initializer='normal',dropout=0.2))
model_lstm_mov.add(Dense(1, activation=sigmoid,kernel_initializer='normal'))
model_lstm_mov.
 ↪compile(loss=binary_crossentropy,optimizer=Adam(),metrics=['acc']) #lr=ta.
 ↪utils.lr_normalizer(5,Adam)
model_lstm_mov.fit(train_X_LSTM_mov, train_y_LSTM_mov, epochs=300, batch_size=64,
                         validation_data=(test_X_LSTM_mov, test_y_LSTM_mov),␣
 ↪verbose=0, shuffle=True)
pred_mov=model_lstm_mov.predict(test_X_LSTM_mov)
pred_lstm_mov=[l[0] for l in pred_mov]
fpr_lstm_mov, tpr_lstm_mov, threshold = roc_curve(test_y_LSTM_mov, pred_lstm_mov)
auc_lstm_mov = auc(fpr_lstm_mov, tpr_lstm_mov)
```

```
[ ]: #store MOV results
pd.DataFrame.from_dict({'tpr':
 ↪tpr_rf_mov+tpr_knn_mov+tpr_svc_mov+tpr_nn_mov+list(tpr_lstm_mov),
             'fpr':
 ↪fpr_rf_mov+fpr_knn_mov+fpr_svc_mov+fpr_nn_mov+list(fpr_lstm_mov),
             'model':
 ↪['RF']*len(tpr_rf_mov)+['kNN']*len(tpr_knn_mov)+['SVC']*len(tpr_svc_mov)
                        +['NN']*len(tpr_nn_mov)+['LSTM']*len(tpr_lstm_mov)}).
 ↪to_csv(
'results\MOV_roc.csv',index=False)
pd.DataFrame.from_dict({'auc':
 ↪[auc_rf_mov,auc_knn_mov,auc_svc_mov,auc_nn_mov,auc_lstm_mov],
                        'model':['RF','kNN','SVC','NN','LSTM']}).to_csv(
'results\MOV_auc.csv',index=False)
```

NIBP

```
[ ]: #RF
model_rf_nibp =␣
 ↪RandomForestClassifier(n_estimators=100,min_samples_split=5,min_samples_leaf=2,
                                        max_depth=10,n_jobs=-1,random_state=42)
model_rf_nibp.fit(train_X_nibp, train_y_nibp)
pred_rf_nibp = model_rf_nibp.predict_proba(test_X_nibp)
tpr_rf_nibp,fpr_rf_nibp=per_patient(pred_rf_nibp[:,1],test_y_nibp)
auc_rf_nibp = auc(fpr_rf_nibp, tpr_rf_nibp)


#kNN
model_knn_nibp =␣
 ↪KNeighborsClassifier(n_neighbors=100,weights='distance',algorithm='auto',
                                       leaf_size=5,n_jobs=-1)
model_knn_nibp.fit(train_X_nibp, train_y_nibp)
pred_knn_nibp = model_knn_nibp.predict_proba(test_X_nibp)
tpr_knn_nibp,fpr_knn_nibp=per_patient(pred_knn_nibp[:,1],test_y_nibp)
auc_knn_nibp = auc(fpr_knn_nibp, tpr_knn_nibp)


#SVC
model_svc_nibp =␣
 ↪SVC(C=10,kernel='rbf',gamma='auto',probability=True,random_state=42)
model_svc_nibp.fit(train_X_nibp, train_y_nibp)
pred_svc_nibp = model_svc_nibp.predict_proba(test_X_nibp)
tpr_svc_nibp,fpr_svc_nibp=per_patient(pred_svc_nibp[:,1],test_y_nibp)
auc_svc_nibp = auc(fpr_svc_nibp, tpr_svc_nibp)
```

[ ]:
```
#NN
num_neurons_nibp=90
model_nn_nibp=Sequential()
model_nn_nibp.add(Dense(num_neurons_nibp,input_dim=59,activation='sigmoid'))
model_nn_nibp.add(Dropout(0.2)) #avoid overfitting
model_nn_nibp.add(Dense(num_neurons_nibp,activation='sigmoid'))
model_nn_nibp.add(Dropout(0.2))
model_nn_nibp.add(Dense(num_neurons_nibp,activation='sigmoid'))
model_nn_nibp.add(Dropout(0.2))
model_nn_nibp.add(Dense(1,activation='sigmoid',kernel_initializer='normal'))
model_nn_nibp.
 ↪compile(optimizer=Adam(),loss='binary_crossentropy',metrics=['acc']) #lr=ta.
 ↪utils.lr_normalizer(5,Adam)
model_nn_nibp.fit(train_X_nibp, train_y_nibp, epochs=10, batch_size=16,
                        validation_data=(test_X_nibp, test_y_nibp), verbose=0,␣
 ↪shuffle=True)
pred_nn_nibp=model_nn_nibp.predict(test_X_nibp)
tpr_nn_nibp,fpr_nn_nibp=per_patient(pred_nn_nibp,test_y_nibp)
auc_nn_nibp = auc(fpr_nn_nibp, tpr_nn_nibp)
```

[ ]:
```
#LSTM
num_neurons_lstm_nibp=75
model_lstm_nibp=Sequential()
model_lstm_nibp.add(LSTM(num_neurons_lstm_nibp, input_shape=(train_X_LSTM_nibp.
 ↪shape[1], train_X_LSTM_nibp.shape[2]),
                         ␣
 ↪activation=sigmoid,kernel_initializer='normal',dropout=0.1))
model_lstm_nibp.add(Dense(1, activation=sigmoid,kernel_initializer='normal'))
model_lstm_nibp.
 ↪compile(loss=binary_crossentropy,optimizer=Adam(),metrics=['acc']) #lr=ta.
 ↪utils.lr_normalizer(3.5,Adam)
model_lstm_nibp.fit(train_X_LSTM_nibp, train_y_LSTM_nibp, epochs=200,␣
 ↪batch_size=16,
                        validation_data=(test_X_LSTM_nibp, test_y_LSTM_nibp),␣
 ↪verbose=0, shuffle=True)
pred_nibp=model_lstm_nibp.predict(test_X_LSTM_nibp)
pred_lstm_nibp=[l[0] for l in pred_nibp]
fpr_lstm_nibp, tpr_lstm_nibp, threshold = roc_curve(test_y_LSTM_nibp,␣
 ↪pred_lstm_nibp)
auc_lstm_nibp = auc(fpr_lstm_nibp, tpr_lstm_nibp)
```

[ ]:
```
#store NIBP results
pd.DataFrame.from_dict({'tpr':
 ↪tpr_rf_nibp+tpr_knn_nibp+tpr_svc_nibp+tpr_nn_nibp+list(tpr_lstm_nibp),
             'fpr':
 ↪fpr_rf_nibp+fpr_knn_nibp+fpr_svc_nibp+fpr_nn_nibp+list(fpr_lstm_nibp),
```

```
            'model':
 ↪['RF']*len(tpr_rf_nibp)+['kNN']*len(tpr_knn_nibp)+['SVC']*len(tpr_svc_nibp)
                        +['NN']*len(tpr_nn_nibp)+['LSTM']*len(tpr_lstm_nibp)}).
 ↪to_csv(
'results\\NIBP_roc.csv',index=False)
pd.DataFrame.from_dict({'auc':
 ↪[auc_rf_nibp,auc_knn_nibp,auc_svc_nibp,auc_nn_nibp,auc_lstm_nibp],
                        'model':['RF','kNN','SVC','NN','LSTM']}).to_csv(
'results\\NIBP_auc.csv',index=False)
```

Annex E: Model Explanation Script

For simplicity, I'll just use feature importance of RF. More advanced stages of the project should use SHAP methodology to evaluate all models.

[ ]:
```
#Basic libraries
import os
import time
import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


#Machine Learning libraries
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

[ ]:
```
#train data
train_X_bis=pd.read_csv('data\Data_model\\train_X_BIS.
 ↪csv',header=None,index_col=False).values
train_X_mov=pd.read_csv('data\Data_model\\train_X_MOV.
 ↪csv',header=None,index_col=False).values
train_X_nibp=pd.read_csv('data\Data_model\\train_X_NIBP.
 ↪csv',header=None,index_col=False).values

train_y_bis=pd.read_csv('data\Data_model\\train_y_BIS.
 ↪csv',header=None,index_col=False)[0].tolist()
train_y_mov=pd.read_csv('data\Data_model\\train_y_MOV.
 ↪csv',header=None,index_col=False)[0].tolist()
train_y_nibp=pd.read_csv('data\Data_model\\train_y_NIBP.
 ↪csv',header=None,index_col=False)[0].tolist()


#test data
test_X_bis=pd.read_csv('data\Data_model\\test_X_BIS.
 ↪csv',header=None,index_col=False).values
test_X_mov=pd.read_csv('data\Data_model\\test_X_MOV.
 ↪csv',header=None,index_col=False).values
test_X_nibp=pd.read_csv('data\Data_model\\test_X_NIBP.
 ↪csv',header=None,index_col=False).values
```

```
test_y_bis=pd.read_csv('data\Data_model\\test_y_BIS.
 ↪csv',header=None,index_col=False)[0].tolist()
test_y_mov=pd.read_csv('data\Data_model\\test_y_MOV.
 ↪csv',header=None,index_col=False)[0].tolist()
test_y_nibp=pd.read_csv('data\Data_model\\test_y_NIBP.
 ↪csv',header=None,index_col=False)[0].tolist()
```

[ ]:
```
#bis
model_rf_bis =␣
 ↪RandomForestClassifier(n_estimators=200,min_samples_split=5,min_samples_leaf=4,
                                      max_depth=10,n_jobs=-1,random_state=42)
model_rf_bis.fit(train_X_bis, train_y_bis)

#mov
model_rf_mov =␣
 ↪RandomForestClassifier(n_estimators=400,max_depth=30,min_samples_split=10,
                                     ␣
 ↪min_samples_leaf=2,n_jobs=-1,random_state=42)
model_rf_mov.fit(train_X_mov, train_y_mov)

#nibp
model_rf_nibp =␣
 ↪RandomForestClassifier(n_estimators=100,min_samples_split=5,min_samples_leaf=2,
                                       max_depth=10,n_jobs=-1,random_state=42)
model_rf_nibp.fit(train_X_nibp, train_y_nibp)
```

[ ]:
```
predictors=['BIS', 'EMGBIS',
    'BSBIS', 'SQI09', 'CpREMI', 'CeREMI', 'InfVolREMI', 'InfRateREMI',
    'CpPROPO', 'CePROPO', 'InfVolPROPO', 'InfRatePROPO', 'Age',
    'Height', 'Weight', 'LBM', 'BSA', 'HR', 'NIBPsys',
    'NIBPdia', 'NIBPmean', 'RespiRate', 'SPO2',
    'qCON', 'qCONEMG', 'qCONBS', 'qCONSQI', 'qCONqNOX']+[str(i)+'Hz' for i in␣
 ↪range(31)]
```

[ ]:
```
importance = pd.DataFrame([predictors,
              model_rf_bis.feature_importances_,
              model_rf_mov.feature_importances_,
              model_rf_nibp.feature_importances_]).transpose()
importance.columns=['Predictor','BIS','MOV','NIBP']
```

[ ]:
```
importance.to_csv('results\\feature_importance.csv',index=False)
```