WARD
Severe Adverse Event Prediction

By (Ag)Nete and Emilie

Overview of the Problem

What is the problem, motivation and goal?

WARD: Project to monitor patients of high risk.

WARD model: Which is ~16 if statements from doctors → Gives warning when patient is very “sick” (ALERTS). If patient actually very “sick” → EVENT noted

Motivation:

Our ***problem*** and ***goal***:

- Given data of patients monitored → Find out how and when patients are very sick with ML that is “better” than WARD’s model → Find out when and what kind of event happens for a patient.

Where does the data come from?

Data of the patients is measured 3 places

- Ring on finger
- On the arm
- On the chest

→ Given in principle 8 features from these. (Heart rate, Respiration rate, Oxygen saturation, Oxymeter pulse, Systolic blood pressure, Diastolic blood pressure, Blood pressure pulse, Patient orientation)

PLUS → Alert from the WARD model is also given!

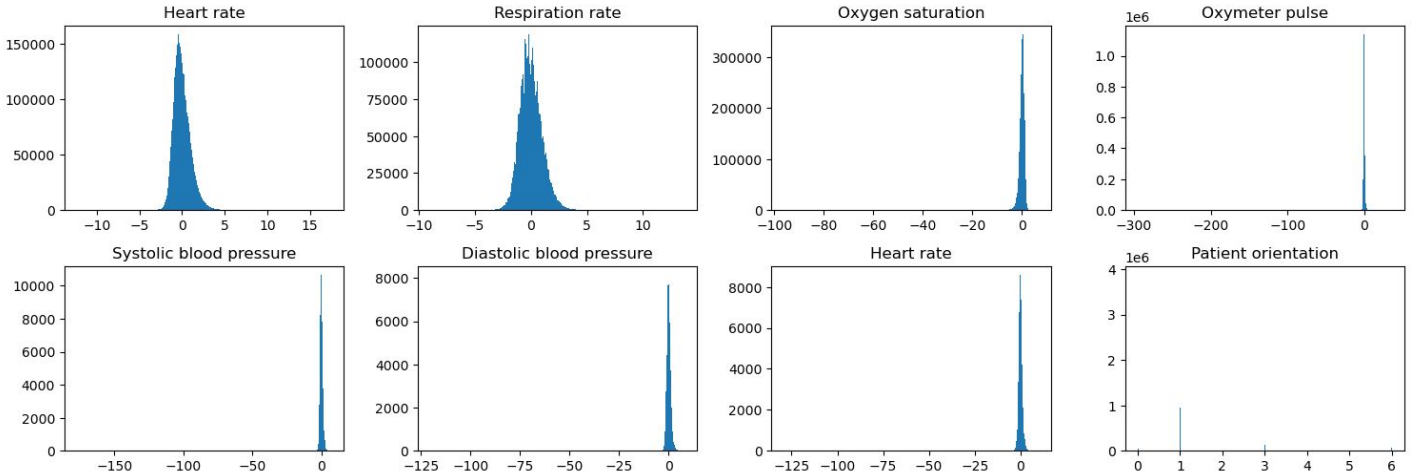
Introduction to the data

Introduction to the data

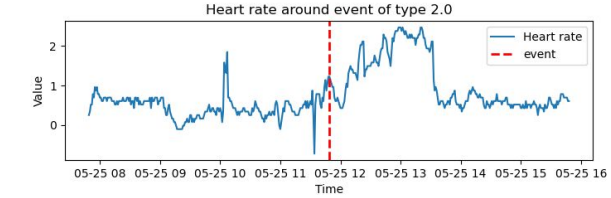
- **6379135** data points from **1393** patients with **729** Adverse Events / Severe Adverse Events
  - 5 different SAE groups and 35 different event types
- 8 features:
  - Heart rate, respiration rate, oxygen saturation, oxymeter pulse, patient orientation: **Every minute**
  - Systolic blood pressure, diastolic blood pressure, blood pressure pulse: **Every 15 minutes**
  - WARD alert as a feature
- Challenges:
  - Small dataset: Not many SAE cases for machine learning!
    - Main focus: Binary classification → SAE or not?
  - Data is very unbalanced
    - Data augmentation or focal loss
  - Many missing data points
    - How should we handle NaNs

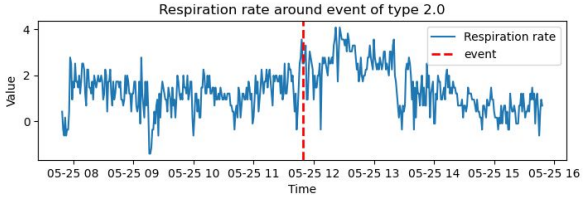
Distributions of feature variables

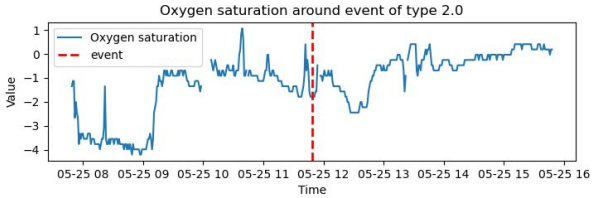
- Data is normalized to mean 0 and variance 1 → Clear outliers

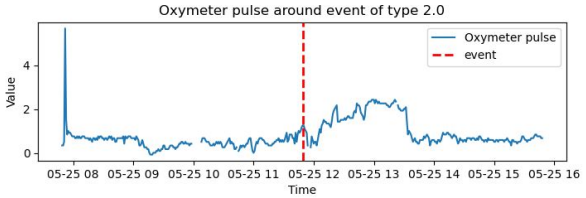


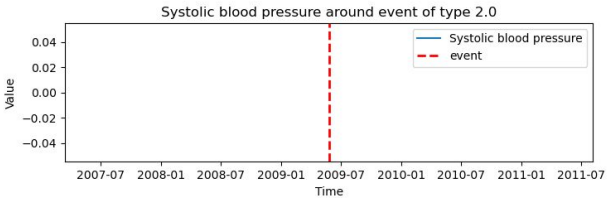
Example of data around SAE

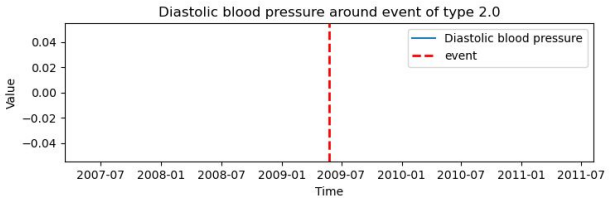


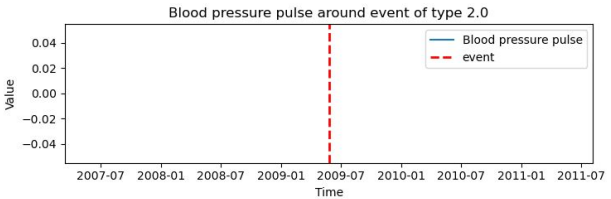


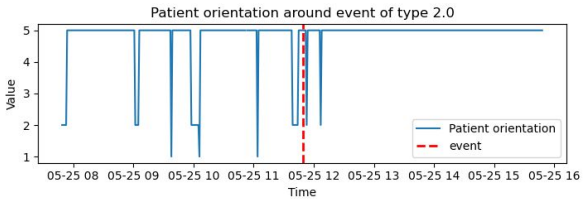








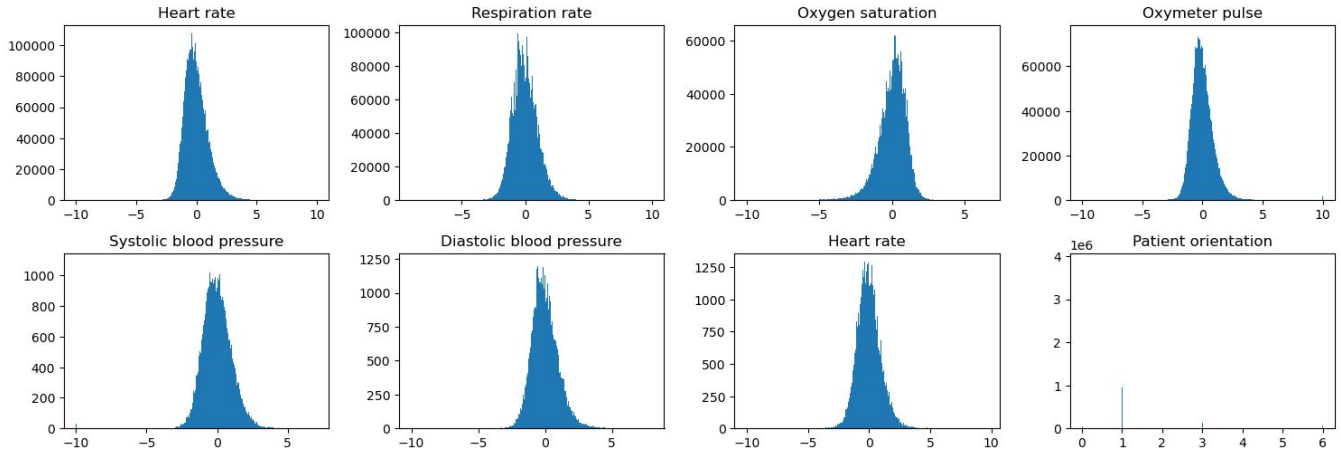




Preprocessing of data

Truncate outliers

- Truncate outliers more than 10 standard deviations from mean



Split data into intervals

- Data for each patient is divided into **8 hour intervals**
- 2 or more instruments on 75 % or more of the time
  - Otherwise too much information is lost
- SAE at the end of interval
  - We want to predict SAE *before* it happens!

Train - test - validation split

- Time series data: We should be careful not to train on future and test on past!

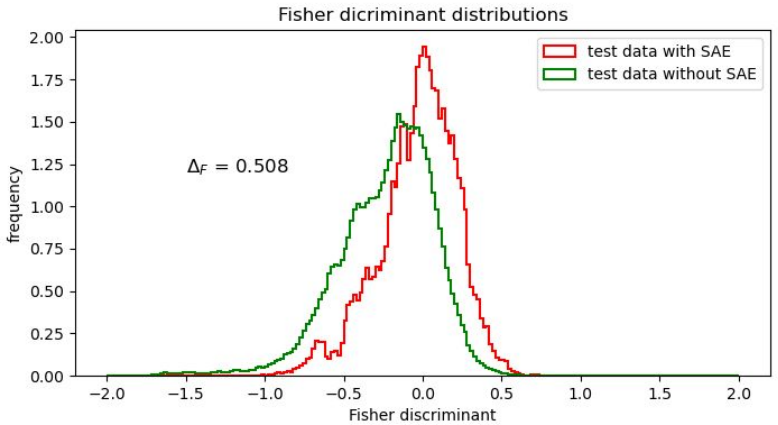
- Solution: **Split data on the basis of patients**
  - Some patients are purely trained on
  - Others are purely used to validate intermediate models
  - The rest are purely tested on

- Few data points: **5 fold cross validation**
  - Allows us to compute *uncertainties* on the AUC scores of models

Our 3 models

Linear model: Linear Discriminant Analysis

- What is the largest *linear* separation between classes?
  - Quickest and simplest model
  - Visualized with histograms
- Expectations:
  - Purely linear → Fast but “bad “model
- No inherent way to deal with temporal relations
  - Cannot input a whole interval: Each time step is input individually
  - (Will add temporal information through features later)



Gradient Boosted Decision Tree: LightGBM

- Our tree based method
- No inherent way to deal with temporal relations:
  - Cannot input a whole interval: Each time step is input individually
- Early stopping to avoid overfitting
- Hyperparameter optimization with Bayesian optimization

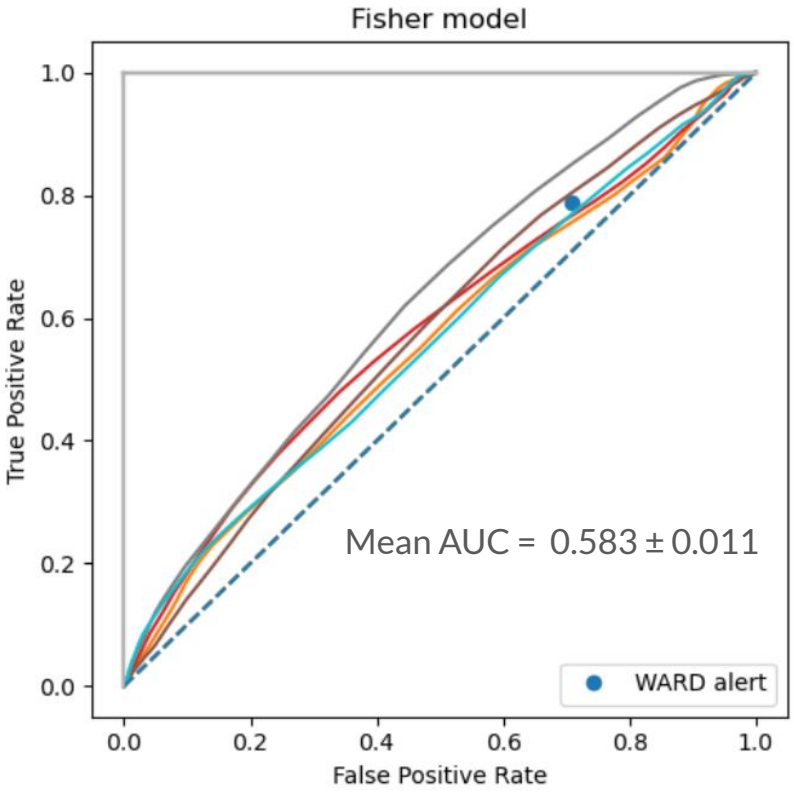
Recurrent Neural Network: LSTM TensorFlow

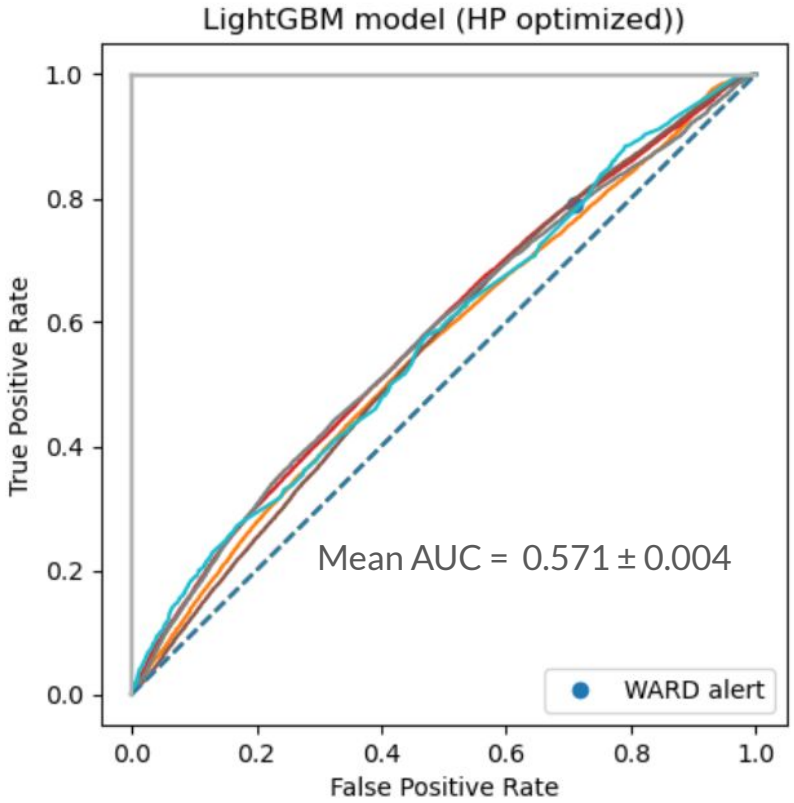
- Recurrent Neural Networks are designed for time series
- Input: An 8 hour interval with a class label: SAE or no SAE at the end
  - We expected this method to work the best
- Early stopping to avoid overfitting
- LSTM vs. GRU
  - GRU initially a bit better (within the uncertainty) but slower → continued with LSTM

A first try at a model

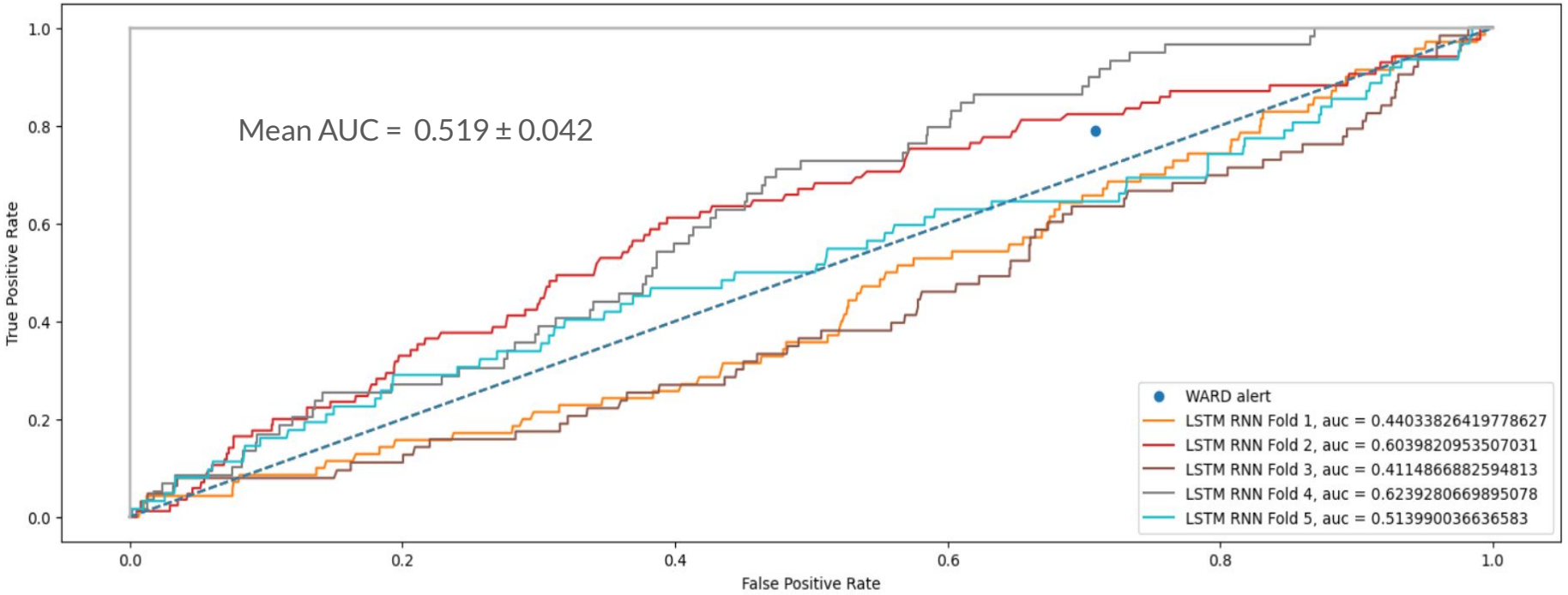
Quick and simple preprocessing

- Remove NaNs (necessary for linear and RNN models)
  - Fill forward with latest measurement
  - Fill backward with next measurement when interval starts with NaN
  - Insert 0 (mean) everywhere else
- Insert “Is fake?” column for each variable
  - 0 when a datapoint is original
  - 1 when datapoint is a NaN that has been filled
- Now we are able to create runnable models…


Fisher model
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Mean AUC = 0.583 ± 0.011
WARD alert


LightGBM model (HP optimized))
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Mean AUC = 0.571 ± 0.004
WARD alert

RNN model


Mean AUC = 0.519 ± 0.042
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
WARD alert
LSTM RNN Fold 1, auc = 0.44033826419778627
LSTM RNN Fold 2, auc = 0.6039820953507031
LSTM RNN Fold 3, auc = 0.4114866882594813
LSTM RNN Fold 4, auc = 0.6239280669895078
LSTM RNN Fold 5, auc = 0.513990036636583

Can we do better?
- more preprocessing

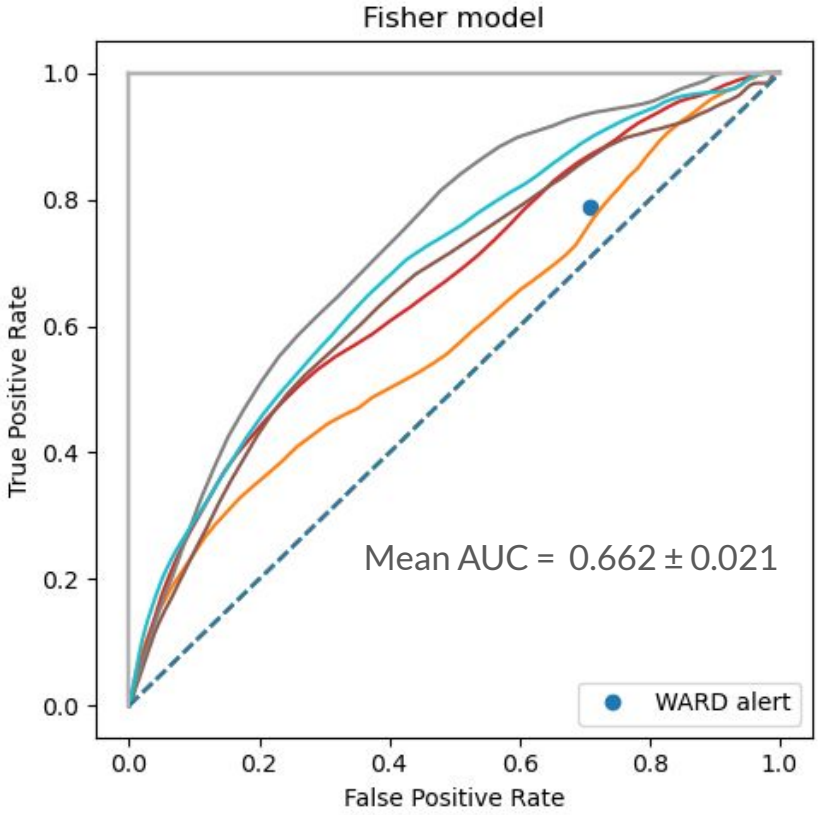
How to add time information?

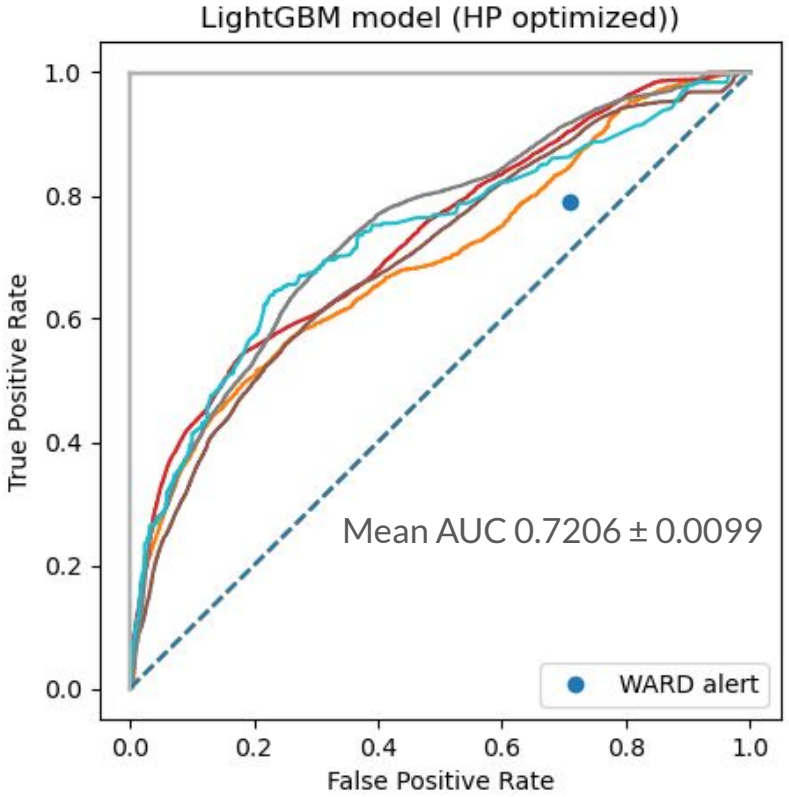
- RNN inherently understands time
  - But how do we convey temporal relations to the linear and tree based model?
- Regression over 15 minutes for the first 3 variables
  - Is heart rate, respiration rate and oxygen saturation rising or falling?
- Each patient is given a unique integer
- For each patient, each interval is also given a unique integer
  - Tells models that these data points are related

Remove NaNs: Regression

- Remove Nans by linear regression between existing data points for continuous variables
  - More realistic than simple fill forward/backward
- Discrete data: Patient orientation
  - Still use fill forward/backward

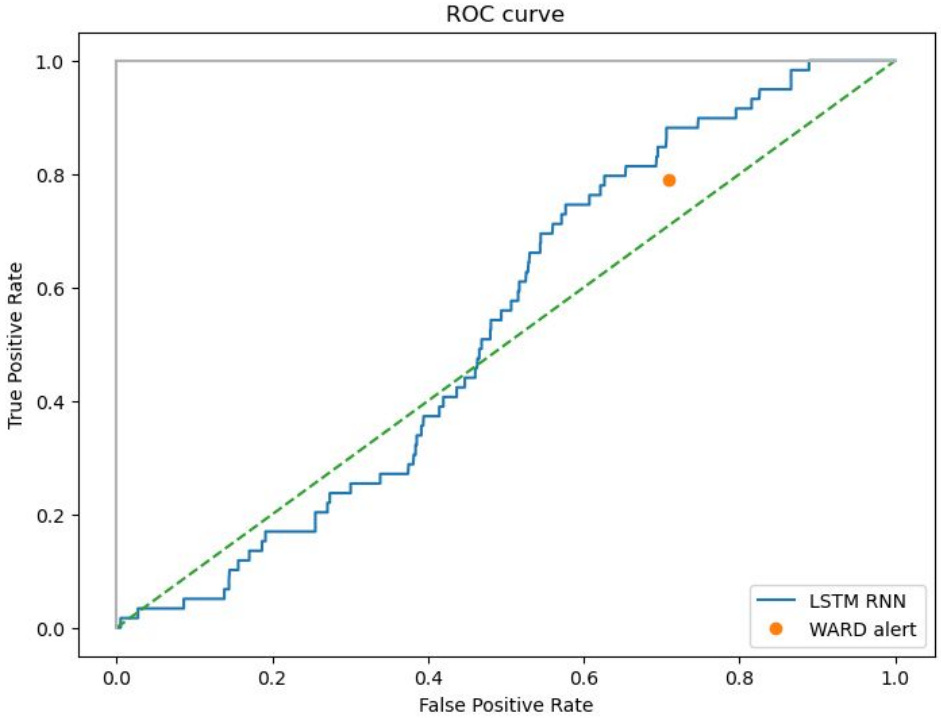
A better model with preprocessing?
(Results)





Linear model (from "raw data"→ with preprocessing)**:**
0.583→ 0.662

BDT model (from "raw data"→ with preprocessing)**:**
0.571 → 0.721



RNN LSTM model:

Mean AUC = 0.525 ± 0.022

From “raw data”→ with preprocessing:
0.519 → 0.525

Note: On plot is shown **fold 4** as it had an auc (0.537) similar to the mean auc.

Data Augmentation

1. Removing non-SAE data

Remove non-SAE data points

Remove 8 hour intervals not containing an SAE until data is more balanced

Before:

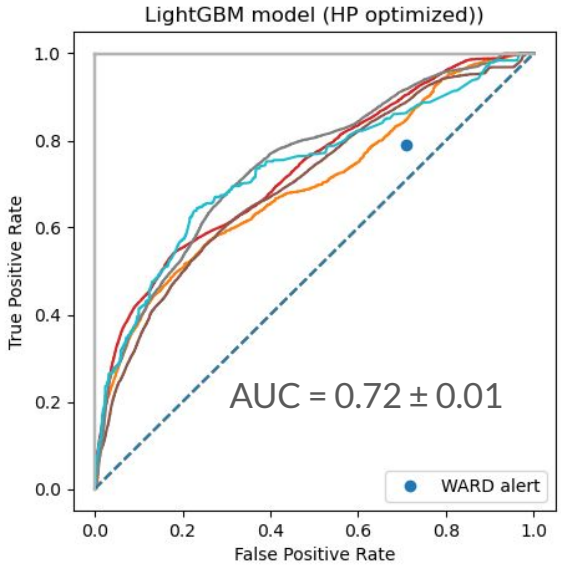
About 2 % of intervals contain SAE

After:

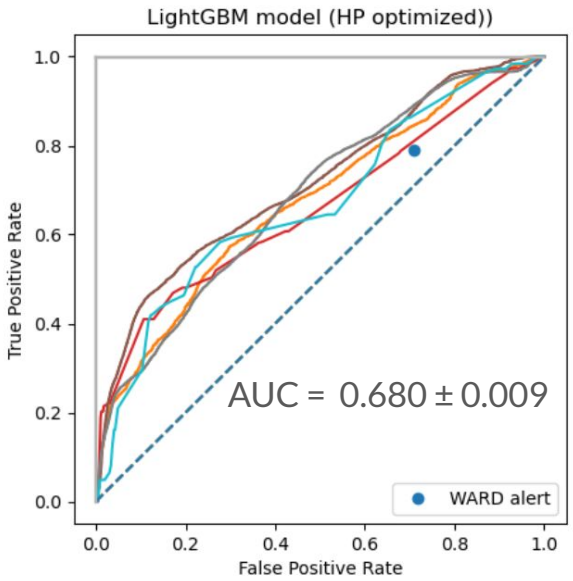
About 33 % of intervals contain SAE → Balanced! (but smaller dataset)

Results of Remove non-SAE data points: GBDT

Only preprocessing:

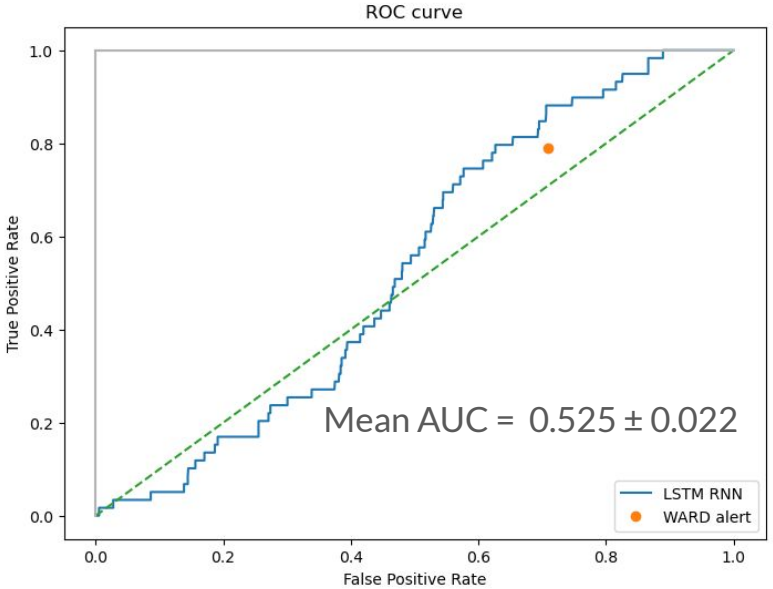


Preprocessing + Remove non-SAE:

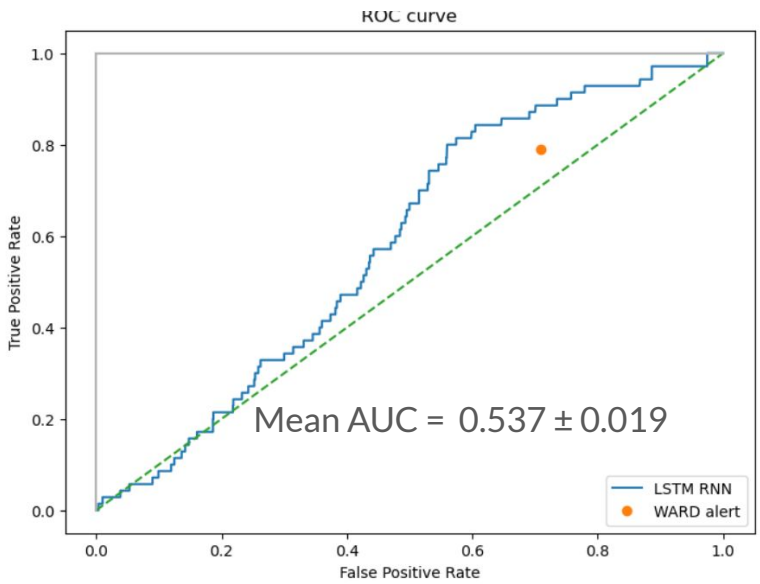


Results of Remove non-SAE data points: RNN

Only preprocessing:

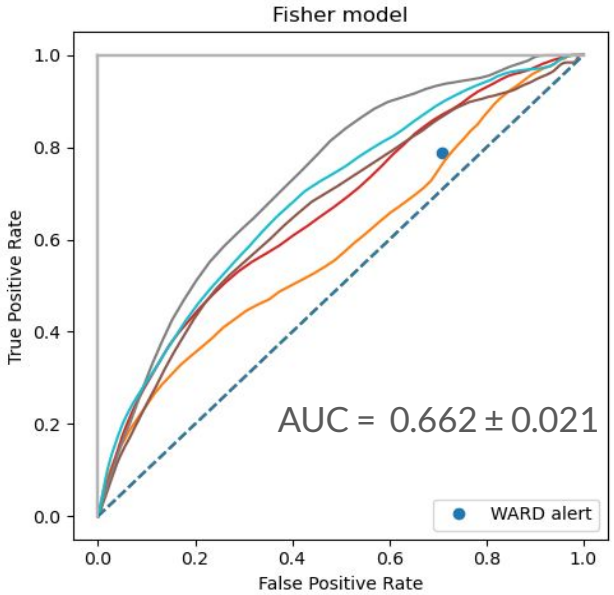


Preprocessing + Remove non-SAE:

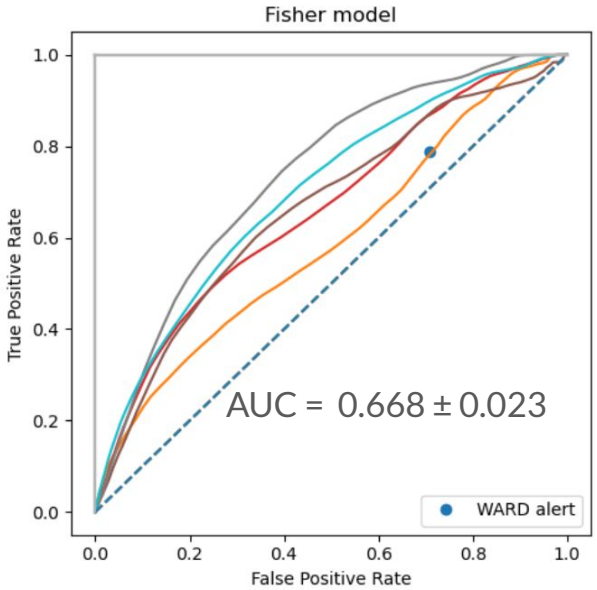


Results of Remove non-SAE data points: Linear

Only preprocessing:



Preprocessing + Remove non-SAE:



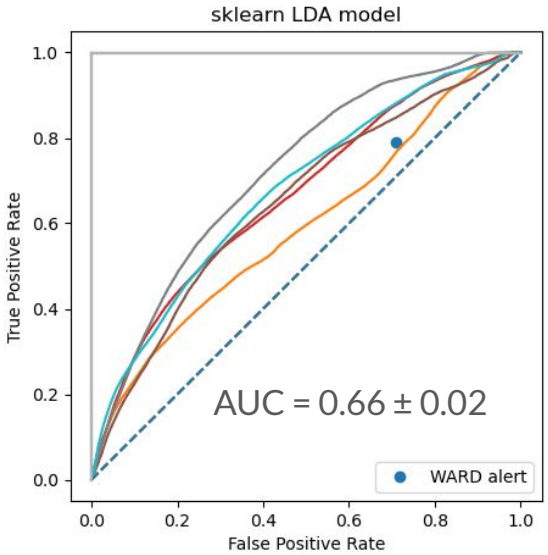
2. Repeated SAE data

Repeated SAE data

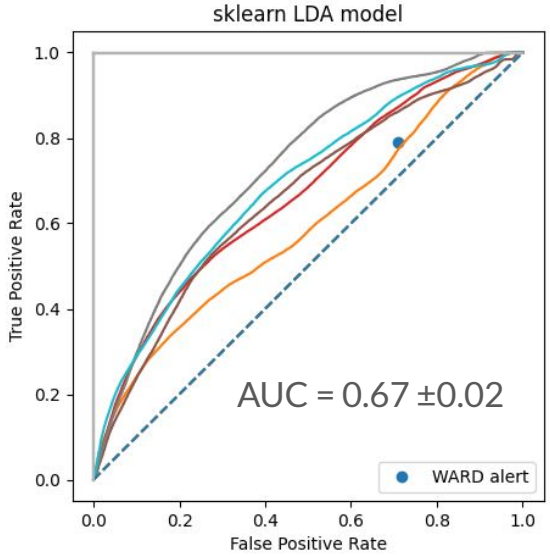
- Find all 8 hour intervals with SAE and duplicate them 10 times
- *Before:*
  - About 2 % of intervals are SAE intervals
- *After:*
  - About 20 % of intervals are SAE intervals → Balanced!

Results of Repeated data: Linear model

Only preprocessing:

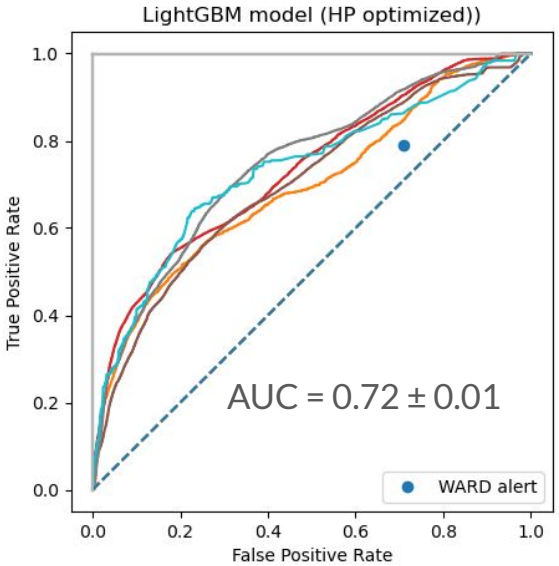


Preprocessing + Repeated data:

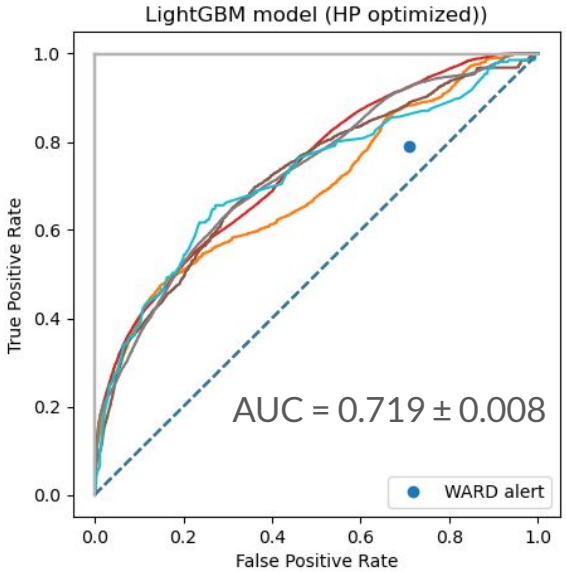


Results of Repeated data: GBDT model

Only preprocessing:

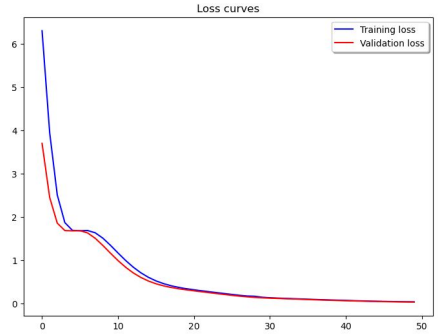


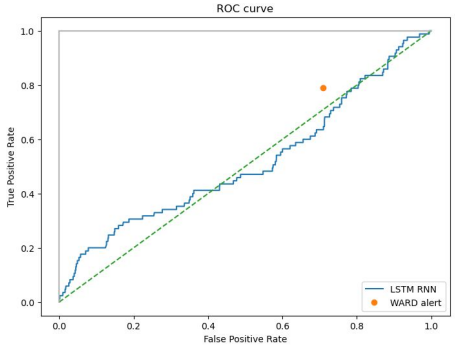
Preprocessing + Repeated data:



Results of Repeated data: RNN LSTM model

Only preprocessing:

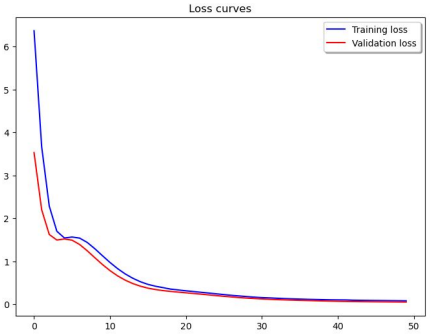


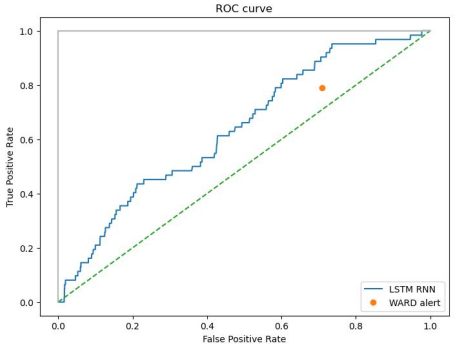


AUC = 0.52 ± 0.02

* Only fold 2 shown since it has a AUC closest to the mean (AUC(fold 2) = 0.514)

Preprocessing + Repeated data:





AUC = 0.61 ± 0.03

* Only fold 5 shown since it has a AUC closest to the mean (AUC(fold 5) = 0.639)

3. Making fake patients

Make “Fake patients”

Find 8-hour interval where SAE ***happens!***

Find the Group (1, 2, 3, 4, 5) of SAE that happens → We make **assumption** that people are alike

→

SO change 3 features around with people in the same group of SAE (under the **assumption**)

Results of Fake patients

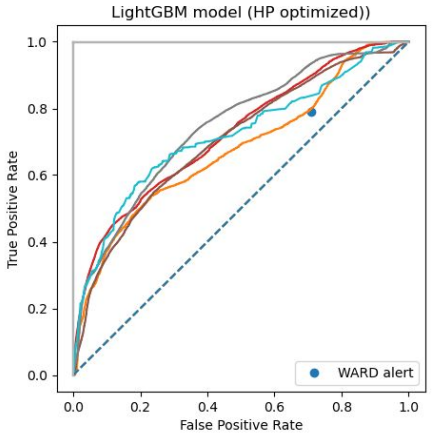
BDT model

Mean AUC=0.713±0.011

"Raw" → Fake patients:

auc=0.721→ auc=0.713

Why? Too board assumption



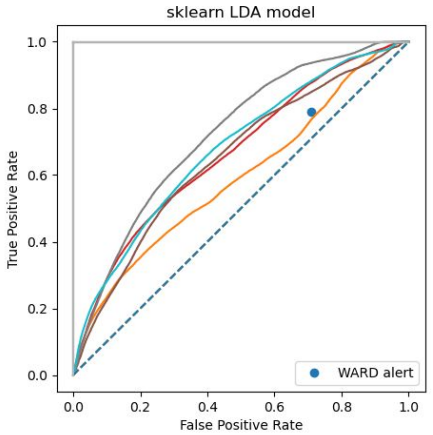
Linear model

Mean AUC=0.662±0.021

"Raw" → Fake patients:

0.661→0.662

No real difference.



Results of Fake patients

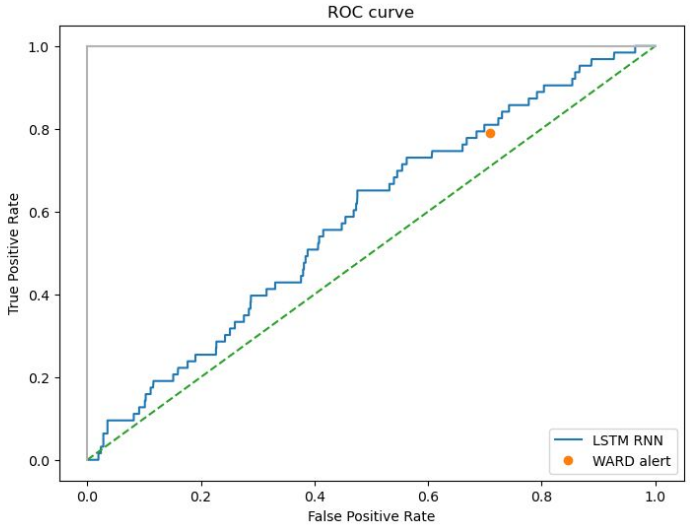
RNN LSTM model

Mean AUC= 0.553 ± 0.041

"Raw" → Fake patients:

auc=0.524→ auc=0.553

Improves but within uncertainty. Why little improvement? → Makes data more balanced



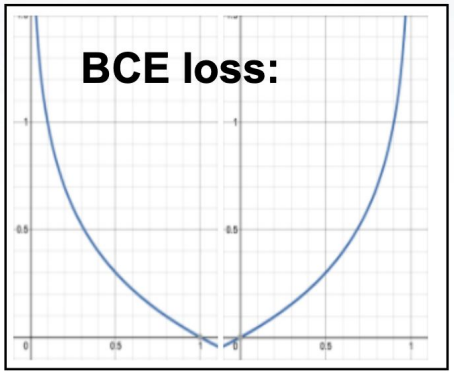
*Plot of **fold 3** as its auc (0.582) is the closest to the mean auc

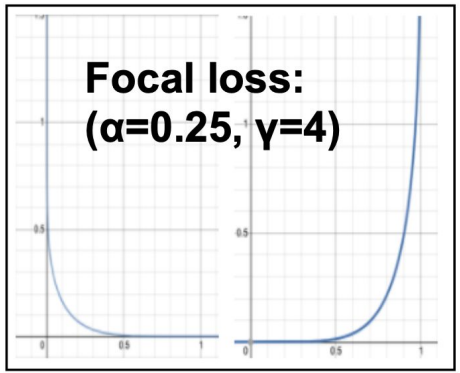
3.5 Implementing Focal Loss for BDT

Alternative: Focal loss in BDT (LightGBM)

- So far: Binary Cross Entropy as loss function
  - But this loss will for unbalanced data be low for a model that always predicts no SAE!

- Alternative loss function: **Focal loss**
  - This puts more emphasis on the lesser represented class!
  - Tried this as an alternative to data augmentation

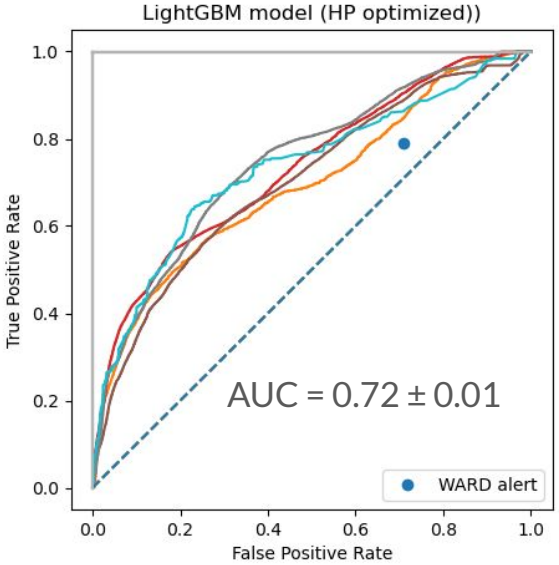




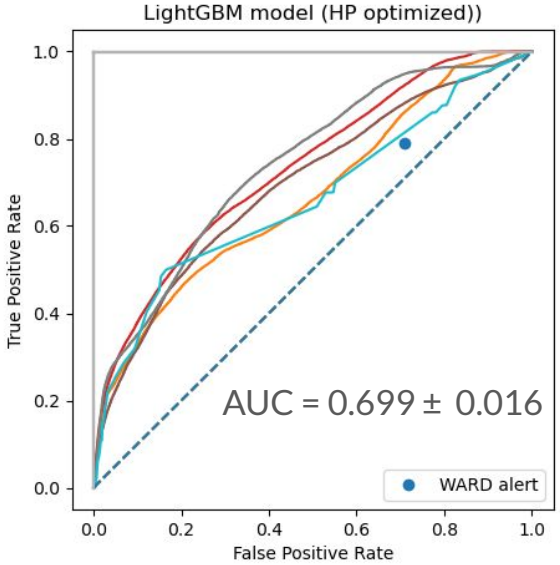
Source:
ML2024_LossFunctions, slide 5

Results of using Focal loss (BDT)

Preprocessing + Binary Cross Entropy loss:



Preprocessing + Focal loss:



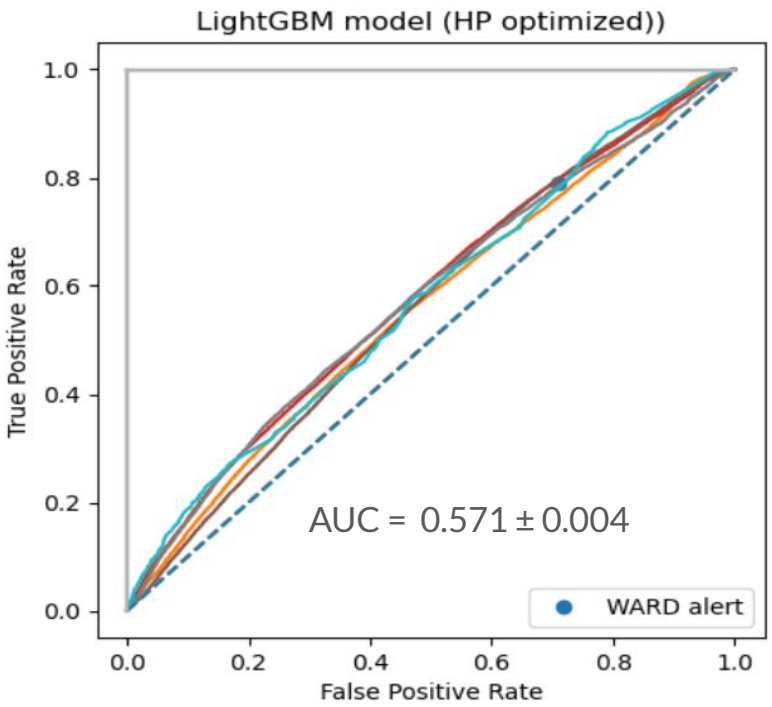
Binary Classification Best model

Our best model

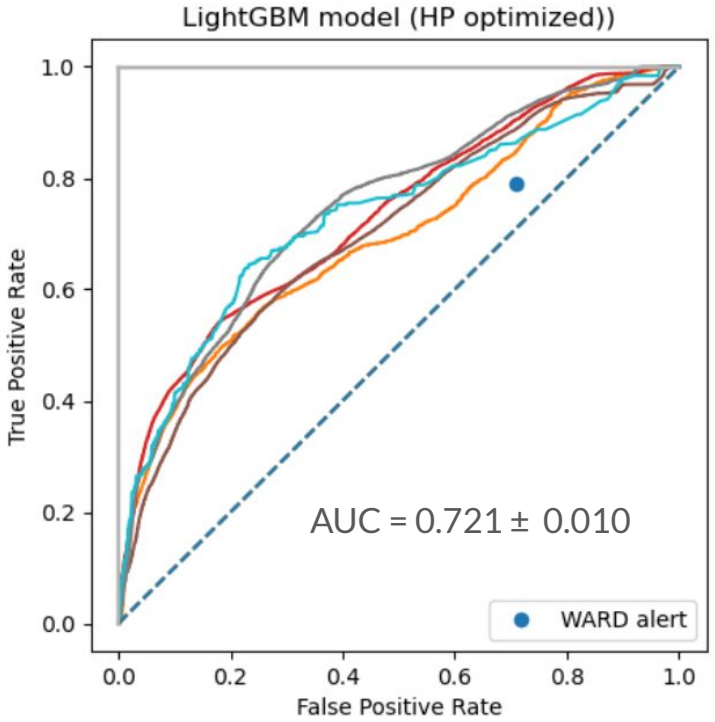
Model	Mean AUC first try	Mean AUC best
GBDT	0.571 ± 0.004	**0.72 ± 0.01**
RNN LSTM	0.52 ± 0.04	0.61 ± 0.03
Linear	0.583 ± 0.011	0.67 ± 0.02

GBDT

First try:



Preprocessing, no data augmentation (best model):



Multi-Class Classification Model

Multi-class prediction

WANT: Prediction of what event **group** → 5 different groups

PROBLEM: Already few SAEs → even **fewer** SAEs in different groups → very unbalanced data!

Solution(?): Use repeated event data.

Performance of duplicating SAE data intervals?

Multi Class BDT LightGBM, only preprocess:

Group 1: Mean AUC score is 0.6811 ± 0.079

Group 2: Mean AUC score is 0.7074 ± 0.038

Group 3: Mean AUC score is 0.6209± 0.089

Group 4: Mean AUC score is 0.5202 ± 0.085

Group 5: Mean AUC score is 0.6739 ± 0.019

Multi Class BDT LightGBM, with duplicated data:

Group 1: Mean AUC score is 0.7235 ± 0.062

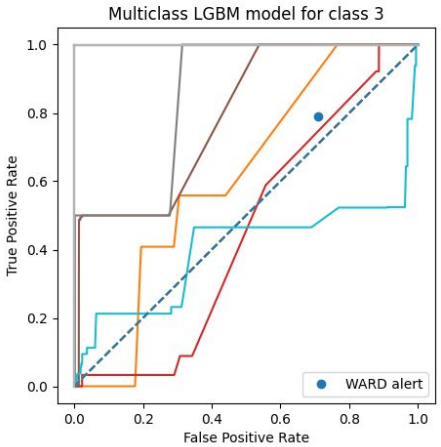
Group 2: Mean AUC score is 0.7689 ± 0.022

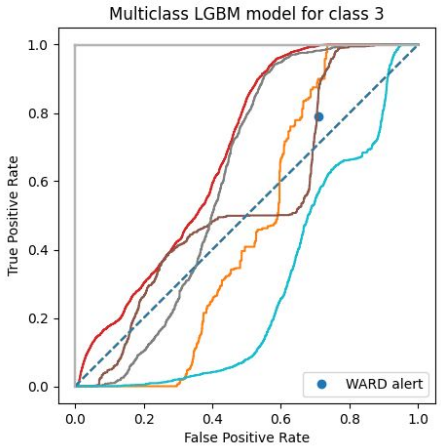
Group 3: Mean AUC score is 0.5183 ± 0.066

Group 4: Mean AUC score is 0.7443 ± 0.054

Group 5: Mean AUC score is 0.6759 ± 0.014

ROC-curves for multi class classification (BDT)





Group 3

Left: Original data (preprocessed)

Right: With duplicated SAE intervals

Unbalanced(not enough different classes) data visible in the less smooth roc curves.

(colors are the different folds)

Performance of duplicating SAE data intervals?

Multi Class RNN TensorFlow, only preprocess:

Group 1: The mean AUC score is 0.3471 ± 0.081

Group 2: The mean AUC score is 0.5932 ± 0.043

Group 3: The mean AUC score is 0.6124 ± 0.075

Group 4: The mean AUC score is 0.5260 ± 0.12

Group 5: The mean AUC score is 0.5727 ± 0.063

Multi Class RNN TensorFlow, with duplicated data:

Group 1: The mean AUC score is 0.5946 ± 0.069

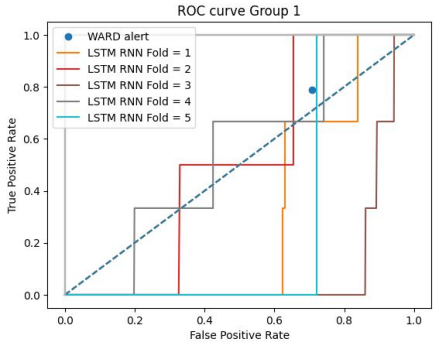
Group 2: The mean AUC score is 0.6495± 0.014

Group 3: The mean AUC score is 0.6876 ± 0.076

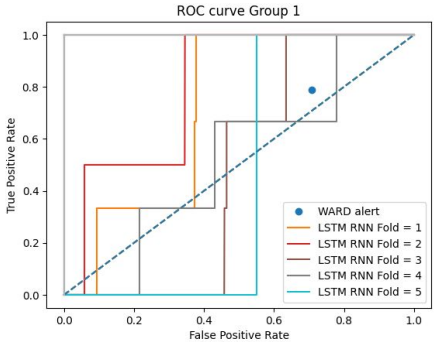
Group 4: The mean AUC score is 0.5280 ± 0.081

Group 5: The mean AUC score is 0.5943 ± 0.040

ROC-curves for multi class classification (RNN LSTM)



The mean AUC score is 0.3471 ± 0.081



The mean AUC score is 0.5946 ± 0.069

Left: Only preprocessing

Right: With duplicated event intervals

No smooth curves due to the input data being 8-hour intervals.

Visible that the ROC curves with duplicated data skew to the left of the WARD alert point.

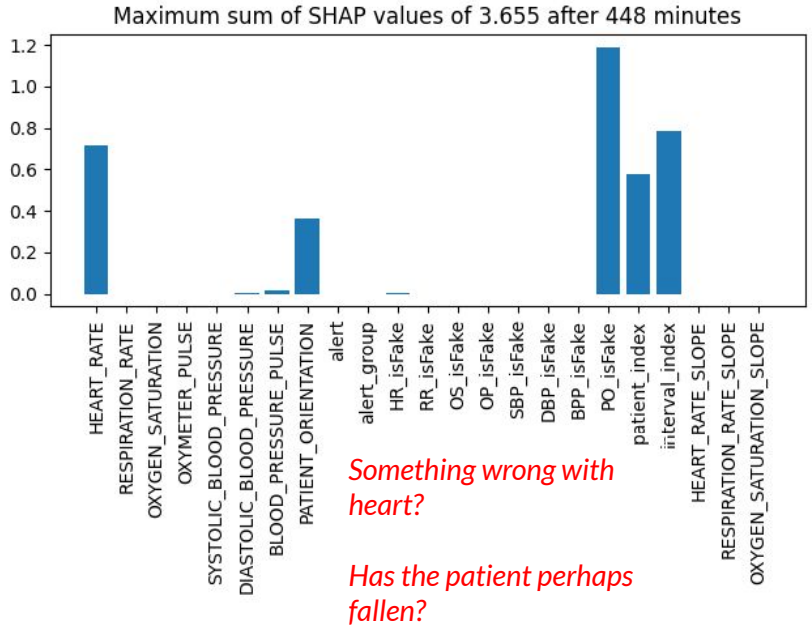
What made the alarm go off?

(SHAP-values)

What made the alarm go off?

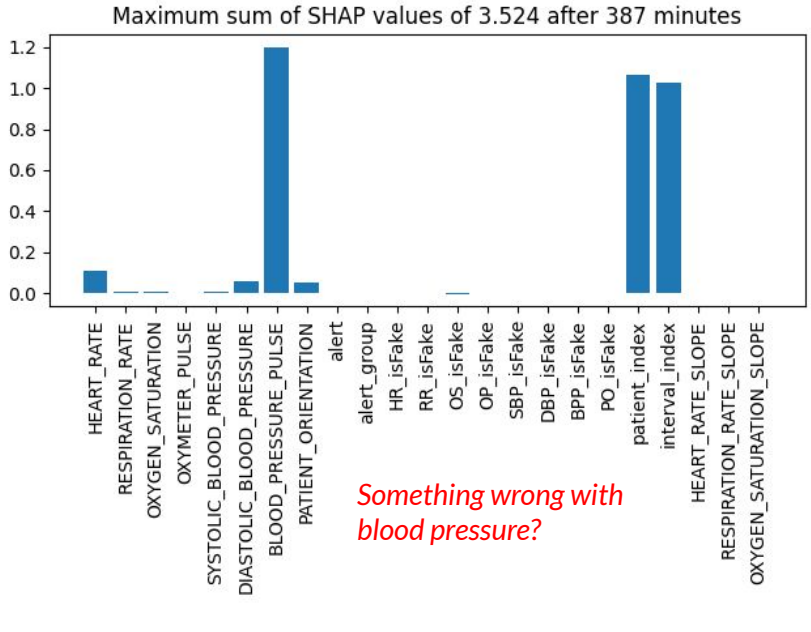
- We were told that the nurses at the hospital would like to know why an SAE is predicted
  - What is wrong with the patient?
- SHAP values can possibly give insight:
  - If a feature has a large positive SHAP value in a data point:
    - This feature pushed to model towards predicting an SAE
- A bar plot of the SHAP values of the features in a datapoint where an SAE is predicted can tell us WHY the model predicted an SAE in the interval

SHAP for individual predictions



Something wrong with heart?

Has the patient perhaps fallen?



Something wrong with blood pressure?

Conclusion

Conclusion

- GBDT with preprocessing but no data augmentation performed best
- Surprised us that it was better than RNN (made for time series)
- Probably due to so little data + unbalanced
    - Which LightGBM is good at handling but RNN is not

Appendix

Appendix
Our models: Recurrent Neural Network

Our choice of models: Recurrent Neural Network

Given that the WARD data is time series data, the most natural choice of model is a recurrent neural network, specifically a Gated Recurrent Unit since the data is numerical.

During the weeks that we worked on the project, we tried both RNNs in the form of Gated Recurrent Units (GRUs) and in the form of Long Short Term Memory (LSTM). We found that the LSTM based models and the GRU based models performed almost equally well, but GRU based models took significantly longer to train.

Therefore, we chose to use LSTM based RNNs throughout the final runs of our RNN models due to time constraints.

Input data

Since RNNs inherently understand temporal relations, the input data is simply an 8 hour interval (or an interval of some other length) showing how each feature variable evolves in time. The truth data is 0 if there is no SAE at the end of the given interval and 1 if there is an SAE at the end of the interval.

RNN structure

Our final RNN consisted of 3 LSTM layers and an output layer.

We made sure to have more nodes than input features in the first LSTM layer. Otherwise, we found that the performance would worsen significantly, probably due to too much of the information in the feature variables being “compressed” too quickly. In the second and third layer, we gradually decreased the number of nodes until we reached the output layer with a single node for binary classification (SAE or no SAE) and 6 nodes for multiclass classification (no SAE and one for each of the 5 SAE groups)

RNN hyper parameters

Loss function: Focal loss (since the data is unbalanced) - TensorFlow's BinaryFocalCrossentropy

Early stopping when validation loss stops improving to prevent overfitting

Layers: [30 , 25, 15, 1] for data with no (minimal) preprocessing. [35, 25, 15, 1] for data with preprocessing and data augmentation as we "add" features to tell the model that things are augmented.

Activation is tanh and hard_sigmoid (best for RNNs).

Appendix
Our models: Linear model

Our choice of models: Linear model

We wanted to try a variety of models and see how high an AUC score we could get for different models of varying complexity.

The linear model (Linear Discriminant Analysis) is our most simple model. This model tests what the maximum *linear* separation between classes (SAE vs. no SAE) is. The fact that this model is purely linear makes it incredibly fast compared to for example a Neural Network or Decision Tree. However, we did expect our linear model to perform the worst out of our 3 models since our other two learning based models are able to separate the classes non-linearly.

We both tried implementing our own Fisher Discriminant Analysis and tried the built in LinearDiscriminantAnalysis from sklearn and got very similar results.

Input data

Unlike RNNs, the linear model does not have a direct way of dealing with temporal relations, and therefore it cannot take a whole 8 hour interval of data points as input at a time. Instead we chose to input each time step individually, and assigned the truth value 1 to the data point if it originates from an 8 hour interval with an SAE at the end and 0 otherwise.

In an attempt to “put back” some of the temporal information, we created a new feature column telling which patient the data point originates from and which 8 hour interval from this patient that the data point originates from.

Note, that since we split the data up into training, testing and validation sets based on patients, we have eliminated the risk of testing on patients that the model knows about from training.

Fisher Discriminant Analysis

Fisher Discriminant Analysis works by calculating a weight for each of the input feature variables. These weights correspond to the linear combination of the input features that gives the maximum linear separation between classes.

The weights are calculated from:

$$\vec{w} = (\Sigma_{\text{SAE}} + \Sigma_{no\ SAE})^{-1}(\vec{\mu}_{SAE} - \vec{\mu}_{no\ SAE})$$

where Σ_{SAE} is the covariance matrix for data points in an interval with an SAE, and $\vec{\mu}_{SAE}$ is a vector with the mean of the input feature variables for data points in an interval with an SAE.

Linear class separation

In histograms showing the distributions of Fisher discriminants for the two classes, the maximum linear class separation has been plotted based on the linear separation between classes based on the Fisher discriminant.

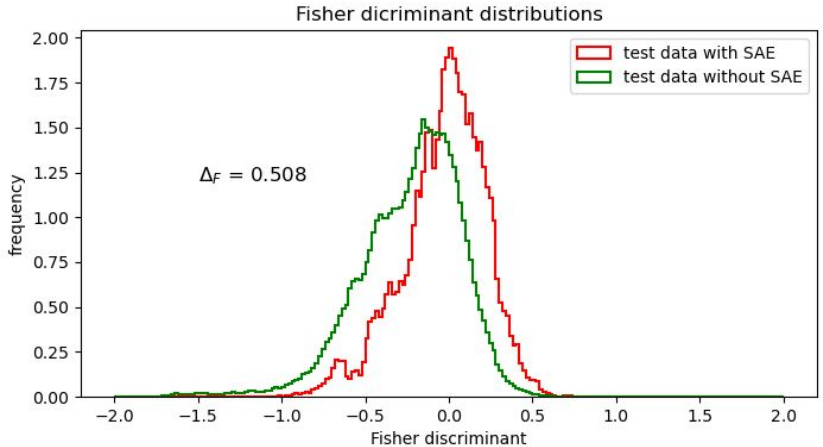
The linear separation is defined as:

$$\Delta_F = \frac{|\bar{x} - \bar{y}|}{\sqrt{\sigma_x^2 + \sigma_y^2}}$$

Linear class separation

Here, an example of a histogram showing the maximum linear separation between classes for one of the folds in our best linear model is shown. It is evident that there is some small separation between intervals with and without an SAE at the end, but overall there is also huge overlap between the two classes. This is in accordance with our expectation that the purely linear model is not very good at separating the two classes.

The corresponding ROC curves can be found in the main body of the slides.



The linear separation is 0.508.

Appendix
Our models: Gradient Boosted Decision Tree

Our choice of models: Gradient Boosted Decision Tree

We also wanted to see how well we could get a tree based model to perform. From our initial projects, we all had very good experience with using Gradient Boosted Decision Trees from LightGBM, and therefore we went with this algorithm.

We expected this model to perform better than the purely linear model since decision trees are able to separate classes non-linearly.

Unlike RNNs, this model does also not have a direct way of handling temporal relations. We were therefore very interested to see that we could actually get this model to perform better than an RNN despite this.

Input data

Like the linear mode, the GBDT model cannot take a whole 8 hour interval of data points as input at a time. Instead, we chose to input each time step individually, and assigned the truth value 1 to the data point if it originates from an 8 hour interval with an SAE at the end and 0 otherwise.

In an attempt to “put back” some of the temporal information, we created a new feature column telling which patient the data point originates from and which 8 hour interval from this patient that the data point originates from.

Note, that since we split the data up into training, testing and validation sets based on patients, we have eliminated the risk of testing on patients that the model knows about from training.

Hyper parameters and optimization

In an attempt to get as much performance out of this model as possible, we performed hyperparameter optimization on a few of the hyperparameters using **Bayesian optimization**. The hyperparameters that we optimized were learning_rate (range: 0.01 to 0.2), max_depth (range: 5 to 50) and num_leaves (range: 10 to 100).

Furthermore, we implemented **early stopping** to stop training when validation loss had not improved for 20 rounds. This prevents overfitting. Therefore, even though we set n_estimators=1000, the training actually stopped before this due to the early stopping.

The rest of the parameters were set to default.

Loss function: Cross entropy vs. Focal loss

In most of our GBDT models, we have simply used Cross Entropy loss. However, when data is very unbalanced with very few SAE cases (as in our case), Cross Entropy loss will also be low for a model that always predicts no SAE. This is obviously unwanted.

A different loss function that puts more emphasis on the lesser represented class is **Focal loss**. We therefore implemented Focal loss in a GBDT model to try this as an alternative to data augmentation.

Hyperparameters used in Focal loss:

$$\alpha = 0.25, \ \ \gamma = 4.0$$

GBDT with Focal loss: Results

Preprocessing + Binary Cross Entropy loss:

AUC = 0.72 ± 0.01

Preprocessing + Focal loss:

AUC = 0.699 ± 0.016

Comments:

It is evident that the Focal loss did not improve the performance of the GBDT model. In fact, the performance dropped a bit (but only within a few std so it is not very significant). This probably shows that LightGBM is relatively good at handling the unbalanced data on its own.

Furthermore, we might have been able to tweak the Focal loss hyperparameters even more to get the model to perform better. But due to time constraints, we did not prioritize this.

Appendix
Evaluation of models

ROC curves and AUC

Since our data is very unbalanced, special care must be taken when evaluating model performance. If we used accuracy as a performance measure and only 2 % of our intervals are SAE intervals, then a model that always predicts no SAE will be 98 % accurate. This obviously defeats the purpose and therefore accuracy is not an appropriate performance measure in this case.

Instead, we evaluated our models based on ROC curves and Area Under Curve (AUC). Since these are based on the False Positive *Rate* and True Positive *Rate*, this behaves nicely even though data is unbalanced.

An AUC score of 0.50 corresponds to a random classifier, and an AUC score of 1.0 corresponds to a perfect classifier. Therefore, the best model is the one that has an AUC score closest to 1.0 on the test set.

Appendix: Eight hour intervals

(Any hour intervals)

How the 8-hour intervals were made:

1: Go through patients, and define time-interval frame.

2: If event is in time-interval make sure it happens at the very end. [95% into the interval]. If not, move the time-interval such that it does.

3: Then check if there are enough measurement devices ON → those that are done 1 pr. min above 75% and those that are 1 pr. 15/30 min > 1%. ONLY append if so.

4: When we move interval back, I want to remove the previous, IF the previous time-interval has been appended! (To avoid repeated data)

5: Return: List of lists of dataframes.

6: Each list is a patient and the dataframe are the time-intervals.

NB: We use 8 hour interval primarily but can use any length of interval.

Appendix
Train-Test-Validation split

Cross validation

We have relatively few intervals with an SAE present, and therefore not many SAE cases to train, test and validate on. Due to the low statistics, we want to do cross validation which allows us to compute the uncertainty on our model performance.

We chose to use 5 fold cross validation since 5 folds are enough to get a decent uncertainty estimate but not so many folds that our validation and testing data sets become unreasonable small.

Hence, within each fold, ⅕ of the data is test data, and ⅘ is training data. 10 % of the training data is separated to be validation data for testing intermediate models during training.

The linear model does not use validation data, and thus for this model we did not separate the 10 % from the training data.

Split on the basis of patients

Since we have time series data, we need to be careful that we don't train on data from "the future" and then test on data from "the past".

In order to avoid this problem, we split the data into training, testing and validation sets based on **patients**. This means that all data for some patients are trained on, other patients are validated on and the rest are tested on. Since patients are independent from each other, this solves the problem.

Appendix
Separation of truth data from input data

Truth data selection

Binary classification: If an entry in the feature column “event_group” is not NaN in an 8 hour interval, then the truth value for this 8 hour interval is 1 (SAE present at the end of the interval). Otherwise the truth value is 0 (SAE not present). After this selection, the columns with the class label information (“event_group” and “event”) are dropped.

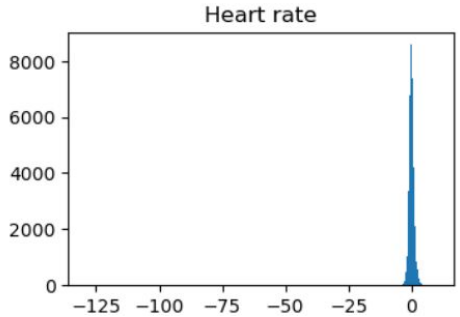
Multiclass classification: If an entry in the feature “event_group” is not NaN in an 8 hour interval, then the truth value for this 8 hour interval is the value that “event group” takes (SAE type between 1 and 5 at the end of interval). Otherwise the truth value is 0 (SAE not present). After this selection, the columns with the class label information (“event_group” and “event”) are dropped.

Appendix
Removing outliers

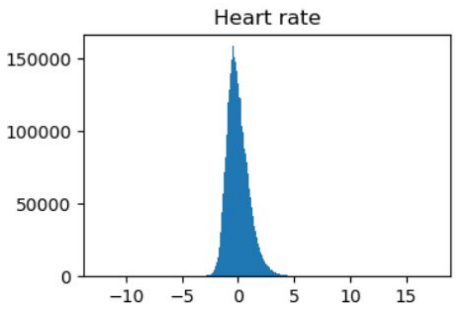
Removing outliers

An important part of preprocessing the data, was removing the outliers. This is important because narrow, spiky distributions can be detrimental to neural network performance. Since the data was normalized around 0 with variance 1, this was simply done by going through the feature columns and replacing values smaller than -10 with -10 and values larger than 10 with 10. Hence, we truncated any feature variables further than 10 standard deviations from the mean.

This way we went from histograms that looked like this:



To histograms that looked like this:



Appendix
Remove NaN entries

Simple remove NaN

For our first attempt at a model, we used a simple method for removing any NaNs from the data. We had to remove the NaNs because the linear and RNN based models cannot take NaNs as input (while the decision tree can).

First, we created eight new feature columns for stating whether a feature value is imputed or not (for example “HEART_RATE_is_fake”. The entry is set to 0 for entry values that are original to the data set and 1 for entry values that are NaN, and thus are going to be imputed.

First, any NaN values are filled with the latest non-NaN measurement of the feature (fill forward). Sometimes an interval starts with a feature which is NaN, and in this case we impute with the next non-NaN measurement (fill backward). Any remaining NaNs are filled with 0s (the mean).

Remove NaNs with regression

Next we tried a more advanced way of removing NaNs. This time, we used linear regression to fill out the NaNs between existing measurements.

First, the nearest non-NaN measurements before and after a given index is found. Then the missing value is added through linear interpolation. For NaN values not between two existing measurements, linear regression is not possible. Thus, NaNs before the first existing measurement and NaNs after the last existing non-NaN value were simply replaced by zeros.

For the feature “PATIENT_ORIENTATION”, we still use the simple method since this is a discrete variable.

As with the “simple remove NaN” method, eight extra feature columns indicating whether the data was imputed (1) or not (0), were also created.

Appendix
Adding temporal information through slopes

Calculating the slopes

Unlike with the RNN, using a time series for the BDT or linear model is not possible. So in order to preserve some sense of time evolution in the data for these models, we tried adding the slopes of the heart rate, respiration rate and oxygen saturation as feature variables in addition to the measurements.

In order to calculate the slopes, the data for these variables is first divided into 15 minute intervals. Then, using linear regression, a line is fit to the measurements for each feature variable. The slopes of the resulting linear fits are then returned and saved in a new DataFrame.

The reason that we calculate the slopes for the heart rate, respiration rate and oxygen saturation is that these are measured every minute.

Another step we took to put temporal information back into the BDT and linear models was to add a feature telling the model which patient a data point belongs to and another integer telling the model which interval for the patient that a data point is in. Our reason for adding these indices is to give the models a sense for which data points are "closer" to each other both in the sense of belonging to the same patient and in the temporal sense of belonging to the same interval.

Performance of adding slopes and patient_index + interval_index?

Binary BDT LightGBM (not HP optimized):

It did better with the slopes and indices

No HP-optimization, first try:

Mean AUC score is 0.5802 ± 0.0091

No HP-optimization, using slopes and indices:

Mean AUC score is 0.6957 ± 0.015

Binary BDT LightGBM (HP optimized):

It did better with the slopes and indices

With HP-optimization, first try:

Mean AUC score is 0.5709 ± 0.0039

With HP-optimization, using slopes and indices:

Mean AUC score is 0.7206 ± 0.0099

Performance of adding slopes and patient_index + interval_index?

Binary Linear

First try:

Mean AUC score is 0.583 ± 0.011

Using slopes and indices:

Mean AUC score is 0.66 ± 0.02

Binary TensorFlow LSTM RNN

First try:

Mean AUC score is 0.52 ± 0.04

Using slopes and indices:

Mean AUC score is 0.52 ± 0.02

Comments on results

Both the linear and GBDT models improved significantly after our attempt to input temporal information through linear regression and the mentioned indices. This indicates that this strategy successfully made these models more aware of time.

The LSTM RNN model did not improve after adding these features. However, this is also what we would expect since RNNs are designed for time series and therefore should be able to understand evolution in time already before adding the slopes. Hence, adding these features should not give the model any additional information.

Appendix
Balanced data: Remove data points

Randomly remove non-SAE intervals

A possible way to deal with the fact that our data is very unbalanced is to remove random 8 hour intervals without an SAE at the end until the data is approximately balanced:

1. The 8 hour intervals with and without an SAE at the end are separated into two different lists.
2. Then random 8 hour intervals without an SAE are selected from the list until we have twice as many intervals without an SAE as intervals with an SAE.
3. Then all intervals with an SAE and the selected ones without an SAE are put back into a common list, and the list is shuffled.

Now the data is approximately balanced with ⅓ of the intervals being SAE intervals and ⅔ being non-SAE intervals.

Results

Linear model

Only preprocessing:

AUC = 0.66 ± 0.02

Preprocessing and removed data points:

AUC = 0.67 ± 0.02

GBDT LightGBM model

Only preprocessing:

AUC = 0.72 ± 0.01

Preprocessing and removed data points:

AUC = 0.680 ± 0.009

RNN LSTM model

Only preprocessing:

AUC = 0.52 ± 0.02

Preprocessing and removed data points:

AUC = 0.537 ± 0.019

Comments on results

The *linear model* did not improve significantly after removing data points to make the dataset balanced (results before and after are consistent within 1 std). This is in accordance with our expectations since this model is purely linear and therefore should not “care” about whether the data is unbalanced or not.

The *GBDT model* worsened slightly after removing data points. We argue that this is due to LightGBM being relatively good and handling an unbalanced dataset on its own, and the drop in performance is due to the information lost with the removed data points since the dataset is very small after this.

The *LSTM RNN model* improved slightly but it is not very significant since the results are consistent within 1 std. We argue that the reason for this is that any increase in performance due to the more balanced data is probably mostly “cancelled” by the information lost with the removed data points since the dataset is very small after this.

Appendix:
Duplicate SAE data intervals

How and what is duplicating SAE data intervals?

Motivation and reasoning:

We wanted to make the data more balanced, so adding more of the already existing data without doing anything to it was a simple way (even though this of course does not add any new information).

```
1: Find all time-intervals that contain an event.

2. Copy them and randomly insert those intervals into
the data.
```

```
NB: We can control how many times we want to add extra
intervals into our data. So we add 10x as much as there was
originally present. This increases our fraction of SAE
intervals from about 2% to about 20% and hence making the
data decently balanced.
```

Performance of duplicating SAE data intervals?

Binary BDT LightGBM (not HP optimized):

It did more or less equally well with and without the duplicated data.

No HP-optimization and only preprocess:

Mean AUC score is 0.6957 ± 0.015

No HP-optimization, with duplicated SAE data

Mean AUC score is 0.7062 ± 0.012

Binary BDT LightGBM (HP optimized):

It did more or less equally well with and without the duplicated data.

With HP-optimization and only preprocess:

Mean AUC score is 0.7206 ± 0.0099

With HP-optimization, with duplicated SAE data:

Mean AUC score is 0.7193 ± 0.0076

Performance of duplicating SAE data intervals?

Multi Class BDT LightGBM, only preprocess:

Group 1: Mean AUC score is 0.6811 ± 0.079

Group 2: Mean AUC score is 0.7074 ± 0.038

Group 3: Mean AUC score is 0.6209± 0.089

Group 4: Mean AUC score is 0.5202 ± 0.085

Group 5: Mean AUC score is 0.6739 ± 0.019

Multi Class BDT LightGBM, with duplicated data:

Group 1: Mean AUC score is 0.7235 ± 0.062

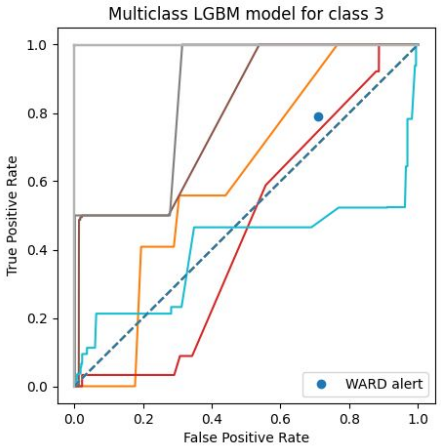
Group 2: Mean AUC score is 0.7689 ± 0.022

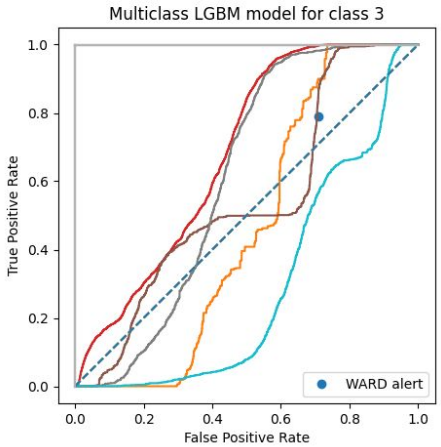
Group 3: Mean AUC score is 0.5183 ± 0.066

Group 4: Mean AUC score is 0.7443 ± 0.054

Group 5: Mean AUC score is 0.6759 ± 0.014

ROC-curves for multi class classification (BDT)





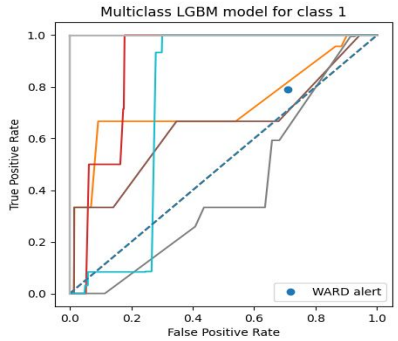
Group 3

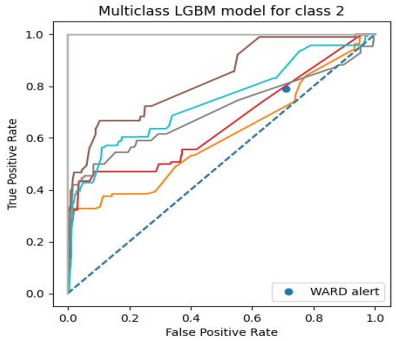
Left: Original data (preprocessed)

Right: With duplicated SAE intervals

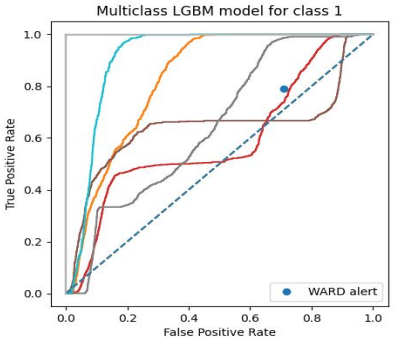
Unbalanced(not enough different classes) data visible in the less smooth ROC curves.

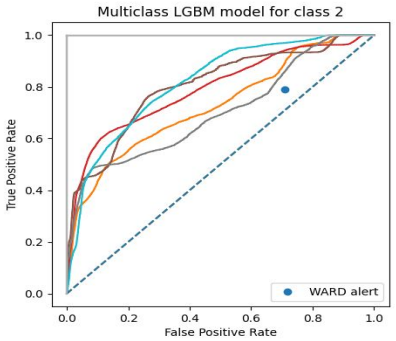
(colors are the different folds)



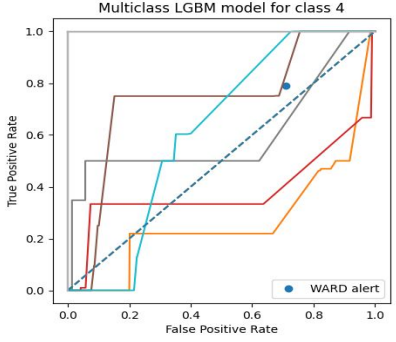


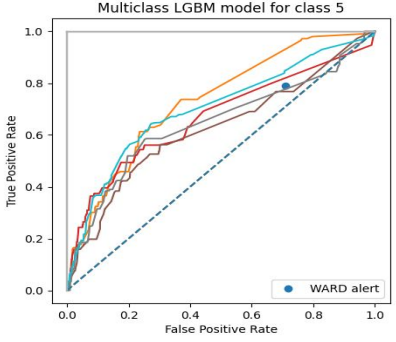
Original data (only pre-processed)
(colors are the different folds)

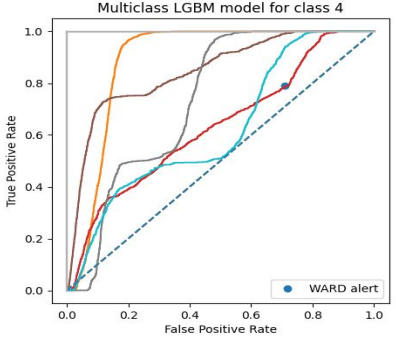


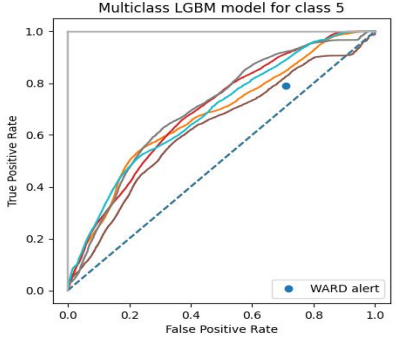


With duplicated SAE intervals
(colors are the different folds)









Performance of duplicating SAE data intervals?

Binary Linear

Only preprocess:

Mean AUC score is 0.6618 ± 0.021

With duplicated SAE data:

Mean AUC score is 0.6668 ± 0.022

Binary TensorFlow RNN

Only Preprocess:

Mean AUC score is 0.5246 ± 0.022

With duplicated SAE data:

Mean AUC score is 0.6073 ± 0.027

Performance of duplicating SAE data intervals?

Multi Class RNN TensorFlow, only preprocess:

Group 1: The mean AUC score is 0.3471 ± 0.081

Group 2: The mean AUC score is 0.5932 ± 0.043

Group 3: The mean AUC score is 0.6124 ± 0.075

Group 4: The mean AUC score is 0.5260 ± 0.12

Group 5: The mean AUC score is 0.5727 ± 0.063

Multi Class RNN TensorFlow, with duplicated data:

Group 1: The mean AUC score is 0.5946 ± 0.069

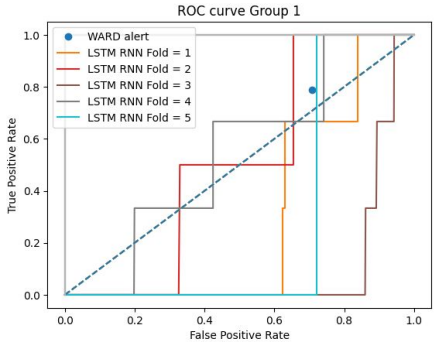
Group 2: The mean AUC score is 0.6495± 0.014

Group 3: The mean AUC score is 0.6876 ± 0.076

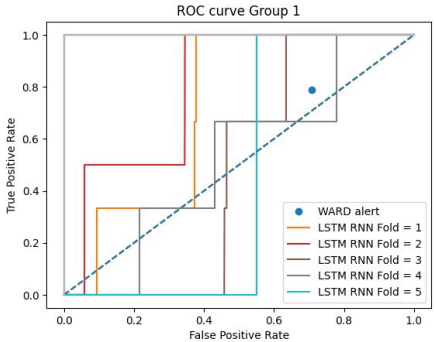
Group 4: The mean AUC score is 0.5280 ± 0.081

Group 5: The mean AUC score is 0.5943 ± 0.040

ROC-curves for multi class classification (RNN LSTM)



The mean AUC score is 0.3471 ± 0.081



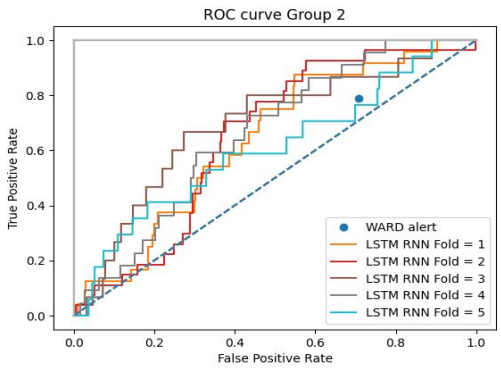
The mean AUC score is 0.5946 ± 0.069

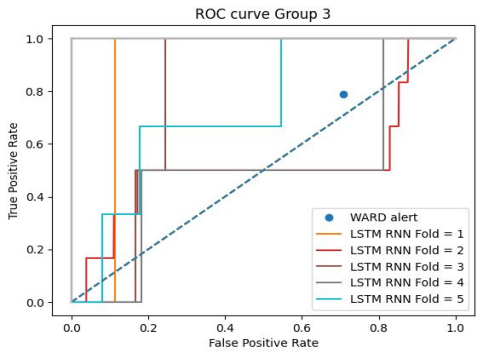
Left: Only preprocessing

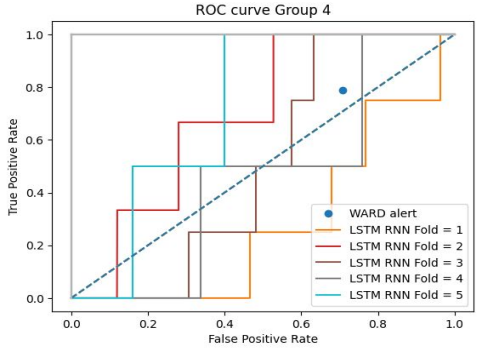
Right: With duplicated event intervals

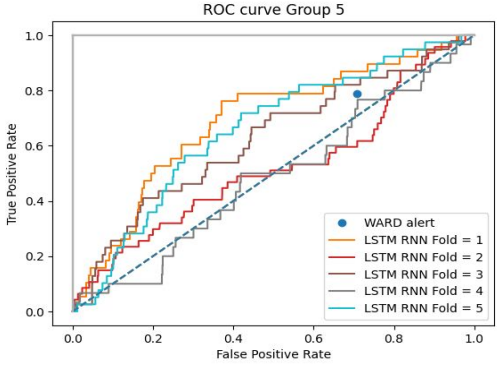
No smooth curves due to the input data being 8-hour intervals.

Visible that the ROC curves with duplicated data skew to the left of the WARD alert point.

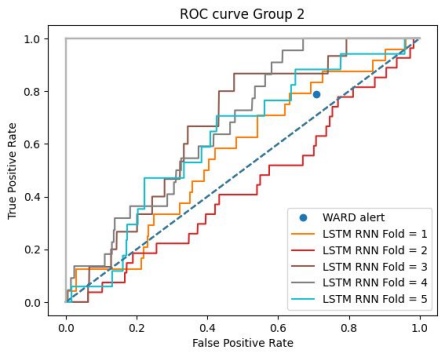


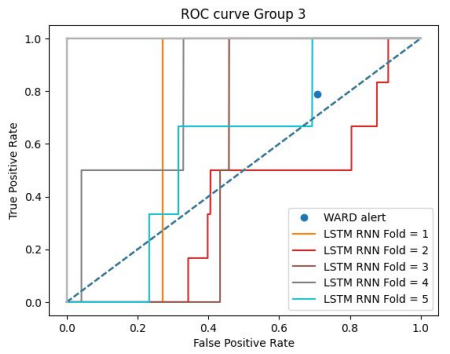


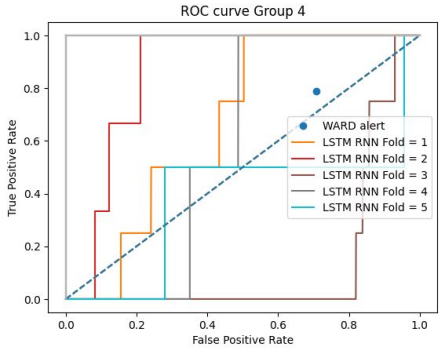


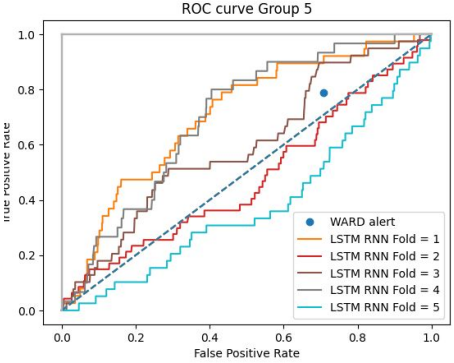


With duplicated data









Only preprocessed

Why the same/little worse performance than just preprocessed data?

For binary (linear and BDT):

We assume since the the linear and BDT models don't struggle with unbalanced original data, then just adding more of the exact same data doesn't help the models.

And therefore the performance is roughly the same with and without the duplicated data.

For multi class classification (BDT):

We *assumed* a more noticeable difference here as there are so few data points in the different groups, making the data extremely unbalanced. So giving the model more cases of a group makes the data less unbalanced, such that the model might actually make predictions.

After duplicating SAE data, the ROC curves became smoother but the uncertainty on the AUC scores are still large.

Why RNN gets better?

For Binary (RNN LSTM)

The RNN AUC went from 0.5246 → 0.6073 by duplicating (10x) the intervals with event in it.

The RNN model seems to be very affected by the unbalanced data → Therefore we see a clear improvement outside of the uncertainty when duplicating the SAE intervals.

For multi Class (RNN LSTM)

In general, as with the BDT, we assumed a more noticeable improvement by doing duplication for the multiclass classification.

We do for some groups see a very noticeable improvement not within the uncertainty.

The reason why some of the groups may have less of an improvement might come from the fact that even 10 doubling the amount is not enough, as some groups have almost no data present.

Appendix: Making Fake Patients

How we define and make fake patients

Motivation and reasoning:

We wanted to make the data more balanced.

We assume people are alike so we can shift their vitals around if they fall in the same group (event group).

```
1: Find patients(in 8-hour interval) with the same event group.

2: Save those time-intervals with the same event group.

3: Run through time-intervals for each group, and then randomly choose other patient to switch 3 vital measurements (can be any three) with.

4: Make column to note that they are fake data points.
```

Performance of making fake patient data?

Binary BDT LightGBM (not HP optimized):

It did more or less equally well with and without fake patients.

No HP-optimization, only preprocess:

Mean AUC score is 0.6957 ± 0.015

No HP-optimization, with fake patients:

Mean AUC score is 0.6879 ± 0.013

Binary BDT LightGBM (HP optimized):

It did more or less equally well with and without fake patients.

With HP-optimization, only preprocess:

Mean AUC score is 0.7206 ± 0.0099

With HP-optimization, with fake patients:

Mean AUC score is 0.7134 ± 0.011

Performance of making fake patient data?

Binary Linear

It did more or less equally well with and without fake patients.

Only preprocess:

Mean AUC score is 0.6618 ± 0.021

With Fake Patients:

Mean AUC score is 0.6624 ± 0.021

Binary TensorFlow RNN

Only Preprocess:

Mean AUC score is 0.5246 ± 0.022

With Fake Patients:

Mean AUC score is 0.5525 ± 0.041

Why the same/little worse performance than just preprocessed data?

Our assumption:

We assume that having the same event group means that people are alike, HOWEVER that is probably not enough granularity as group maybe too broad to make that assumption.

SO, basically we made data that is too far from reality which makes the model not better, despite making the data more balanced.

Why do we not then use types (35 different ones) *instead of groups (5 different ones)?*

Due to time constraints and the way that the rest of our processing of data was made, it would be a bit hard to implement. Furthermore, some types possibly didn't even have more than one data point.

BUT one could try and check!

Appendix: SHAP-Values

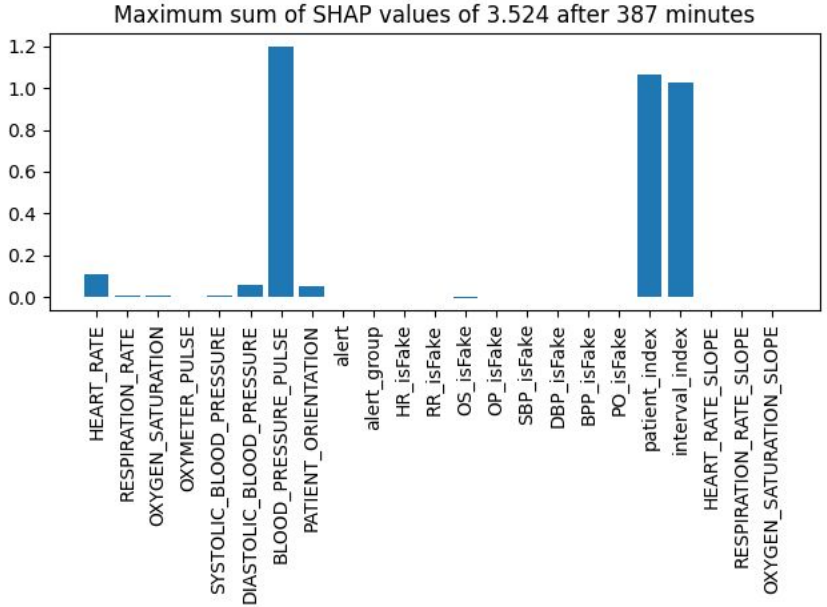
(Finding the feature that predicts SAE)

How we do SHAP-values?

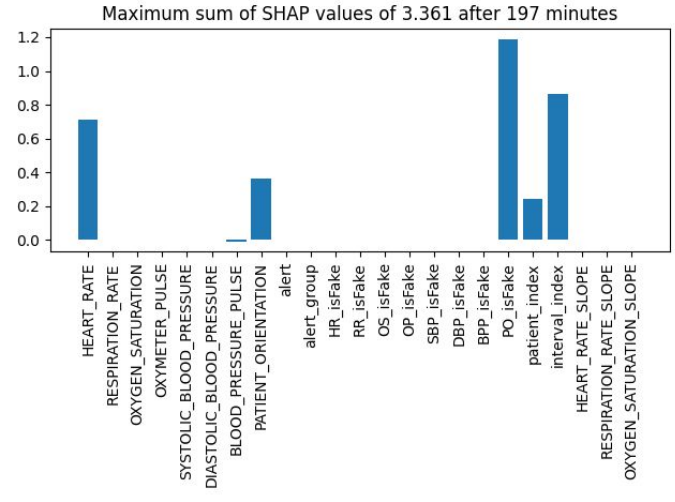
1: Compute SHAP values on test data for our best model (LightGBM model, only preprocess, no data augmentation)

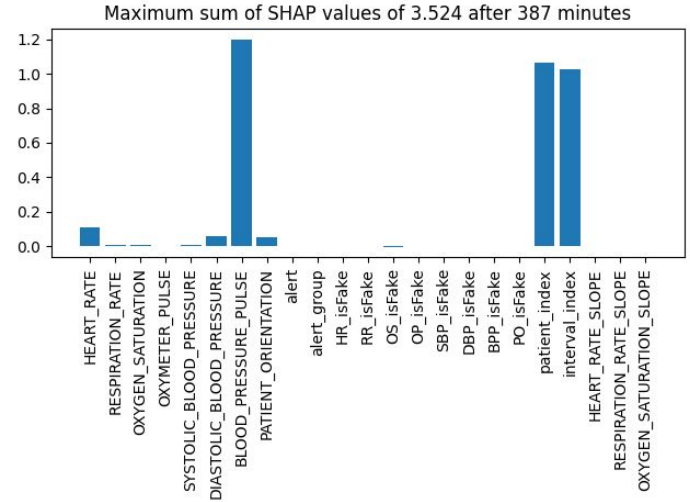
2: Search through 8-hour intervals with predicted SAE and find the highest sum of SHAP-values across features in that interval.

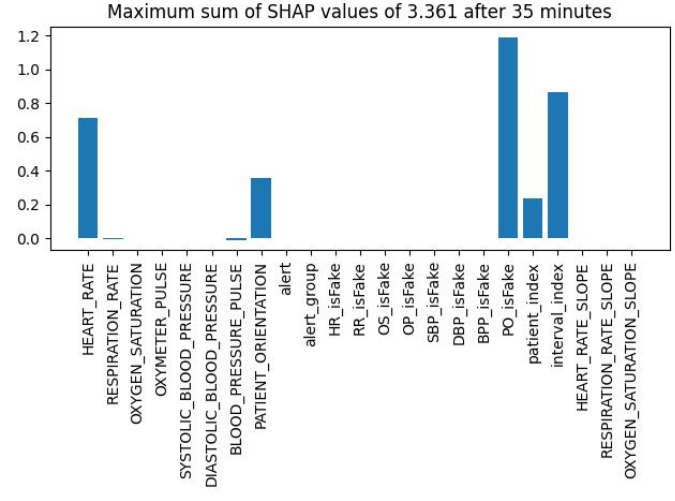
Note that we do NOT use the absolute values of the SHAP values when computing the sum since we want to find the features that made the model predict SAE (large positive SHAP values) and NOT the features that made the model predict no SAE (large negative SHAP values). We also find the maximum sum across features instead of just the feature with the largest positive SHAP value since there might be another feature with a large negative SHAP value for the same data point leaving the model overall to predict no SAE for this datapoint

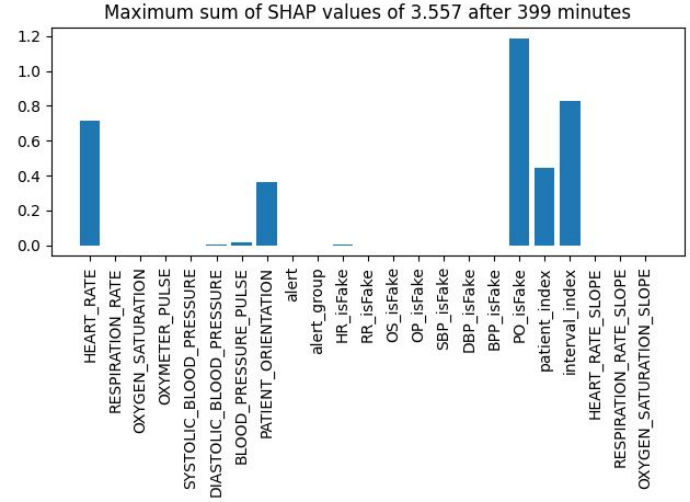


Example:
In this interval the **blood pressure pulse** is the feature that most contributes to the model predicting the SAE!

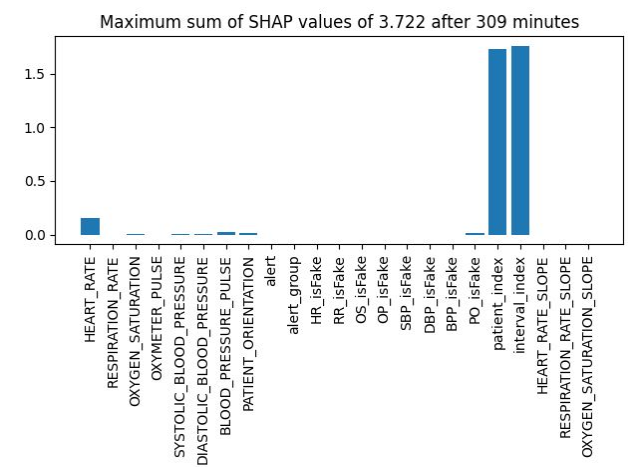


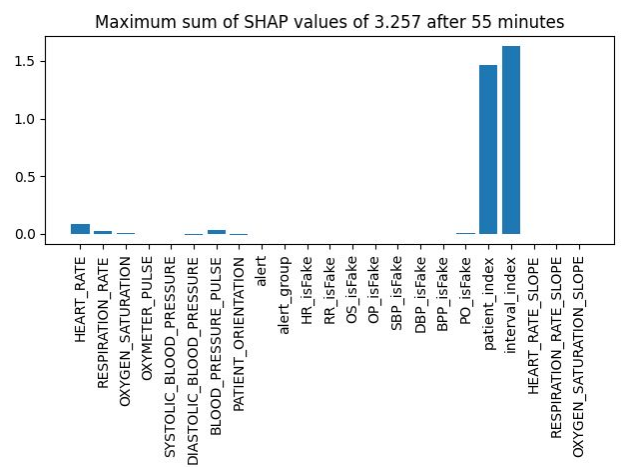


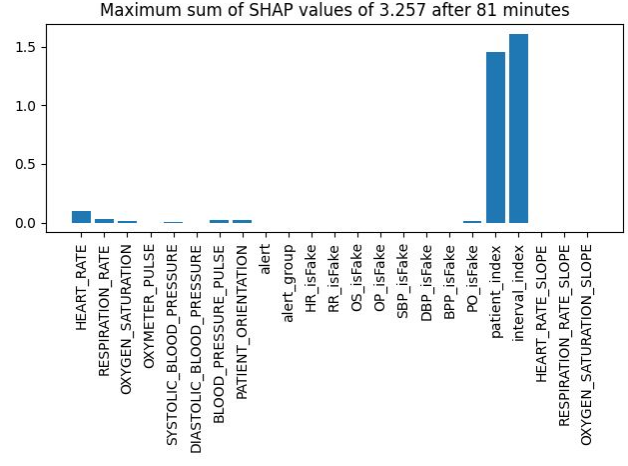


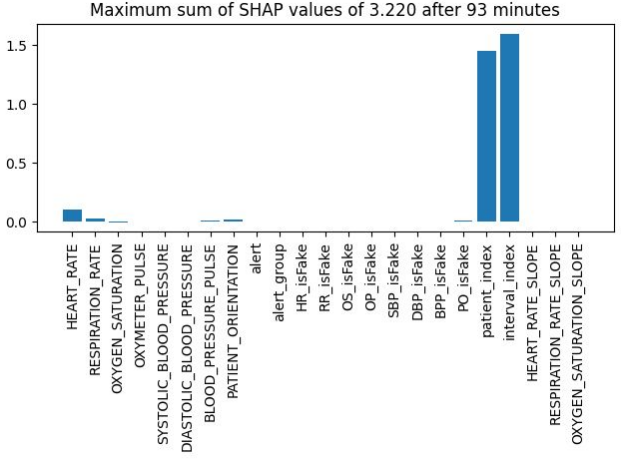


Examples of 8-hour intervals where SAE is predicted and the point (minutes) in that interval where the sum of the features' SHAP is largest.

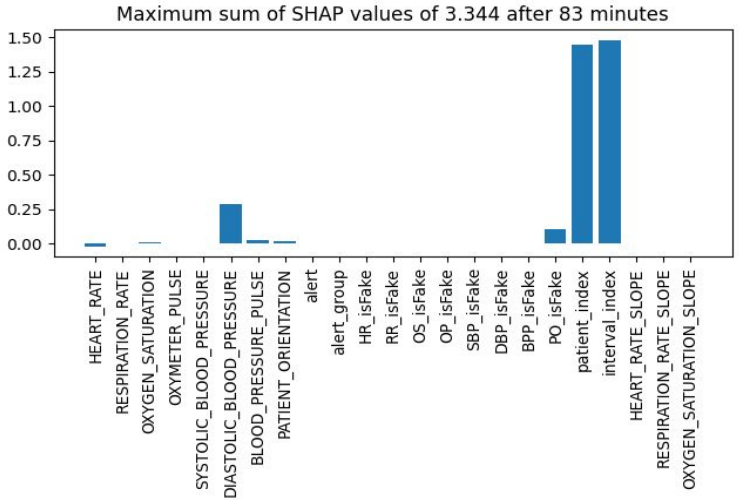


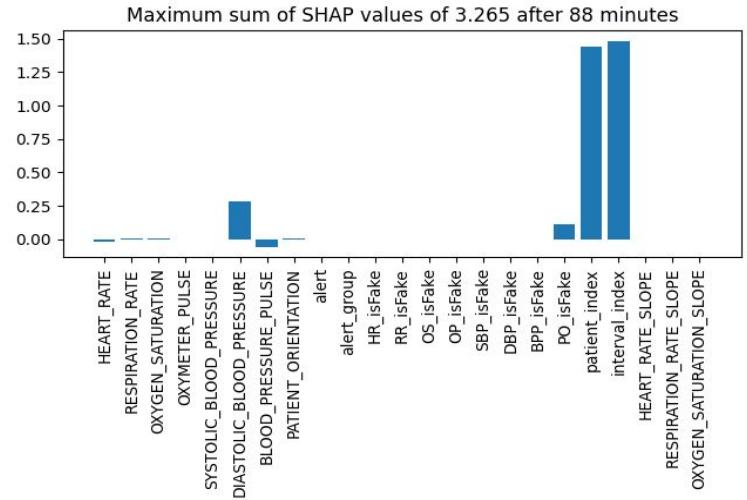


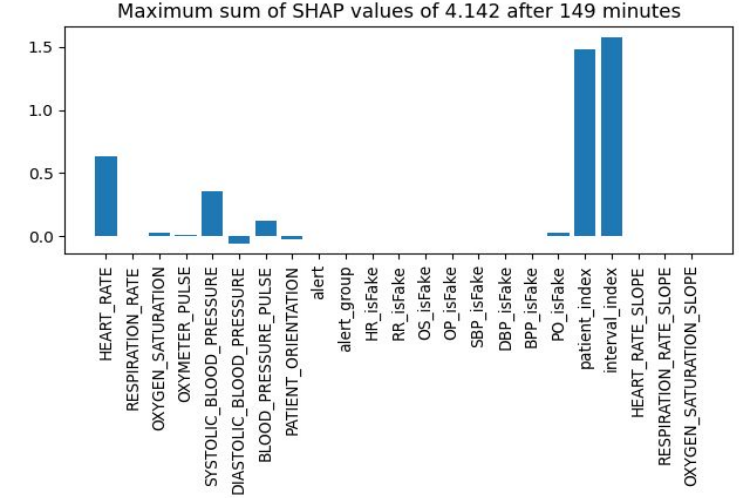


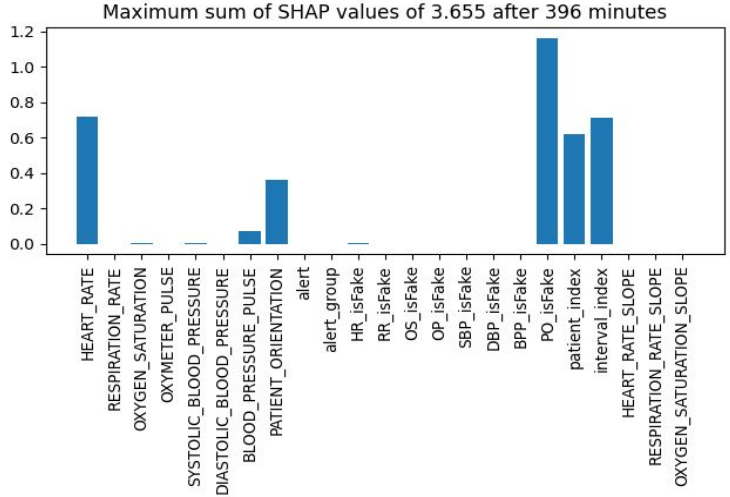


Examples of 8-hour intervals where SAE is predicted and the point (minutes) in that interval where the sum of the features' SHAP is largest.

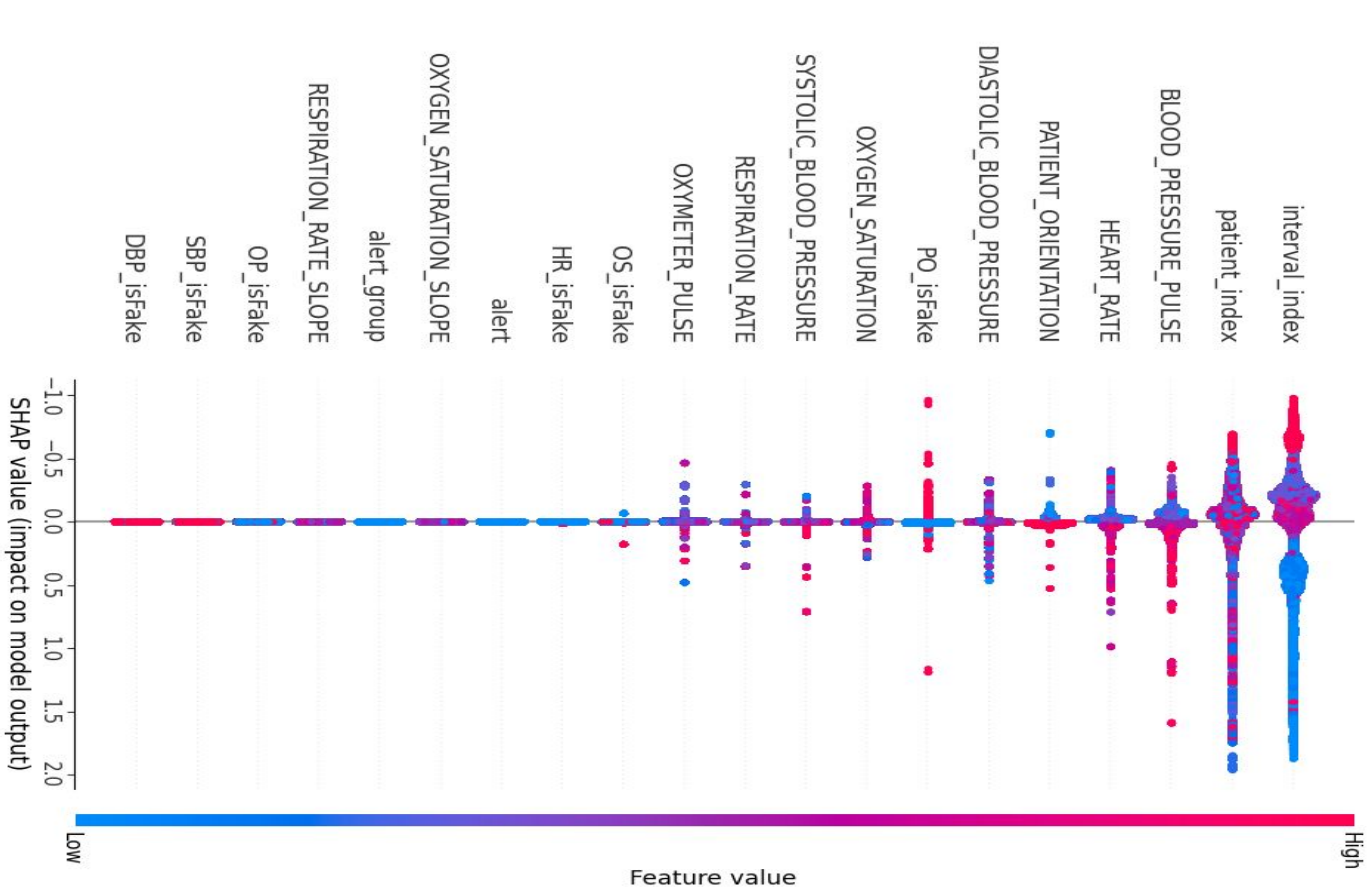








Examples of 8-hour intervals where SAE is predicted and the point (minutes) in that interval where the sum of the features' SHAP is largest.



SHAP Summary Plot (showing what features were generally most important for predictions)

SHAP-values continued

The summary plot and examples show that it is often the “patient_index” and “interval_index” that have the highest SHAP values. This indicates that the LightGBM model has used the temporal information that we tried to put into the features “patient_index” (which patient does the data point belong to) and “interval_index” (which 8 hour interval from this patient does the data point belong to). Hence, it seems like these variables have successfully made the model understand which data points are closer related to each other across patients (which patient) and across time (which interval), and the model uses this information to make predictions. Our interpretation of this is that for example the model might note that if the heart rate is very high for multiple data points for the same patient and interval, then an SAE is more likely to happen!

The patient orientation SHAP is also high in some instances. We argue that it might just correlate a patient lying down with being “sick”. This is makes sense as you are more inclined to be lying down when very sick!

Extra: 4-hour intervals in LightGBM GBDT models

Results of running all GBDT models with 4 hour intervals

1) First try: AUC = 0.6030 ± 0.008

2) With Extra Preprocessing: AUC = 0.589 ± 0.008

3) With "Remove Non SAE Points": AUC = 0.583 ± 0.009

4) With "Repeated Data": AUC = 0.589 ± 0.008

5) With "Fake Patients": AUC = 0.588 +- 0.009

6) With Focal Loss: 0.590 +- 0.008

It is evident that changing the interval length from 8 to 4 hours made the performance of our GBDT models worse. Hence, 8 hour intervals seem to be a more appropriate choice. This might reflect that severe adverse events have been developing for more than 4 hours before being detected.

Had we had more time, it would have been interesting to run the model with for example 16 hour intervals and see what the performance would be.