THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Interpretable Machine Learning for Time Series Data in an ICU Setting

Gareth Booth

Supervised by Dr Sally Shrapnel

Submitted for the degree of Bachelor of Engineering (Honours) for the School of ITEE

November 1, 2020

Prof Amin Abbosh
Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, Q 4072

Dear Professor Abbosh,

In accordance with the requirements of the degree of Bachelor of Engineering in the division of Electrical Engineering, Electrical and Biomedical Engineering, Electrical and Computer Engineering, Software Engineering, Mechatronic Engineering, I present the following thesis entitled "Interpretable Machine Learning for Time Series Data in an ICU Setting". This work was performed under the supervision of Dr Sally Shrapnel.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

Gareth Booth.

Acknowledgments

I would like to acknowledge Sally Shrapnel for the time she spent advising my thesis and it's direction, and for introducing me to machine learning research in the first place. Thanks to Samuel Hinton, Kyle Young and Michael Kewming for creating some of the data processing code.

Abstract

Health emergencies such as the current COVID-19 pandemic put great strain on ICU clinicians. Using machine learning to predict patient outcomes using time series ICU data seems to be a promising way to reduce information overload and help doctors better manage their patients. Using the MIMIC-III database, we create an ICU dataset of 2870 patients and evaluate it on an LSTM model to predict patient mortality, achieving an 87% accuracy. We show that SHAP, a popular interpretability method, is able to create rich global explanations of this model, however is inadequate for local explanations over full time series. Modifications to the SHAP library are proposed to better handle local explanations, and these modifications are evaluated. This new SHAP is shown to provide local accuracy, but is unsuitable for global explanations. For validation, the accuracy and explanations from the LSTM are compared with those of random forests trained on fixed amounts of admission and final day data.

Contents

1 Introduction

Health emergencies such as pandemics cause great strain to the healthcare system and create a shortage of intensive care unit (ICU) beds. During these emergencies, many hospitals gather data from their patients in ICU including vital signs, procedures and diagnoses. This data could be used to inform doctors about effective treatments or provide early warnings. However, there is often an information overload for ICU specialists, considering the number of patients, the time each patient spends in ICU and the number of measurements taken [1].

As machine learning requires a large amount of data, it has great potential in this field. Recently, there have been a large number of studies released in this area. Some of the problems that these studies focus on include predicting clinical events such as death, myocardial infarction or discharge [1].

There are a few factors that make this a challenging domain. While there is a relatively large amount of open access data available compared to other fields, it can be challenging to process. Also, the time series nature of ICU data limits the choice of machine learning models substantially.

Few studies incorporate explainability techniques. Machine learning models normally give a single output, which is a prediction for the given data. This does not given any explanation as to why that prediction might be true. Explainability (or interpretability) techniques allow seeing which features are important for a prediction, and can help give users of the model confidence [2].

One use of explainability is to recognize biases. Machine learning models are capable of learning very complex relationships, and if a dataset contains a bias, the model is likely to also learn this bias. Searching the dataset manually for biases is not feasible, due to the amount of data and the presence of unknown biases [2]. Explainability techniques can reveal these biases by highlighting spurious relationships, which domain experts can identify.

The importance of ICU data has been highlighted in the ongoing COVID-19 pandemic. One possible application of this work is to perform transfer learning to a new dataset with data from COVID-19 patients. Interpretability techniques can highlight relationships between features, which could potentially reveal novel insights which could be used by doctors. Another possible application of robust explainability techniques for time series data is to sanity check existing models. This can be possible for some model agnostic techniques.

The key question of this thesis is if it's possible to predict death using time series ICU data in an interpretable way.

2 Background

2.1 Machine Learning on ICU Data

The first difficulty arising from using ICU data is the nature of the time series data. Time series data is 3 dimensional. Specifically, an ICU dataset will have many patients, each with many features (e.g. vital signs) over many different points in time (e.g. every hour). Time series machine learning models should be able to handle any length of time series. This means the model should take inputs of the shape (x, y, z) where x and y are any length, and z is the number of features which is fixed. The model will output a list of predictions, one predictions for each x.

One of the problems that we can ask using this time series data is given a patients data, predict if this patient was discharged or died, where discharged means discharged from ICU alive. However, this problem means that the machine learning model could be looking at the final point of data to make a prediction, i.e. the data just before the patient died. Care needs to be taken to ensure that the model is not only using this data. Another related problem is of predicting 90-day mortality, which is an easier problem [3].

A machine learning model to predict ICU death could be useful in practice to find out if a patient is going to die soon. If the model predicts that the patient will die given the current data, then extra care can be given to them. If the model predicts that the patient will be discharged soon, then other patients can be prioritized.

Another way of using ICU data is to make predictions using a fixed amount of data, e.g. only admission data. Data from the first n days can be used, but n must be known before the model has been created. This problem is suitable for traditional machine learning models. These models can handle 2 dimensional data, i.e. many data points with many features. These models will output 1 prediction for each data point (where a data point is a single set of features). Similarly, data from the last n days of a patients ICU stay can be used instead. However, due to these models using less data to make predictions, it is reasonable to expect these models do not have as much predictive power as a time series model. [1, 3] find that models benefit

from access to longer time series data.

One of the most widely used ICU datasets is MIMIC-III, a dataset of deidentified data from 53,423 distinct hospital admissions [4]. Due to its size and public availability, it has been widely used for benchmarking machine learning tasks. The data includes bio-markers, drug administrations, procedures and diagnoses (using International Classification of Diseases).

Most studies using time series data use a fixed temporal resolution, for example 1 day in [1] and 1 hour in [3]. However, some techniques exist to handle continuous data [5].

2.2 LSTMs

Recurrent neural networks (RNNs) are neural networks with a feedback loop [6]. An RNN consists of a neural network that takes and outputs some data, but also takes some state from the previous application of the neural network as input. This means that an RNN is able to make predictions based on what it's seen before, and is also able to handle an arbitrary length of input sequence.

Long short-term memory (LSTM) is an RNN architecture that achieves greater performance than a traditional RNN. It allows models to learn long-term dependencies, i.e. information from many inputs ago [6]. LSTMs take two inputs from the previous application, one is called the "cell state" and the other is the previous output vector. This extra cell state is updated each application by removing unimportant data and adding new relevant data from the current input.

The below figure shows an example of an LSTM network. Here, the network has been unravelled for demonstration, but the different inputs are applied to exactly the same model. The top 'line' represents the cell state, which is modified by the original input and the bottom 'line' is the output from the model at that time step. It is important to note that the size of the vectors passed on through the model is a hyper-parameter of this model.

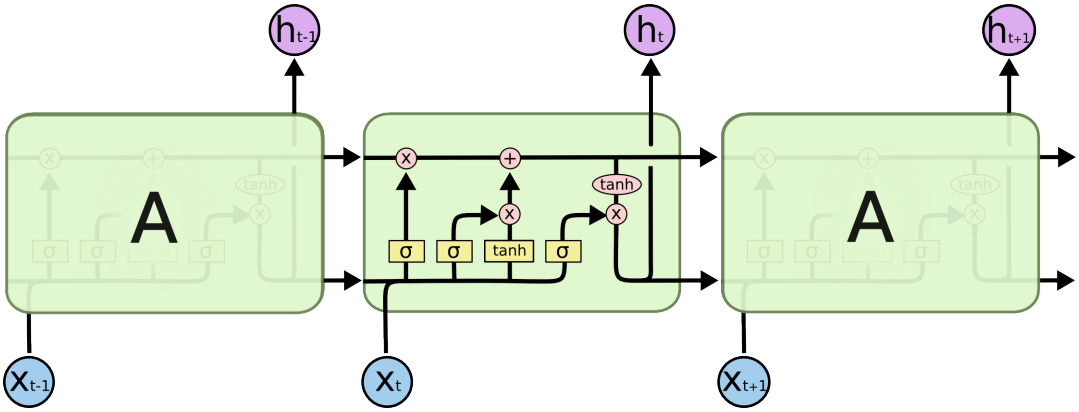


Figure 1: Example LSTM network [6]

An example of an LSTM used with ICU data to predict mortality is in [3]. They use a vanilla LSTM with hyperparameter training. Their model achieves an AUCROC of 0.88, however their dataset is not publicly available and uses a very high temporal resolution of 1 hour.

2.3 Random Forests

There are many machine learning models that handle a fixed amount of input data. LSTMs can be made to handle fixed amounts of data by limiting the length of input sequences when defining the model. Random forests are a popular type of model. Random forests (RF) and gradient boosted trees achieve state of the art performance in many domains, particularly when features are individually meaningful and in tabular form [7]. Random Forests involve ensembling many decision trees and getting a single aggregated result. Gradient boosting methods are slightly different in the way they're constructed.

Mortality is predicted in patients with unplanned extubation in [8]. The paper compares the performance of four different machine learning architectures, including neural networks, logistic regression models and random forests. The study concluded that random forests were the most suitable for this task, having the highest AUROC, precision and recall. However, the dataset only contains data from 341 patients in this study, and cannot be used as a benchmark as it does not use the MIMIC-III database.

2.4 SHAP

SHAP is a model agnostic, local interpretability method [9]. It is a unification of six existing methods, and outputs the importance value for each feature based on the contribution that feature makes to the prediction. It is based on Shapley values from co-operative game theory, and has a strong theoretical justification [2, 3]. The SHAP value for a feature is it's compound effect when interacting with all other features. A positive value means the feature moves the prediction in the positive direction. For example, if the model outputs 1 for death, then a positive value represents an increased risk of death [10].

SHAP is used in [3] to tell if a given feature increases or decreases the probability of mortality. It is used to show that, for example, the presence of a small age in a patient greatly decreases risk, while a higher age increases it.

As an example application of SHAP, consider a model predicting mortality trained on an ICU database [10]. The below figure shows the effect of systolic blood pressure. Each dot represents one point of data input to the model. For each point, a SHAP value is calculated (y-axis). This shows that SHAP is able to find complex relationships, such as how lower systolic blood pressure generally decreases the patients risk of death, however a sufficiently low value will again increase the patients risk. Also, SHAP shows how age affects systolic blood pressure, which can be seen by the colour of the points.

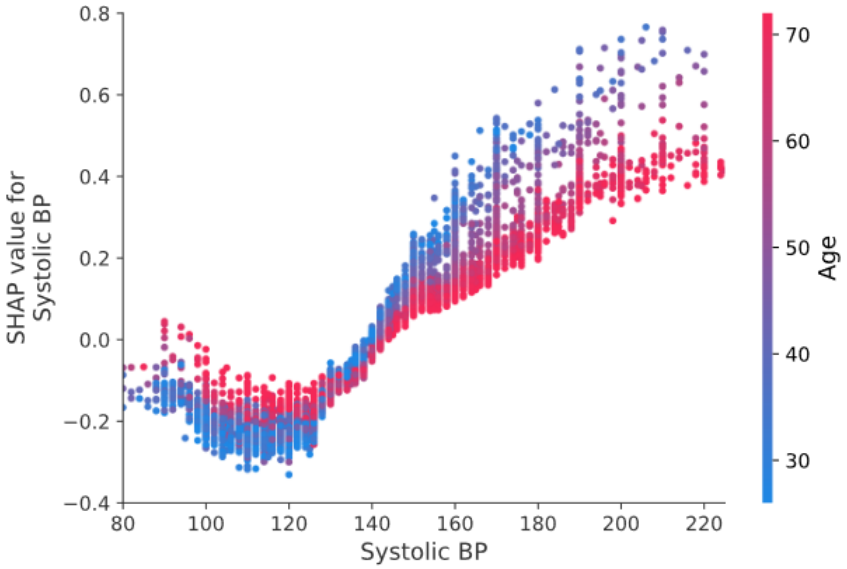


Figure 2: Example SHAP application from [10]

The local explanations from SHAP can be combined to make global explanations [7], as the above figure shows from plotting all local explanations. Further global explanations can be found by taking the mean absolute SHAP value of all features, which allows the comparison of model sensitivity between all features [11].

An enhanced version of SHAP for random forests is introduced in [7] and [10]. This includes the ability to calculate interaction values between features. This can allow quantifying which features interact the most. As an example, the above figure shows that age and systolic blood pressure interact strongly.

Unfortunately, there is no support in the framework for time series data. Using vanilla SHAP in a time series data setting involves flattening the 3 dimensional time series data to 2 dimensions by removing the time dimension.

2.5 Project Overview

This investigation has been broken down into three parts.

Firstly, a time-series ICU dataset for this project must be created. This will involve processing the freely available MIMIC-III database, and careful feature selection in order to provide sufficient prediction power and enough data points for machine learning.

Next, two different architectures of machine learning models will be trained on this dataset. This will include an LSTM model trained on time series data, and random forests trained on admission and final data. The results from these models will be evaluated and compared.

Finally, interpretability methods will be investigated. Vanilla SHAP on the LSTM will be investigated, and a new modified version of SHAP designed for time series data will be proposed. These explanations will be compared to those of the random forest when using TreeExplainer.

3 Data Processing

The first state of the project is to investigate the available data and create a suitable dataset.

The use of daily data was chosen early in the project. The initial plan as outlined in the project proposal was to create a dataset from MIMIC-III that resembles a COVID-19 dataset which used daily data. However, insufficient data was obtained for this new dataset and this line of investigation was cancelled.

The Python programming language was chosen due to it being the de facto standard for machine learning. Pandas [12] was chosen to manipulate the data due to previous experience with the framework, and matplotlib [13] was used to visualize the results for the same reason.

3.1 MIMIC-III

MIMIC-III consists of data from two different ICU database systems, CareVue and Metavision. There are 57,272 unique hospital admissions. The raw data is a series of csv files of database tables, as is documented in [4]. The tables of interest for this study include CHARTEVENTS and D_ITEMS.

CHARTEVENTS contains a list of events, also called charted data or measurements. In this table, the key columns are HADM_ID, which is a unique ID given to hospital patients for a single admission. CHARTTIME gives the time of the event, ITEMID gives the ID of the event and VALUE gives the value of the event. In total, there are 330,712,483 rows in the chartevents table.

D_ITEMS contains the definition of all ITEMIDs in the CHARTEVENTS table. There are 6463 unique items, i.e. 6463 different types of measurements. The below table shows 12 of the most common items in the CHARTEVENTS table, with the Item Name from the D_ITEMS table.

Note the two different “Heart Rate” features and oxygen saturation. These are from the different ICU databases (the lower ITEMIDs are always from CareVue).

ITEMID	Count	Item Name
211	5180809	Heart Rate
742	3464326	calprevflg
646	3418917	SpO2
618	3386719	Respiratory Rate
212	3303151	Heart Rhythm
161	3236350	Ectopy Type
128	3216866	Code Status
550	3205052	Precautions
1125	2955851	Service Type
220045	2762225	Heart Rate
220210	2737105	Respiratory Rate
220277	2671816	O2 saturation pulseoxymetry

Table 1: Most Common Items

The below figure shows the occurrences for the 300 most commonly appearing items.

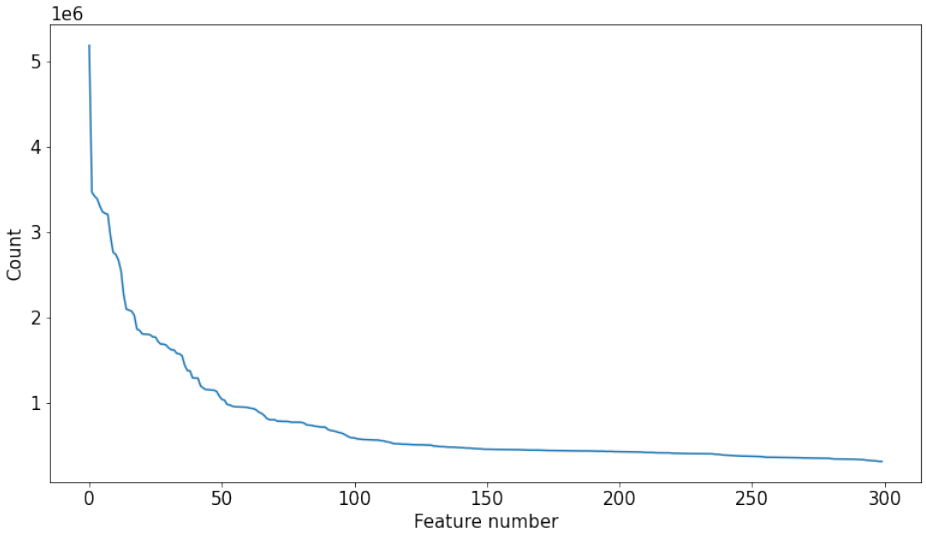


Figure 3: Number of Data Points for Most Common Items

While there is a lot of data available in MIMIC-III, unfortunately a significant amount of it cannot be used in this study. This is due to the nature of the problem

where we are using daily data, meaning only 1 record can remain for each feature for each patient for each day. All data points from the same day for the same feature for the same patient are dropped except for the first element found in the dataset. Using only 1 point per day has a significant effect on the amount of available data, and also introduces the problem of missing data.

The below table shows how the count is affected. Counts now represent the number of measurements when there can only be 1 measurement per day per patient. The number of measurements for the top item has decreased by an order of magnitude. Also, all Metavision items are no longer present in the most common list, in fact there are no Metavision items in the top 90 most common elements. This suggests that the Metavision data is of a higher temporal resolution, and that a higher temporal dataset may be possible using this subset of MIMIC-III.

ITEMID	Count	Item Name
211	241092	Heart Rate
31	236115	Activity
80	230807	Bowel Sounds
212	157814	Heart Rhythm
742	157746	calprevflg
161	157608	Ectopy Type
618	157423	Respiratory Rate
646	157363	SpO2
432	157020	Level of Conscious
617	156824	Respiratory Pattern

Table 2: Most Common Items (Once per Day per Patient)

The below figure is analogous to Figure 3 except for the new data restriction. Clearly, there is much less data available for this kind of problem in general, which is a challenge in the ICU data setting.

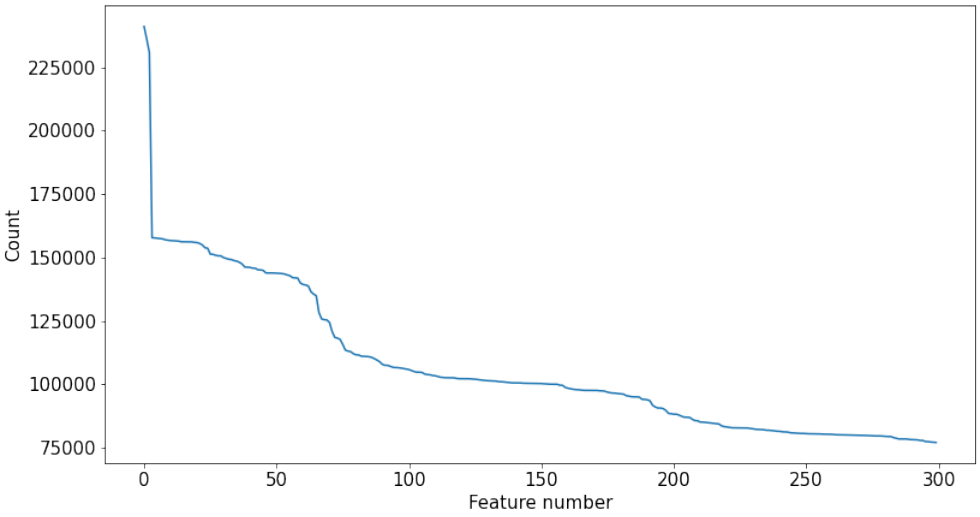


Figure 4: Number of Data Points (Once Per Day per Patient)

3.2 Feature Selection and Missingness

In order to create a usable dataset, features must be judiciously chosen. There is a trade off between getting enough relevant features and getting enough data, as too many features will mean too many features are empty. This can be demonstrated by the figure below.

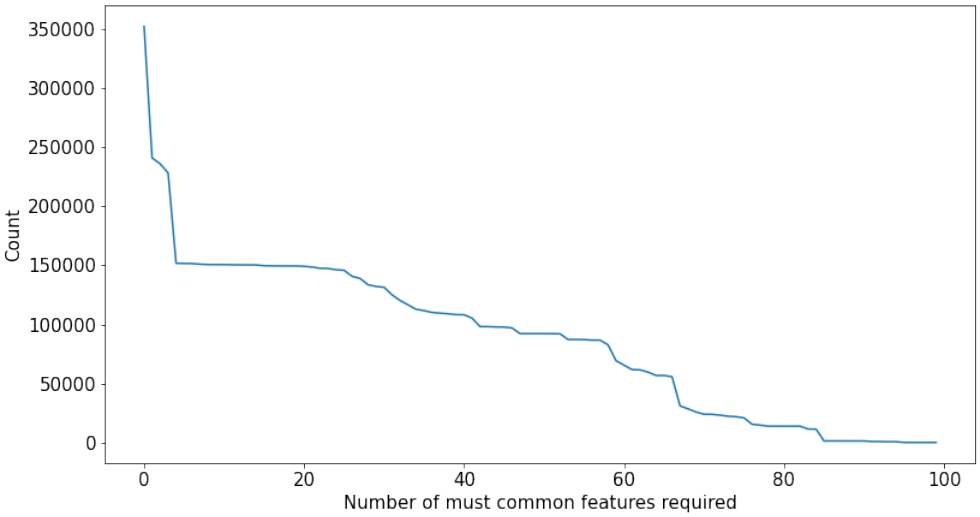


Figure 5: Days of Full Data

The y-axis represents the number of full days of data, and the x-axis is the number of the most commonly appearing features used. For example, the point (3, 228383) means that if you want to make a dataset using the 3 most common features (see Table 2 to see these features), you will have 228383 days of data. Patient days with any missing data features are not included. The point (0, 352325) shows that there are 352325 different combinations of days and patients in the database.

While this may look like a large amount of data, a lot of these features are not useful or not documented. For example, there is a "Temperature F" and "Temperature C (calc)" field. Also, units of measurement in CareVue are not always specified. There are three different items that all represent oxygen saturation in the top 26 most common ITEMIDs.

In order to maximize the number of records obtained, it is important to use both databases and all duplicated fields. This involves finding, for a single feature, all ITEMIDs that represent that feature from both databases. This was done for a series of features that were thought to be of clinical significance and features that contain a significant number of records, e.g. some of those in the above tables. This list of features can be found in the file 'variables.csv' in the repository (See Appendix A).

After this, a subset of these features needed to be chosen that satisfied a few criteria. After dropping all days with missing data, there had to be enough records overall to perform machine learning, which in general is at least 5000 data points [14]. Also, there needed to be a high ratio of patient mortality in this subset to avoid a class imbalance problem, which always provides a challenge for machine learning. Finally, there needed to be enough data on the day of the outcome.

Demographic features such as the patients weight and height are only recorded once during the hospital stay, and so care must be taken to propagate them to all rows for each patient.

Feature selection involved choosing features that resulted in a significant amount of records remaining from this initial subset of clinically relevant features. The final numeric features chosen are as follows: Creatinine, HCO3, Heart rate, haemoglobin, platelet count, potassium, respiratory rate, sodium, weight, SaO2.

More important features for performing machine learning are included. HADM_ID is a hospital admission ID, which is needed to uniquely identify sequences and is used as a patient ID. Each day of data also has a CHARTDATE, and a DAYS_TO_DISCH, and the day relative to the start of their ICU stay. The patient's age, gender and a boolean flag for death is also added.

As well as the above features, comorbidities can also be strong predictors of mortality. The following commodities were found in MIMIC from looking at ICD-9 codes in the DIAGNOSES_ICD table: Chronic kidney disease, obesity, heart disease, asthma. The corresponding ICD-9 codes can be found in the 'get_all_diagnoses' function in 'process_mimic.py' (See Appendix A).

The next step of data processing is ensuring that, for each hospital admission, the "final day" of data is complete, i.e. the day before discharge or death. However, the other days in a patients stay are not required. For example, a patient who has a stay of two weeks can have any days missing except for the final day. This requirement was done because of how critical the final days data is to this project. However it comes at a cost, as this single step decreases the number of hospital admissions in the data set from 21443 to 2935. Combining this with requiring a complete sequence removes too much data, and so isn't done.

Finally, outliers are removed by removing all rows with data points having a z-score > 4. This is done to remove very extreme outliers, which were most likely problems with data entry or incorrect units of measurement.

After processing, this data is saved in a flat file csv.

3.3 Final Dataset

After the data processing, there are 2870 unique patient admissions. This represents a total of 15097 days of complete data, with 21 features.

The below figure shows the length of each patients' ICU stays. Clearly, most of the data in the dataset is from single day hospital admissions. Also, is it clear that the distribution between patients who died and patients who survived is different, and patients who died had a longer hospital stay on average (12.1 days compared to 9.3 days).

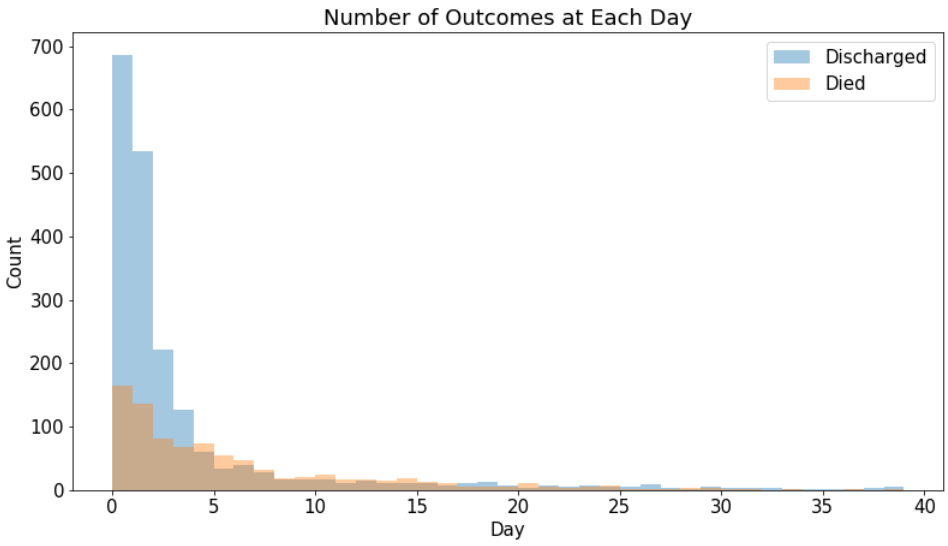


Figure 6: Number of Outcomes at Each Day

The below figure shows the number of data points at each day. Day 1 has more data available than day 0 because many patients do not have complete admission data available. For the purposes of this study, admission data is data from the first day in the sequence. This figure shows that there are a significant number of data points at later days, for example there are 431 patients with an ICU stay longer than or exactly 10 days.

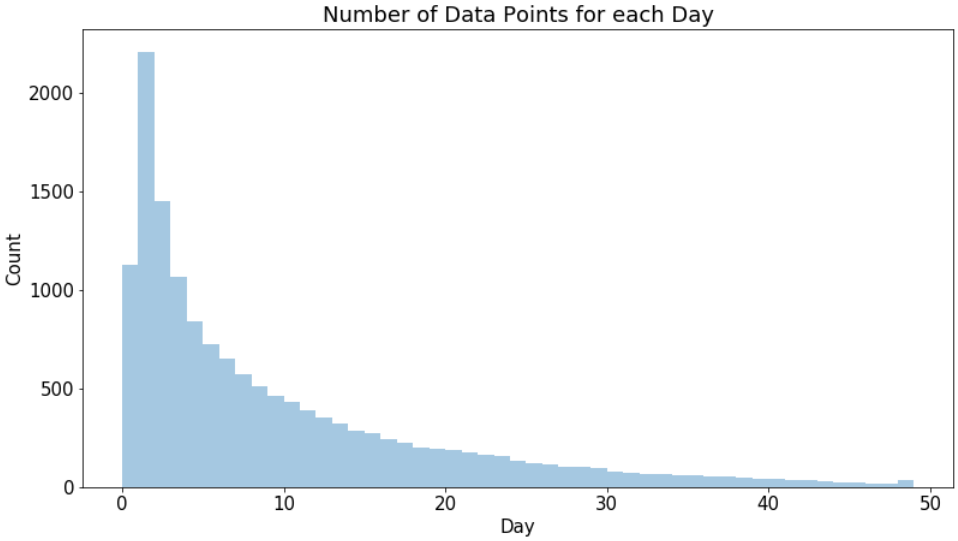


Figure 7: Number of Data Points for each Day

The imbalance between discharge and death is large. Out of all 2870 patients, 1977 were discharged alive while 893 died. Discharged patients represent 68.8% of the population. Therefore, an accuracy of 68.8% could be achieved through guessing discharge. This number will form a minimum baseline for prediction accuracy. Note that the mortality in this subset is much higher than the in-hospital mortality rate of 11.5% in the MIMIC-III study [4].

Below is a correlation matrix, showing all features used for prediction, as well as "DEATH", which is the target feature. Some obvious correlations are highlighted, such as between obesity and weight, gender and weight, and some medically significant relationships such as chronic kidney disease and creatinine, sodium and HCO3 and others such as age and death.

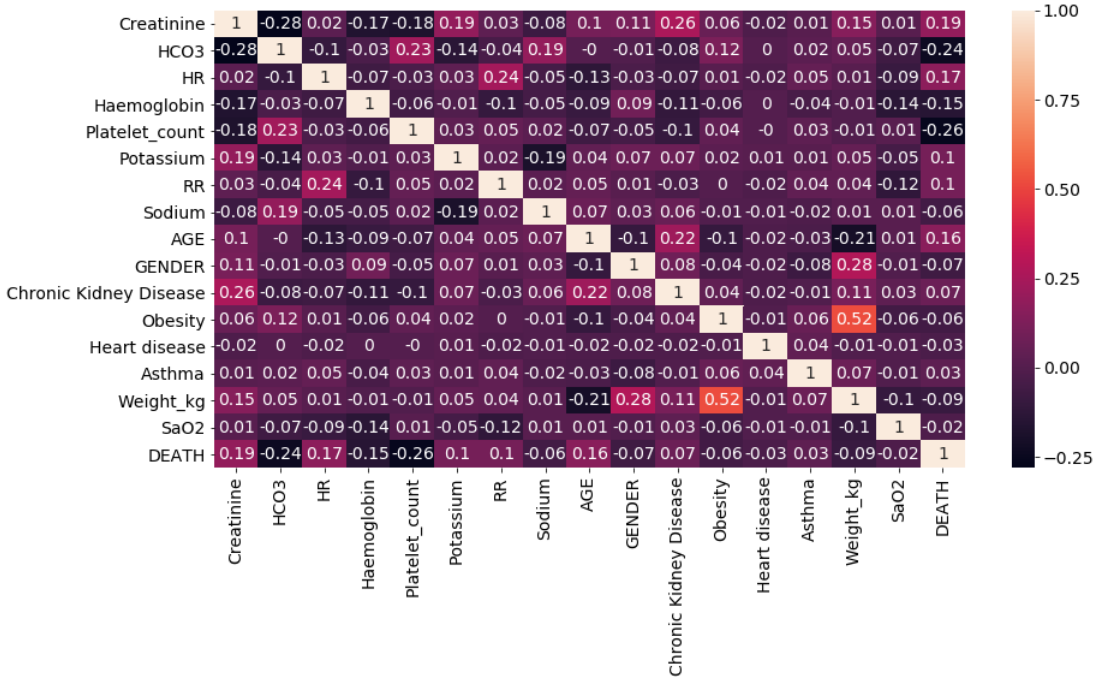


Figure 8: Correlation Matrix

An example histogram of Heart rate is shown below. The distribution of heart rate between death and discharge appear to be different. Histograms for all features can be found in the notebook 'MIMIC death.ipynb' (See Appendix A).

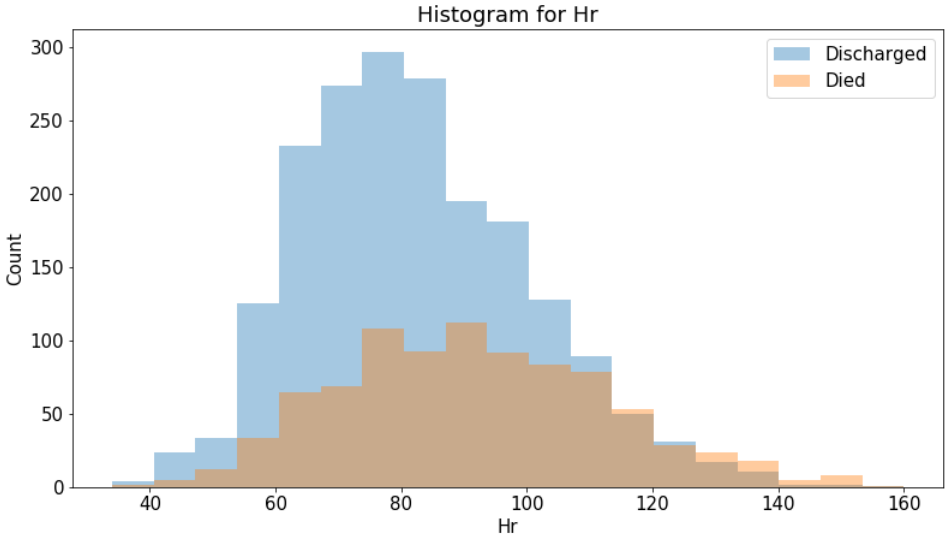


Figure 9: Histogram for Heart Rate

Code for this section is in the repository under the 'mimic' folder which contains the python file 'process_mimic.py'. The dataset is created in the notebook 'MIMIC dataset.ipynb'. (See Appendix A)

4 Classifier

Two different types of machine learning models are used. LSTMs are used because they operate on time series data, which is the key focus of this investigation. LSTMs will be trained on the time series data and evaluated.

However, random forests will also be used, even though they do not operate on time series data. They are used instead to validate the LSTM. Care must be taken to ensure the LSTM is actually exploiting the time series nature of the data, and not just using the final day of data to make a prediction (i.e. the day of discharge or death). To help test if this is the case, a random forest can be trained on the last day of data. If the accuracy of the random forest is worse than the LSTM, then it is reasonable to conclude that the LSTM is using more than the last day of data since random forests represent the state of the art for such classification.

Random forests will also be trained on admission data as this is a popular problem in this space, and a large difference in accuracy between admission only data and LSTMs helps motivate the use of time series data.

Also, random forests with varying amounts of data will be trained. For example, a random forest will be trained with access to the first N days. This will be done to investigate an alternative problem where a fixed amount of data is known, in which case an LSTM may not be required.

4.1 Model Building & Results

Firstly, for model training and evaluation, a 20% split test and train set was created. The training set contains 2296 admissions and 12310 days of data, while the test set has the remaining 574 admissions and 2787 days. This same dataset split will be used for both different types of models.

4.1.1 Random Forest

XGBoost is chosen as the random forest framework [15]. This is a popular gradient boosted method.

The first random forest experiment is to use admission data. Various lengths of admission data is used. For example, 1 day admission means only the information from the first day is used, 2 day admission means data from the first two days is used, etc. Note that for n day admission data, ICU stays that are less than n in length cannot be used. Therefore, increasing the number of days will decrease the data available. The below figure shows this.

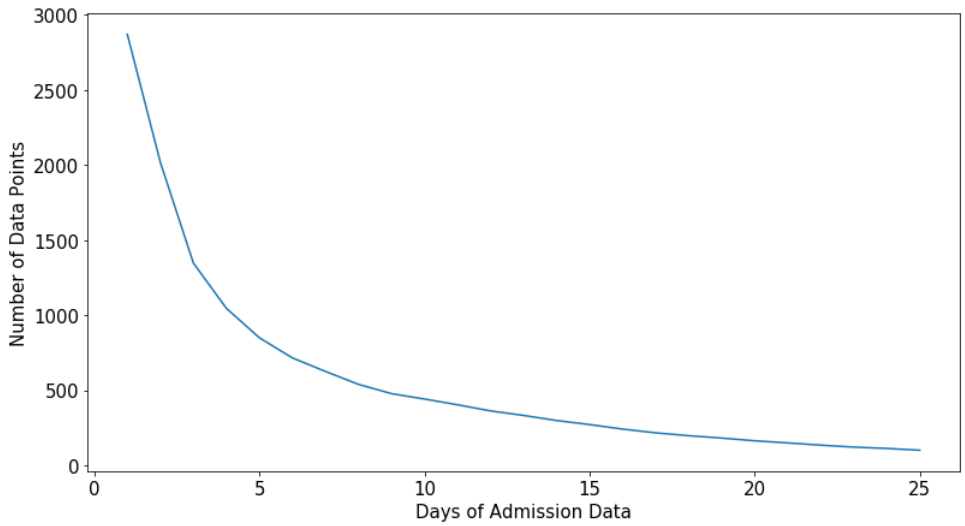


Figure 10: Random Forest Admission Data

Hyper-parameter grid search was done using scikit-learn's GridSearchCV function [16]. The parameter ranges were: max_depth from 3 to 10, alpha from 1 to 19, subsample from 0.1 to 1, colsample_bytree from 0.1 to 1. The optimal values for these parameters are: max_depth of 8, alpha of 9, subsample of 1, colsample_bytree of 0.4. Max_depth was later changed to 5 to reduce overfitting. Other parameters chosen are: learning_rate of 0.05 and n_estimators of 1000. This grid search was done on 2 days of admission data. Class weighting is used.

For each different number of days of data, the model is recreated and trained. This must be done since the random forest cannot take a varying amounts of data. Also, a 20% holdout test set is created for each different length and the model is evaluated on this. The 'discharged' and 'death' lines in the below figure show the percentage of patients who were discharged alive and the percentage of patients who died in the dataset. The maximum of these two lines represent the baseline accuracy for the model.

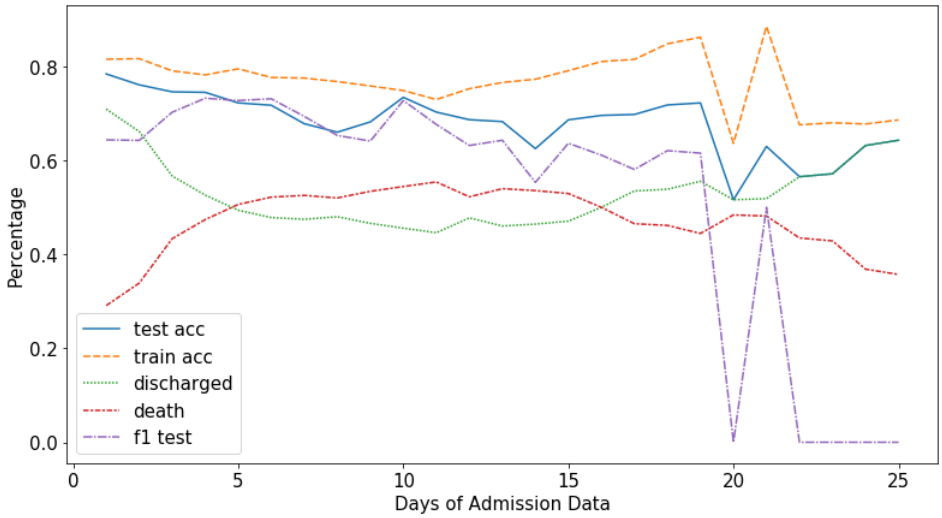


Figure 11: Random Forest Admission

The above figure shows the random forests performance as the number of days of admission data increases.

The f1 score for the death class on the test set is also featured in the above figure. The f1 score shows how a model performs on an individual class, and is calculated using the precision and recall of that class. The f1 score for death is very low for the first few days of data when compared to the accuracy, suggesting the model is much better at predicting if the patient discharged successfully. However, as the death and discharge percentage approach 50%, the f1 score begins to converge on the test accuracy, meaning both classes performed equally.

The random forests are able to generalize well while there is a large amount of data, such as in the first 5 days (see Figures 10 and 11). However, the model does not generalize well past a certain point. For example, with 1 week of data the test accuracy becomes much lower than the train accuracy. When there is 20 days of admission data, the f1 score for the test set becomes zero (same for the train set). This means the model is always predicting discharge, since it is the most prevalent class. This is due to the small number of records available for these long sequences, which causes the random forests to be unable to learn meaningful relationships.

The next experiment tests a similar problem, where instead of admission data you are given data from the days before the outcome. The experiment is repeated exactly as before, except the x-axis is now the number of days of data before the

final outcome.

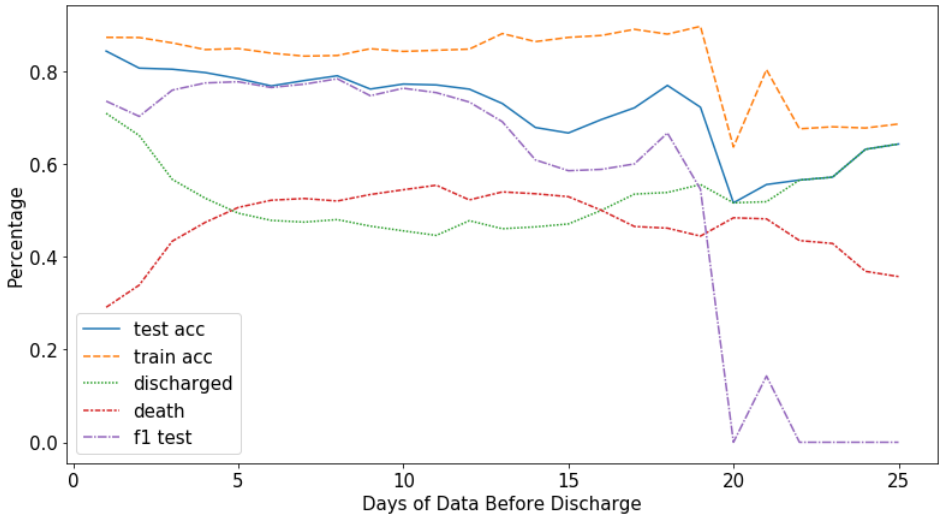


Figure 12: Random Forest Discharge

Note that these random forest models attain significantly higher accuracy when compared to the admission models. For example, the 1 day admission data model accuracy is 78%, while the 1 day discharge data is 84%.

Code for these experiments can be found in the 'MIMIC death RF' notebook.

4.1.2 LSTM

Keras with a Tensorflow backend is chosen as the framework to create this model [17]. Keras is uesd due to its ease of use and contains a built in implementation of an LSTM. The keras functional API allows creating a complete machine learning model by building up layers of functionality [17]. This API can also accommodate multiple inputs and outputs in a single model.

The base functionality of the LSTM model used in this study is very simple. First, it consists of an input layer. In order for the model to be able to handle arbitrary length time series, the input shape should be (?, ?, num_features), where ? is an unspecified length and num_features is the number of features in the data, which is 16 in this case (when using the notation from Section 2.1). Next, a masking layer is added which masks out time steps where all features are zero. This is done so

3D numpy arrays arrays can be used, since 3d arrays require all dimensions to be known when they are created so all time series must be padded to the same length. It was verified that no actual data involved all features being zero before this step.

An LSTM layer is added next. This layer takes a 3D input from the previous layer and outputs a 2D output. It outputs a vector of length n for each sequence, where n is a hyper parameter of this model. This is connected to a hidden dense layer. Finally, a sigmoid dense layer is added to create a final numerical output between 0 and 1. A diagram of this model with 16 features and 128 chosen as the LSTM and hidden layer hyper-parameter is below. As is standard for binary classification, binary cross-entropy loss is used. All other parameters are left as their defaults from the Keras library [17].

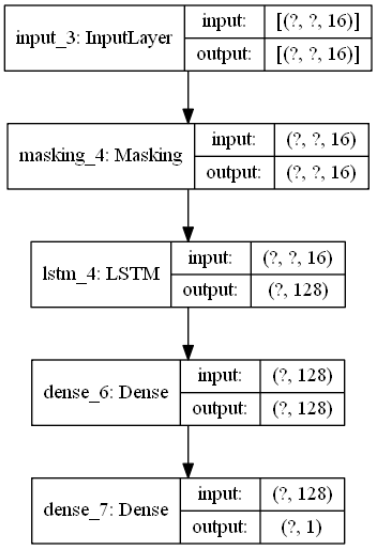


Figure 13: LSTM Model Diagram with Hidden Dense Layer

The chosen hyper-parameters are 128 for the LSTM vector output size and an Adam optimizer with a learning rate of 0.0005 and an epsilon of 1e-8. Models are trained for 70 epochs.

The stochastic nature of the Adam optimizer causes the training of the model to be unstable. However, this problem is especially prevalent when training LSTMs, as has been reported in [18, 19]. This instability problem exacerbates the difficulty of doing a hyper-parameter search. Now, each hyper-parameter combination must be trained and evaluated many times over, which takes a prohibitive amount of time. For this reason, manual hyper-parameter tuning was done.

This instability problem is observed in this project. A total of 102 models, each with the same hyper-parameters and data were trained and evaluated on the same test set. The test and train set accuracy are reported in the below box and whisker plot. The figure also shows the same statistics when overfit models are not included (models with a training accuracy 2% greater than the test accuracy are deemed to be overfit, leaving a total of 37 models).

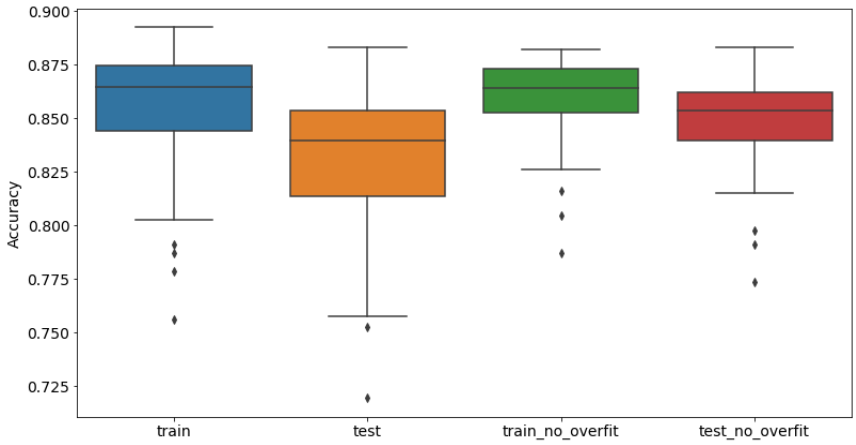


Figure 14: LSTM instability

The architecture without the extra hidden dense layer was also evaluated. The architecture is the same as in Figure 13 but without the dense_6 layer.

This model proved to be much more stable. In 129 iterations, only one model had a training accuracy 2% greater than the test accuracy. The mean test accuracy is %85 with a standard deviation of 1.6%. This is compared to the previous model, with a mean test accuracy of 83% and a standard deviation of 3%. However, the models training accuracy was slightly decreased, from 85.6% to 85%. The below figure shows the box and whisker plot for all 129 models.

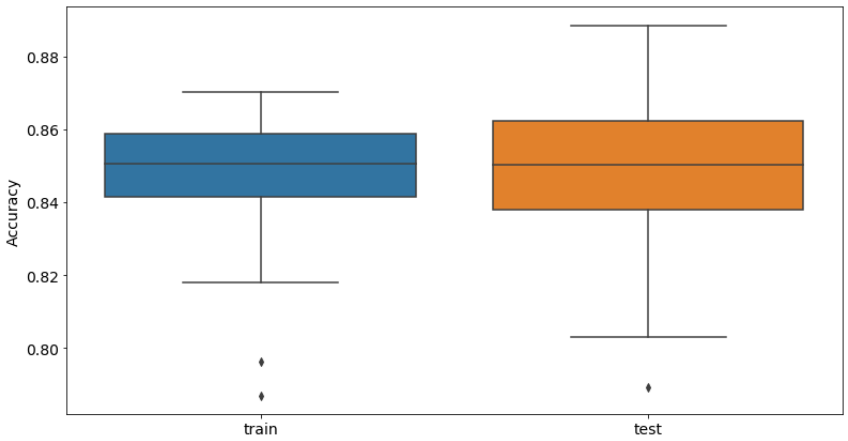


Figure 15: LSTM instability no hidden

Due to the significantly decreased chance of overfitting and greatly decreased variance, the model with no hidden dense layer will be used for future experiments.

Next, a single model must be chosen which will be used for future experiments. The next trained model better than 70% of the previous 129 models was chosen. This model has a test accuracy of 86.8% and an AUC of 91%. It has an f1 score for death of 0.75 and for discharge 0.91. This difference between f1 scores is due to the class imbalance. The model can achieve a higher accuracy in the discharge class due to a larger amount of data. Note that class weights have been used during training.

The below figure shows how accuracy is affected by increasing the sequence length of the test set from admission. This is comparable to Figure 11 except that in this case, the model is not retrained at every different length. The model is trained to, given a complete time series, predict if the patient died. Therefore, the model can only be expected to perform well on complete sequences. The percentage of complete sequences is the green line. Out of these complete sequences, the percentage of patients discharged is shown in orange. As the days of admission data increases, this converges to the baseline accuracy for always guessing discharged. Note that this does not represent the baseline accuracy, as incomplete sequences are also evaluated.

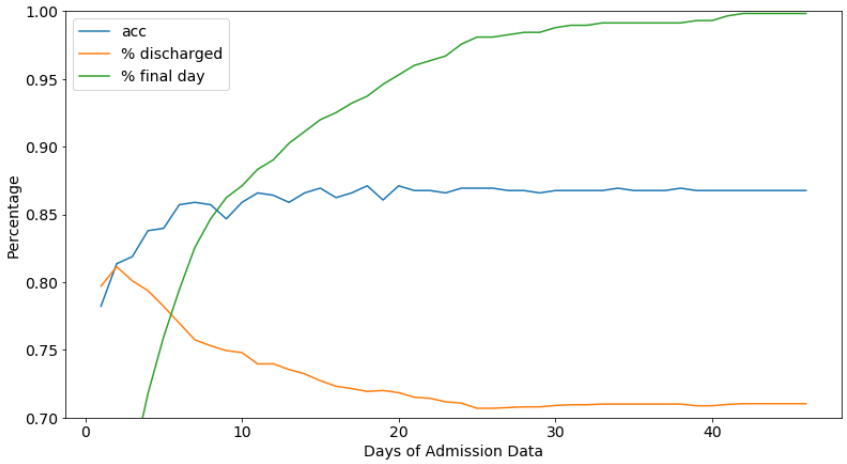


Figure 16: LSTM Admission Data

The below figure shows the effect of increasing the sequence length of the data, but from the end of the sequence instead of the start (as in the above figure).

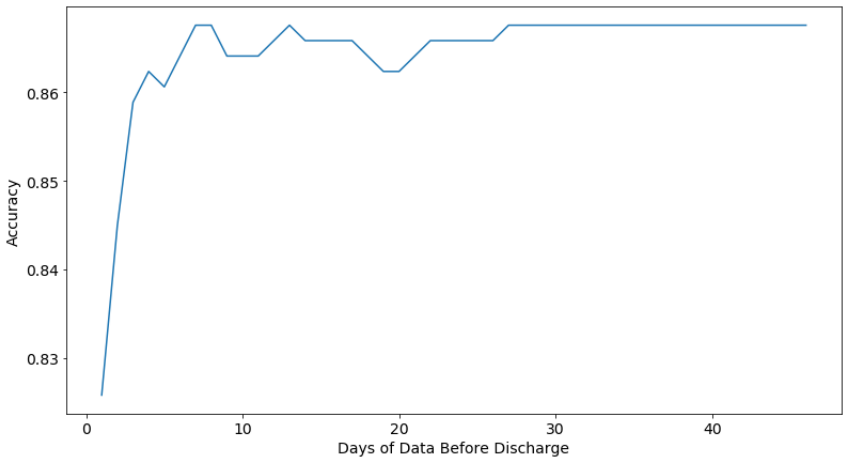


Figure 17: LSTM Accuracy Using Number of Days from Discharge or Death

Code for this experiment can be found in the 'MIMIC death' notebook.

4.2 Discussion

The LSTM has an accuracy of 87% compared to the random forests accuracy of 84%. This is a significant increase for machine learning, and was expected since the LSTM has access to more data and therefore should have more predictive power. However, the f1 scores for death are very similar. The LSTM had an f1 score of 0.75 for the death class and 0.91 for the discharge class, while the RF had an f1 score of 0.74 for death and 0.89 for discharge.

These models have lower f1 scores for death compared to discharge, showing that they are better at classifying discharge than death. This is despite employing class weights for both models. One paper uses oversampling of the minority class (death) to overcome this imbalance [3]. However, it is shown in [20] that oversampling can come at the cost of overall accuracy. Future work would be to investigate the effect of oversampling on the models.

The admission random forest has an AUC of 0.815. This is in line with the findings of the following literature review, which found that machine learning studies with 1000 to 10000 patients had a median AUC of 0.82 for predicting mortality (n=47) [21].

When trained on only data from the day of discharge or death, the random forest achieves an accuracy of 84% (See Figure 12). The LSTM has a smaller accuracy of 82.5% when evaluated on the same test set, using only the final day data. This difference is quite significant, and it shows that the random forests are a better choice for simply doing classification using 2D data.

We can also conclude that the LSTM model is using more than the final day data from Figure 17. The accuracy greatly increases from 82.5% to when there's only final day data to 86% when there's 5 complete days of data, and again to 86.8% for 7 complete days. The model was trained to predict if the patient died after the final day of data, and adding more data before the final day increases accuracy.

In Figures 11 and 12, increasing the number of days of data decreases both test and train accuracy (for at least the first 8 days, where there is sufficient data). This could be due to the smaller amount of data available as the number of days increased, as shown in Figure 10. While the overall accuracy decreases, the f1 score for the test set increases. This is due to the classes becoming balanced, which occurs when the discharged and death lines (red and green) approach 0.5.

Figure 10 shows another advantage of using time series models. Since these models are able to use any amount of data, there is more data available for training. This is in comparison with some models that can only take a fixed amount of data. For example, a model trained on 1 week of ICU data is useless to a patient only in ICU for 5 days. Also, since time series models are able to use more data for training, they are more likely to be good models in general.

5 Explainability

In this section, the SHAP explainability method is investigated.

5.1 SHAP

SHAP was previously introduced in Section 4.4.

There are three properties that SHAP is shown to show in [9]. The first is local accuracy, which states that the explanations for all features add up to the actual output from the model. The second is missingness, which states that 'missing' features have no impact. The third is consistency, which states that if a model changes so that some input's contribution increases or stays the same regardless of the other inputs, that input's attribution should not decrease.

There are different versions of SHAP made for different purposes. The SHAP versions of interest in this paper are TreeExplainer and Kernel SHAP. Both of these methods require a model, which is any python function for Kernel SHAP and an XGBoost model for TreeExplainer. SHAP also always requires a dataset which are the points that will be explained. In the following sections, the test set is used for this.

Kernel SHAP is a the model agnostic version of SHAP. In order to use Kernel SHAP, a background dataset is required. This is used to simulate a 'missing' feature by replacing the value with those in the background dataset, which lets SHAP observe the change in the model and calculate the features importance (From the SHAP documentation for KernelExplainer [9]).

TreeExplainer was first introduced in [10] and detailed further in [7]. It is an op-

timized version of SHAP made for random forests, and works with random forests created with XGBoost. It is of special interest due to the ability to calculate interaction values, which quantify how features interact with other features. This approach also doesn't require a background dataset, instead it creates a background distribution from following the individual trees and finding how many samples visit each leaf (From the documentation on TreeExplainer [9]).

Firstly, Kernel SHAP is used on the time series data. A modified version of Kernel SHAP is created to better suit time series data and is investigated. Finally, TreeExplainer is used to explain the predictions from the random forests from the previous section.

5.1.1 Time Series SHAP

As mentioned in Section 2.4, there is no support for time series data in SHAP. The 3D data must therefore be turned into 2D data. In this section, Kernel SHAP is used. Kernel SHAP uses a model, which is a user supplied python function. This function should take the a 2D input (array of inputs) and return a 1D array with the model outputs for each input. This is different to the input and output of the LSTM model. The input is reshaped to (num_input_days, 1, num_features), meaning the model treats each of the inputs as a different time series with length 1. The output from the LSTM is flattened to a 1D array.

SHAP also requires a background dataset. Using the entire test set as a background dataset would take a prohibitive amount of time. Instead of this, a summary dataset of 10 points is chosen using the k-means function. Finally, the SHAP values are calculated from the test dataset, where the data is made to be 2D by removing the sequence IDs.

SHAP is now run for every data point, which takes approximately 37 minutes. The output from this is a SHAP value for every feature for every day. An example of some SHAP values from a single day is visualized in the force plot below. In this day, a heart rate with a value of 120 pushes the prediction towards mortality, while the platelet count, HCO3, haemoglobin and age all push the prediction towards survival. The SHAP values are the length of the bar for the feature, where the red bars have a positive value and the blue bars have a negative value. These values sum up to the actual prediction from the model for this day.

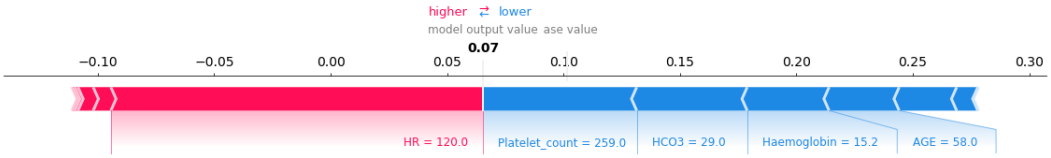


Figure 18: SHAP Explanation Example

The below figure shows the global explanations. These are formed by taking every SHAP value for every feature, plotting the value and shading the point by the feature value. This can help create global explanations such as that decreasing HCO3 increases likelihood of mortality, older patients are at a greater risk and a higher heart rate is associated with a higher mortality rate. However, there appears to be a more complex relationship for platelet count, where low values increase the likelihood of mortality but so do very high values.

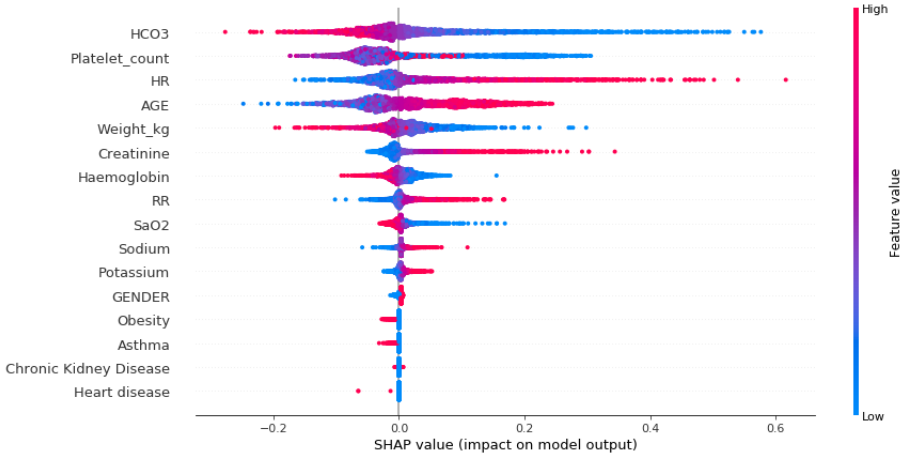


Figure 19: Vanilla SHAP Global Explanations

The above plot also sorts features by importance. This can help provide more global insight, such as that HCO3 is the most important feature that the model is using to make their prediction, and the comorbidities (the bottom 4 features) seem to have little effect, and when they do they bring the model towards survival. The importance is quantified by calculating the mean absolute SHAP value, as can be seen in the below figure. This allows conclusions such as HCO3 is twice as important as weight in general.

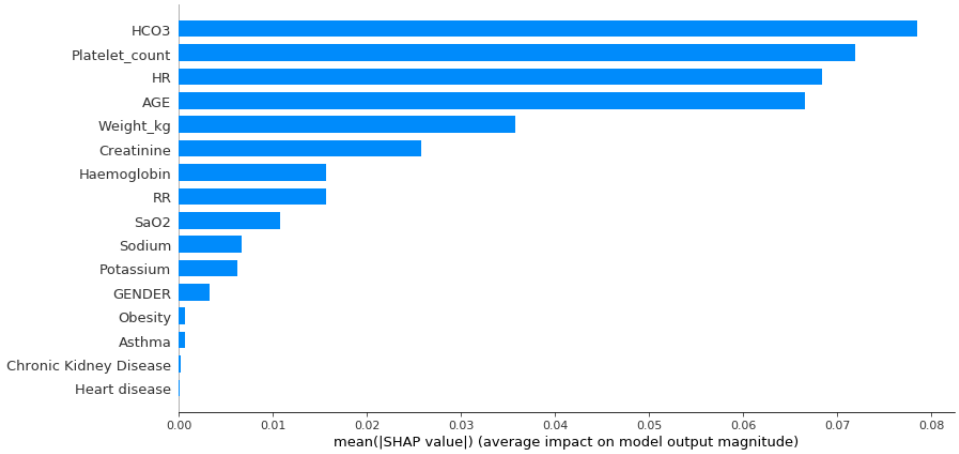
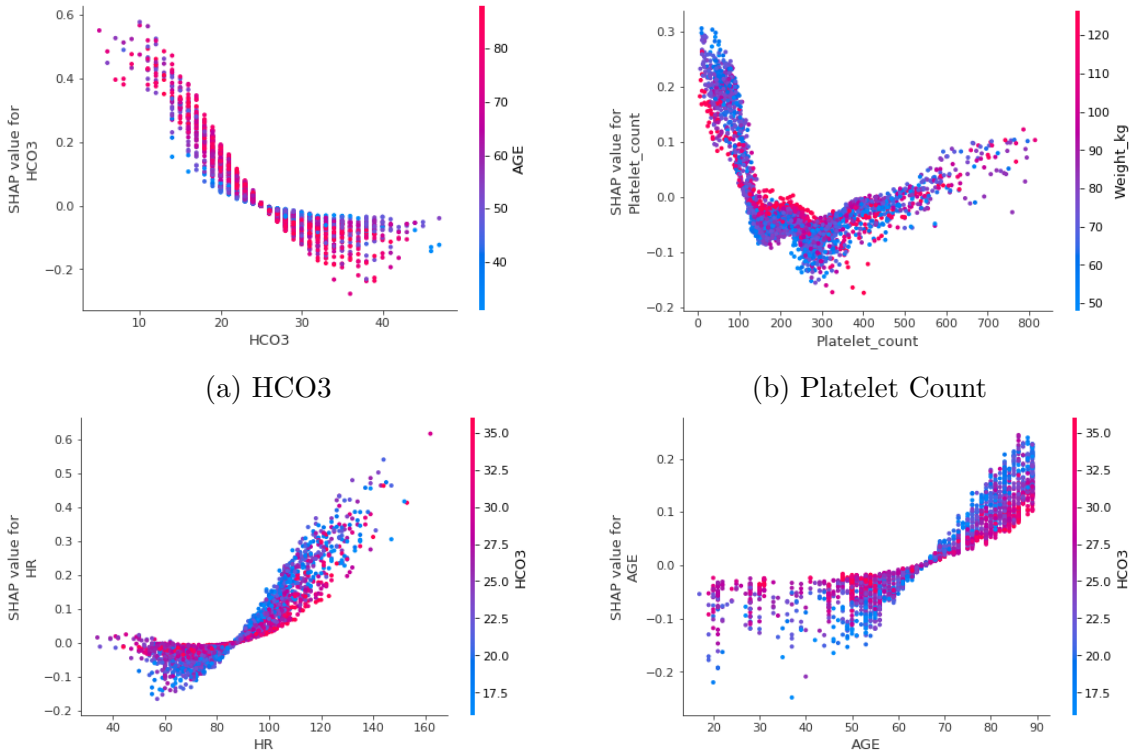


Figure 20: SHAP Feature Importance

[3] also uses vanilla SHAP to create global explanations. They find similar trends for age, heart rate, creatinine and platelets. However, the importance of these features are different to the results obtained in this study. Age is the most important feature (i.e. appears on top in the global explanation figure), followed by heart frequency, creatinine and platelets.

This paper also finds that comorbidities have a lesser effect than the majority of biomarker features, and some result in a lower mortality rate. This paper concludes that this is likely due to the model being unable to learn meaningful relationships due to the small number of patients with comorbidities.

SHAP can be used to show more interesting global relationships. The below figure shows plots for the top 4 most important features for this model (see Figure 19), as previously seen in Figure 2. Note that the axis are different for each subplot. Similar plots for all other features can be found in (Appendix A).



(a) HCO3

(b) Platelet Count

(c) Heart Rate

(d) Age

Figure 21: SHAP Explanations for Individual Features

The average from the entire dataset for HCO3, platelet count, heart rate and age are 25.5, 233, 86.7 and 65.8. For three of the plots in Figure 21, all points that are equal to the mean have a SHAP value of 0. A SHAP value of 0 implies the feature has no effect on the prediction. This means the model has learned that having the mean value for any of these features does not help classify the point.

The exception is for platelet count, where the mean value has a range of SHAP values between -0.1 and 0. The model has clearly learned a more complex relationship with this feature than others. Values too low or too high increases mortality. Also, patients with a greater weight have a higher mortality than those with a lower weight only when the platelet count is between 150 and 300. It is proposed that the W shape of this function is caused by the use of k-means to approximate the background samples. While the 10 points of background samples are appropriate for many other features, platelet count has a larger range of 810, a mean of 240 and a standard deviation of 147.

Besides the mean, most other values of features result in a spread of SHAP values. For example, a heart rate of 120 has resulted in a range of SHAP values between 0.1 and 0.5. This is due to the interactions that heart rate has with other features. Each feature in Figure 21 is also plotted with a colour bar representing the strongest interacting feature. We can see that heart rate interacts strongly with HCO3. For example, a heart rate of 120 on a day with high HCO3 results in a lower SHAP value than it does on a day with a low HCO3.

We are able to use SHAP to create rich global explanations. However, local explanations, specifically explanations about a complete time series, are not accurate. This is because each day of data in a sequence is evaluated individually, meaning the actual output from the model is different when compared to the outputs from the SHAP model.

An example of this is featured below. The model predictions are calculated from SHAP by summing up the SHAP values for all features in that day and adding the expected value. The model output line is created by evaluating the time series with all days up to and including the current day, representing the actual prediction by the model. Note that the two match only on the first day.

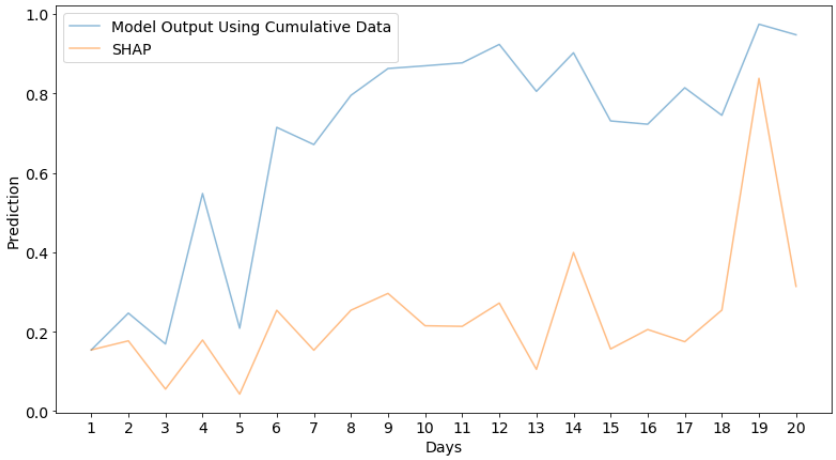


Figure 22: SHAP Local Explanation Example

The below shows the same trend as the above but for the first 20 sequences in the test set instead of just one. Note that the sequence ID goes back to 0 for readability. This also shows a general trend where the actual model always has a prediction greater than that of using just one day of data. This could suggest that the model has learnt that a longer stay in ICU increases risk of dying.

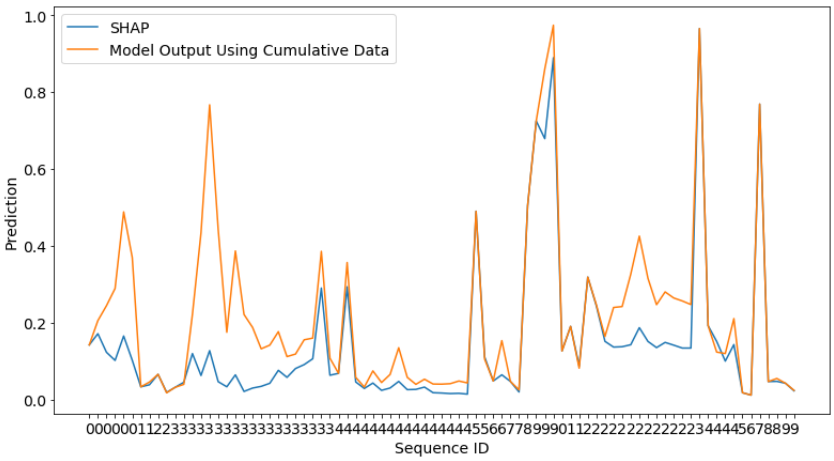


Figure 23: SHAP Local Explanation Examples

This shows that the local accuracy property of SHAP is broken, and therefore that the local explanations do not match the model.

5.1.2 Time Series SHAP Modifications

By exploiting the nature of LSTMs and their hidden state, SHAP can be modified to better handle time series data. The idea of using the hidden state of the LSTM in SHAP is inspired by [22], however the implementation and ML framework used in this section is entirely new. Again, Kernel SHAP is used.

The general strategy is to store the hidden state from the LSTM and use it for the next element in the sequence in the time series. This way, the model will have the exact same inputs as when it is being used on time series data. The data is still passed to SHAP as in the above section (i.e. 2D data).

Firstly, modifications to the model had to be made. It must be able to take the internal states as inputs, therefore two new InputLayers are required. Also, it must return these states as outputs, resulting in two extra outputs. The new model architecture is shown in the below figure.

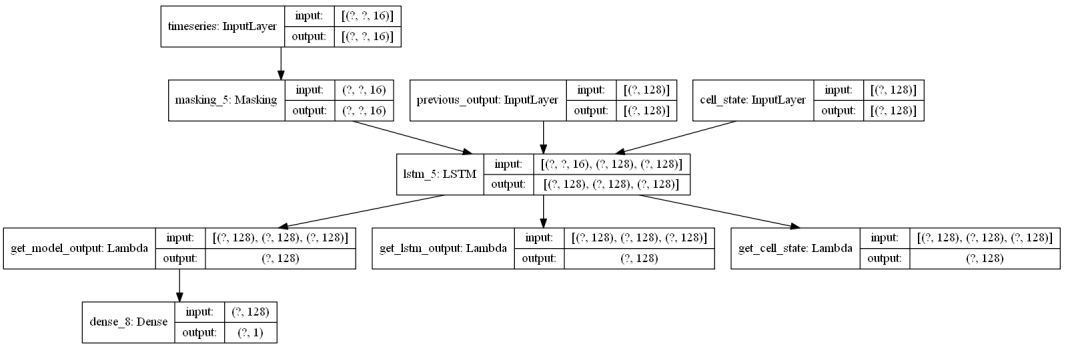


Figure 24: Updated LSTM Model

Note that the model is still able to take arbitrary length sequences. The hidden state is only used at the first element in the sequence, and the hidden state is only returned from the last element. There are now two ways to run the model on time series data: Inputting the entire time series of length N with an empty initial state. Running the model N times, once with each day of data, using an empty initial state for the first day, and on subsequent days using the returned hidden state from the previous day.

The SHAP function 'shap_values' was extended to take an extra argument, 'association', as well as the existing X argument for the dataset. 'association' is an array where the ith element is the ID of the sequence that the ith piece of data in X belongs to. For example, if the first two time points in the dataset X correspond to the same time series, the association array could be [1,1], or anything as long as the first two values in are equal. In this dataset, the 'HADM_ID' from each element is used as the association array.

SHAP is modified to save the LSTM output and cell state for each time. This saved state is fed into the model the next time data that has the same sequence ID is evaluated.

SHAP is able to pass in and get hidden state from the LSTM through a simple API. The 'model' function that is given to SHAP must now take an optional argument 'internal_state'. If None, this will evaluate the input with default initialization. Otherwise, this function should expect a list of the form [h, c] where h is the previous LSTM output and c is the previous cell state. This input should be given to the underlying LSTM. Finally, this 'model' function should return a list of the form [output, h, c], where output is the model output, h is the LSTM output and c is the new cell state. This means a wrapper function is needed to run the model.

These SHAP changes are backwards compatible, and the above API is only used when an 'association' array is given.

The above API works for LSTM models or other models which take 3 inputs and give 2 outputs. Very minor modifications would be required to run the equivalent model on other time series models that take a different number of arguments, e.g. gated recurrent units (GRUs).

This API of using an association matrix is chosen to simplify the implementation. The alternative is to modify SHAP further to take an array and run all the elements in SHAP in a single call, as in [22]. However, this adds unnecessarily complexity to the implementation. If this API is desired, it can be created using the association array method by making the library create an association matrix automatically if a 3D data set is provided.

Details on code modifications to the SHAP library and a link to the repository can be found in Appendix B. Functionality required to run this new SHAP can be found in the 'MIMIC death' notebook under the 'New SHAP LSTM Design' section.

5.1.3 Time Series SHAP Evaluation and Sanity Checks

SHAP has three desirable theoretical properties. The properties are as follows.

Property 1 - Local accuracy

The SHAP paper describes this property as follows. "When approximating the original model for a specific input, local accuracy requires the explanation model to at least match the output for the simplified input" [9]. In other words, the sum of all SHAP values in an explanation should be the same as what the model actually output. This property is clearly desirable, since if it was not satisfied the predictions might not be relevant to the actual model.

This property is held in the modified SHAP on the scale of entire time series. If one assumes that the property holds in the original SHAP, since the modified SHAP is running exactly the same model as the actual LSTM the property will hold for the modified SHAP. This can be verified in practice by checking that the sum of all SHAP values from the final time step in a sequence is the same as the output from the model when using the entire sequence.

This validation is done for the ICU dataset.

Property 2 - Missingness

This property is described as follows. "If the inputs represent feature presence, then missingness requires features missing in the original input to have no impact"[9]. This property is used when calculating the SHAP value. To determine the impact of a feature, that feature is set to "missing" and the change in the model output is observed (From the SHAP documentation for KernelExplainer [9]). In practice, this is simulated by replacing the feature with values from the background dataset. Interestingly, this property is not satisfied in Figure 21 for platelet count, as there is no value that acts as a missing input.

This property cannot be satisfied in general for time series models due to the presence of the hidden state. An example that would be impossible to satisfy is a model that counts the length of a time series. Every data point will increase the prediction by 1, meaning that if there is only 1 feature the SHAP value will be 1 no matter what the input is.

Property 3 - Consistency

This property is described as follows [9]. "Consistency states that if a model changes so that some simplified input's contribution increases or stays the same regardless of the other inputs, that input's attribution should not decrease."

In order to find if the contribution increases or stays the same regardless of the other inputs, this property relies on the missingness property. This means that this property also cannot satisfied in general.

Addition LSTM

The new SHAP is evaluated using a simple addition LSTM model. This addition LSTM is designed to add numbers in a sequence and return the result. The model has two features, the first is a random integer between 0 and 10, and another feature that is always 1. The extra feature is added both to test SHAPs behaviour and also to differentiate between an empty feature and a value of 0. The LSTM is trained to add all of these numbers together.

The training set is created with sequences of a random length between 1 and 10. The training and test sets both are created with 1000 sequences. The same LSTM

as in Section 4.1.2 is used, however the loss is now mean squared error since this is a regression problem. The model attains a mean squared error of 0.7 on the train set and 0.3 on the test set.

The below figure shows the global explanations from vanilla SHAP for this LSTM. Note that this SHAP value is relative to the expected value from the model that SHAP estimates, which in this case is 3.1. We can see the inputs for Feature 0 are evenly grouped from -2 to 7. Since all these groups have the same colour (i.e. feature value), it is clear that the same integer results in exactly the same SHAP value. Remembering that the model is adding 1 at each time step, this comes to a range from 0 to 9, which is the same as the input values for this feature. Furthermore, Feature 1 always has a SHAP value of 0. This suggests that the model has learned to ignore this feature.

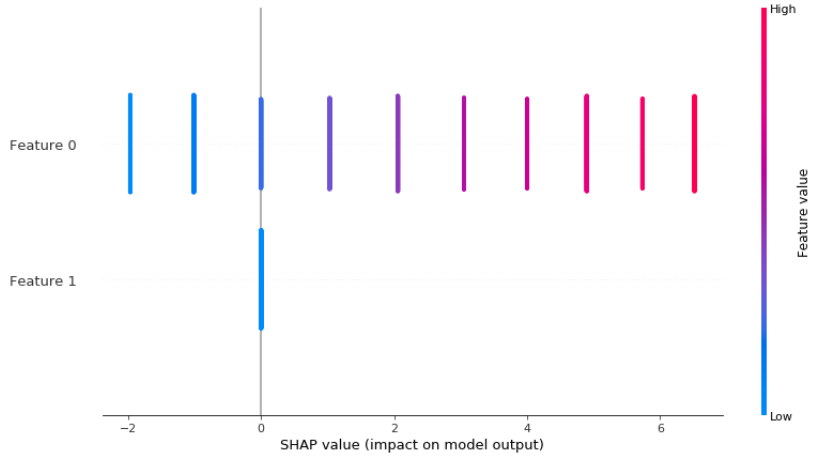


Figure 25: Addition LSTM Global Explanations

This is also shown in the below figure.

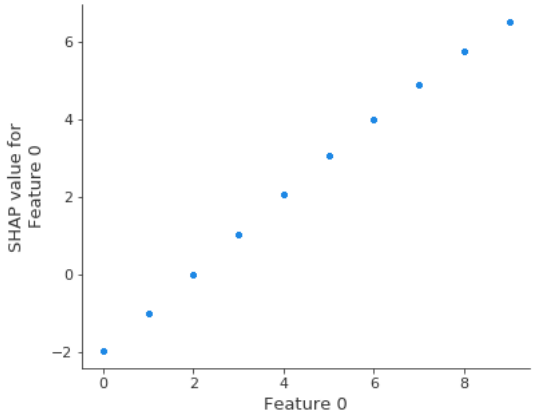


Figure 26: Addition LSTM Global Explanation for Feature 0

As before, the local explanations do not match the actual models output. This can be seen in the figure below, which shows the SHAP values (y axis) for the different element ID (x axis). The LSTM is adding the sequence (7,0,7,1,8,1,2,3,7). Note that the feature switches from red to blue depending on which side of the expected value the result is, and feature 1 has no effect.

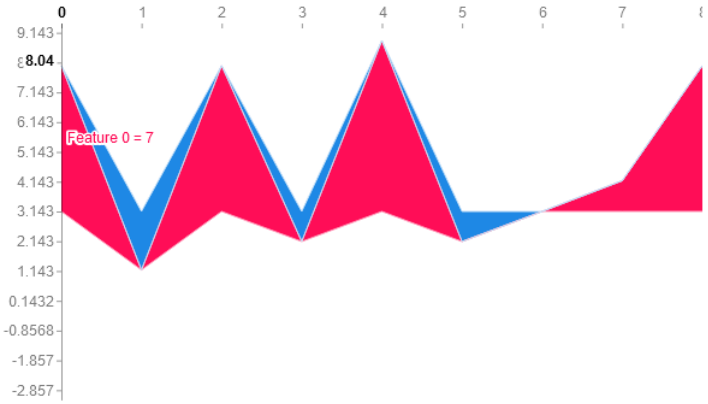


Figure 27: Addition LSTM Local Explanation

The results are very different for the new SHAP. Since the new SHAP adheres to the local accuracy property, the same input will no longer have the same SHAP value. This is because the output will depend on what elements the LSTM has already seen. For example, if the previous sum is 5 and the new SHAP value is 0, the model will output 6 for the new element, which means the features must have a SHAP value of 6.

This behaviour is shown in the below figure. The expected value from SHAP is now 5.5. Clusters of matching SHAP values can still be seen. However, there is now no longer a clear relationship between feature value and SHAP value. Note that Feature 1 still has no importance.

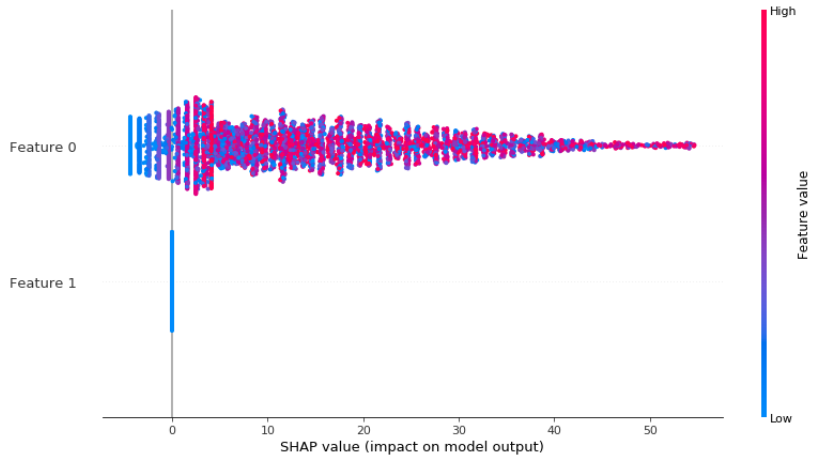


Figure 28: Addition LSTM New SHAP Global Explanations

This is shown more clearly in the below figure. When Feature 0 is 0, the SHAP value can be anywhere up to 50.

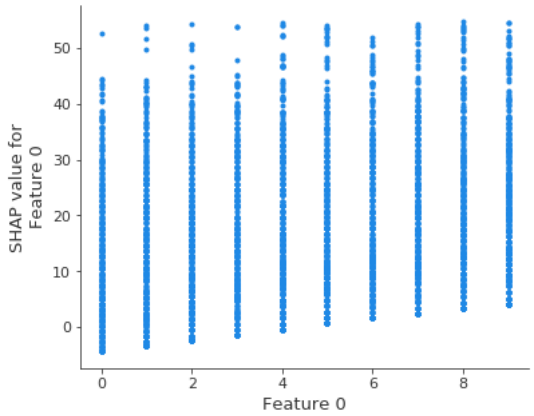


Figure 29: Addition LSTM New SHAP Global Explanation for Feature 0

While the global explanations are not functional, the local explanations are correct across an entire time series, as can be seen below. This is the same sequence as in27

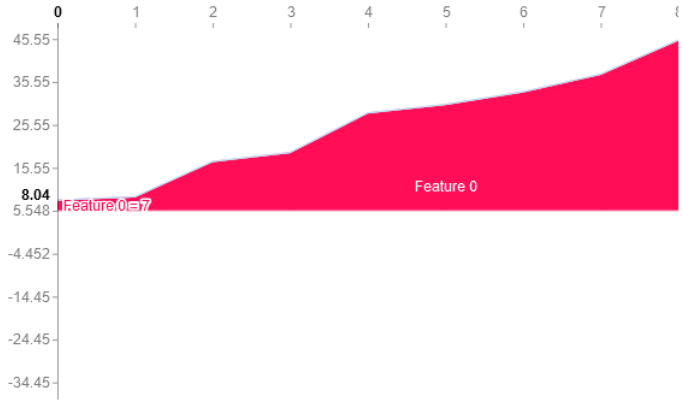


Figure 30: Addition LSTM New SHAP Local Explanation

This experiment has shown that the new SHAP gives accurate local explanations at the cost of inaccurate global explanations. One possible modification to attempt to get both accuracy local and global explanations is to try extract the contribution from the hidden state. For example, if the previous sum was 15 and the current sum is 20, then 15 can be extracted for the hidden state, and the current features will make up the remaining 5.

Unfortunately, this is a simplification that is unlikely to work in general. For example, consider a counter LSTM that counts the number of elements in the sequence. No features should have any importance since the LSTM will increment irrespective of the feature value, however both old and new SHAP will atribute 1 to the features. Also, consider an LSTM that acts as both a counter but also has complex interactions between the hidden state and the current values. There is no way to tell how much of the hidden state is due to the LSTM counting and how much is due to these other interactions.

Code for this experiment is available in the 'SHAP Sanity Checks' notebook, see Appendix A.

ICU LSTM

Firstly, it is verified in code that the ICU LSTM satisfies the local accuracy property for all elements in the test set.

The global explanations for the same model as in Section 5.1.1 is done using the new SHAP. Because the output from the model is between 0 and 1, the global

explanations do not suffer from the same problems as the addition LSTM. In fact, the global explanations seem to match those in Figure 19, and the features have the same order of importance.

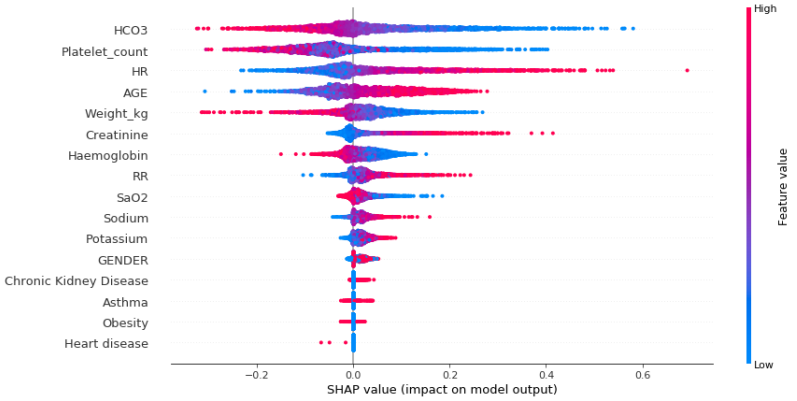
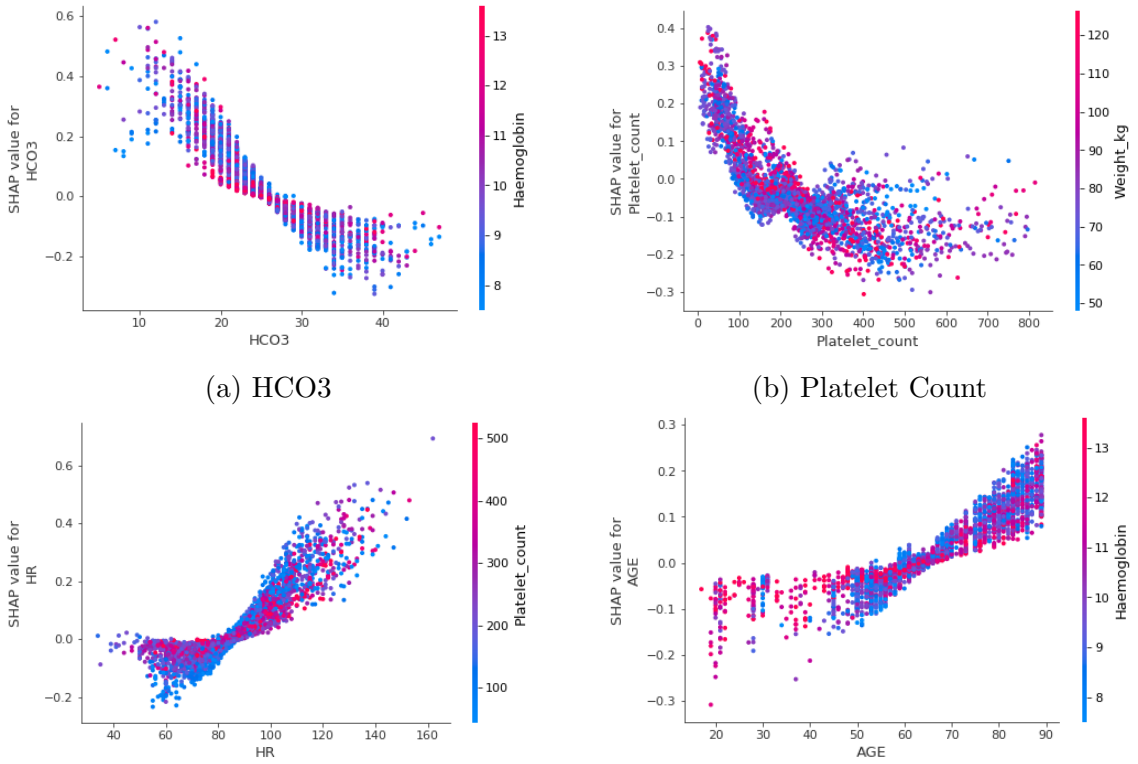


Figure 31: New SHAP Global Explanations

However, the below individual plots for features show the problems that the new SHAP has. Specifically, the mean value for HCO3, heart rate and age no longer have 0 SHAP value. The most important interactions are also different.



(a) HCO3

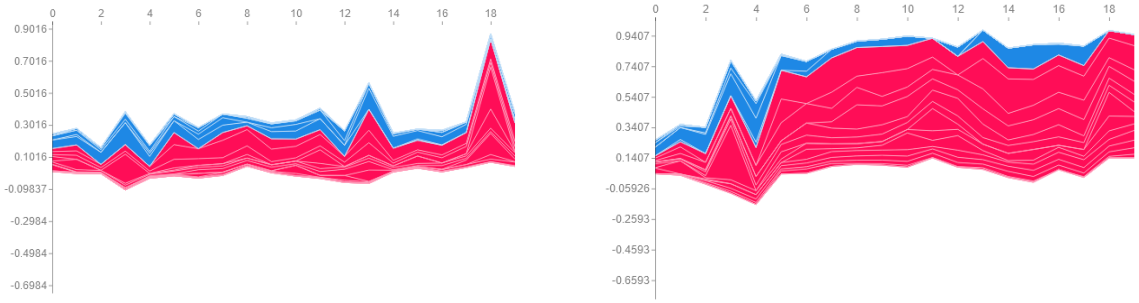
(b) Platelet Count

(c) Heart Rate

(d) Age

Figure 32: New SHAP Explanations for Individual Features

A local explanation over an entire time series is shown below for vanilla and new SHAP. The same 19 day ICU stay is explained. Note that the actual model output is where the blue and red lines meet (each red line represents a feature that impacts towards mortality, and blue lines impact towards survival). This shows how inaccurate the LSTM is when only evaluated on single day slices. According to Vanilla SHAP, the patient will be discharged alive but the actual model is very confident the patient will die, even after the first week of ICU data.



(a) Vanilla SHAP (b) New SHAP

Figure 33: SHAP Local Explanations

This also highlights an interesting relationship, that longer series have a higher chance of mortality. The mean prediction over all days of data is 0.25 for vanilla SHAP, but it is 0.37 for the new SHAP. For 1969 out of 2787 days of the data, the SHAP values from new SHAP are larger than those from vanilla SHAP. This could suggest that the LSTM has learned that the longer the stay in ICU, the higher the chance of mortality. In order to test this, the 'day' feature could be added as an input to the model.

The vanilla SHAP plot in Figure 33 shows some moderate jumps in mortality (spikes), such as in day 3 and 13 and also a large spike at day 18. In this specific example, the spikes at day 3 and 19 are caused by a very high heart rate. This shows another use for vanilla SHAP. It can be used to detect such spikes which may go unnoticed by the new SHAP due to a consistently high mortality prediction. Future work would be to investigate this possibility.

5.1.4 TreeExplainer

In this section, TreeExplainer is used on the random forest trained on admission data, final day data and the final 2 days of data. In all of these experiments, SHAP runs in less than 1 second.

Admission Data

The first experiment is running the admission random forest that uses only first day data from Figure 11 with TreeExplainer. The results of the global explanations are in the figure below.

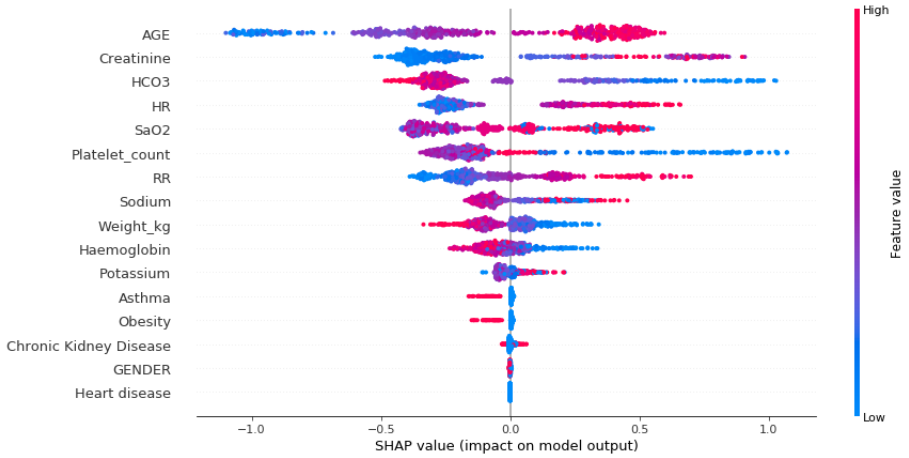


Figure 34: TreeExplainer Global Explanations Admission

The features that are deemed important are different to the LSTM from the previous section. Age now has the highest impact on the model. It makes sense why the model would find age more relevant to death since the age is constant throughout the ICU stay, but the biomarker features are likely to be very different from day 1 to the final day. The importance of all features are shown below. Interestingly, weight has a lesser effect than that of the LSTM. This could be due to the lack of interactions formed between weight and other features.

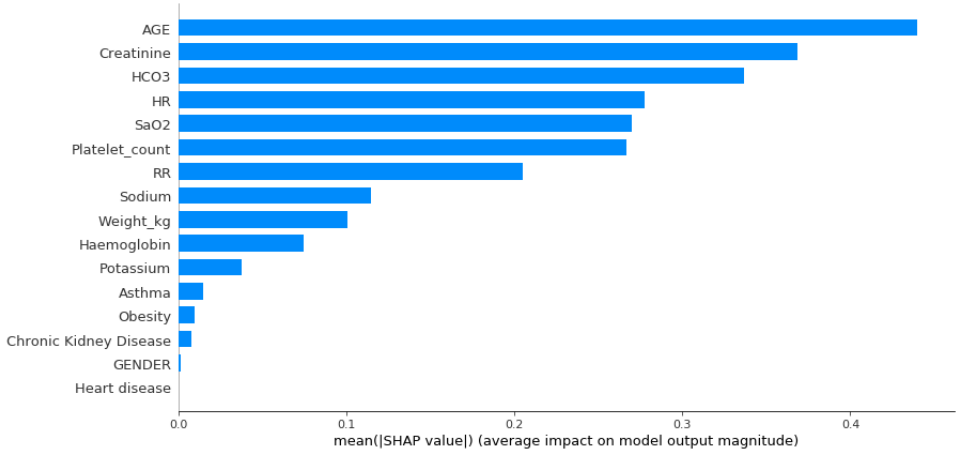
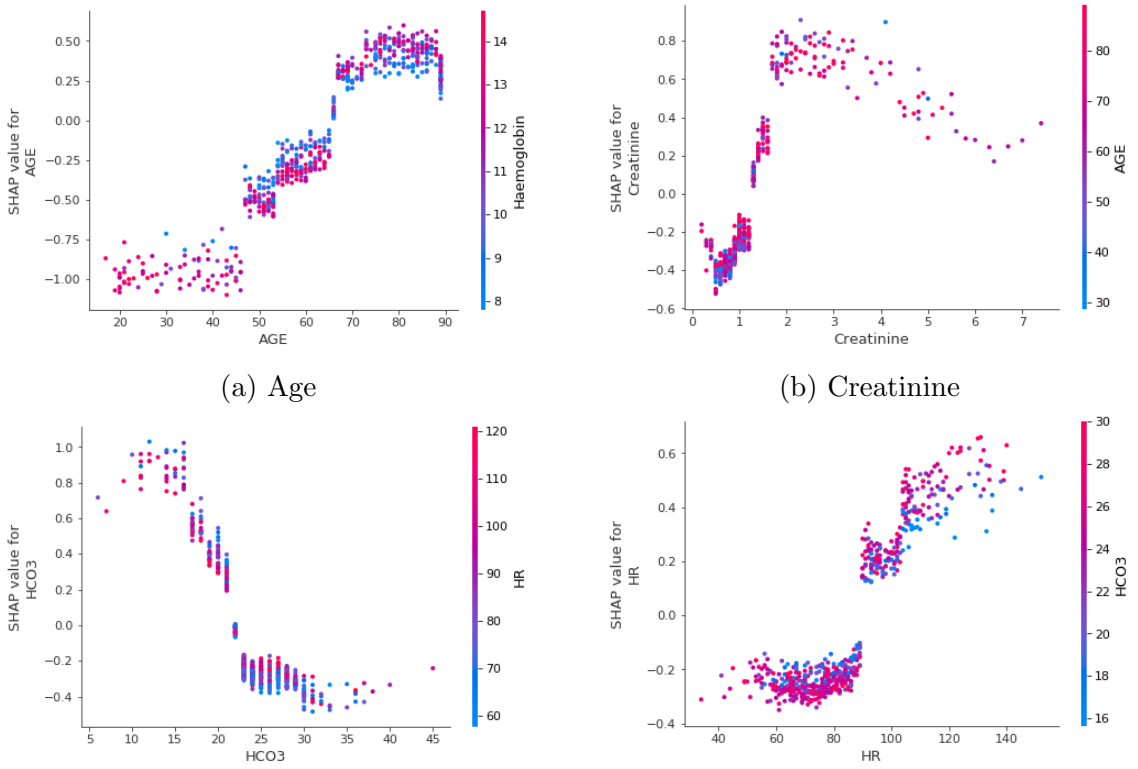


Figure 35: TreeExplainer Feature Importance Admission

The top 4 features are shown below. Similar trends are noticed for heart rate, age and HCO3 when compared to Figure 21. However, the lines are less smooth

and are not as continuous as they were before. This could be due to the nature of the decision boundaries of random forests, which can be very sharp. However, it is likely due to the model being trained with insufficient data, as the XGBoost survival model featured in [10] does not have this discontinuity problem. There are also significantly less data points in the test set for this problem, only 574 compared to the LSTM's 2787.



(a) Age

(b) Creatinine

(c) HCO3

(d) Heart Rate

Figure 36: SHAP Explanations for Individual Features

Final Day Data

This experiment is more directly comparable with the LSTM, as it is predicting if the patient died after these readings. This random forest also had a much higher accuracy than the admission random forest by 6%.

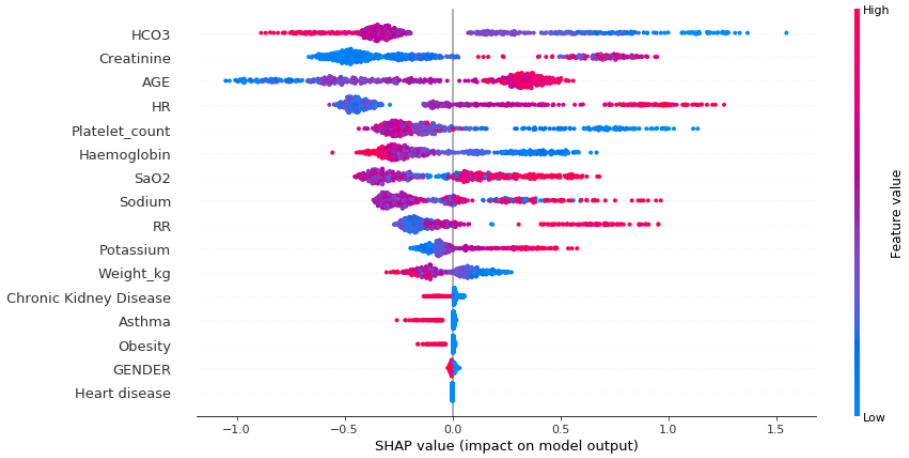


Figure 37: TreeExplainer Global Explanations Final Day

The trends (the general shape of the curves) for HCO3, platelet count, age, and heart rate are similar to Figure 21. However, the trends for creatinine and SaO2 are different. The SHAP values for creatinine in particular have a much larger range. SaO2 has the opposite trend, where a higher SaO2 increases mortality.

The feature importance also differs from the LSTM. Creatinine is one of the most important features for the random forest, similarly to the admission random forest.

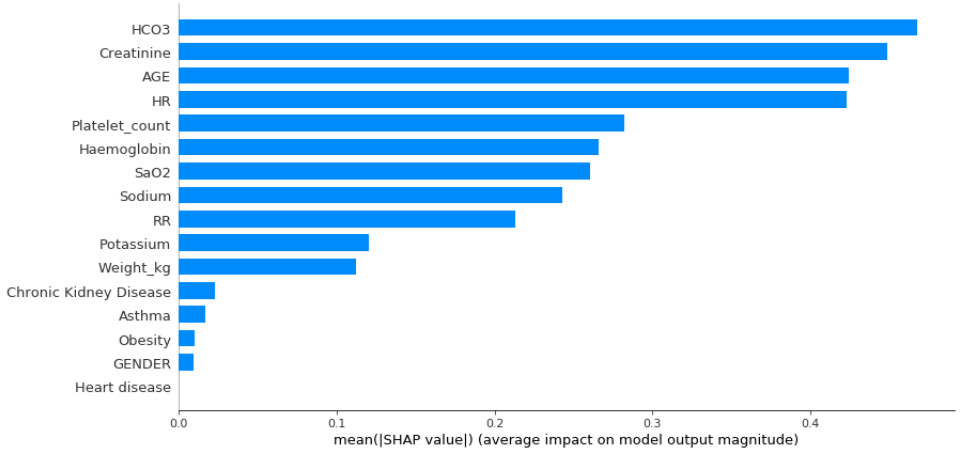
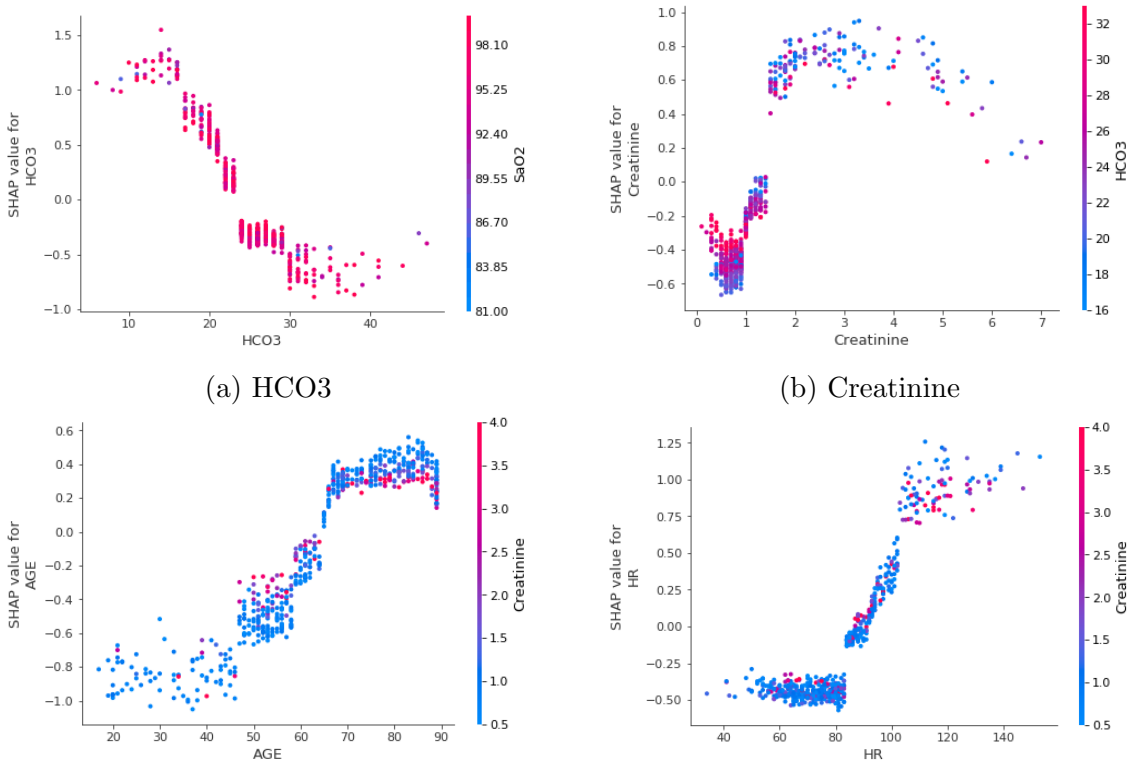


Figure 38: TreeExplainer Feature Importance Final Day

As before, the top 4 features are shown.



(a) HCO3

(b) Creatinine

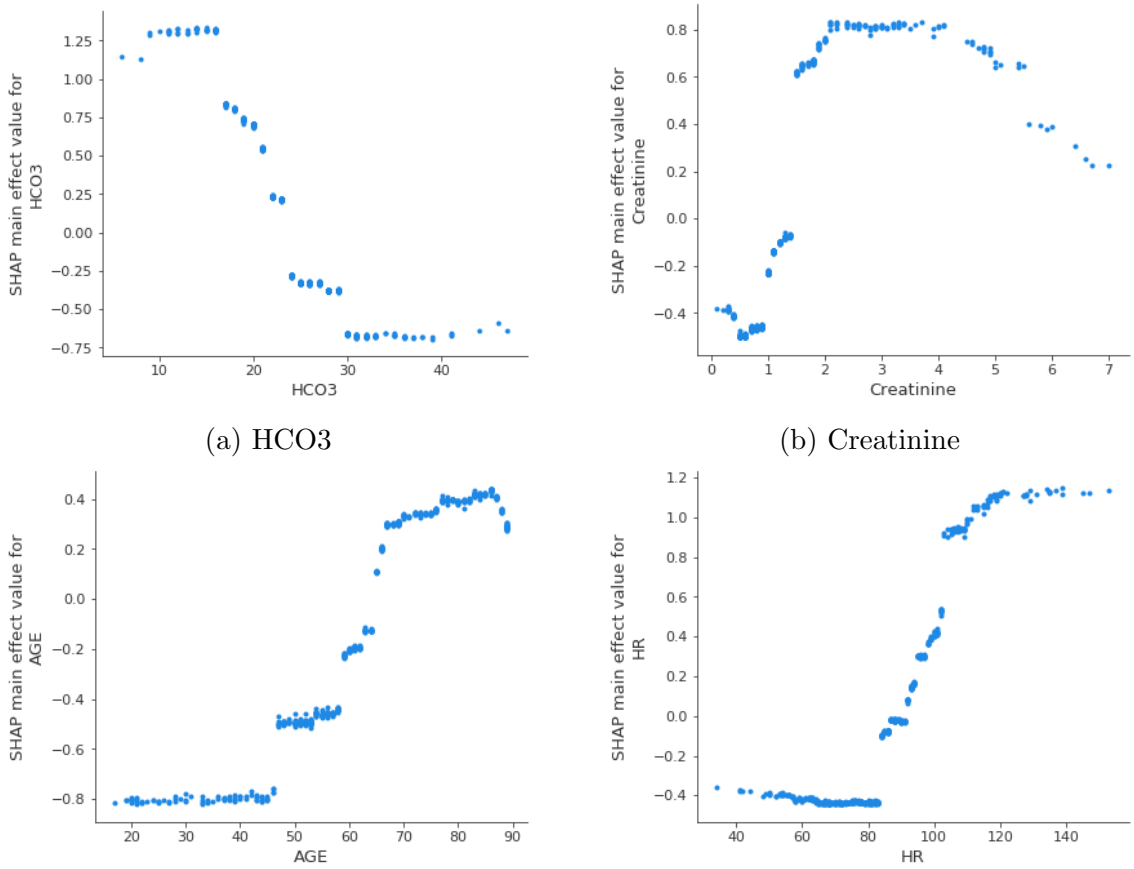
(c) Age

(d) Heart Rate

Figure 39: SHAP Explanations for Individual Features

TreeExplainer provides the option to extract interaction values (i.e. the extent that features interact with each other). This makes it possible to see exactly how two features interact, and even plot features without the interactions of other features.

The figure below shows the main effect of the top 4 features. Since there are no interactions with any other features, there is no longer any variance between the SHAP value for the same value of the feature. When compared with Figure 39, it can be seen that HCO3 and heart rate don't interact much with other features compared to age or creatinine. The discontinuities are clearly evident in these plots.



(a) HCO3

(b) Creatinine

(c) Age

(d) Heart Rate

Figure 40: SHAP Main Effect for Individual Features

An example interaction plot between creatinine and HCO3 is shown below. This plot shows the same interaction as from Figure 39. However, the interaction is now quantifiable. For example, a very high HCO3 is impacts the prediction more when creatinine is between 0 and 1 (0.2 to 0.3) than when it is greater than 1 (-0.1 to -0.2).

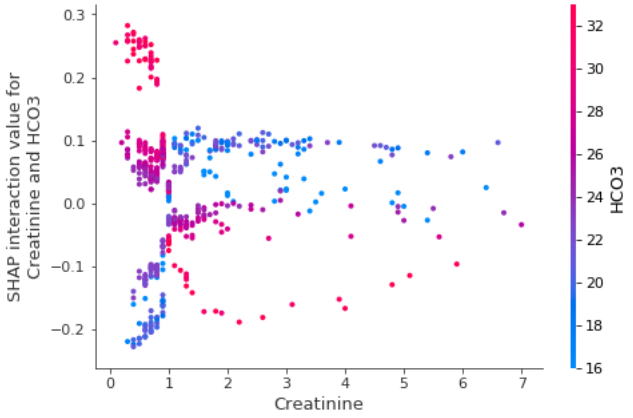


Figure 41: TreeExplainer Interaction between Creatinine and HCO3

Final 2 Days of Data

In this experiment, the final 2 days of data is used. Day 1 is the data from the second last day of the patients ICU stay, while day 2 is the final day. There are 32 features in total. Global explanations can be seen below. Note that chronic kidney disease, gender, asthma and heart disease for both days are not shown as they have almost no effect.

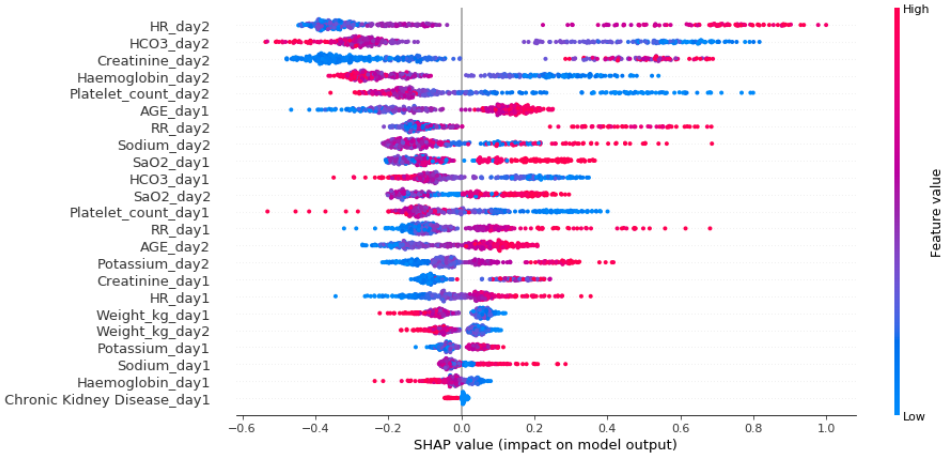


Figure 42: TreeExplainer Global Explanations Final 2 Days

As expected, the day 2 features have a much greater impact on the model output than those of day 1. The trends for the same feature on different days have the

same appearance overall. For example, a low HCO3 increases mortality and a high HCO3 decreases mortality. The quantified feature importance is featured below.

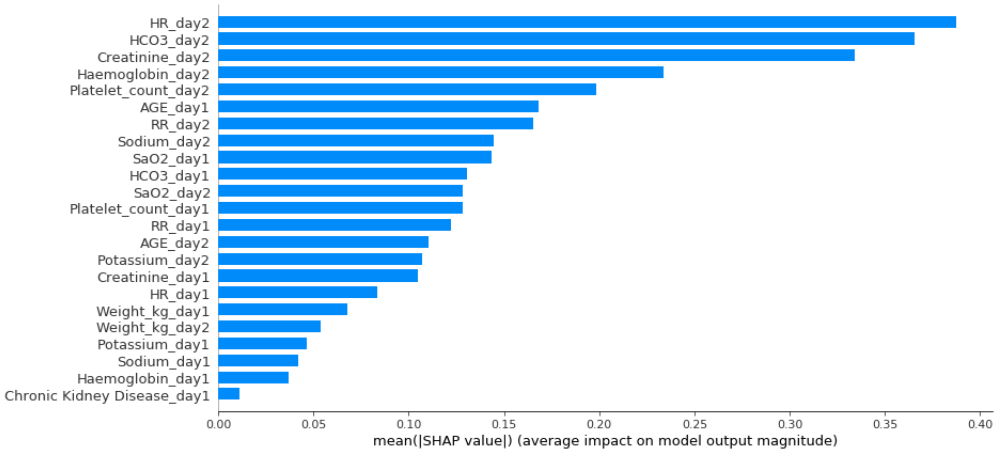
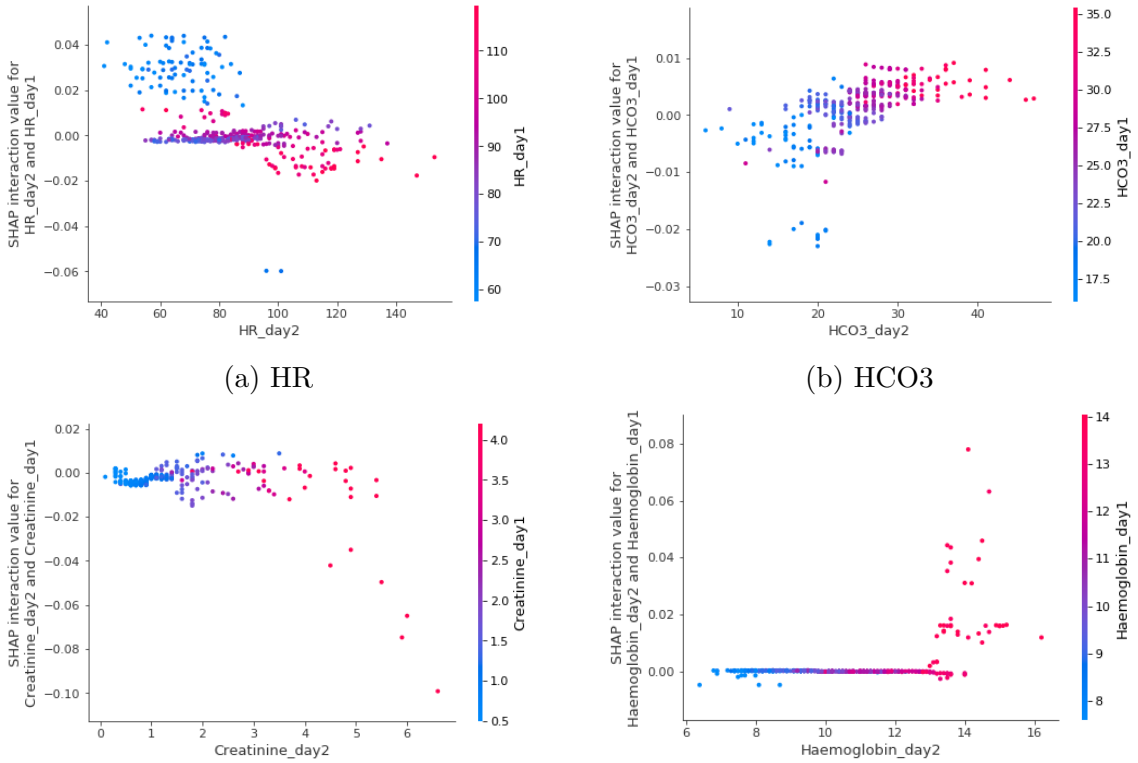


Figure 43: TreeExplainer Feature Importance Final 2 Days

The individual feature plots are very similar to those in Figure 39. There are minor differences which could be caused by the difference between the two datasets.

We can investigate whether or not the model is learning the relationships between the days. One hypothesis for the decrease in accuracy when increasing the days of data is that the random forest cannot learn the relationship between the days. The LSTM is forced to learn this relationship due to it's architecture, but the random forest is not. Interaction between day 2 and day 1 variables. This mimics the way LSTMs work, where features on day 2 interact with previous values, not visa versa.



(a) HR (b) HCO3

(c) Creatinine (d) Haemoglobin

Figure 44: SHAP Interaction Effects For Same Feature

There is very little interaction between the features for creatinine and haemoglobin. It appears that HCO3 and heart rate are impacted by the previous day's value. However, the scale on the Y axis shows how little the values are affected. From Figure 43, the mean absolute SHAP value for day 2 HR is 0.4, however the day 2 heart rate only changes due to day 1 heart rate by at most 0.04. The relationships shown in the above figure may therefore not be significant.

From the haemoglobin graph in Figure 44, it is evident that the day 2 haemoglobin goes up to 16 but the day 1 haemoglobin only goes up to 14. The model has learned that if the day 2 haemoglobin is very high it is important to look at the day 1 haemoglobin.

Since the random forest is able to handle all features from multiple days at once, there are also interactions between day 1 features and day 2 features (i.e. day 2 features impact on SHAP scores of features from day 1). In fact, the explanations for interaction from day 1 to day 2 look very similar to those in Figure 43. If these

relationships are significant, the random forest has learned a relationship that does not make sense in the context of the problem.

It is possible that there are many other significant relationships that SHAP interaction plots can show. However, it is unreasonable to interrogate them all. For example, for a 1 week ICU stay with 16 features per day means there will be a total of 112 features, therefore a total of 12544 interactions. Even using 2 days of data has 1024 interactions. This highlights one problem with trying to get detailed explanations for long time series. Even an ideal explainability method that could show the relationship between every feature in the time series so far, it would not be feasible for anyone to check them all. However, if the interactions can be quantified, finding only the most important interactions would be possible.

5.2 Discussion

It is found that SHAP is very useful for explaining relationships between features and outputs on a global scale. One of the aims of explainablility is to be able to validate or invalidate a model. SHAP makes this easy to do on a high level, as it can be easily discovered if the model has learned a nonsensical relationship.

The proposed modifications to SHAP provide local accuracy for entire time series at a cost to the strong theoretical justification of SHAP and global explanations. The Addition LSTM is shown to have meaningless global explanations with the new SHAP but meaningful local explanations. A hybrid use of vanilla SHAP for global explanations and new SHAP for local explanations could be of interest.

It was found that most of the features from the random forests investigated had the same trends. The extra functionality (extracting interactions between features) from TreeExplainer was also investigated, and found to provide rich global explanations. Sensible explanations are also mostly obtained from running a random forest with multiple days of the same features.

It was found that the explanations from the final day random forest and the LSTM are similar. Future work would involve finding if these relationships are sound in the context of the ICU.

By comparing explanations from the vanilla SHAP to the new SHAP, evidence is found for a relationship between sequence length and mortality. Further investigation is required.

A method that can attribute importance to every feature from all days would be the ideal outcome. This has been shown to be possible for random forests using SHAP, however it is infeasible in it's current state due to an explosion of interactions for long time series. Future work would be to investigate quantifying the most useful interactions to show users.

This highlights a trade off between greater accuracy and better explanations, since LSTMs are clearly better suited to this problem and attain a significantly higher accuracy.

Another distinction between random forests and LSTMs highlighted in this section is the run time for SHAP. Kernel SHAP takes nearly 40 minutes to run, while TreeExplainer takes less than 1 second for a quarter of the data. This is due to the simplicity of the random forest compared to the LSTM. Note that these run times are on a CPU, however attempts to run the LSTM on a GPU failed.

6 Discussion

The first step in this project was to create a dataset for machine learning. This was done using the MIMIC-III dataset. In order to create the final dataset, a certain subset of features were chosen. These features were chosen due to the number of data points present in the dataset. It was also decided that no imputation would be done in order to simplify the dataset creation.

However, the lack of imputation is a significant limitation. This meant that the study is only using a specific subset of the data, which may have introduced a bias. Clearly there is a larger mortality rate in the final dataset (31%) compared to the in-hospital mortality rate of all sites in the MIMIC-III study (11.5%). In future, the MICE method could be used to investigate such biases [23].

Random forests were created and trained on different amounts of admission or final days of data. Random forests trained on longer time series consistently had a lower accuracy than random forests trained on smaller time series. This is because in the created dataset, the longer the sequence, the less data that is of at least that length. This highlights a further advantage of using time series models, that they are able to use data from every ICU stay, regardless of length.

Two different LSTM models were trained. It was found that a model with an extra

hidden layer generalized poorly, while the model without generalized very well. It was also found that the model was not using only the last day of data to make it's prediction. As expected, the LSTM performed better than any of the random forests.

Three different experiments were done involving SHAP. This includes running SHAP with the LSTM without considering interactions within individual time series sequences. Global explanations are found to be useful and rich. However, it is found that local explanations are incorrect.

A modified version of SHAP is proposed that handles time series data by keeping track of internal state within sequences. Modifications to the SHAP library were created and a new general API was proposed. While these modifications break some ideal properties, it satisfies the local accuracy property on the scale of entire time series.

An example is shown where this new method does not provide useful global explanations. It is proposed that a hybrid system using vanilla SHAP for global explanations and using the new SHAP for local explanations could be used together.

7 Conclusion

This thesis had one key aim; to see if it's possible to predict death using time series ICU data in an interpretable way.

It was shown that an LSTM attains a high accuracy on time series data when predicting death and discharge, however it is better at predicting discharge than death. The accuracy of the model was verified to be higher than that of a model that uses data of a fixed sequence length.

This model was made interpretable by using the SHAP explainability method to create global explanations, and a newly proposed modified version of SHAP to provide local explanations in the scale of individual time series. This allows the model to be interpreted for validation by checking global explanations, ensuring the model aligns with medical knowledge. Local explanations can be used to tell which features are important for a patients current state.

However, the local explanations do not perfectly describe the model. In particular,

it is not able to see precisely how features from previous days affect the current day. This is shown to be possible using SHAP, but only for models that take data of a fixed sequence length.

While the time series model may have good accuracy and be explainable, is it useful? In practice, models that predict specific times to death or discharge for example would be more useful. However, time series models that predict mortality have the potential to be useful to clinicians to provide live risk prediction, and the model's explainability can reveal actionable elements [3]. Further validation and research into this area is required.

7.1 Future Work

This study would benefit from more data, especially of longer ICU stays. This would allow the random forest experiments to become more accurate. This should be possible by doing data imputation on MIMIC-III.

It is expected that a hyperparameter search could find an LSTM that has a higher accuracy. Further investigation should be done into using methods to combat the class imbalance problem, specifically oversampling of the minority class.

In order for this work to be more easily validated, a unified system combining both versions of SHAP should be created. This should make it easy for end users to be able to see both global and local explanations and be informed on the limitations of the method.

Further investigation into using SHAP to look for jumps in mortality predictions as an early warning system could be promising.

Try quantify LSTM dependence on counting by adding the day number as a feature

If a random forest can be trained that has a greater accuracy than the LSTM, then using TreeExplainer SHAP should be further investigated. In particular, a way to quantify the most important interactions between features should be found in order to combat the immense number of features.

8 Bibliography

References

[1] D. A. Kaji, J. R. Zech, J. S. Kim, S. K. Cho, N. S. Dangayach, A. B. Costa, and E. K. Oermann, "An attention based deep learning model of clinical events in the intensive care unit.(research article)(report)," *PLoS ONE*, vol. 14, no. 2, p. e0211057, 2019.

[2] K. Young, G. Booth, B. Simpson, R. Dutton, and S. Shrapnel, "Deep neural network or dermatologist?" *arXiv.org*, vol. 11797, 2019. [Online]. Available: http://search.proquest.com/docview/2276715881/

[3] H.-C. Thorsen-Meyer, A. B. Nielsen, A. P. Nielsen, B. S. Kaas-Hansen, P. Toft, J. Schierbeck, T. Strøm, P. J. Chmura, M. Heimann, L. Dybdahl, L. Spangsege, P. Hulsen, K. Belling, S. Brunak, and A. Perner, "Dynamic and explainable machine learning prediction of mortality in patients in the intensive care unit: a retrospective study of high-frequency data in electronic patient records," *The Lancet Digital Health*, vol. 2, no. 4, pp. e179–e191, Apr 2020. [Online]. Available: https://doi.org/10.1016/S2589-7500(20)30018-2

[4] A. E. Johnson, T. J. Pollard, L. Shen, L.-W. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific Data*, vol. 3, no. 1, 2016.

[5] E. De Brouwer, J. Simm, A. Arany, and Y. Moreau, "Gru-ode-bayes: Continuous modeling of sporadically-observed time series," *arXiv.org*, 2019. [Online]. Available: http://search.proquest.com/docview/2232265113/

[6] C. Olah. (2015, Aug) Understanding lstm networks. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[7] S. M. Lundberg, G. G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S. Lee, "Explainable AI for trees: From local explanations to global understanding," *CoRR*, vol. abs/1905.04610, 2019. [Online]. Available: http://arxiv.org/abs/1905.04610

[8] M. Hsieh, M. Hsieh, C.-M. Chen, C.-C. Hsieh, C.-M. Chao, and C.-C. Lai, "Comparison of machine learning models for the prediction of mortality of patients with unplanned extubation in intensive care units," *Scientific Reports*, vol. 8, 12 2018.

[9] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[10] S. M. Lundberg, G. G. Erion, and S. Lee, "Consistent individualized feature attribution for tree ensembles," *CoRR*, vol. abs/1802.03888, 2018. [Online]. Available: http://arxiv.org/abs/1802.03888

[11] C. Molnar. (2019) Interpretable machine learning. https://christophm.github.io/interpretable-ml-book/.

[12] T. pandas development team. (2020, Feb.) pandas-dev/pandas: Pandas. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[13] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[14] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," *MIT Press*, 2016.

[15] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[18] J. Miller and M. Hardt, "Stable recurrent models," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Hygxb2CqKm

[19] J. Brownlee. (2020) Instability of online learning for stateful lstm for time series forecasting. https://machinelearningmastery.com/instability-online-learning-stateful-lstm-time-series-forecasting/.

[20] X. Gao, H. Chen, and Y. Guo, "Mortality prediction of icu patient based on imbalanced data classification model," in *LISS2019*. Singapore: Springer Singapore, 2020, pp. 83–94.

[21] D. Shillan, J. A. C. Sterne, A. Champneys, and B. Gibbison, "Use of machine learning to analyse routinely collected intensive care unit data: a systematic review," *Critical care (London, England)*, vol. 23, no. 1, pp. 284–284, 2019.

[22] A. Ferreira. (2019, Jul) Interpreting recurrent neural networks on multivariate time series. [Online]. Available: https://andrecnf.github.io/2019/07/31/InterpretingRecurrentNeuralNetworksOnMultivariateTimeSeries.html

[23] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, "Multiple imputation by chained equations: what is it and how does it work?" *International journal of methods in psychiatric research*, vol. 20, no. 1, pp. 40–49, 2011.

9 Appendix

9.1 Appendix A - Repository Overview

Code is available at `https://github.com/Gareth001/thesis`. Notebooks can be viewed directly inside Github without being re-run.

Code to process the mimic database into a single flat file csv is found in the 'mimic' folder. The remainder of the code is in the form of jupyter notebooks in the 'notebooks' folder. The 'MIMIC dataset.ipynb' notebook is used to further process the data into a usable dataset for machine learning. 'MIMIC features.ipynb' contains some visualizations of the features available in the MIMIC 3 database.

'MIMIC death.ipynb' contains some feature visualizations, the code for building and evaluating the LSTM model, and the code required to run both vanilla and new SHAP on the LSTM. 'MIMIC death RF.ipynb' contains the random forest related code.

The other folders contain the other pieces of assessment for this thesis.

To reproduce the experiments in this thesis:

1. Clone the repository
2. Download the MIMIC 3 database from `https://mimic.physionet.org/about/mimic/` and place all .csv.gz files into the mimic/data_raw folder

3. Run all cells in the 'MIMIC dataset.ipynb' notebook

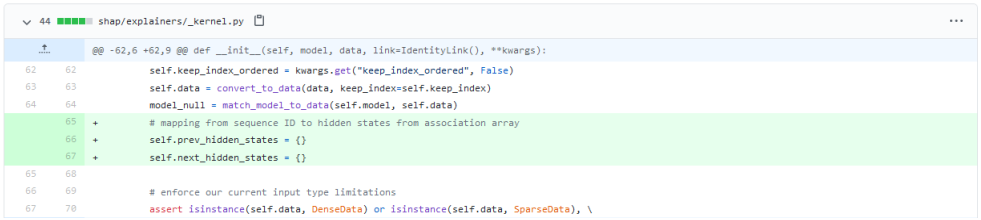
To run the modified SHAP in the MIMIC death notebook, ensure `https://github.com/Gareth001/shap` is cloned into the parent directory of this repository. Note that a fresh SHAP install via pip or conda may be required to run TreeExplainer SHAP as it requires dependencies to be compiled.

9.2 Appendix B - Annotated SHAP Modifications

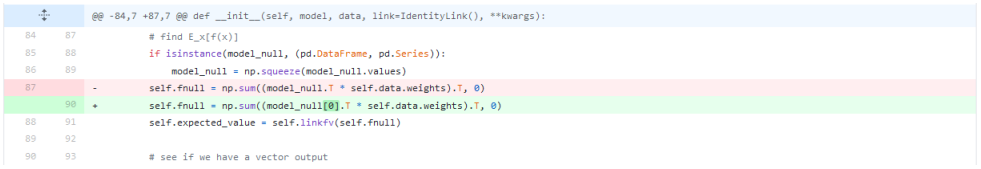
Code is available at `https://github.com/Gareth001/shap`.

File: shap/explainers/_kernel.py

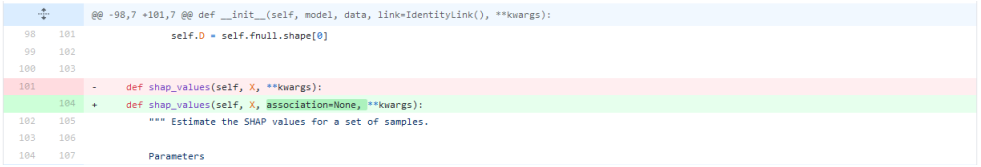
These dictionaries map sequence ID from association array to state. 'next_hidden_states' is used to store the state for the next application in the sequence, and 'prev_hidden_states' stores the state fed into the previous application.



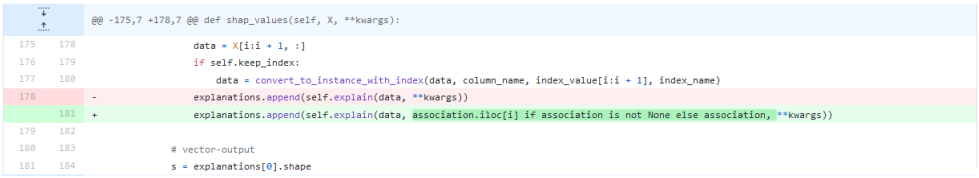
'model_null' is now of a different shape due to the changes.



Add the optional association argument to the 'shap_values' function.



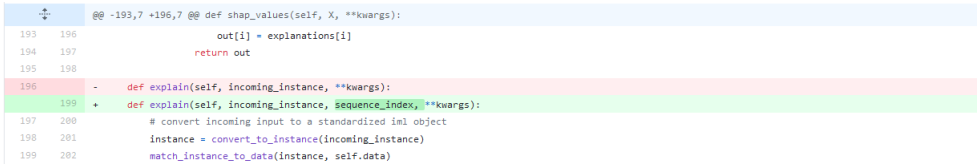
Pass through the association sequence ID if this current element (i).



```
@@ -175,7 +178,7 @@ def shap_values(self, X, **kwargs):
                    data = X[i:i + 1, :]
                    if self.keep_index:
                        data = convert_to_instance_with_index(data, column_name, index_value[i:i + 1], index_name)
-                   explanations.append(self.explain(data, **kwargs))
+                   explanations.append(self.explain(data, association.iloc[i] if association is not None else association, **kwargs))

                # vector-output
                s = explanations[0].shape
```

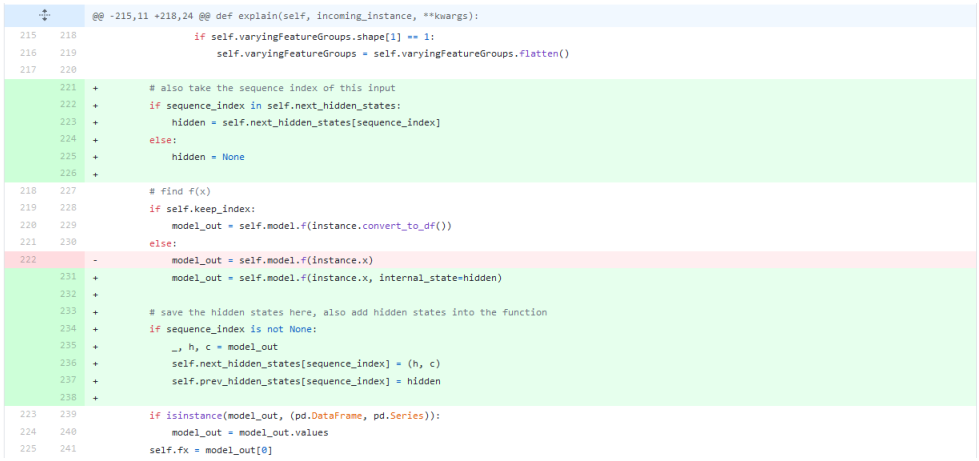
Modify 'explain' to take a sequence index.



```
@@ -193,7 +196,7 @@ def shap_values(self, X, **kwargs):
                        out[i] = explanations[i]
                return out

-    def explain(self, incoming_instance, **kwargs):
+    def explain(self, incoming_instance, sequence_index, **kwargs):
        # convert incoming input to a standardized iml object
        instance = convert_to_instance(incoming_instance)
        match_instance_to_data(instance, self.data)
```

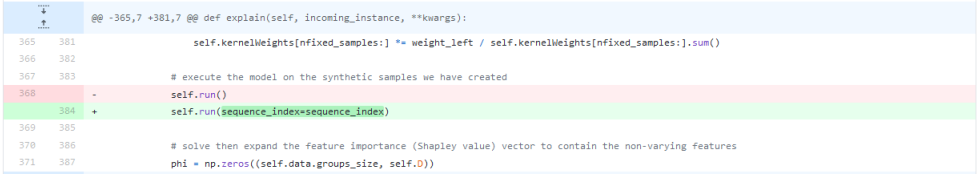
If we have a 'next_hidden_states', then use it, otherwise this is the first element from this sequence. Modify call to 'f' (the user supplied function) to pass in 'internal_state'. Finally, store the output in 'next_hidden_states' and the previously used state into 'prev_hidden_states'.



```
@@ -215,11 +218,24 @@ def explain(self, incoming_instance, **kwargs):
                    if self.varyingFeatureGroups.shape[1] == 1:
                        self.varyingFeatureGroups = self.varyingFeatureGroups.flatten()

+       # also take the sequence index of this input
+       if sequence_index in self.next_hidden_states:
+           hidden = self.next_hidden_states[sequence_index]
+       else:
+           hidden = None
+
        # find f(x)
        if self.keep_index:
            model_out = self.model.f(instance.convert_to_df())
        else:
-           model_out = self.model.f(instance.x)
+           model_out = self.model.f(instance.x, internal_state=hidden)
+
+       # save the hidden states here, also add hidden states into the function
+       if sequence_index is not None:
+           _, h, c = model_out
+           self.next_hidden_states[sequence_index] = (h, c)
+           self.prev_hidden_states[sequence_index] = hidden
+
        if isinstance(model_out, (pd.DataFrame, pd.Series)):
            model_out = model_out.values
        self.fx = model_out[0]
```

Pass through sequence index to run call.



```
@@ -365,7 +381,7 @@ def explain(self, incoming_instance, **kwargs):
                    self.kernelWeights[nfixed_samples:] *= weight_left / self.kernelWeights[nfixed_samples:].sum()

                # execute the model on the synthetic samples we have created
-               self.run()
+               self.run(sequence_index=sequence_index)

                # solve then expand the feature importance (Shapley value) vector to contain the non-varying features
                phi = np.zeros((self.data.groups_size, self.D))
```

```
@@ -492,7 +508,7 @@ def addsample(self, x, m, w):
492 508          self.kernelWeights[self.nsamplesAdded] = w
493 509          self.nsamplesAdded += 1
494 510
495     -    def run(self):
    511 +    def run(self, sequence_index=None):
496 512          num_to_run = self.nsamplesAdded * self.N - self.nsamplesRun * self.N
497 513          data = self.synth_data[self.nsamplesRun*self.N:self.nsamplesAdded*self.N,:]
498 514          if self.keep_index:
```

The model is called twice in kernel SHAP. This second call runs all of the extra samples. Therefore, we need to use the same hidden state as what we first used, which is why we save 'prev_hidden_states'. Note that 'h' and 'c' must be padded to match all of these samples.

```
@@ -502,7 +518,19 @@ def run(self):
502 518              data = pd.concat([index, data], axis=1).set_index(self.data.index_name)
503 519              if self.keep_index_ordered:
504 520                  data = data.sort_index()
505     -        modelOut = self.model.f(data)
    521 +
    522 +        if sequence_index is not None:
    523 +            # extend hidden states array for all samples
    524 +            if self.prev_hidden_states[sequence_index] is None:
    525 +                modelOut, _, _ = self.model.f(data, internal_state=None)
    526 +            else:
    527 +                h, c = self.prev_hidden_states[sequence_index]
    528 +                h = np.array([h[0]] * len(data))
    529 +                c = np.array([c[0]] * len(data))
    530 +                modelOut, _, _ = self.model.f(data, internal_state=(h, c))
    531 +        else:
    532 +            modelOut = self.model.f(data)
    533 +
506 534          if isinstance(modelOut, (pd.DataFrame, pd.Series)):
507 535              modelOut = modelOut.values
508 536          self.y[self.nsamplesRun * self.N:self.nsamplesAdded * self.N, :] = np.reshape(modelOut, (num_to_run, self.D))
```

File: shap/utils/_legacy.py Model now supplies 3 outputs.

```
4 shap/utils/_legacy.py
@@ -115,8 +115,10 @@ def match_model_to_data(model, data):
115 115          raise
116 116
117 117      if model.out_names is None:
118     -        if len(out_val.shape) == 1:
    118 +        if len(out_val) == 1:
119 119              model.out_names = ["output value"]
    120 +        elif len(out_val) == 3:
    121 +            model.out_names = ["output value", "h", "c"]
120 122          else:
121 123              model.out_names = ["output value "+str(i) for i in range(out_val.shape[0])]
122 124
```