1030 Final Report - Andrew Wang

Tuesday, October 12, 2021

Introduction:

During COVID, ventilators were a key tool used to keep patients in critical conditions alive as they were having difficulties breathing. Mechanical ventilators used in hospitals nowadays are also clinician-intensive, requiring manual tuning due to the delicate nature of assisting and replacing natural breathing.

From this dataset, the goal is to simulate a ventilator, and predict the airway pressure measured in the respiratory circuit, measured in cmH20, and so it is inherently a regression problem. The dataset itself is comprised of 6,036,000 data points, representing measurements taken at 80 points per breath, each of which lasting around 3 seconds. The 7 input features to base predictions off of are as follows[1].

This is from an ongoing public competition on kaggle.com, and approaches of other competitors are available. Two such models used - Long-Short-Term Memory[2]; and Temporal Convolutional Networks[3] - were particularly of interest.

The first approach went through some bare preprocessing and EDA, and sought to pare down the dataset by dropping columns that correlated too closely with other features already present in the dataset. The approach with the LSTM model did not split data up by its time series nature, and opted for a neural net. The structure of the model allows data to be 'forgotten' or 'remembered' and used in order to hold memory for past information and predict future results. This model ended up with a mean absolute error of 2.0754.

Given these past approaches, I might personally aim to beat the results of the LSTM model. Given the data ranges from ~-1.6 to ~64.8, a MAE of 0.5297 seems particularly difficult to surpass, especially given the deep-learning nature of the model used.

Exploratory Data Analysis:

From plotting out the distributions/frequencies of the data individually, it's clear that the R, C, and u_out features are categorical in nature, while the time_step, u_in, and pressure features are continuous. R and C have only three categories/settings each (5, 20, 50 cmH2O/L/S; and 10, 20, 50 mL/cmH2O respectively), and u_out is a binary feature indicating the control input to the exploratory solenoid valve.

Given the time-series nature of the data, one of the first plots of interest was to plot the target variable over the time step, which results in this scatterplot:

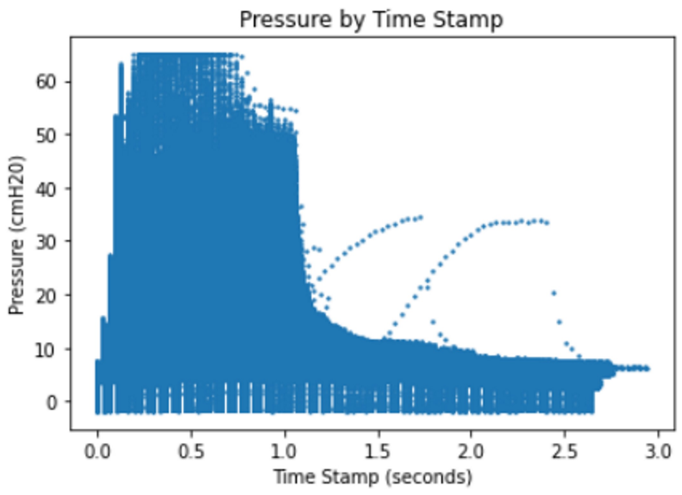


Figure 1: Scatterplot of the target variable, pressure, over time

Most of the data falls in the same general area, though there seems to be outliers with data after 1.5 seconds and above ~11-12 cmH20, which look like it could belong to only two or three distinct breaths. It's important to consider outliers: isolating the points beyond the visually determined thresholds yielded only two distinct breaths out of the 75450 total breaths in the dataset, and comparing to a subsection of the rest of the data produces the following:

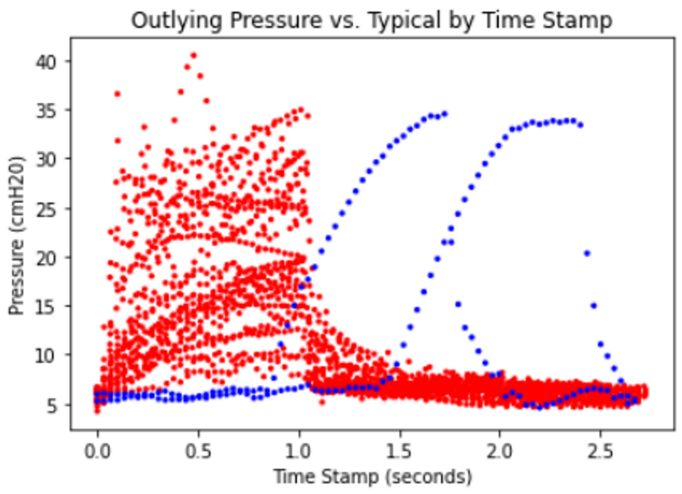


Figure 2: Scatterplot of typical data compared to outlying data on pressure plotted against time stamp

where the 'typical' data is in red, and the outlying data is in blue. Since there are only two outlying breaths out of over 75,000, I decided there was no harm in leaving them in the data, and since these still represent real breaths from real patients it is still important to leave them in.

Second, when plotting the distributions of 'u_in' and 'pressure', I noted the shapes of the distributions were somewhat similar: both were heavily right-skewed.

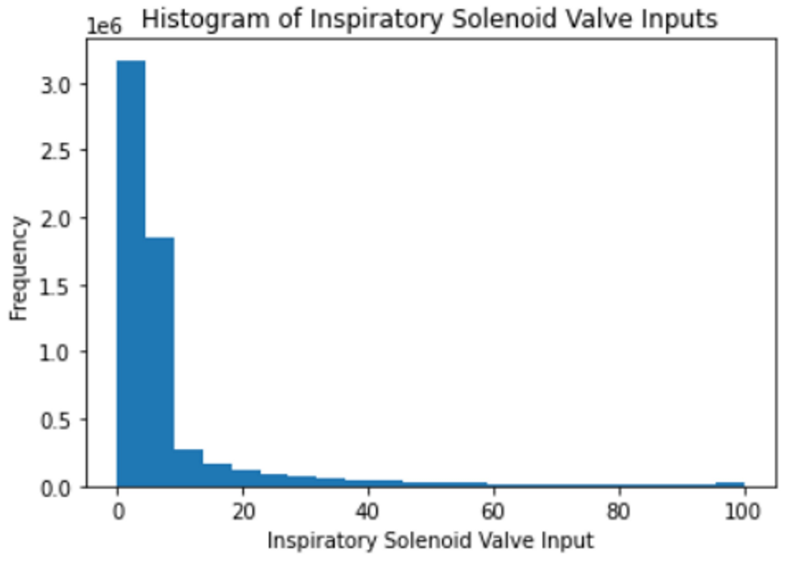


Figure 3: Distribution of u_in, or the Inspiratory Solenoid Valve Control Input. The distribution is significantly right-skewed.

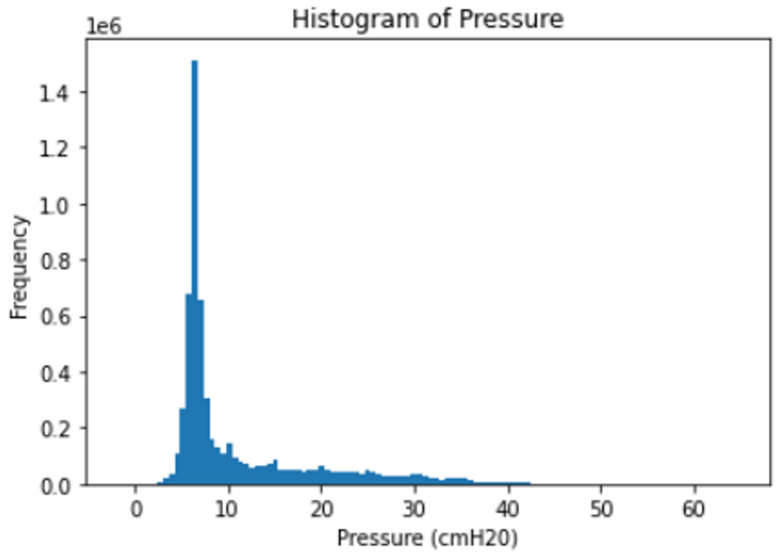


Figure 5: Distribution of the target variable, airway pressure. The distribution is significantly right-skewed.

To verify any correlation, and to look at the correlations between all features in general, I plotted the correlation matrix between the relevant features, and the feature u_in is most positively correlated with pressure, though still only weakly (0.31), so the shapes of the distributions is more coincidental, which is confirmed by the scatterplot of the two distributions:

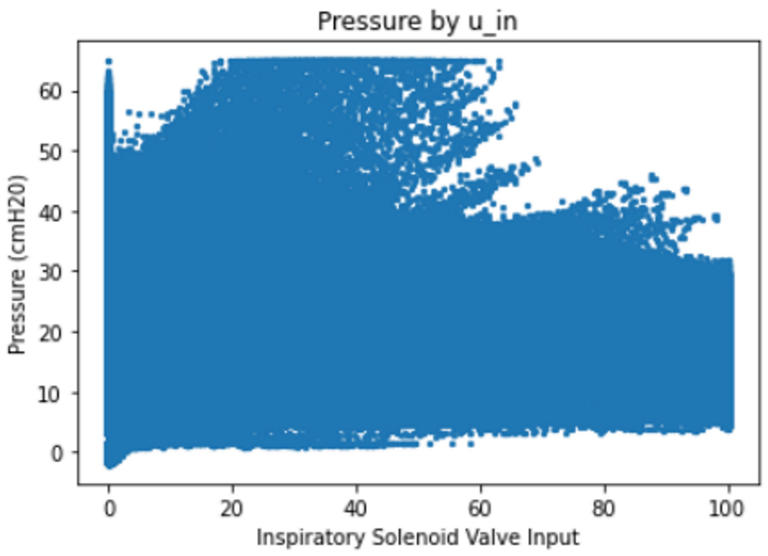


Figure 5: Scatterplot of the target variable (pressure) over u_in (solenoid valve control input). The features are slightly positively correlated.

Methods:

Data Preprocessing:

As mentioned above, of the features remaining for training, C, R, and u_out are categorical features while time_step, u_in, and pressure are continuous features. This dataset from Kaggle is relatively clean to start with: there are no missing data from any of the features.

The data is non-iid and is time-series dependent, as has a group structure: rows are grouped by the breath_id, identifying which breath they belong to. The data splitting process is simple, but it's still important to avoid a number of things: only data in the future of the training data should be in the validation and test sets, and data from each breath should not be shuffled around such that data from separate breaths do not 'leak' into other splits (for cross validation).

Because of the time series data, when predicting the target variable it's not necessarily guaranteed that the time at which the prediction is made will follow immediately after the data presented to predict on. To mitigate this issue, I utilized sklearn's TimeSeriesSplit to implement a "day forward chaining"[4] approach, where many train/validation/test splits are created with the latest day, or in this case, set of time stamps, as the test set, and all of the preceding data is used for training, like so:

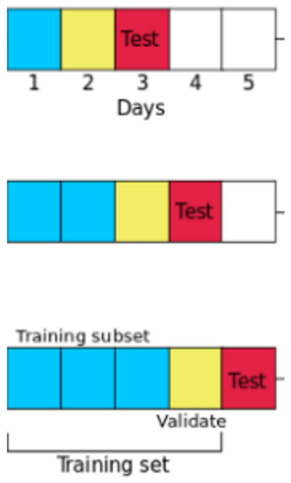


Figure 6: Visual representation of 'day forward-chaining'

applied to day-by-day time-series data

The goal is to produce a more robust estimate of the model error by averaging its performance across multiple splits, and allow the model to properly weight and learn from any time stamp in the overall dataset.

The second way to mitigate the issue was to deal with the 'gap' in the validation data. As mentioned before, a prediction may have to be made for a time stamp not immediately following the data given. In this case especially with a continuous time domain (instead of discrete days of the week, months, etc.) this is even more significant, because not all data collected are equally spaced by time stamp. By artificially introducing a gap between the training data and the validation data like in the following figure, the model will train on data in a way more consistent with test data predictions or a real-world use-case.

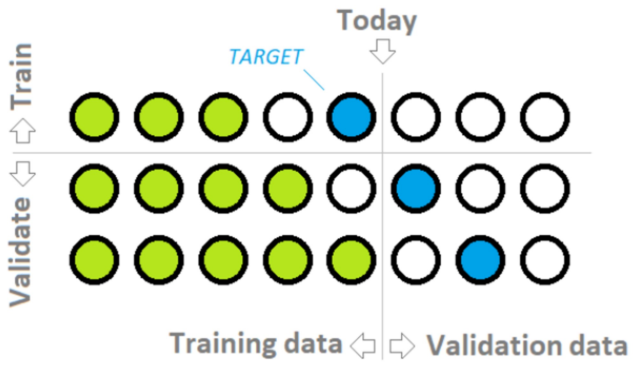


Figure 7: Visualization of the gap in validation data: adding a gap between training and validation sets allows the model error to better reflect real data.

Finally, in terms of feature preprocessing, the values of C, R, and u_id are clearly categorical, and the values of C and R are quantitively related. So, C and R were encoded using the OrdinalEncoder to attempt to preserve some of the numerical relationship between the values, while u_id being a binary feature was encoded using the OneHotEncoder. All of the continuous features were scaled using the MinMaxScaler: all of the plotted distributions of the continuous features looked skewed, and in particular u_in is on finite scale between 0 and 100 while the time_step is bound on the left by 0 with vanishingly small probability that a single breath would last significantly longer than 3 seconds. The pressure is similarly bounded: there's a limit to airway pressure levels, so the MinMaxScaler is perfect for all three of these features.

Algorithms/ML Pipeline:

Due to the volume of machine learning models that are attempted, building an ML pipeline was crucial. Though applying GridsearchCV was an attractive option, in the end it turns out there are compatibility issues between GridsearchCV and using XGBoost's early stopping functionality, so much of the pipeline was built from the ground up. As mentioned above, the data is time-series, and so the overall pipeline steps are:

1. Set up arrays to record information across multiple runs from different random seeds.
2. For each random seed, split the data using TimeSeriesSplit into four splits, with each consecutive split making use of a longer history of past data in the training data.
3. Apply the data preprocessing from above and fit it to the training data before transforming the training and validation sets. Manually search through hyperparameters of interest for each model, and for each random seed determine which set of hyperparameters yielded the best results from the validation set. Apply the best model from each random seed to the test set to observe performance on unseen data.
4. Make visualizations from the results, including determining model variance over the sweeps of

hyperparameter values for the purpose of observing over or underfitting, as well as plots of feature importance for each model. Errorbars work very well for visualizing model uncertainty, while bar plots work just fine for graphing feature importance. Feature importance was only determined for the best preforming model, with mostly built-in methods.

5. Make a final determination on what the best model is based on the chosen evaluation metric.

The four models elected to be pursued were: (1) The ElasticNet-penalized linear regression model; (2) the K-Nearest-Neighbor regressor; (3) the RandomForest regressor; and finally, (4) XGBoost. The size of the dataset is considerably large, well over one million data points, and so while models such as SVR were considered, in the end it was simply more economical to stick with models that scaled better than SVR. All of these models can be evaluated under the same metric for a regression problem, and between mean-squared error and R2-score, in the end R2-score seemed like a slightly more robust metric to use that invariant to scaling issues, so the R2-score became the metric of choice for hyperparameter selection as well as final evaluation of model performance[6].

In terms of hyperparameter tuning, ElasticNet is straightforward: just the L1-penalty term and the weight ratio were explored. For K-Nearest-Neighbors, the method of measuring/weighting distance (uniform vs. distance) was varied, as well as the number of neighbors considered by each point. RandomForest had more features to work with, though in the end only max_depth and max_features seemed appropriate to tackle without being lost in the nuance of adjusting parameters such as min_leaf_nodes; these two parameters deal with the two most salient aspects of RF's outside of n_estimators, which is generally understood to do better with more trees for the most part. Finally for XGBoost, both penalty terms were varied as well as the learning rate and the max_depth. The values swept are below:

Model:	Parameter Varied:	Values:
ElasticNet Regressor	Alpha penalty term	Logspace(-3, 3, 7)
	L1 ratios	Linspace(0, 1, 4)
KNN Regressor	Weights	['uniform', 'distance']
	N_neighbors	[2, 5, 10, 15, 30]
RandomForest Regressor	Max_depth	[5, 20, 35, None]
	Max_features	['auto', 'sqrt', 'log2']
XGBoost	Learning rate	[0.01, 0.03, 0.1]
	L1 regularization	Logspace(-2, 2, 3)
	L2 regularization	Logspace(-2, 2, 3)
	Max_depth	[1, 3, 10, 15]

Table 1: Hyperparameter Search Values

Uncertainty was measured both across features individually in the model, as well as measured due to splitting issues through the use of multiple random seeds and separate splits. Especially due to the nature of the TimeSeriesSplit, model performance drastically changes as more training data is fed in through later splits, and so the variation in performance is quite higher than might be expected in a normal K-fold splitting as can be seen in an example below:

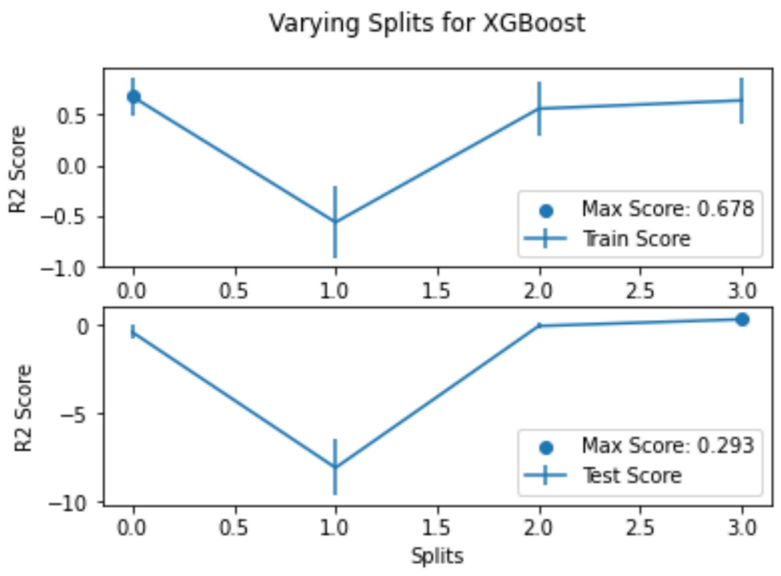


Figure 8: Normalized Performance Across Splits for XGBoost

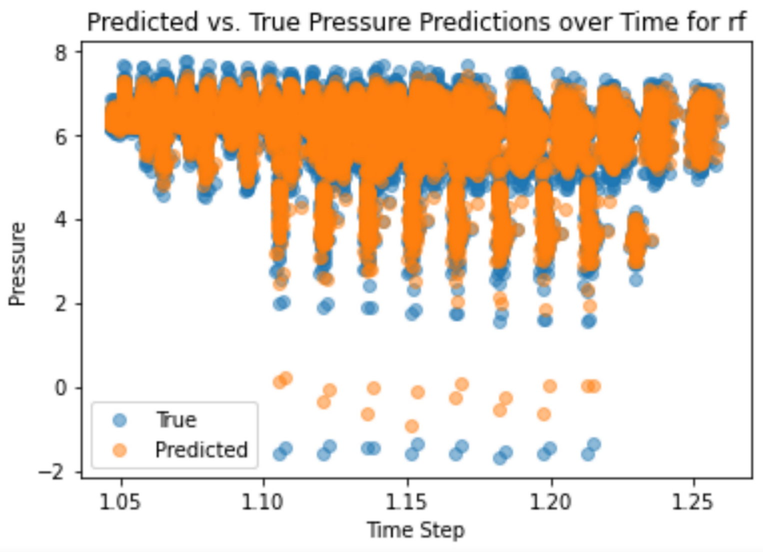
Results:

In terms of highest average performing, as well as highest peak performing model on this dataset, XGBoost generally yielded the best R2 scores followed closely by RandomForest, then KNN. Linear regression as expected did not fare as well, and in some manners seemed to do almost as poorly as a simple baseline model from sklearn that predicts the mean, or the median of the dataset. Tabulated below are the results: with the R2 score, the baseline prediction of the mean yields a score of 0, and standard deviations are calculated based on each model's own uncertainty values:

Model:	Mean	Stddev	Stddev Above Average:
Baseline:	0	0	0
ElasticNet	0.36	4.3	0.084
KNN:	0.57	6.5	0.087
RandomForest:	0.609	0.307	1.98
XGBoost:	0.624	0.222	2.81

Table 2: Comparing R2 Score Performance Across Models

Though looking at the predictions themselves plotted by time, it does seem as if the RandomForest visually appears to do better than XGBoost. This discrepancy is likely due to the fact that the entire dataset is plotted, so even if it looks like the overall distributions are similar they may not be corresponding to the same breaths, and so may still be the complete wrong predictions:



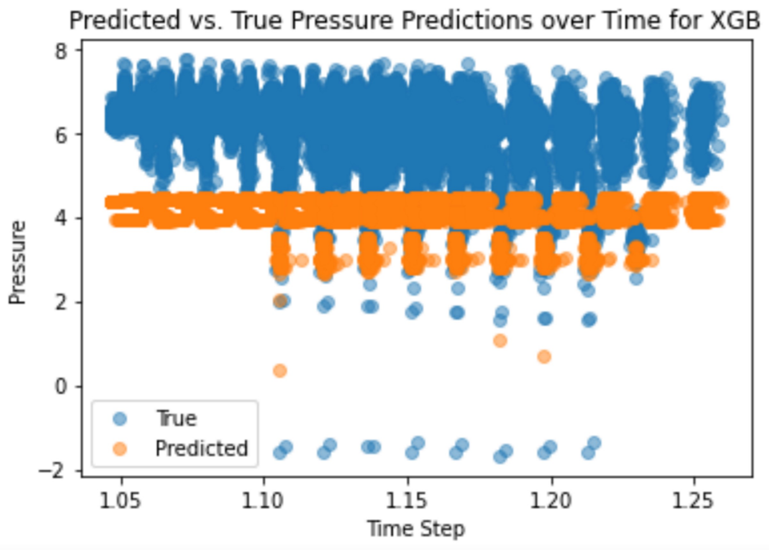


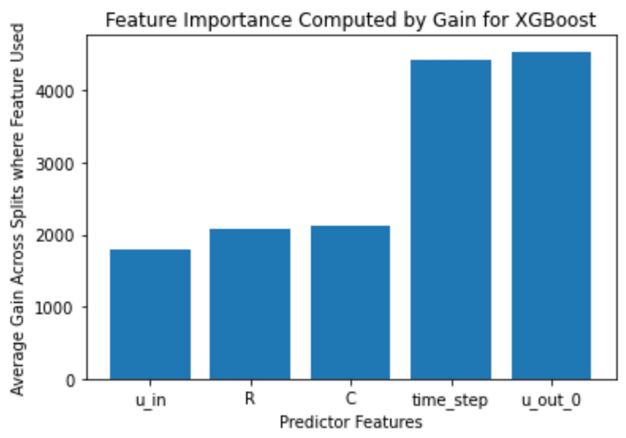
Figure 8: Comparing Visual Predictions of RF vs XGBoost

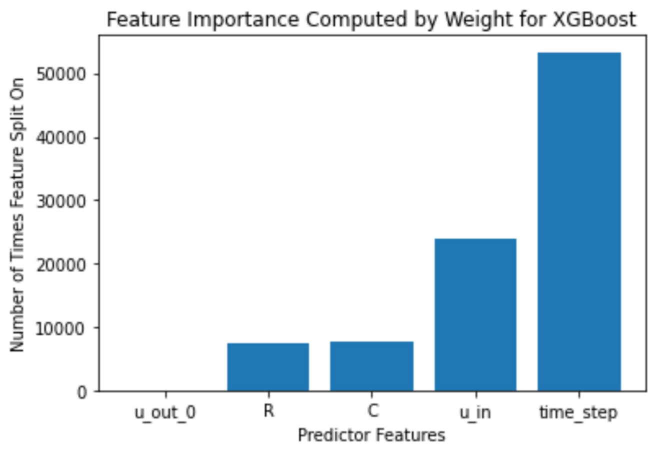
Feature Importance:

Global feature importance was calculated in three methods for the best performing model, XGBoost (learning rate=0.01, reg_alpha=0.01, reg_lambda=0.01, max_depth=10). The three methods were:

1. Gain: computing the average gain across splits where a particular feature was used;
2. Weight: The number of times a feature was split on; and
3. Permutation: randomly shuffling a particular feature to see how much it impacts the predictive power of the model.

Interestingly, all three metrics gave different results:





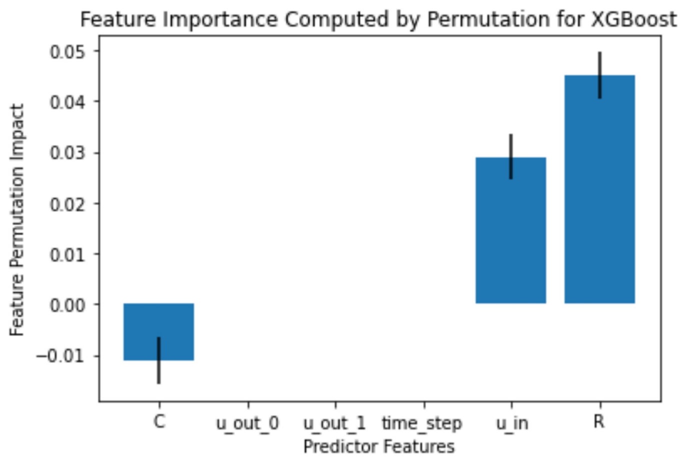


Figure 9: Comparison of Global Feature Importance Calculations

I had expected the time to be the top most important feature for all three, and though it is at the top for the first two it's curious that under the permutation test it doesn't appear to matter that much. Even looking at correlation plots it doesn't appear that R is highly correlated with time, so I'm not quite sure what the discrepancy in expectation is caused by here.

In terms of local feature importance, the SHAP metric was used. Using the Shapley values, it's clear that here at least time is far and away the most important metric:

SHAP Feature Importances for XGBoost

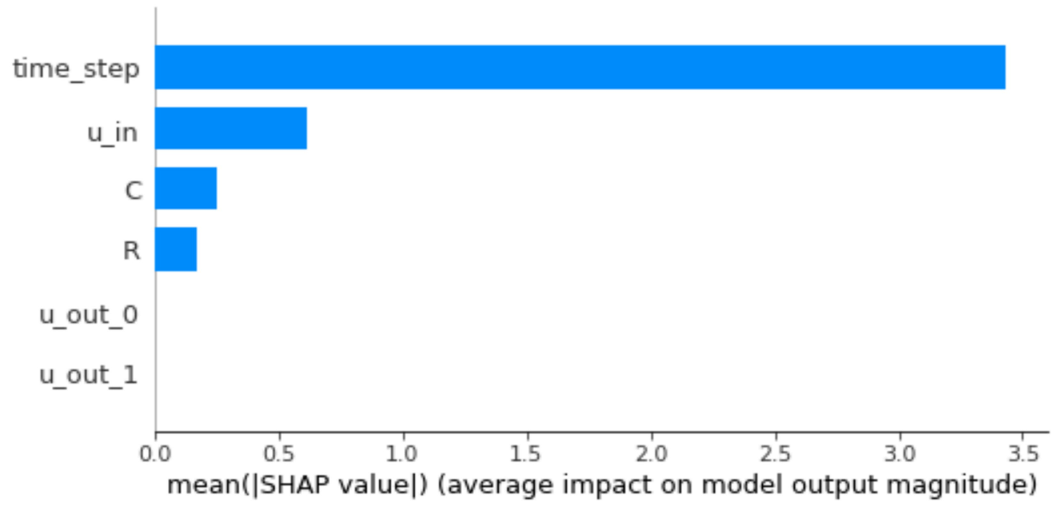


Figure 10: Comparison of Local Feature Importance by SHAP

Outlook:

Though XGBoost was found to be the best performing out of the other models, the overall performance is still quite unsatisfactory with an R2 score hovering below 0.70. In terms of just performance, deep learning methods such as recurrent neural networks or long-short-term-memory architectures are generally the more modern approach towards some time-series based problems, though the time in this dataset is still only comprised of 80 data points across a little over one single minute of time. Beyond simply trying more computationally complex architectures, a deeper comb of hyperparameters would have been beneficial, though more time would have been needed: with the size of the dataset any sort of parameter searching went very slowly. Perhaps making use of a compute cluster at the university or paying for Google Cloud computational resources is another thing to try in the future. In terms of interpretability, overall the methods of determining feature importance, quantitatively measuring improvement over the baseline model, and demonstrating some expected aspects of the training process such as the increase in performance over splits is not unsatisfactory. The real focus looking ahead should be towards more useful datasets: at the end of the day, this data does not come from live patients, and so is still a step abstracted away from a real-world setting where the variation may be higher and gathered data more rich, including anything from other basic physical vitals to medical history, environmental factors, and mental health. There is always more that can be done when it comes to the healthcare setting, whether on the modeling side or the data collection and engineering aspect.

References:

[1]Dataset: Google Brain - Ventilator Pressure Prediction | Kaggle
[2]LSTM Model Approach: https://www.kaggle.com/shubham9455999082/prediction-using-lstm/log
[3]TCN Model Approach: https://www.kaggle.com/carlmcbrideellis/temporal-convolutional-network-using-keras-tcn/log
[4]Day Forward-Chaining: https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9
[5]Validation Data Gap: https://towardsdatascience.com/proper-validation-of-a-time-series-model-5c1b54f43e60
[6]Mean Squared Error or R-Squared - Which one to use? - Data Analytics (vitalflux.com)

Github:

https://github.com/Andrew-X-Wang/DATA1030_Project