1030 Midterm 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. Many well-intentioned, but misinformed and lacking-context solutions were proposed, some of which by automobile manufacturers that tried to engineer monotonous, mechanical ventilators to help plug the gap in hospitals' inventory of medical-grade machines. However, it was pointed out very quickly that a machine that pumps air at a set pressure with a set rhythmic beat is not at all what is needed for intubated patients, but rather, a tailored method that better emulates natural breathing is optimal for boosting survival rates.

Mechanical ventilators used in hospitals nowadays are also clinician-intensive, requiring manual tuning due to the delicate nature of assisting and replacing natural breathing. The hope is to remove some of the reliance on nurses and hospital staff to man the ventilators and manually tune them to each patient, and look towards machine-learning techniques to automate this process.

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]:

> id - globally-unique time step identifier across an entire file
> breath_id - globally-unique time step for breaths
> R - lung attribute indicating how restricted the airway is (in cmH2O/L/S). Physically, this is the change in pressure per change in flow (air volume per time). Intuitively, one can imagine blowing up a balloon through a straw. We can change R by changing the diameter of the straw, with higher R being harder to blow.
> C - lung attribute indicating how compliant the lung is (in mL/cmH2O). Physically, this is the change in volume per change in pressure. Intuitively, one can imagine the same balloon example. We can change C by changing the thickness of the balloon’s latex, with higher C having thinner latex and easier to blow.
> time_step - the actual time stamp.
> u_in - the control input for the inspiratory solenoid valve. Ranges from 0 to 100.
> u_out - the control input for the exploratory solenoid valve. Either 0 or 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.

The second approach did more heavy feature engineering, and focused on the u_in feature. It added in columns detailing summary statistics (last value, mean, median, min, max, range, sum), as well as a column that 'lagged' the data: shifting all of the time series data back by 2 time stamps in order to draw more direct relationships with past data. This approach also chose to drop features, though only removing the categorical u_out, before scaling the features. The data was split in this approach, though only using kfold: again opting not to split in a meaningful way based on the time-series nature of the data. The model used was a TCN, which makes use of 'causal convolutions', where an output at a particular time is only convolved with other elements of time t and earlier from the previous layer of the CNN. It also uses 'dilated convolutions' which increase a larger receptive field for each neuron, allowing it to look back further in the history of the network. This approach was more robust, from the feature engineering to the complexity of the model, and achieved a mean absolute error of 0.5297, much improved over the earlier LSTM model results.

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:

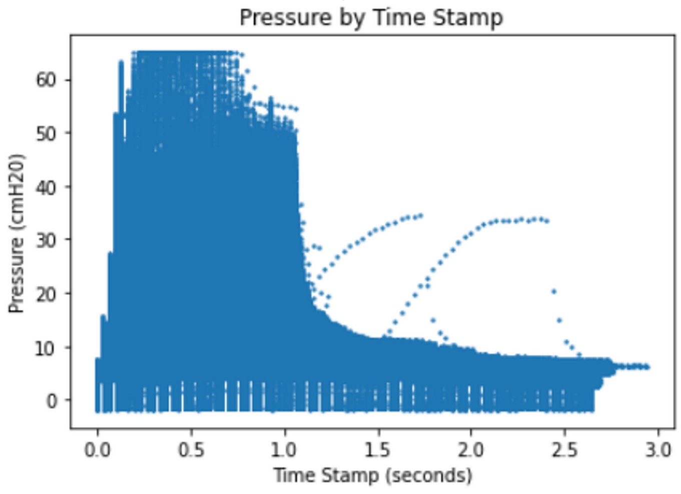

Figure 1: Scatterplot of the target variable, pressure, over time

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:

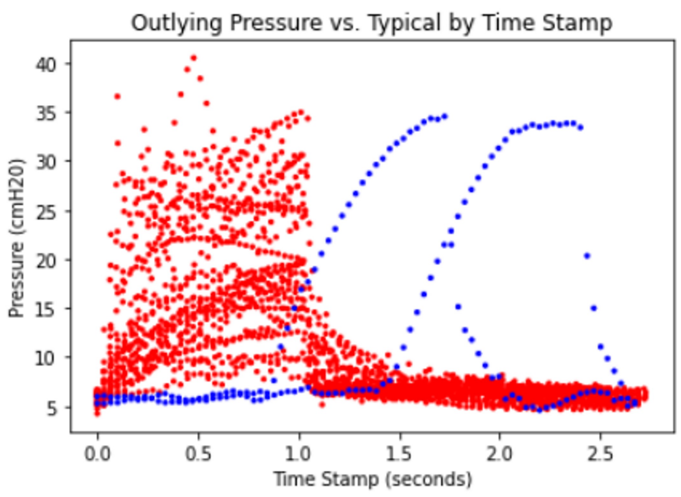

Figure 2: Scatterplot of typical data compared to outlying data on pressure plotted against time stamp

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.

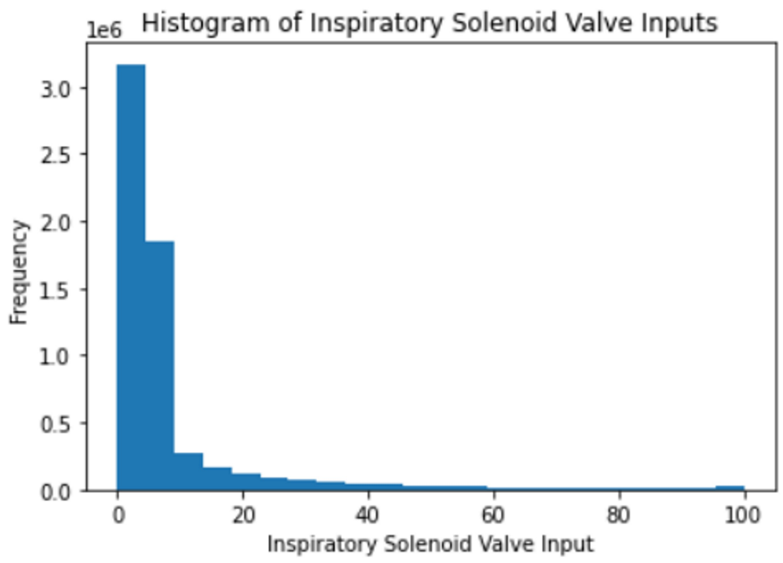

Figure 3: Distribution of u_in, or the Inspiratory Solenoid Valve Control Input. The distribution is significantly right-skewed.

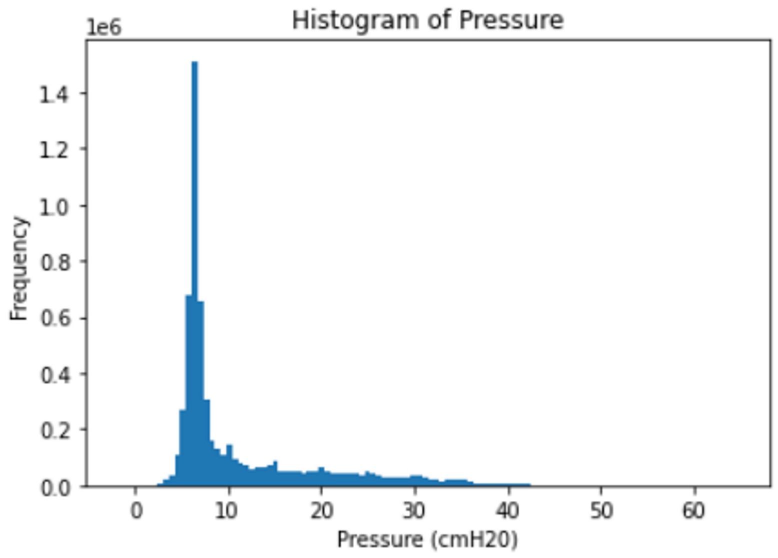

Figure 5: Distribution of the target variable, airway pressure.
The distribution is significantly right-skewed.

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:

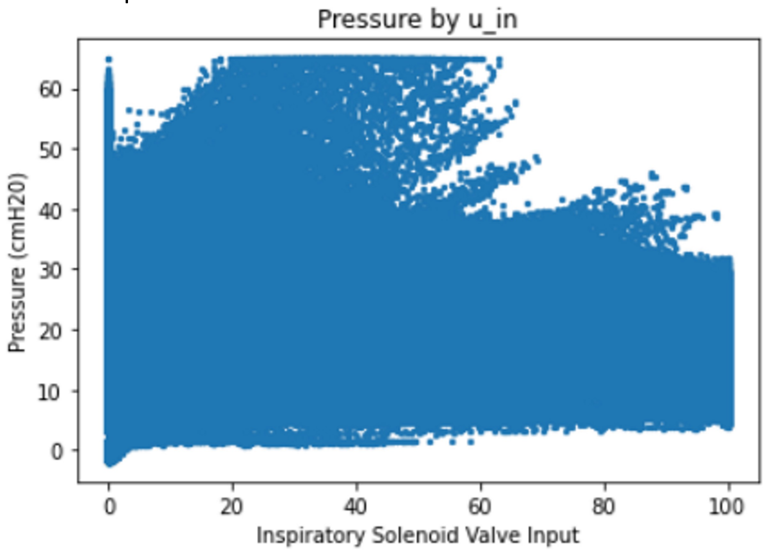

Figure 5: Scatterplot of the target variable (pressure) over u_in (solenoid valve control input).
The features are slightly positively correlated.

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:

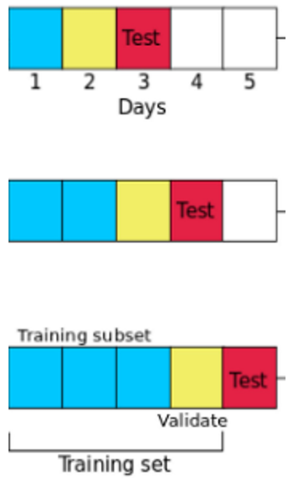

Figure 6: Visual representation of 'day forward-chaining' applied to day-by-day time-series data

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.

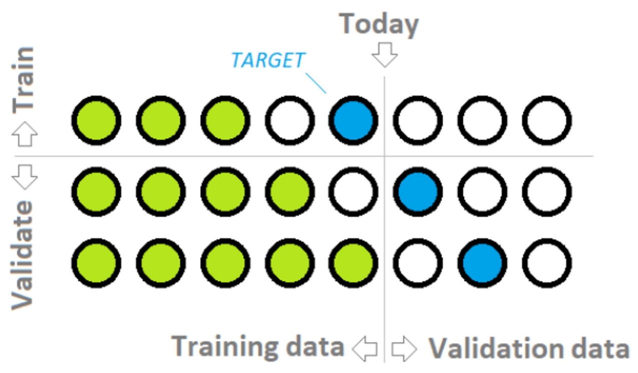

Figure 7: Visualization of the gap in validation data: adding a gap between training and validation sets allows the model error to better reflect real data.

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.

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

Github:

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