8017 GROUP PROJECT
Ventilator Pressure Prediction

BIAN Lin 3036047286
HUANG Yu 3036044545
YE Jiahui 3036044064
YI Xunming 3036044430
YUAN Ye 3036044026

Abstract

Ventilators can help maintain vital signs in patients who cannot breathe on their own, but how they are mechanically ventilated requires close attention from healthcare professionals. Ventilators are used less frequently in the absence of sufficient healthcare professionals. This paper uses machine learning and deep learning algorithms to find the relationship between breathing and lung pressure by simulating real lung breathing, enabling the ventilator to act more efficiently on the patient. By comparing the difference in effectiveness between different models, the optimal model is found to help make the ventilator better available in times of pandemic influenza.

Contents

1 Background & Motivation

When a patient experiences difficulty breathing, doctors may use a ventilator to deliver oxygen to the lungs through a tube inserted into the windpipe of a sedated patient. However, mechanical ventilation can be a demanding procedure that requires the close attention of medical professionals. This was particularly evident during the initial stages of the COVID-19 outbreak. Additionally, the expense of developing new techniques to control mechanical ventilators is often prohibitively high, even before clinical trials begin. To mitigate these challenges, high-fidelity simulators could potentially reduce this barrier.

The objective at hand is to create a simulation of lung respiration and integrate it with a ventilator that induces sedation in the patient's lungs. If successful, this approach could help overcome the financial obstacle to developing novel ways to manage mechanical ventilators and alleviate the pressure on medical professionals during the pandemic and beyond. This, in turn, could make ventilator therapy more accessible for patients who require assistance with breathing.

2 Dataset Description

The information was generated using a modified open-source ventilator linked to an artificial test lung through a respiratory circuit. The diagram below depicts the configuration, with the two control inputs highlighted in green and the state variable (airway pressure) to be predicted in blue. The primary control input is a continuous variable ranging from 0 to 100, signifying the percentage of the inspiratory solenoid valve open to allow air into the lung. At 0, the valve is entirely closed, and no air is allowed in, while at 100, it is fully open. The second control input is a binary variable that indicates whether the exploratory valve is available (1) or closed (0), which dictates the amount of air released.

The time series data corresponds to a breath lasting approximately three seconds. The files are structured so that each row corresponds to a single time step within a breath and provides information about the two control signals, the resultant airway pressure, and pertinent lung attributes described below, as well as the working principle of the ventilator.

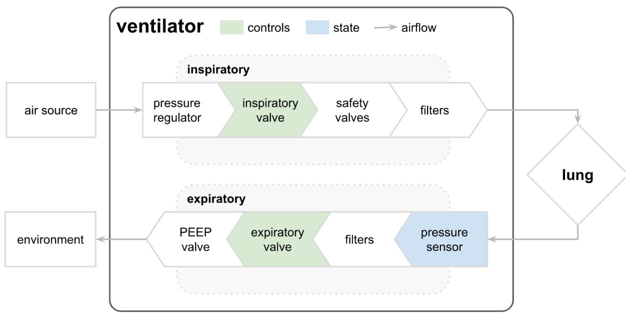


Figure 1: Workflow of the ventilator

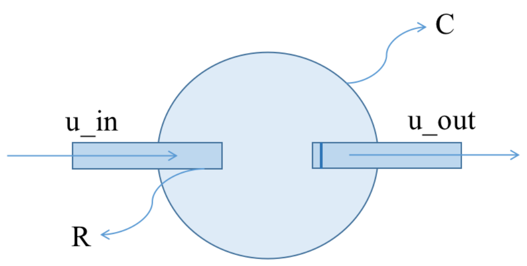


Figure 2: Schematic diagram of the working principle of the ventilator

The columns have the below definition.

- id - globally-unique time step identifier across an entire file
- breath-id - globally-unique time step for breaths

- R - lung attribute indicates the airway's restriction (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 a higher R being harder to blow.

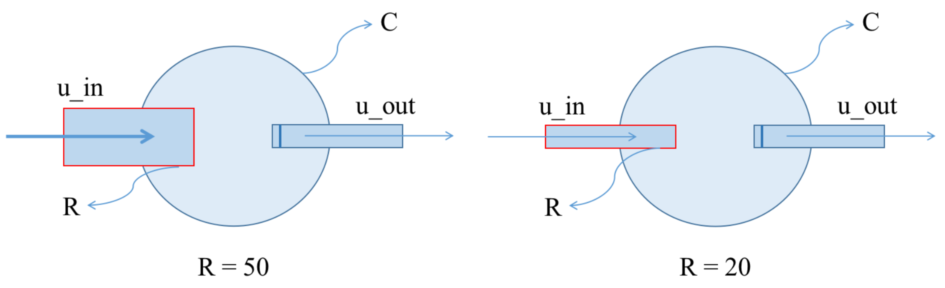


Figure 3: Schematic diagram of R

- C - lung attribute indicates the lung's compliance (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 a higher C having thinner latex and easier to blow.

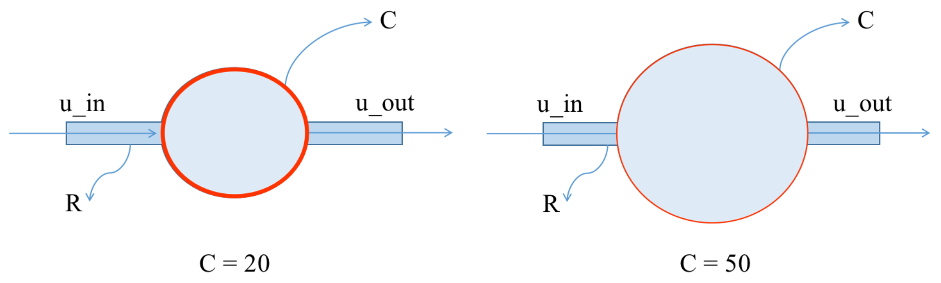


Figure 4: Schematic diagram of C

- 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.

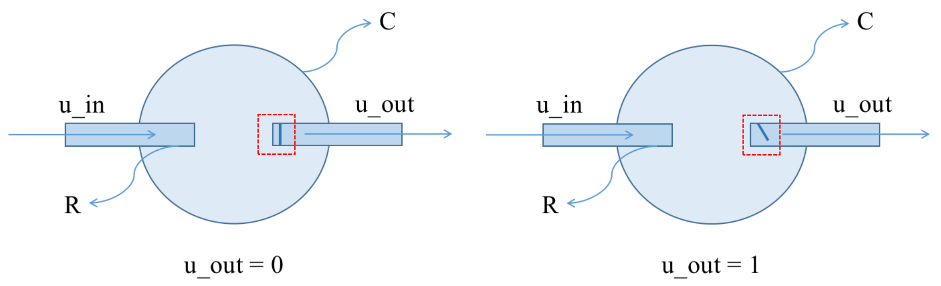


Figure 5: Schematic diagram of u_out

- pressure - the airway pressure measured in the respiratory circuit, measured in cmH2O.

3 EDA

3.1 Univariate analysis

breath-id

The figure below shows the trend of pressure, u_in, and u_out in some breath_id's over time. (In order to see the change of u_out more clearly, that is, to perform scaling, we multiply the value of u_out by 10)

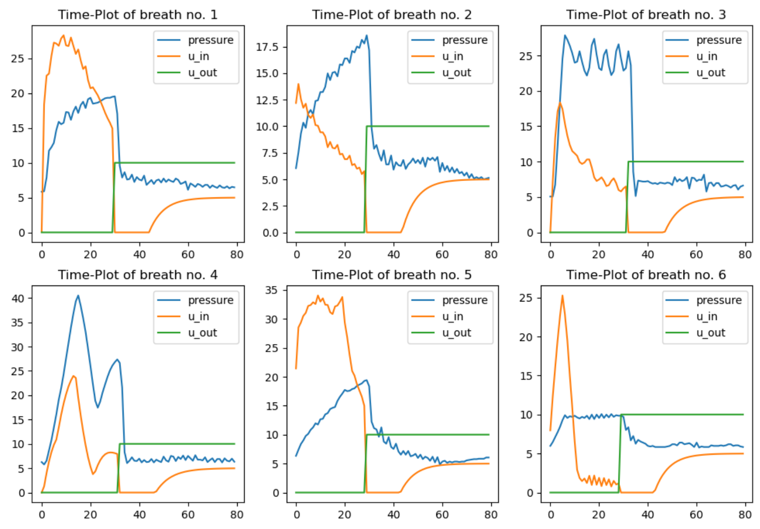


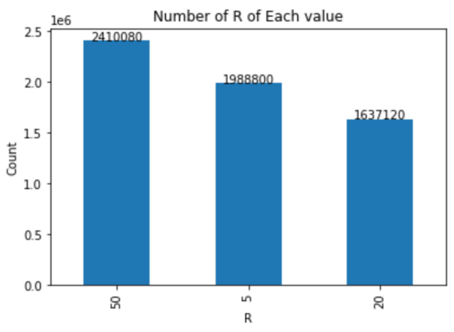
Figure 6: Breath plot

As breath-id is the globally-unique time step for breaths, it will be a distinguished field for us to do further analysis.

- The total records of the dataset: 6036000
- The total breathes of the dataset: 75450
- The number of steps for each breath: 80

R

It is found that there are only 3 values for R in the training set: 5, 20, and 50 (It's the same in the testing set). And R is also the same in a complete breath.



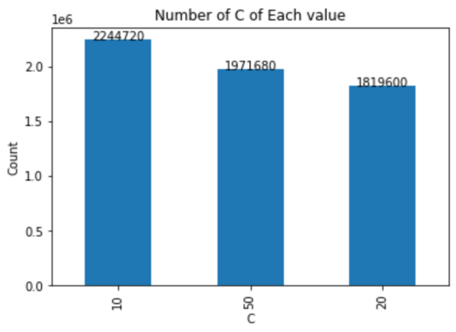


Figure 7: EDA-R

Figure 8: EDA-C

C

The same as R, there are only 3 values for C in the training set: 10, 20, and 50 (It's the same in the testing set). C is the same in a complete breath.

u-in

The variable u_in has 1432253 0 values. In order to avoid interference with the overall data, we remove it, and then draw the histogram and box-plot of u_in as shown below.

It can be seen from the figure the distribution of u_in is relatively uneven, and the specific performance is that the values are concentrated on the left side, which also leads to a small mean value, and at the same time, the sparse outliers values are very large.

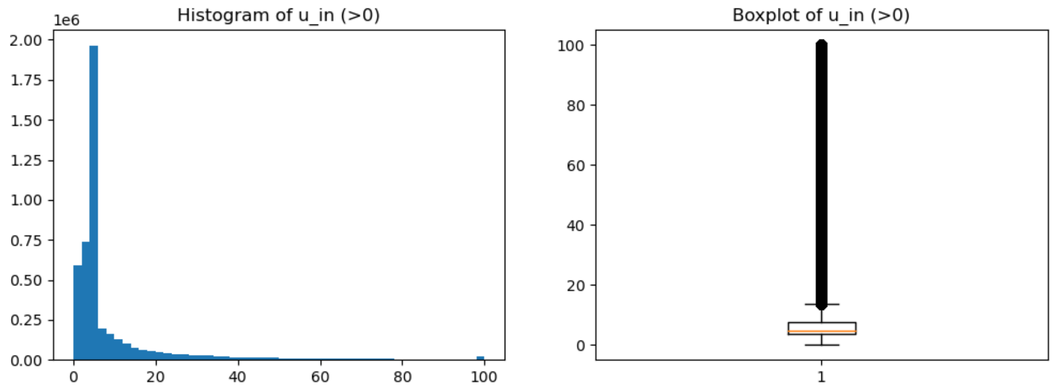


Figure 9: EDA-u_in

u-out

There are only two values of u_out, 0 and 1, representing the valve's closing and opening, respectively. In the sample, a total of 2,290,968 values are 0, and a total of 3,745,032 values are 1.

pressure

The histogram and box-plot of the variable pressure are shown in the figure below. It can be found that the distribution of pressure and u_in present similar characteristics. The data is concentrated on the left side, but there are large outliers on the right side.

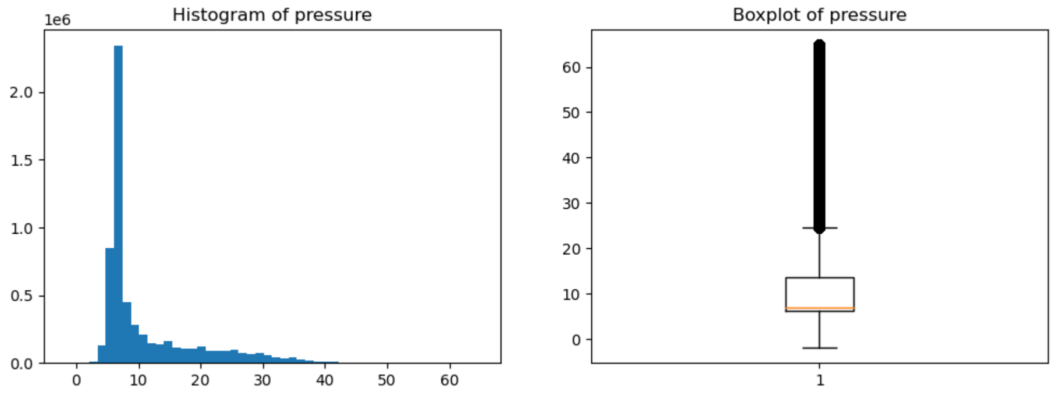
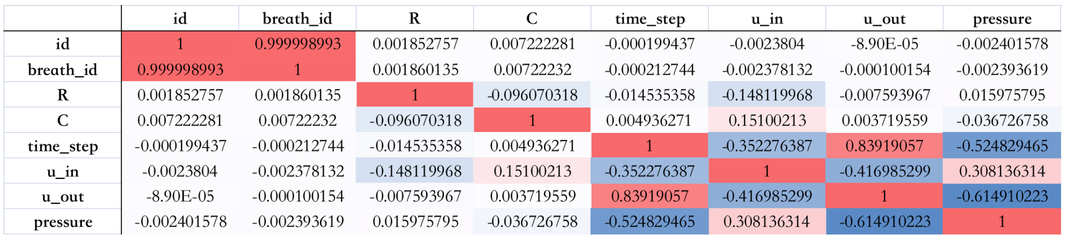


Figure 10: EDA-pressure

3.2 Bivariate analysis

We draw the correlation coefficient matrix between the variables. Among them, red represents a positive correlation between variables, blue represents a negative correlation between variables, and the darker the color, the more significant the correlation.



	id	breath_id	R	C	time_step	u_in	u_out	pressure
id	1	0.999998993	0.001852757	0.007222281	-0.000199437	-0.0023804	-8.90E-05	-0.002401578
breath_id	0.999998993	1	0.001860135	0.00722232	-0.000212744	-0.002378132	-0.000100154	-0.002393619
R	0.001852757	0.001860135	1	-0.096070318	-0.014535358	-0.148119968	-0.007593967	0.015975795
C	0.007222281	0.00722232	-0.096070318	1	0.004936271	0.15100213	0.003719559	-0.036726758
time_step	-0.000199437	-0.000212744	-0.014535358	0.004936271	1	-0.352276387	0.83919057	-0.524829465
u_in	-0.0023804	-0.002378132	-0.148119968	0.15100213	-0.352276387	1	-0.416985299	0.308136314
u_out	-8.90E-05	-0.000100154	-0.007593967	0.003719559	0.83919057	-0.416985299	1	-0.614910223
pressure	-0.002401578	-0.002393619	0.015975795	-0.036726758	-0.524829465	0.308136314	-0.614910223	1

Figure 11: Correlation coefficient matrix

It can be seen from the figure breath_id and id have a high linear correlation, which is very reasonable, as breath_id can be directly calculated from id, and this property is often used in subsequent discussions. u_out and time_step also show an obvious linear correlation, so it can be inferred that u_out always changes close to a certain time, the outlet valve is always opened at a specific time. There is also a significant negative correlation between u_out and pressure, which shows that the opening of the outlet valve can significantly reduce the pressure in the ventilator, which is also consistent with daily experience.

4 Baseline

In this section, we provide some benchmarks for subsequent comparison. The baseline uses random guess and linear regression to get.

4.1 Two modes of data structure

Before going further, it's necessary to introduce the two modes of the data structure. As shown in section 3, our dataset contains many independent breath sequences, and the data structure is:

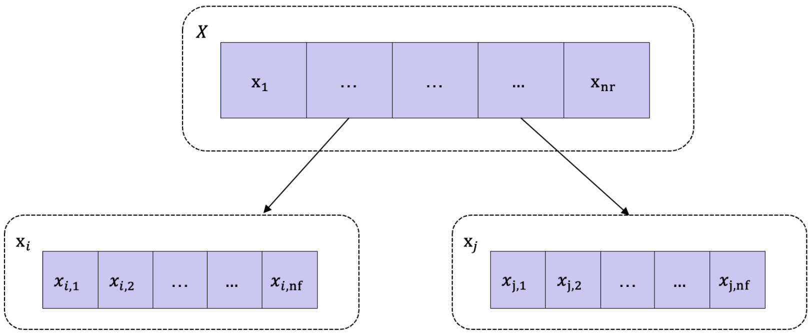


Figure 12: Visualization of data

For an individual breath X,

$$X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{nr})$$

where $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,nf})$, and nr is the number of records for each independent breath (In this project, it is 80 for all breathes), nf is the number of features for each record (In this project, before feature engineering, it is 5 and after feature engineering, it is 39).

Based on this, there are two different modes for data

- Treat each record as a sample for model fitting, which means the inputs will be $\mathbf{x} \in R^{n \times nf}$, where n is the number of records to input. It's suitable for decision tree and some ensemble models.
- Treat each breath as a sample for model fitting, which means the inputs will be $\mathbf{X} \in R^{n \times nr \times nf}$, where n is the number of breathes to input. It's suitable for CNN and RNN.

4.2 Dataset splitting

We split the original dataset into three subsets: training set, validation set and testing set with a split ratio of 2: 1: 1. The split is based on breath (X mentioned in section 4.1). After splitting, The structure of the three subsets are:

- training set: 37725 breathes and 3018000 records
- validation set: 18862 breathes and 1508960 records
- testing set: 18863 breathes and 1509040 records

4.3 Criterion

In order to compare the performance between models, we selected five commonly used evaluation criteria, namely R-squared, MSE, MAE, MAPE and SMAPE.

R-squared

R-squared is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable, which means R-squared shows how well the data fit the regression model. The formula of R-squared can be expressed as follows:

$$SS_{total} = \sum_i (y_i - \overline{y})^2$$

$$SS_{residual} = \sum_i (y_i - \hat{y})^2 = \sum e_i^2$$

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$$

Where $\overline{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$

MSE

Mean squared error (MSE) measures the amount of error in statistical models. It assesses the average squared difference between the observed and predicted values. When a model has no error, the MSE equals zero. As model error increases, its value increases. Its formula can be expressed as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MAE

Similar to MSE, Mean absolute error (MAE) is also a measure of the error of statistical models. The difference is that it uses the average of the absolute value of the difference between the predicted value and the true value, rather than the mean of the square. Below is its formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

MAPE

Mean absolute percentage error (MAPE), is another measure of prediction accuracy of a forecasting method in statistics. It usually expresses the accuracy as a ratio defined by the formula:

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}|\frac{y_i - \hat{y}_i}{y_i}|$$

SMAPE

Symmetric mean absolute percentage error (SMAPE) is another accuracy measure based on percentage (or relative) errors, which is calculated by the following formula:

$$SMAPE = \frac{100\%}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}$$

4.4 Random Guess

In the part of random guess, we hope our estimates can be as close as possible to the overall distribution characteristics of the training data while introducing certain randomness as well, then use this mode to fit the validation set. For this reason, we designed two parts: Within-breath fit, and Between-breath guess. Within-breath fits specifically examines the timing relationship of samples within a breath_id, and Between-breath explores the distribution relationship between samples at the same timestamp between different breath_ids, and we adjusted the weight of the linear combination of the two through the coefficient α, which means:

$$final\_guess = \alpha \times WithinBreath\_fit + (1 - \alpha) \times BetweenBreath\_guess$$

Where $0 \leq \alpha \leq 1$

When $\alpha = 0$, the above formula degenerates into a Between-breath guess, and when $\alpha = 1$, the above formula degenerates into a Within-breath fit.

Within-breath Fit In order to fit the characteristics of the data in breath_id, we first calculate the average value of the corresponding positions of each sequence in the training set to obtain the sample mean sequence, as shown in the green curve in the figure below.

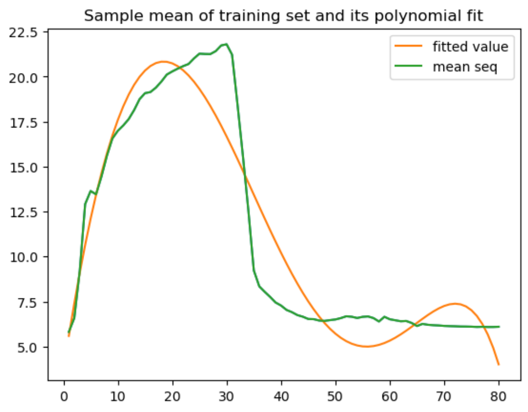


Figure 13: Sample mean of training set and its polynomial fit

It can be found that such a curve is difficult to approximate with a certain distribution, and there is an obvious correlation within the sequence, which does not follow the independence assumption of random sampling, so we consider using a polynomial function to fit it. The order of the polynomial is a hyperparameter. Here we choose five, and the fitting result is shown in the yellow curve in the figure above.

We use the same curve to fit other breath_id sequences in the training set, and the results are shown in the figure below. It can be seen that the polynomial function follows the trend of the true value roughly, but only as a rather rough approximation. Also, we use this curve to estimate the Within-breath fit of each breath_id in the validation set. This part of the information reflects the prediction effect of the model if it directly "remembers" the average value of the entire training set.

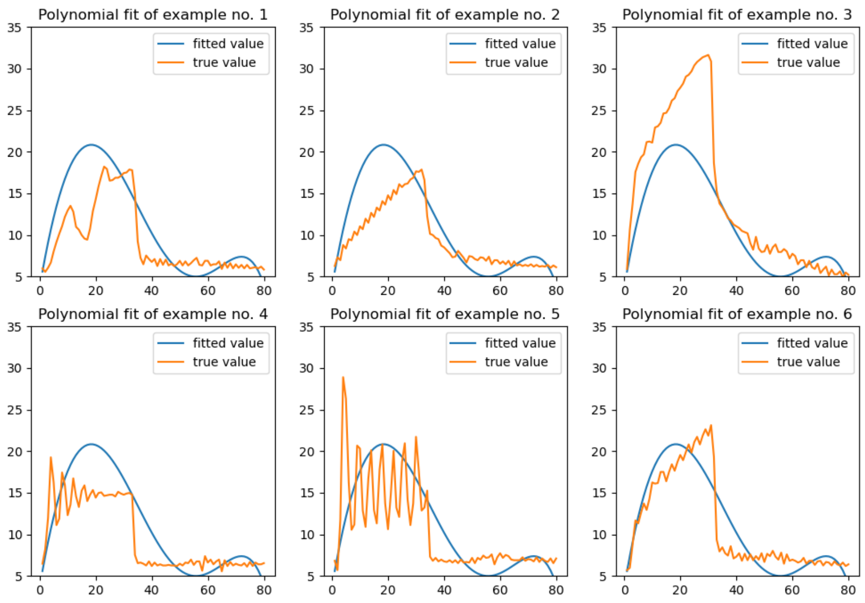


Figure 14: Within-breath polynomial fit

Between-breath Guess

Between-breath guess is based on an important assumption, if each breath is regarded as a whole sample, they should be independent of each other and generally follow the normal distribution. That is to say, for each breath_id, the pressure value under the same timestamp is normally distributed.

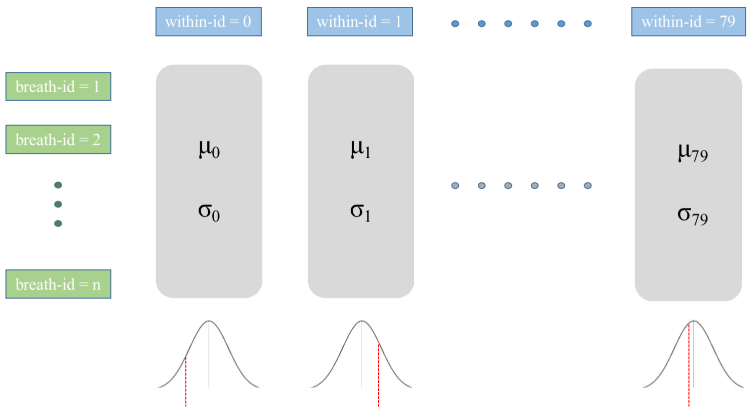


Figure 15: Schematic diagram of the principle of Between-breath gaussian guess

To this end, we first calculate the mean and standard deviation of the pressure values of all training set samples for each within-id (the id of the sample in the current breath_id, which is from 0 to 79), and thus construct a normal distribution, of the random sampling from such a distribution is the value of the Between-breath guess part. The idea of the Between-breath gaussian guess is shown above.

The fitting curve of our normal distribution to some within-id is shown in the figure below. The sample's distribution does not completely obey the normal distribution, but it is not bad as an approximation in a random guess.

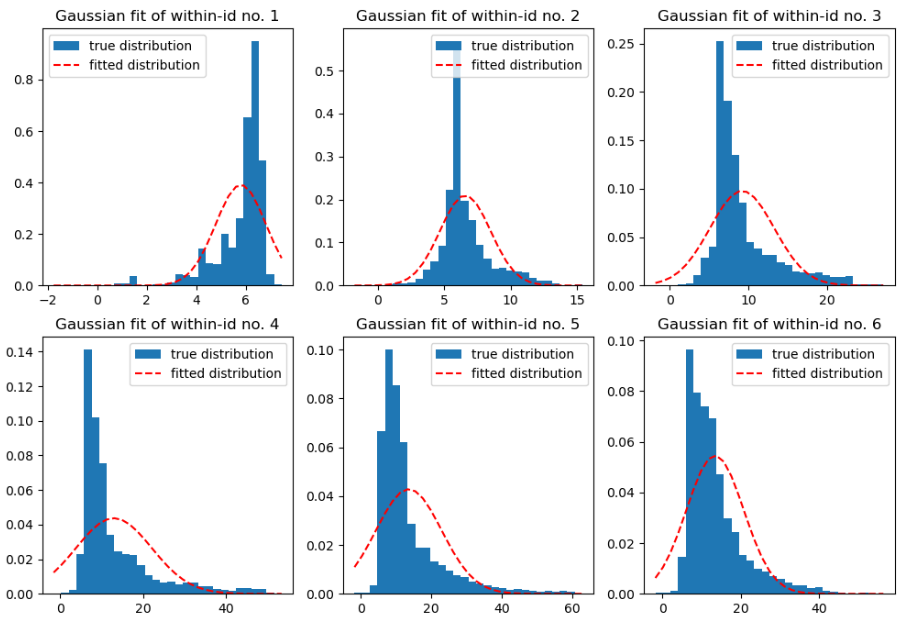


Figure 16: Between-breath gaussian guess

Two Parts Combination

We can construct our random guess by adjusting α based on the above two parts. The following table shows the performance of our random guess method on the validation set for different values.

Table 1: Evaluation of random guess

α	R^2	**MSE**	**MAE**	**MAPE**	**SMAPE**
0	-0.27	82.93	5.07	0.36	0.39
0.5	0.31	45.18	3.71	0.35	0.27
1	0.50	32.68	3.66	0.45	0.29

It can be found that when $\alpha = 0$ degenerates into a Between-breath guess, the performance of random guess is the worst, and R-squared is even lower than 0, which means that the model's performance is worse than directly using the mean to predict. When $\alpha = 1$, it degenerates into a Within-breath fit that performs best, and R-squared reaches about 0.5. When the two parts are considered comprehensively and equally, the performance is somewhere in between. This situation can be used later to judge the performance improvement brought by the model.

4.5 Linear Regression

Linear models are the simplest regression models. We also use it as a baseline. On the one hand, it can be used as a reference for subsequent models. On the other hand, we can infer the role of feature engineering through the difference in linear regression performance before and after feature engineering.

In order to verify the effect of feature engineering, we first fit the original data set using a linear regression model and evaluate its effect on the validation set. In the next section, we will compare it with the effect of feature engineering. Among them, all feature engineering is performed on the same data partition.

5 Feature Engineering

Pre-Definitions

- N: Number of records for the whole dataset
- n: Number of time steps for each breathe (80)
- i: Record id, and $0 \leq i \leq (N-1)$ –> correspond to the column 'id'

Time features

- time_delta: the lag of time between two continuous records in one breath

$$\Delta time_i = ReLU(Timestep_i - Timestep_{i-1}), i \geq 1$$

where

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

The reason for using the ReLU function here is because the Timestep will always start from 0 and accelerate until the breath ends (for example, 2.87), and then there will be another breath which starts from $Timestep = 0$. Actually, the equation above is equivalent to

$$\Delta time_i = \begin{cases} 0 & \text{if } i \bmod n = 0 \\ Timestep_i - Timestep_{i-1} & \text{elsewhere} \end{cases}$$

- total_time: the total time consuming for one breath

$$T_i = Timestep_{end}$$

where

$$end = n \times \lceil \frac{t}{n} \rceil - 1$$

Lag features

- u_in_lagk:

$$UIN_i^{lag\_k} = \begin{cases} 0 & \text{if } i \bmod n < k \\ UIN_{i-k} & \text{elsewhere} \end{cases}$$

- u_out_lagk:

$$UOUT_i^{lag\_k} = \begin{cases} 0 & \text{if } i \bmod n < k \\ UOUT_{i-k} & \text{elsewhere} \end{cases}$$

In our Feature Engineering, we let $k \in \{1, 2, 3, 4\}$

Q: Do we still need differences?

A: Actually no, suppose we apply the first-order differential operator to u_in,

$$\Delta UIN_i = UIN_i - UIN_i^{lag1}$$

They can be **linearly** represented by u_in_lag_k and u_in.

Back features

- u_in_back_k:

$$UIN_i^{back\_k} = \begin{cases} 0 & \text{if } i \bmod n > n - k - 1 \\ UIN_{i+k} & \text{elsewhere} \end{cases}$$

- u_out_back_k:

$$UOUT_i^{lag\_k} = \begin{cases} 1 & \text{if } i \bmod n > n - k - 1 \\ UOUT_{i+k} & \text{elsewhere} \end{cases}$$

Global Statistic features

- textu_in_max: the max value of UIN in one breathe

$$UIN_i^{max} = max\{UIN_{start}, UIN_{start+1}, \dots, UIN_{start+n-1}\}$$

  where

$$start = n \times \lfloor \frac{i}{n} \rfloor$$

- textu_in_mean: the mean value of UIN in one breath

$$UIN_i^{mean} = \frac{1}{n} \sum_{j=1}^{l} UIN_j$$

- textu_in_std: the standard deviation of UIN in one breath

$$UIN_i^{std} = \sqrt{\frac{1}{n} \sum_{j=1}^{l} (UIN_j - UIN_i^{mean})^2}$$

Q: Why don't extract the global statistic features for UOUT?

A: It's a binary variable, and it seems like a step function to time (if it follows a Bernoulli distribution or something else, it's still meaningful to calculate its statistic features).

Sliding Window features

Here we set the size of the sliding window to be $L \in \{3, 5, 7, 9\}$. In the following equations, note that $l = \frac{L-1}{2}$.

- u_in_slidingWindow_maxL:

$$UIN_i^{SWmax\_L} = max\left\{UIN_i^{lag\_l}, \dots, UIN_i, \dots, UIN_i^{back\_l}\right\}$$

- u_in_slidingWindow_minL:

$$UIN_i^{SWmin\_L} = min\left\{UIN_i^{lag\_l}, \dots, UIN_i, \dots, UIN_i^{back\_l}\right\}$$

- u_in_slidingWindow_meanL(dropped)

$$UIN_i^{SWmean\_L} = \frac{1}{L}[UIN_i + \sum_{j=1}^{l} (UIN_i^{lag\_j} + UIN_i^{back\_j})]$$

- u_in_slidingWindow_stdL(dropped)

$$UIN_i^{SWstd\_L} = \sqrt{\frac{1}{L}[D_1 + \sum_{j=1}^{l} (D_2^j + D_3^j)]}$$

  where

$$\begin{cases} D_1 & = (UIN_i - UIN_i^{SWmean\_L})^2 \\ D_2^j & = (UIN_i^{lag\_j} - UIN_i^{SWmean\_L})^2 \\ D_3^j & = (UIN_i^{back\_j} - UIN_i^{SWmean\_L})^2 \end{cases}$$

- u_out_SW_is_keyPoint_in: based on the discussion above about UOUT, the key point (t) is defined as the point where (in one breath)

$$UOUT_i = \begin{cases} 0 & \text{if } start \leq i < t \\ 1 & \text{if } t \leq i \leq end \end{cases}$$

where

$$t \in \{1, 2, \ldots, N-1\}, \quad start = n \times \lfloor \frac{i}{n} \rfloor, \quad end = n \times \lceil \frac{i}{n} \rceil - 1$$

The meaning of u_out_SW_is_keyPoint_in is whether the key point is in this window or not. It can be obtained by

$$max\left\{UOUT_i^{lag\_l}, \ldots, UOUT_i, \ldots, UOUT_i^{back\_l}\right\} - min\left\{UOUT_i^{lag\_l}, \ldots, UOUT_i, \ldots, UOUT_i^{back\_l}\right\}$$

Note that it's still a binary variable.

Q: Why drop the mean value and std value of the sliding window for u_in?

A: It can be seen $UIN_i^{SWmean\_L}$ can be **linearly** represented by UIN, UIN^{lag_k}, UIN^{back_k}, so we drop it. And we also drop $UIN_i^{SWstd\_L}$ since we don't calculate the mean value.

Other features

- Interaction of attributes R and C:

$$R\_C = R \times C$$

Data Scale

After feature engineering, we need to do some data scaling. In this project, we use a robust scaler to do that, which scales features using statistics that are robust to outliers. Given a sample x_i, then the scaled value is obtained by:

$$x_i' = \frac{x_i - x_{median}}{IQR}$$

where IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile)

Ablation experiment of feature engineering effect

We use the training set after feature engineering to train a linear regression model, evaluate its performance on the same validation set but with feature engineering, and compare it with the model in Section 4.2. Their comparison is shown in the table below:

Table 2: Experiment of feature engineering effect

	R^2	**MSE**	**MAE**	**MAPE**	**SMAPE**
without FE	0.38	40.33	3.94	0.49	0.29
after FE	0.68	20.83	3.17	0.42	0.30
improvement	78.9%	48.4%	19.5%	14.3%	-3.44%

As can be seen from the table, the model has been greatly improved overall, especially the R-squared and MSE improvements. MAE and MAPE have also been improved to varying degrees, while SMAPE has not been significantly improved. In general, it can be considered that feature engineering enriches the selection of features, expands the model's state space, and can bring about a good effect improvement, so it is very effective and necessary.

6 Modeling

6.1 Decision tree

Decision tree model, a machine learning model which is highly interpretable and both numerical and categorical inputs accept. Generally, an interpretable variable in medical science will be more helpful in clinical diagnosis. However, decision tree is not suitable for big data as its computation cost is extremely high.

6.2 Random forest

The ensemble learning technique can somewhat improve the decision tree model's overall performance. In this project, a large number of samples as well as nearly 40 features, may have many complex relationships with each other. The random forest method handles many input features and captures non-linear relationships between input features and the target variable. Besides, random forest can make the prediction more accurate by combining different trees.

6.3 XGBoost

XGBoost and Random Forest are both ensemble learning algorithms based on decision trees. However, they differ significantly in several aspects. One of the key differences is that Random Forest is composed of multiple independent decision trees, and the final result is determined by voting on the output of each tree. In contrast, although XGBoost is composed of multiple decision trees, the final result is obtained by a weighted summation of the output by each tree, where the weights are determined by the performance of each tree.

6.4 MLP

MLP(Multilayer Perceptron) is a type of artificial neural network composed of multiple layers of nodes (perceptrons), which have several advantages, such as the ability to learn non-linear relationships between inputs and outputs, tolerance for noisy data, and the ability to generalize to new data.

The model structure used in our project is shown in Figure 17. The model contains 6 layers, including 1 input layer H_0, 1 output layer H_5 and 4 hidden layers H_m, where $m = 2, 3, 4, 5$. The size of each layer is listed below:

$$H_1 \in R^{f \times h_1}, H_m \in R^{h_{m-1} \times h_m}, H_6 \in R^{h_{m-1} \times t}$$

where f is the input size or number of features, $h_i (i = 1, \ldots, 5)$ is the size of layer i, t is the target size.

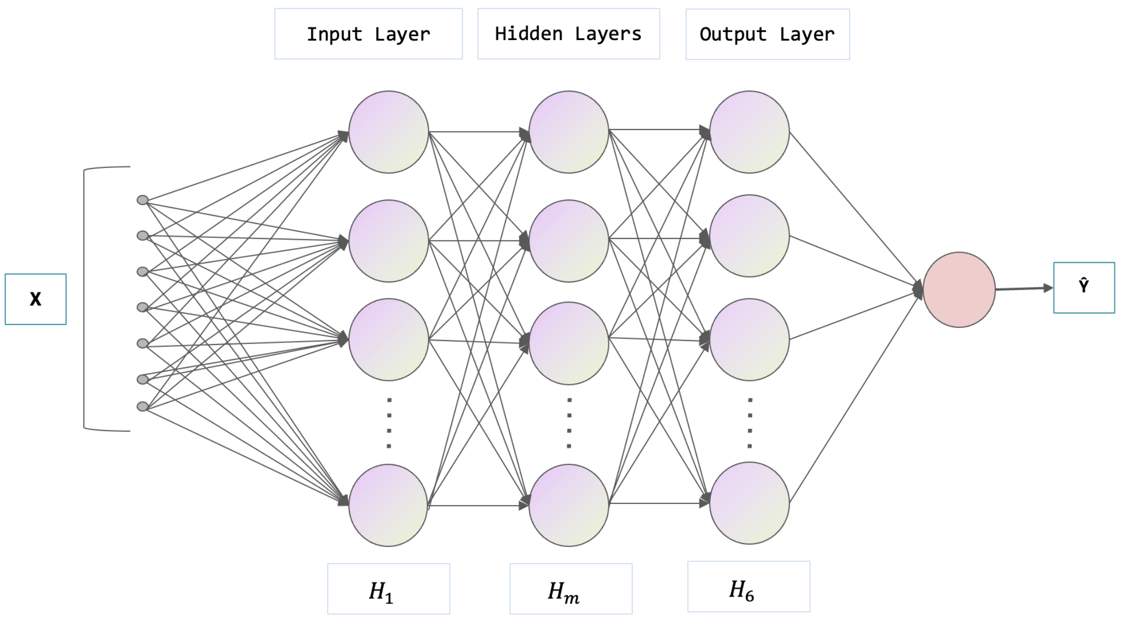


Figure 17: Structure of MLP

As mentioned, input data has two structures: tabular data with size $(n, n\_features)$ or sequence data with size $(n, sequence\_length, n\_features)$. Therefore, two MLP models have the same structure but different sizes for each layer. They are listed below.

Note: 3120 is obtained by 39×80, where 80 is the sequence length.

Table 3: Structure for MLP models

	f	h_1	h_2	h_3	h_4	h_5	t
MLP_tabular	39	64	256	512	256	64	1
MLP_sequence	3120	4096	2048	2048	1024	512	80

6.5 CNN

Convolutional Neural Network (CNN) is a deep learning model for processing two-dimensional or three-dimensional data such as images and speech. We use the sequence data as its input to use the CNN model.

This project uses two CNN models: one is a naive version, and another is added residual connection. They are the same except for the residuals.

The detailed structures are shown below. The first fully connection layer resizes the input, and the last 3 fully connection layers align the data to the target size. Downsampling happened after each max-pool.

6.6 RNN

Recurrent Neural Network (RNN) is a deep learning model used for processing sequence data because it uses the history information well. And our dataset is sequence data. Therefore, we

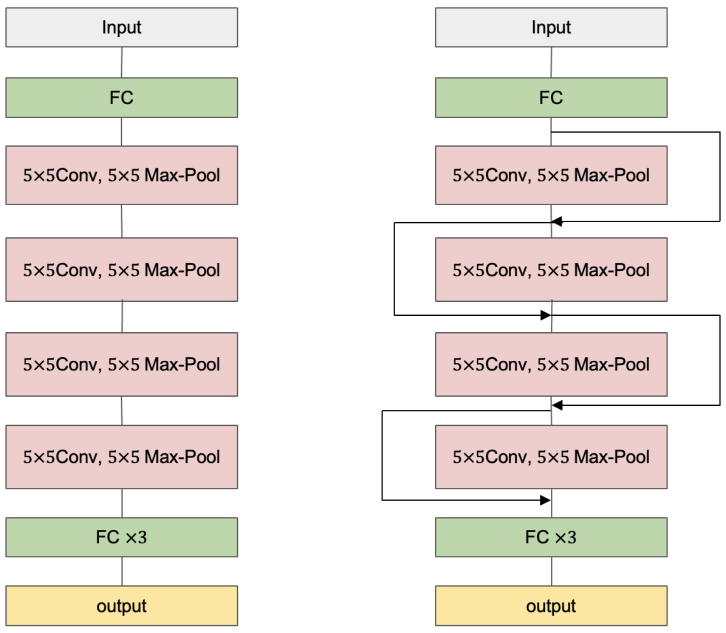


Figure 18: Structure of CNN(left) and CNN with residual (right)

try to model our data by RNN models. In our project, we design three different RNN models with the same general structure but different basic units. The hidden size is 256, and number of layers is 5 for all 3 models.

6.6.1 Basic

A basic RNN cell is shown below:

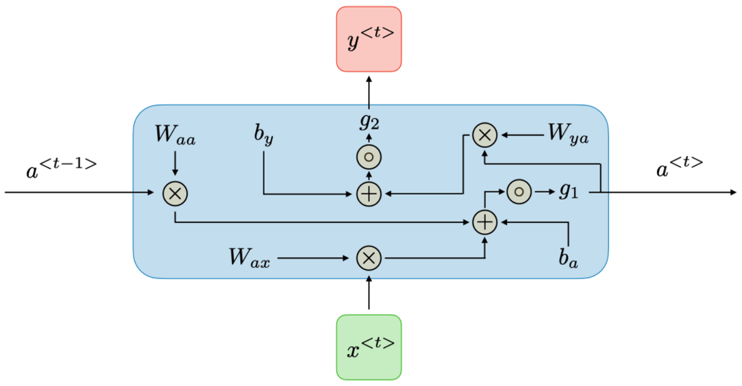


Figure 19: The Architecture of an RNN Cell

For each time step t, the activation $a^{<t>}$ is expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

The output $y^{<t>}$ is

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where W_{ax}, W_{aa}, W_{ya}, b_a, b_y are coefficients that are shared temporally and g_1, g_2 are activation functions.

6.6.2 LSTM

LSTM (Long Short Term Memory Network) is a Recurrent Neural Network (RNN) variant used for processing sequence data.

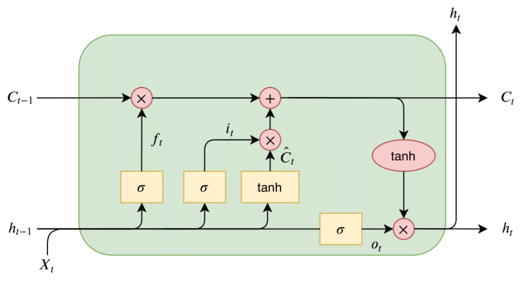


Figure 20: The Architecture of an LSTM Cell

Compared to the basic RNN, some unique gating mechanisms are added to the model. The core idea is to manage long-term dependencies in sequence data by introducing "memory units". A memory unit is one or more state variables stored in units of LSTM and continuously updated and updated throughout the model's cycling process.

6.6.3 GRU

GRU (Generative Recurrent Unit) is another variant of RNN. Similar to LSTM, it also adds a gating mechanism based on RNN. However, GRU's gating tool is relatively simple, with only input and output gates.

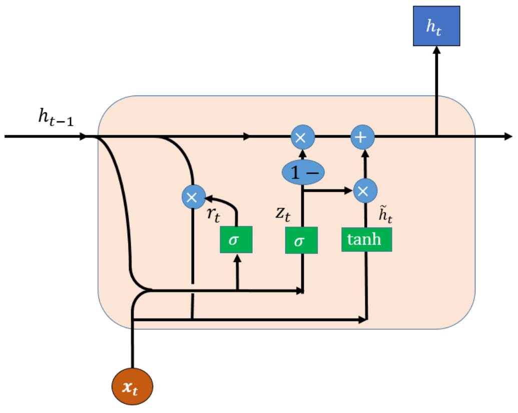


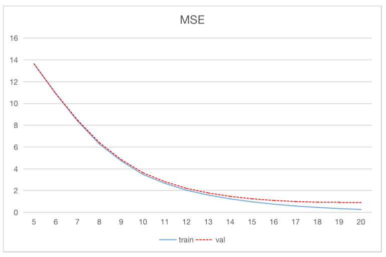
Figure 21: The Architecture of a GRU Cell

Compared with LSTM, GRU has a more straightforward gating mechanism, resulting in faster computation speed and less tendency to fall into local optima. GRU is widely used in natural language processing, speech recognition and time series prediction.

7 Evaluation

7.1 Decision Tree

For the decision tree model, we adjust the parameter max_depth, from 5 to 20, with an interval of 1. Other parameters take default values, and the feature selection criterion is "gini". The resulting performance on the training and validation sets is shown below.



Criterion	Performance
R^2	0.9810
MSE	1.2461
MAE	0.5925
MAPE	0.0593
SMAPE	0.0490

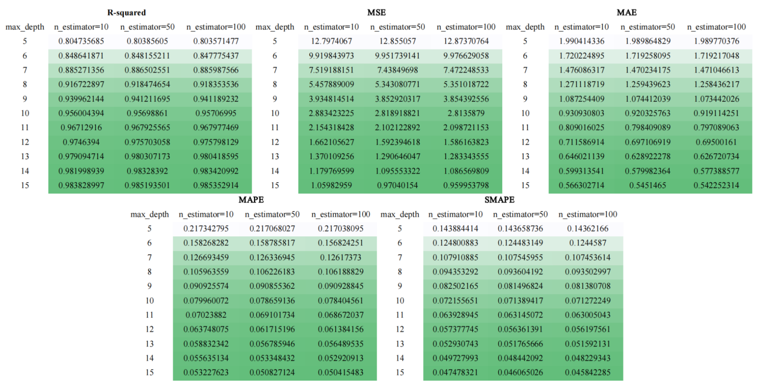
Figure 22: Decision tree evaluation

It can be found that the performance of the model on the validation set is very close to that on the training set. With the increase of max_depth, the model's performance is also improving, indicating that the previous model has been underfitting, and the follow-up is gradually approaching convergence. It can be inferred that with a large number of training samples (more than 3 million records of data) and good features, there is enough information to train a more complex model.

7.2 Random Forest

Using the decision tree as the base learner, we use two ensemble learning methods, Bagging (random forest) and Boosting (XGBoost), to explore performance improvements.

For random forest, considering the model complexity and training cost, we set the downsampling rate to 0.05. We set the parameter max_depth from 5 to 20, the step size is 1, and the parameter n_estimator is 10, 50 and 100, respectively. Other parameters are default values. The resulting performance on the validation set is shown in the figure below. The darker the green, the better the model performance.



R-squared

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	0.804735685	0.80385605	0.803571477
6	0.848641871	0.848155211	0.847775437
7	0.885271356	0.886502551	0.885987566
8	0.916722897	0.918474654	0.918353536
9	0.939962144	0.941211695	0.941189232
10	0.956004394	0.95698861	0.95706995
11	0.96712916	0.967925565	0.967977469
12	0.9746394	0.975703058	0.975798129
13	0.979094714	0.980307173	0.980418595
14	0.981998939	0.98328392	0.983420992
15	0.983828997	0.985193501	0.985352914

MSE

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	12.7974067	12.855057	12.87370764
6	9.919843973	9.951739141	9.976629058
7	7.519188151	7.43849698	7.472248533
8	5.457889009	5.343080771	5.351018722
9	3.934814514	3.852920317	3.854392556
10	2.883423225	2.818918821	2.8135879
11	2.154318428	2.102122892	2.098721153
12	1.662105627	1.592394618	1.586163823
13	1.370109256	1.290646047	1.283343555
14	1.179769599	1.095553322	1.086569809
15	1.05982959	0.97040154	0.959953798

MAE

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	1.990414336	1.989864829	1.989770376
6	1.720224895	1.719258095	1.719217048
7	1.476086317	1.470234175	1.471046613
8	1.271118719	1.259439623	1.258436217
9	1.087254409	1.074412039	1.073442026
10	0.930930803	0.920325763	0.919114251
11	0.809016025	0.798409089	0.797089063
12	0.711586914	0.697106919	0.69500161
13	0.646021139	0.628922278	0.626720734
14	0.599313541	0.579982364	0.577388577
15	0.566302714	0.5451465	0.542252314

MAPE

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	0.217342795	0.217068027	0.217038095
6	0.158268282	0.158785817	0.156824251
7	0.126693459	0.126336945	0.12617373
8	0.105963559	0.106226183	0.106188829
9	0.090925574	0.090855362	0.090928845
10	0.079960072	0.078659136	0.078404561
11	0.07023882	0.069101734	0.068672037
12	0.063748075	0.061715196	0.061384156
13	0.058832342	0.056785946	0.056489535
14	0.055635134	0.053348432	0.052920913
15	0.053227623	0.050827124	0.050415483

SMAPE

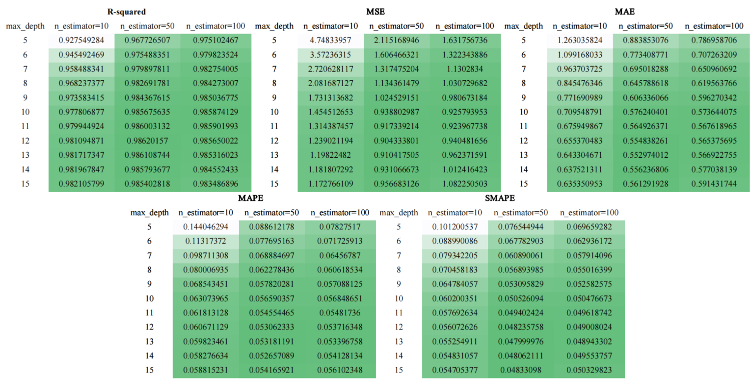
max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	0.143884414	0.143658736	0.14362166
6	0.124800883	0.124483149	0.1244587
7	0.107910885	0.107545955	0.107453614
8	0.094353292	0.093604192	0.093502997
9	0.082502165	0.081496824	0.081380708
10	0.072155651	0.071389417	0.071272249
11	0.063928945	0.063145072	0.063005043
12	0.057377745	0.056361391	0.056197561
13	0.052930743	0.051765666	0.051592131
14	0.049727993	0.048442092	0.048229343
15	0.047478321	0.046065026	0.045842285

Figure 23: Random forest evaluation

It is evident that the closer to the bottom right corner of the table, the better the model performance. The larger max_depth brings better results. The base learner's performance will affect the inheritance algorithm's effect. The larger n_estimator will also get better results. At the same time, one of the advantages of random forest is that it is not easy to overfit with the increase of n_estimator.

7.3 XGBoost

We choose the same parameter settings as Random Forest (sampling rate 0.05, max_depth from 5 to 20, n_estimator is 10, 50 and 100) to train the XGBoost model. The result is shown in the figure below.



R-squared

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	0.927549284	0.967726507	0.975102467
6	0.945492469	0.975488351	0.979823524
7	0.958488341	0.979897811	0.982754005
8	0.968237377	0.982691781	0.984273007
9	0.973583415	0.984367615	0.985036775
10	0.977806877	0.985675635	0.985874129
11	0.979944924	0.986003132	0.985901993
12	0.981094871	0.98620157	0.985650022
13	0.981717347	0.986108744	0.985316023
14	0.981967847	0.985793677	0.984552433
15	0.982105799	0.985402818	0.983486896

MSE

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	4.74833957	2.115168946	1.631756736
6	3.57236315	1.606466321	1.322343886
7	2.720628117	1.317475204	1.1302834
8	2.081687127	1.134361479	1.030729682
9	1.731313682	1.024529151	0.980673184
10	1.454512653	0.938802987	0.925793953
11	1.314387457	0.917339214	0.923967738
12	1.239021194	0.904333801	0.940481656
13	1.19822482	0.910417505	0.962371591
14	1.181807292	0.931066673	1.012416423
15	1.172766109	0.956683126	1.082250503

MAE

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	1.263035824	0.883853076	0.786958706
6	1.099168033	0.773408771	0.707263209
7	0.963703725	0.695018288	0.650960692
8	0.845476346	0.645788618	0.619563766
9	0.771690989	0.606336066	0.596270342
10	0.709548791	0.576240401	0.573644075
11	0.675949867	0.564926371	0.567618965
12	0.655370483	0.554838261	0.565375695
13	0.643304671	0.552974012	0.566922755
14	0.637521311	0.556236806	0.577038139
15	0.635350953	0.561291928	0.591431744

MAPE

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	0.144046294	0.088612178	0.07827517
6	0.11317372	0.077695163	0.071725913
7	0.098711308	0.068884697	0.06456787
8	0.080006935	0.062278436	0.060618534
9	0.068543451	0.057820281	0.057088125
10	0.063073965	0.056590357	0.056848651
11	0.061813128	0.054554465	0.05481736
12	0.060671129	0.053062333	0.053716348
13	0.059823461	0.053181191	0.053396758
14	0.058276634	0.052657089	0.054128134
15	0.058815231	0.054165921	0.056102348

SMAPE

max_depth	n_estimator=10	n_estimator=50	n_estimator=100
5	0.101200537	0.076544944	0.069659282
6	0.088990086	0.067782903	0.062936172
7	0.079342205	0.060890061	0.057914096
8	0.070458183	0.056893985	0.055016399
9	0.064784057	0.053095829	0.052582575
10	0.060200351	0.050526094	0.050476673
11	0.057692634	0.049402424	0.049618742
12	0.056072626	0.048235758	0.049008024
13	0.055254911	0.047999976	0.048943302
14	0.054831057	0.048062111	0.049553757
15	0.054705377	0.04833098	0.050329823

Figure 24: XGBoost evaluation

We found that the same as the random forest, a larger max_depth can bring better results, but a larger n_estimator may not. When n_estimator is too large, the model's performance on some parameters decreases. This may be because the base learners in XGBoost are trained serially, so too many base learners may lead to overfitting.

7.4 MLP

In training the MLP model, we chose two methods. One must learn according to the record (the id in the original data set, the feature size is 39) and ignore the timing relationship as independent samples. It takes the entire breath_id as a sequence sample (feature size is 80*39), and the model can learn complete sequence information. The loss curve of the two learning modes is shown in the figure below.

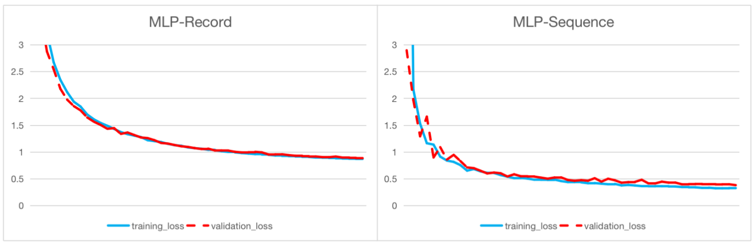


Figure 25: MLP evaluation

It can be seen that the model learned by sequence converges faster and finally converges on a smaller loss. During the training process, we select the best model by MSE (the same below), and the best performance during training is shown in the table below:

Table 4: Best performance of MLP

	R^2	**MSE**	**MAE**	**MAPE**	**SMAPE**
MLP_tabular	0.9866	0.8792	0.5695	0.0585	0.0517
MLP_sequence	0.9942	0.3791	0.3836	0.0421	0.0375

It can be seen that the performance of MLP_residual is ultimately better than that of MLP_tabular, which shows that the sequence relationship between features has a significant impact on prediction. It has achieved an extraordinary prediction performance.

7.5 CNN

Since the feature we input above is a two-dimensional structure with a size of 80*39, it is easy to think of directly training the CNN model. We also tried to add a residual module to avoid model performance degradation caused by too deep layers. The following figures are the loss curves during the training process of CNN and CNN with the residual module added.

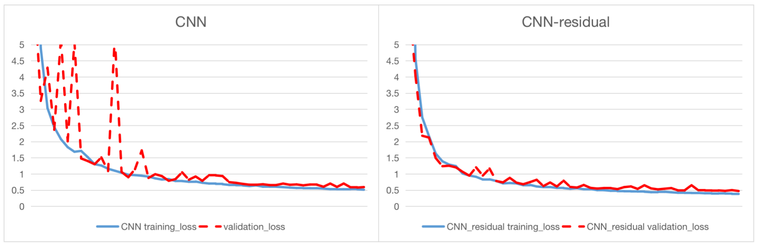


Figure 26: CNN evaluation

It can be seen that the convergence process of the CNN network training after adding the residual module is more stable and can also converge to a smaller loss. The best performance during training is shown below:

Table 5: Best performance of CNN

	R^2	**MSE**	**MAE**	**MAPE**	**SMAPE**
CNN	0.9912	0.5789	0.4665	0.0501	0.0446
CNN_residual	0.9928	0.4724	0.4296	0.0467	0.0414

It can be seen that the CNN with the added residual module performs better than the basic CNN, indicating that the residual module has indeed played a role and the model has achieved excellent prediction performance.

7.6 RNN

Finally, RNN and its two variants, LSTM and GRU. Compared with traditional RNN, LSTM is more suitable for processing longer sequences, so we expect it to get a huge improvement. GRU is a simplification based on LSTM and has a smaller training cost. We expect it to achieve performance comparable to LSTM. The following figures are the loss curves during the training process.

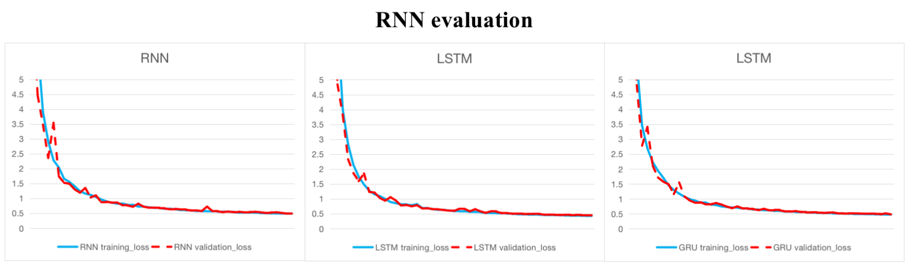


Figure 27: RNN evaluation

It can be seen that the training curves of the three are very close, and there is no obvious difference. So we hope to gain some inspiration from more quantitative model performance. Following is the best performance during the training process:

Table 6: Best performance of RNN

	R^2	**MSE**	**MAE**	**MAPE**	**SMAPE**
RNN	0.9925	0.4923	0.4437	0.0535	0.0438
LSTM	0.9933	0.4401	0.4260	0.0558	0.0431
GRU	0.9927	0.4809	0.4436	0.0575	0.0437

We found that LSTM has the most obvious improvement compared to RNN and performs best in other indicators except MAPE. GRU is between RNN and LSTM, and the performance has been improved to a certain extent, but not much.

7.7 Models comparison

We synthesize all the test results and summarize the performance of each model on the validation set, as shown in the following table:

Table 7: Performance comparison between models

	R^2	**MSE**	**MAE**	**MAPE**	**SMAPE**
Baseline: RandomGuess	0.31	45.18	3.71	0.35	0.27
LinearRegression	0.38	40.33	3.94	0.49	0.29
LinearRegression (FE)	0.68	20.83	3.17	0.42	0.30
DecisionTree	0.9810	1.2461	0.5925	0.0593	0.0490
RandomForest	0.9854	0.9599	**0.5423**	**0.0504**	**0.0458**
XGBoost	**0.9862**	**0.9043**	0.5548	0.0531	0.0482
MLP_tabular	0.9866	0.8792	0.5695	0.0585	0.0517
MLP_sequence	**0.9942**	**0.3791**	**0.3836**	**0.0421**	**0.0375**
CNN	0.9912	0.5789	0.4665	0.0501	0.0446
CNN_residual	**0.9928**	**0.4724**	**0.4296**	**0.0467**	**0.0414**
RNN	0.9925	0.4923	0.4437	**0.0535**	0.0438
LSTM	**0.9933**	**0.4401**	**0.4260**	0.0558	**0.0431**
GRU	0.9927	0.4809	0.4436	0.0575	0.0437

By comparison, we found that the **MLP_sequence** model achieved the best results under our evaluation criteria.

7.8 Generalizability test for the best model

We test our best model **MLP_sequence** on the testing set (mentioned in section 4.2), the results are shown in the following table:

Table 8: Generalizability test results for MLP_sequence

R^2	**MSE**	**MAE**	**MAPE**	**SMAPE**
0.9939	0.3988	0.3854	0.0504	0.0376

It's found that the performance on the testing set is very close to that on the validation set, which indicates our selected model is generalizable and robust.

8 Insights & Summary

After feature engineering, the models' performance has vastly improved, proving our feature engineering is reasonable. The best model for our mission is the **MLP, with the R^2 of 0.9942 and MSE of 0.3791**.

The ventilator aims to deliver oxygen to the lungs through a tube, which is more concerned with providing the exact amount of oxygen for the specified period. It, therefore, does not pursue the same level of model interpretation ability as conventional medical tests. We say a deep learning model is acceptable.

Using our model, we can simulate the breathing conditions of an actual patient and the amount of oxygen delivered. With more data in the future, we can adjust the model parameter and update it to provide more accurate results. This will undoubtedly help develop ventilators and reduce the pressure on doctors during the covid-19 period.