Google Brain
Ventilator Pressure Prediction

Authors

Samuele Cucchi, Riccardo Toti, Fabio Michele Russo, Marianna Abbattista

Distributed Data Analysis and Mining

Data Science & Business Informatics

Academic Year 2021/2022

Contents

1 Introduction

The dataset contains data produced using a modified open-source ventilator connected to an artificial lung via a respiratory circuit.

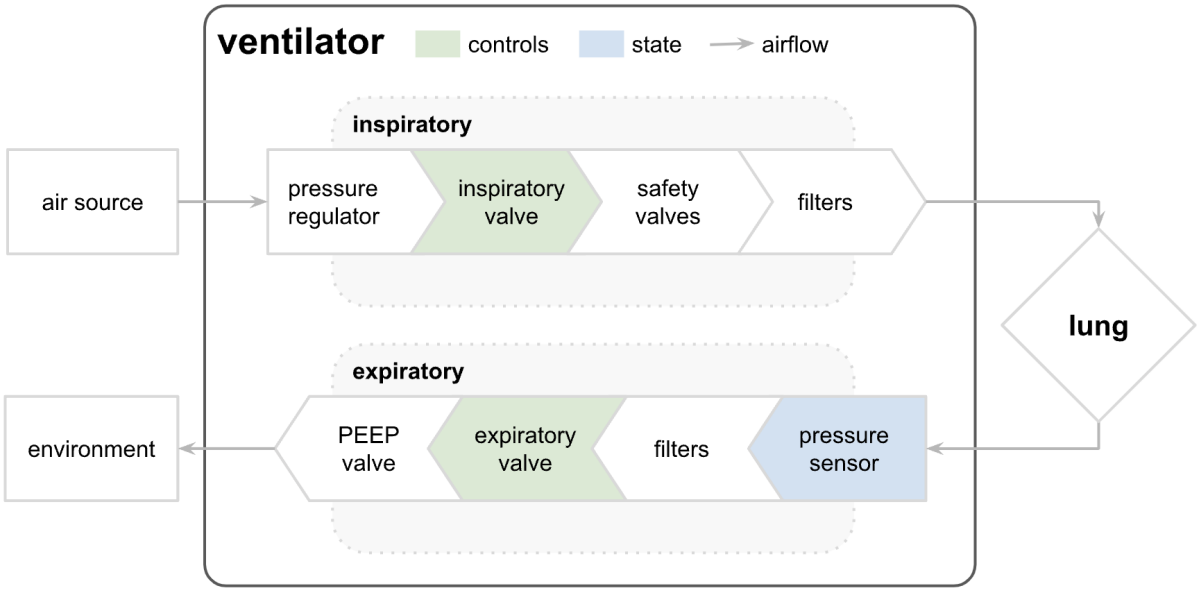


Figure 1.1: Setup of the ventilator

The diagram on the Figure 1.1 illustrates the setup, with the two control inputs highlighted in green and the state variable (pressure sensor) to predict in blue.
The first control input is a continuous variable from 0 to 100 representing the percentage the inspiratory valve is open to let air into the lung (i.e., 0 is completely closed and no air is let in and 100 is completely open). The second control input is a binary variable representing whether the expiratory valve is open (1) or closed (0) to let air out.

2 Data Understanding

The dataset is composed of 6036000 rows, each of which represents the reading of the machine's sensors in a certain *time_step*. The data is then organized as a collection of 125749 time series, each representing a breath recorded by the machine, with every single breath composed by 80 distinct rows.
Our data is composed by 8 columns, containing information regarding lung attributes (R and C), data regarding the input and output valves of the machinery (*u_in* and *u_out*) and data regarding the pressure inside the lungs connected to the machine (*pressure*). The dataset contains 8 columns:

- *id* - a unique identifier for each row on the dataset.
- *breath_id* - a unique identifier for each breath.
- *time_step* - a time feature that indicates the instant of time when the reading has been done. It does not coincide for every breath, as it does not have regular intervals but shapes itself according to the trend of *u_in*.
- R - lung attribute indicating how restricted the airway is, physically, it is the change in pressure per change in flow.
- C - lung attribute indicating how thick the lung is. Physically, this is the change in volume per change in pressure.
- *u_in* - the control input for the inspiratory solenoid valve. Ranges from 0 to 100.
- *u_out* - the control input for the expiratory solenoid valve. Either 0 or 1.
- *pressure* - the airway pressure measured in the respiratory circuit.

2.1 Exploration

We analyzed each column individually in order to get a better overview of the distribution of the data.

- *id* and *breath_id* are two fields used to identify rows and collection of rows, so they are numbers, both start from one and increment.
- R and C are two categorical fields, the first is composed by the numbers 5, 20 and 50, the second has the values 10, 20 and 50. They remain fixed for the entire "duration" of a breath.
- *u_out* binary field, indicates whether the output valve is opened or not: 0 closed, 1 opened.
- *u_in* continuous attribute, ranges from 0 to 100, seen as the percentage of how the input valve is opened when the patient inspires. Its mean and standard deviation are 7.3 and 13.4.
- *pressure* continuous attribute, ranging from -1.9 to 64.8, its mean and standard deviation are 11.2 and 8.1.

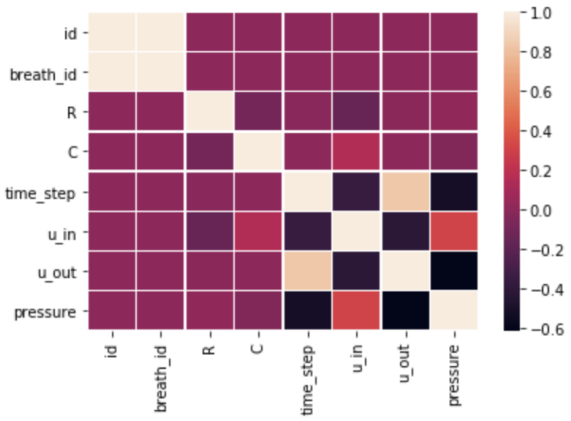


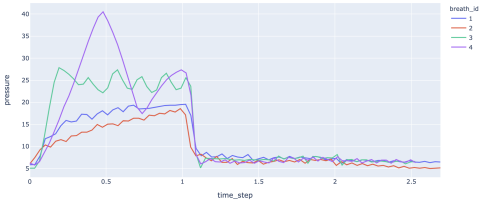
Figure 2.1: Correlation matrix for our dataset

The correlation matrix in the figure 2.1 highlights a positive correlation between the variables *time_step* and *u_out*, this is probably due to the machine was set to inhale for a certain amount of time, then exhale, since the first is a temporal information and the second will go from 0 to 1, this led on having these two variables correlated.
We then have negative correlations linking the couples of variables *time_step* with *u_in* and *pressure*, then *u_in* - *u_out* and *pressure* - *u_out* have a negative correlation too. In order to explain those negative correlation, we need to remember again the fact that the machine was set to start the time series inhaling, then exhale; we will explain each correlation:

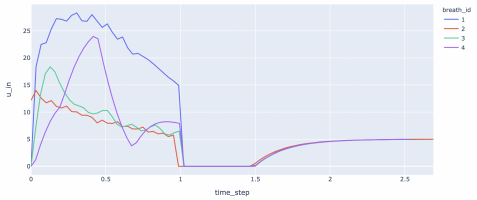
- The correlation between *time_step* and *u_in* and *pressure*, is due to those values will start being high (since the lungs are inhaling) then going to zero, due to the exhaling process, moreover, *time_step* is and incremental field.
- The reason behind the correlation between *u_in* and *u_out* is that, when the lungs are inhaling, the values for *u_in* are high and *u_out* is zero, when the "patient" will then start exhaling, the value for *u_out* will go to 1, and the values for *u_in* will immediately decrease.
- The correlation between *pressure* and *u_out* can be explained similarly as we did for *time_step* and *u_in* and *pressure*.

2.2 Outliers

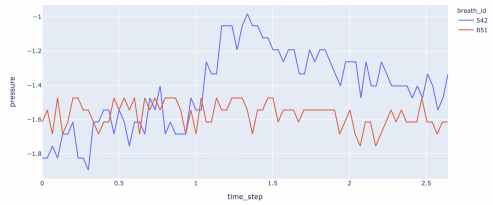
Analyzing the content of our dataset we found no missing values, so we decided to not include this section on our paper.
Regarding outliers, analyzing each columns we found some time series having negative values for pressure during all the breath. In order to decide whether to modify those cells or to delete the entire row, we decided to plot the data contained into the columns *u_in* and *pressure*, both when the pressures are positive and negative.
We can see, examining the figures 1.2 (a) and (b) that, the pressure and u_in tend grow when the output valve (*u_out*) is closed, i.e. during the first second of the breath, they then drop to zero next we see *u_in* starting to grow again at the *time_step* 1.5. Comparing the plots just analyzed with the bottom two (c) and (d) we can see that *pressure* tends to grow and decrease with apparently no logic. In those breaths we see that *u_in* instead tends to grow for 1.2/1.3 seconds, then decreases gradually, without being influenced by the output valve *u_out* (that always changes from 0 to 1 after 1 second from the start of the breath). Since we found those strange behaviours from these two variables, we decided to delete the 3713 rows having negative pressures.



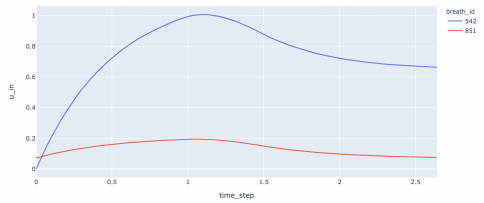
(a) Trend for positive pressure



(b) Trend for *u_in* having positive pressure



(c) Trend for negative pressure



(d) Trend for *u_in* having negative pressure

Figure 2.2: Comparison for *u_in* and *pressure* when the latter is positive or negative

3 Data Preparation

3.1 Approach

The data we have are clearly time series. However, working with time series on Spark is fairly awkward because of the distributed nature of the data in memory and using external libraries for time series has not proved successful. Therefore, we decided to use a double phase prediction model.
We have observed that the time-dependent features are only *u_in*, *u_out* and *pressure* while R and C stay the same over the whole time series. Therefore we put R and C in a separate dataset and use them in the following chapter 4, *Regression Models*.
Therefore, we must model the effect of time for the three "temporal" features. Shunning away from a complex single multivariate regression function, we take the approach of modeling each of the three using two distinct linear regressions and one logistic regressor. This has the disadvantage that any interactions between these three features are not considered.
Here follow the equations of the three regression lines:

$$u\_in = q\_u\_in + m\_u\_in \times Time$$

$$u\_out = q\_u\_out + m\_u\_out \times Time$$

$$pressure = q\_pressure + m\_pressure \times Time$$

By doing this we predict the coefficients:

q_u_in, m_u_in; q_u_out, m_u_out; q_pressure, m_pressure.

In the following step (described in chapter 4, *Regression Models*) we use 4 of these coefficients as features, together with the non-time-dependant R and C, to train a model that would predict *q_pressure* and *m_pressure*, which are our target variables. In this step we try three different categories of algorithms (Linear Regression, Decision Tree Regressor, Gradient Boost Regressor).

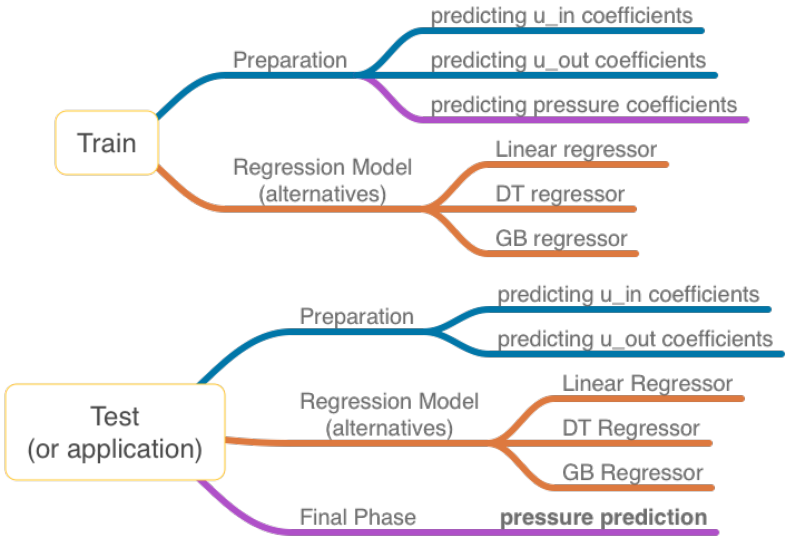


Figure 3.1: Diagram of entire predictor

After having the overall double-phase model trained, we can run predictions on test records that are missing the *pressure* target variable. The pipeline is: run the first two regressions of phase one (this *Data Preparation*) on Time, u_in and u_out. Then, run the phase-two regression step (that is, one of the following *Regression Models* alternative approaches) and get as output *q_pressure* and *m_pressure*. The final pipeline step is running the third of the first-phase regressions with the coefficients we just predicted and Time of the test instance. For any Time given as input, we get a **pressure** prediction.

3.2 Specifics on Data Preparation

In this section we will go over the details of how we prepared our dataset for the following step.
In order to make the regression as we wanted, we had to modify the "shape" of the data that we made the prediction on.
First of all we started by keeping all the information that don't change during the span of the time series, so we kept R, C and *breath_id* as they were for the first dataset, for the second one we managed R and C by creating a parallel dataset in which we transformed them into dummies format. For both we do all the next phase, we also discarded the column *id*, because it was an identifier relative to a single "snapshot" of the machine during a breath detection.
The columns that required computations were *u_in*, *pressure*, that were treated in the same way using a linear regression, and *u_out*, shaped using a logistic regressor. Firstly, we divided the dataset in a way to be able to have each breath singularly, then we trained two linear regression models, both of them having as feature the column *time_step* and either *pressure* or *u_in* as target. After having trained those two regressors, we kept the coefficients for their slope and intercept. We applied the same procedure for the columns *u_out*, with the difference that, instead of using a linear regressor, we used a logistic regressor, since its values starts from zero, then moves to one after a certain amount of time.

breath_ids	R	C	m_u_in	q_u_in	m_u_out	q_u_out	m_pressure	q_pressure
148	50	10	-0.627587	3.279188	3.583992	-4.387057	1.712045	4.787831
463	50	10	-2.730342	11.496349	3.598094	-4.875284	5.452312	14.032852
471	20	20	-1.101224	10.255016	3.589597	-4.547209	2.988894	7.084789
496	20	20	-17.101354	54.395898	3.590041	-4.548817	2.781363	18.636305
1238	50	50	-4.046308	34.687972	3.598168	-4.872869	1.094701	27.028427

Figure 3.2: Five lines of the pre-processed dataset

In the figure 3.2 the final version of the dataset is shown. As said before, the columns R, C and *breath_id* were kept as they were, the columns *m_u_in*, *m_u_out* and *m_pressure* indicate the slope values (assigned by the regressors trained) for the fields *u_in*, *u_out* and *pressure*, similarly, the columns *q_u_in*, *q_u_out* and *q_pressure* contain the same information, but regarding the intercept with the y axis of those fields.

During the next phase (Regression Models) we used the fields R, C, *m_u_in*, *q_u_in*, *m_u_out* and *q_u_out* as features to predict the variables *m_pressure* and *q_pressure*.

4 Regression Models

For every type of algorithm we trained (Linear Regression, Decision Tree Regression, Gradient Boost Regression) we always used the same procedure, in order to keep our code cleaner and the results comparable. Particularly, we defined a pipeline structure with the following stages:

1. We used a *VectorAssembler* to shape our data in a way that was suitable for the training phase,
2. Then we normalize all features with a *StandardScaler*,
3. Then we used a *TrainValidationSplit* object to run a grid search to tune the hyperparameters and retrieve the best model based on the R^2 metric using the *.bestModel* attribute.

For each of the three algorithms used, we tried three distinct approaches, in each of them we run two models, with targets *q_pressure* or *m_pressure* (respectively intercept and slope of the regression line modelling the Pressure on Time). The approaches are:

1. a **"Parallel approach"** where we predict the slope and the intercept of our regression line separately, in two parallel regressions.
2. a **"Waterfall 1"** where we first predict the slope of the regression line and then we use the same dataset, but with the slope just predicted **added in as feature**, to predict the intercept.
3. a **"Waterfall 2"** which is the same, but inverted: first the intercept, then the slope.

4.1 Linear Regression

4.1.1 Parallel approach

We trained our models for predicting the values of *m_pressure* and *q_pressure* using the same training set. Here's the regression equations:

$$q\_pressure = \alpha_0 + \alpha_1 R + \alpha_2 C + \alpha_3 q\_out + \alpha_4 m\_out + \alpha_5 q\_in + \alpha_6 m\_in$$

$$m\_pressure = \beta_0 + \beta_1 R + \beta_2 C + \beta_3 q\_out + \beta_4 m\_out + \beta_5 q\_in + \beta_6 m\_in$$

4.1.2 Waterfall 1

In this approach we firstly predict *m_pressure* with all the features - as before. Then we take our predicted *m_pressure* and we add it as feature in the training dataset and use it to predict *q_pressure*. These are the two equations:

$$m\_pressure = \beta_0 + \beta_1 R + \beta_2 C + \beta_3 q\_out + \beta_4 m\_out + \beta_5 q\_in + \beta_6 m\_in$$

$$q\_pressure = \alpha_0 + \alpha_1 R + \alpha_2 C + \alpha_3 q\_out + \alpha_4 m\_out + \alpha_5 q\_in + \alpha_6 m\_in + \alpha_7 m\_pressure$$

4.1.3 Waterfall 2

This is the same as Waterfall 1 with *q_pressure* and *m_pressure* inverted:

$$q\_pressure = \alpha_0 + \alpha_1 R + \alpha_2 C + \alpha_3 q\_out + \alpha_4 m\_out + \alpha_5 q\_in + \alpha_6 m\_in$$

$$m\_pressure = \beta_0 + \beta_1 R + \beta_2 C + \beta_3 q\_out + \beta_4 m\_out + \beta_5 q\_in + \beta_6 m\_in + \beta_7 q\_pressure$$

4.2 Decision Tree Regressor

Decision tree is a powerful supervised machine learning algorithms that can be used for classification and regression problems. The model is based on decision rules extracted from the training data. In the regression problem the model uses discrete buckets of values as class, and mean squared error as decision accuracy.
In our scenario the hyperparamaters that are involved are max_depth and max_bins. We test the value of max_depth into a range of (2-28) every 2 step in our Pipeline, in our case is max_depth = 5. Instead the value of max_bins, i.e. the number of bins used when discretizing continuous features, goes in a range (2-50) every 2 step, in our case is max_bins = 38.
We can see that if the maximum depth of the tree (controlled by the max_depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit.
We still apply all three approaches: *Parallel*, *Waterfall 1* and *Waterfall 2*.

4.3 Gradient Boost Regressor

Gradient Boost Regressor builds an additive model in a forward stage-wise fashion; it allows the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function. It is considered one of the best Ensemble Methods.
In our scenario the hyperparamaters that are involved are max_depth, max_iter and max_bins. We test the value of max_depth into a range of (2-28), the value of max_iter into a range of (2-50) and the max_bins goes in a range (2-10).
We still apply all three approaches: *Parallel*, *Waterfall 1* and *Waterfall 2*.

5 Evaluation and Results

In this chapter we will explain and evaluate the results that we got from training and testing a Linear Regressor, a Decision Tree Regressor and a Gradient Boost Regressor on our data. In order to choose the best classification approach for each model between the ones presented, we decided to aggregate the performances of R2 and RMSE by calculating the average of these two values.

5.1 Linear Regressor

For this classification model we saw that the model performs better when using the dataset having the values of R and C dummified.
The classification method that performed best with this type of model is the *Waterfall 1*, which consists on train firstly the model to regress *M_pressure*, then add it to the training data and train another model to predict *Q_pressure*.
Evaluating the performance as we explained above, it is immediate to see that the parallel approach is the one performing worst among all. Calculating the aggregated R2 and RMSE for the two approach, we saw that the best way to compute the two target variable is the *Waterfall 1*, which has an aggregated R2 of 0.47 and 3.07 as RMSE, against 0.4675 of R2 and 3.09 of RMSE from the *Waterfall 2* approach.

		Single Input	
		M_pressure	**Q_pressure**
Parallel approch	*RMSE*	**2.140**	4.099
	R2	**0.355**	0.574
		One in Input, one predicted	
		M_pressure	**Q_pressure**
Waterfall 1	*RMSE*		**3.998**
	R2		**0.585**
Waterfall 2	*RMSE*	2.081	
	R2	0.361	

Table 5.1: Linear Regression with dummies

5.2 Decision Tree Regressor

The second model that we tried is the Decision Tree Regression algorithm, different from the preceding, wee see a clear growth on the performances. Also with this model, the difference between the Parallel approach and the Waterfall one are not so much but in this case we can underlay the differences between the performances on the regression on *M_pressure* and *Q_pressure*. In the Parallel approach, the prediction of the single Q is better than the M that because has an higher R2 and a lower RMSE, meaning that the slope value of the line has more predicting power on the intercept than vice versa. We can confirm this hypotheses also trying to explain the Waterfall one. The first, having also the *M_pressure* in input increase the value of R2 and decrease the RMSE, meaning that also helped by the other coefficient the predicting of the *Q_pressure* has better performance.
The second one has the lower results of the Decision Tree method, that because we are trying to predict the *M_pressure* using the *Q_pressure* coefficient, but as explained before, is better to predict this value instead of using it as input value.
For this model we can than say that the best approach to classify the target variables is the parallel one, using the dataset without dummy variables for the regression on *M_pressure* and the dataset without dummy to predict *Q_pressure*. For this approach, the aggregated score for the predictions are 0.89 for R2 and 1.22 for RMSE.

			Single Input	
			M_pressure	**Q_pressure**
Parallel approch	***No Dummy***	RMSE	**1.144**	1.340
		R2	**0.816**	0.954
	Dummy	RMSE	1.158	**1.304**
		R2	0.811	**0.957**
			One in Input, one predicted	
			M_pressure	**Q_pressure**
Waterfall 1	*No Dummy*	*RMSE*		1.406
		R2		0.949
	Dummy	*RMSE*		1.366
		R2		0.952
Waterfall 2	*No Dummy*	*RMSE*	1.123	
		R2	0.814	
	Dummy	*RMSE*	1.230	
		R2	0.777	

Table 5.2: Decision Tree Regressor

5.3 Gradient Boost Regressor

The Gradient Boost Regressor model, as highlighted on the Table 5.3, has the best results of all our analysis. Also here we can underlay the difference between the Parallel and the Waterfall approach. In this case the single prediction of the *Q_pressure* has better results than the Waterfall one, meaning that the *M_pressure* coefficient doesn't help the Regression, and we can observe that this happens also in the second Waterfall, the use of the regressed values of *Q_pressure* decreases the metrics performance.
At the end, we can say that the best approach when using a Gradient Boost Regressor is the parallel one, having 0.9 as aggregated R2 and 1.18 as RMSE.

		Single Input	
		M_pressure	**Q_pressure**
Parallel approch	*RMSE*	**1.008**	**1.359**
	R2	**0.853**	**0.952**
		One in Input, one predicted	
		M_pressure	**Q_pressure**
Waterfall 1	*RMSE*		1.385
	R2		0.950
Waterfall 2	*RMSE*	1.022	
	R2	0.849	

Table 5.3: Gradient Boost Regressor without dummies

6 Conclusions

In this project we became familiar with the Spark framework by exploiting its features for managing datasets and distributed processes. The large family of machine learning methods developed for tasks involving large amounts of data allowed us to exploit them in a simple way. With regard to the chosen dataset it was difficult to identify a possible solution due to the low number of features. As part of a competition proposed by Google on the Kaggle portal, we followed the developments and the notebooks of the winners, but tried to approach the problem from a different point of view. We believe that having tried to predict the regression coefficients using for each record its own time step, which differs from one breath to another, is an interesting and alternative attempt to study and analyse a problem of extreme importance such as that of a respirator valve, especially in the period we are going through.

As a possible future development, since from the Figure 2.2 we can see that the trend of the variables *pressure* and *u_in* are not clearly linear, we thought that it would be a good idea, in order to increment the performances of our model, to apply a quadratic regressor, instead of a linear one, to approximate the trend of these variables.

✍