Machine Learning Engineer Nanodegree

Capstone Project

Jamal Abidar
March 28th, 2019

I. Definition

Project Overview

The chosen project is in the industrial domain. More specifically in predictive maintenance. Predicitive maintenance is a well know subject which is very sought after by various industries as its Return on Investment (ROI) is often very high. It consists in analyzing machine generated data (states, operations, aggregations, logs) and trying to predict when the machine will fail.

The data points are indexed/listed in time order. Such data is called “Time Series” data. Analysis of Time Series data is an old (19th century and late 60s of 20th century) and well-known subject (widely used in Finance).

The project we chose here is from Kaggle. It is the only project that is explicitly around the subject of predictive maintenance. The Kaggle competition deals with equipment data and asks Kaggle competitors to predict when the equipment will fail. Thus allowing corrective maintenance and preventing downtime and potentially saving a lot of money.

I personally chose this subject as I am passionate about IOT and the first use case IOT enables is predictive maintenance. I would like to use this project as an introduction to IOT analytics.

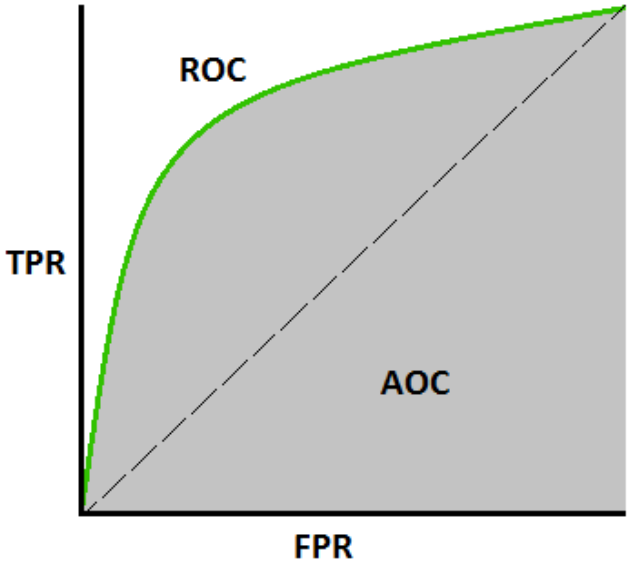
Problem Statement

The problem to be solved is the following: predict when a piece of equipment will fail. We have data for 983 days of a piece of equipment. The labels (OK, Failure) are given for the first 683 days. Kagglers need to predict the 300 missing data points. The data given each day is multivariate meaning that we have not only one but many features for each day (130 different features).

The results are assessed using a well-defined measure: Area Under ROC curve aka AUC. All the features can be translated to numerical values (mathematical values) and the problem (failure) occurs more than once (approximately 7% of the time). For more details, please refer to: https://www.kaggle.com/c/predictive-maintenance1

Metrics

The Kaggle competition evaluates the results using AUC metric for the label (0: equipment OK, 1: failure). So, we will just follow the same methodology in order to have the best results as per the Kaggle competition requirements. The ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much model is capable of distinguishing between classes. The ROC curve is calculated using the TPR (True Positive Rate) and FPR (False Positive Rate).



$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN} \qquad \text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}$$

II. Analysis

Data Exploration

The dataset consists of 2 files:

- feature.xlsx
- train_label.csv

The excel file has 983 lines. One line for each day. The covered time period is from 5/3/2015 (3rd of April 2015) to 1/17/2018 (17th of January 2018).

In the feature.csv file, for each day we have 5 sets of 26 features:

- errors counts (number of errors that day) for 26 types of errors
- errors max ticks (time of the last error of the day) for 26 types of errors
- errors min ticks (time of the first error of the day) for 26 types of errors
- errors mean tick (mean time for the errors of the day) for 26 types of errors
- errors standard deviation of the ticks for 26 types of errors

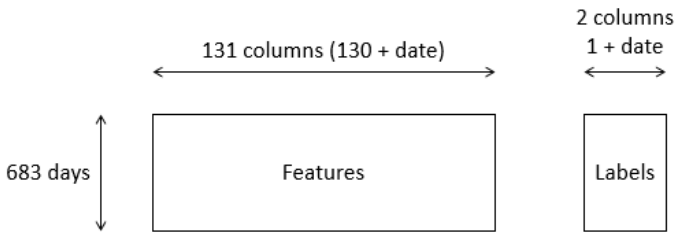
Total number of features in the feature.xlsx file is 26 * 5 + 1(date) = 131.

The train_label.csv file has 683 lines. One line for each day which presents only one label: 0 when equipment is OK, 1 when equipment failed that day. The covered period is from 5/3/2015 (3rd of April 2015) to 3/24/2017 (24th of March 2017).

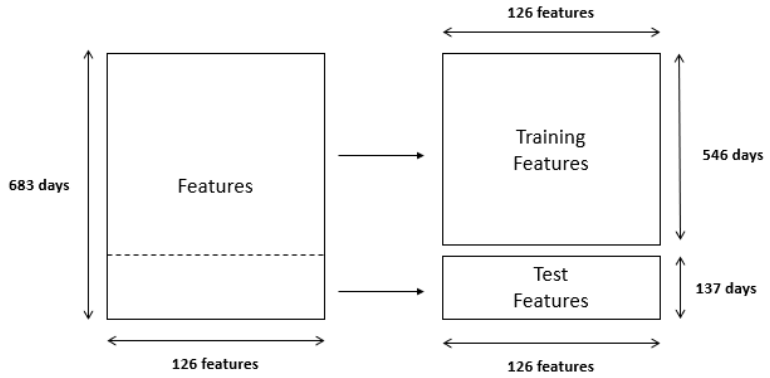
The objective it to use the provided data to predict when the equipment will fail for the missing 300 days in the train_labels.csv file: from 3/25/2017 to 1/17/2018.

In our project we will not follow exactly the requirments from the Kaggle competition. We will use the 683 days data for training/validation and test. The 300 days to predict can only be predicted when submitting to the competition but the competition is no longer open for submission.

Our dataset will then consist of 683 days of data from 5/4/2017 to 3/24/2017.



This dataset will be split in training dataset and test dataset as follows:



The number of features/columns is different (126 instead of 131) in the above image as some features were removed from the initial dataset:

- The Date column (useless for the 2 algorithms)
- Error19 (See renaming below to understand the column naming) had all its tick values (mean_tick, min_tick, max_tick and std_dev) at 0 or NaN (Not a Number). We decided to remove these 4 columns.

Renaming Columns

Columns had to be renamed as the csv files had the same name for the all errors for each category. Example: “error136088802” was the column name for error136088802 counts but also for the error 136088802 min ticks etc etc… The same column name was repeated 5 times.

The pandas inference then appended a suffix for each category. Example: error136088802 error count was renamed by pandas as error136088802, the second error136088802 which is the error 136088802 min tick was renamed error136088802.1 and so on and so forth. The pandas dataframe had 136088802, 136088802.1, 136088802.2, 136088802.3 and 136088802.4.

I decided to rename them with simple names:

- error1 to error26 for all error counts
- error1.max_tick to error26.max_ticks for max tick error
- error1.min_tick to error26.min_ticks for min tick error
- error1.mean_tick to error26.mean_ticks for mean tick error
- error1.std_dev_tick to error26.std_dev_ticks for standard deviation tick error

Types

The features types in the source files are integers for all error counts, max, min but mean and std are floats. The dataset is read by pandas and the types are automatically inferred to float for all features. The labels type was automatically inferred to integer (0 or 1).

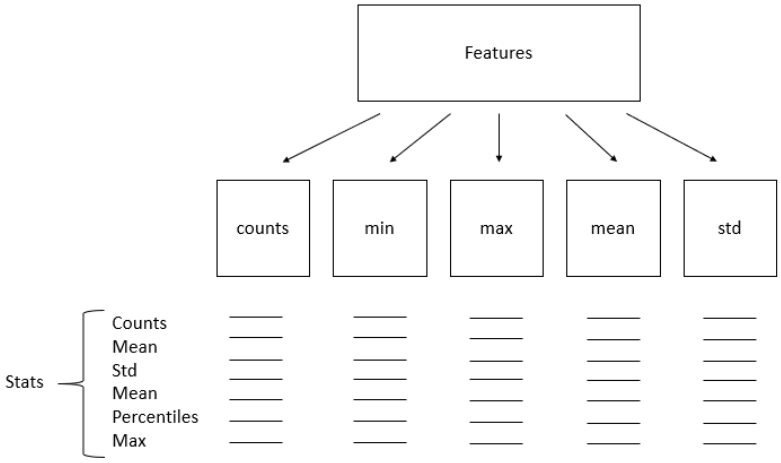
Statistics

The features are grouped in categories in the source file:

- error counts
- error max tick
- error min tick
- error mean tick
- error standard deviation tick

Each category is split from the main pandas dataframe (containing all categories) and basic statistics are displayed for each category and each column of the category (pandas describe):

- count
- mean
- std
- min
- 25, 50 and 75 percentiles
- Max



Distinctive Characteristic

I noticed that the number of machine failures was very low compared to the number of days when the machine was working fine. That is fine but I needed to know how many days had failures (percentage of failure days in the label dataset).

This calculation was done, and the number of failure days is 6.44% of the whole dataset period.

That is low and the training might be difficult because the algorithm must see enough failures to be able to recognize one.

This distinctive characteristic highlights the fact that the training will be difficult, and that the dataset will most likely have to be augmented to have good results.

Exploratory Visualization

In order to complete the previous exploration, we needed to visualize the features to see if patterns arise from the data.

Scaling

In order to display the various errors together even though the scales are different, we had to rescale all the features between 0 and 1. We used a custom-made Min Max scaling function for that.

Focus on Errors counts

For the sake of brevity, I choose to focus the visualization on the error counts. That would give 26 different graphs at least.

The errors min, max, mean and std dev could also bring interesting information but I wanted to be concise here.

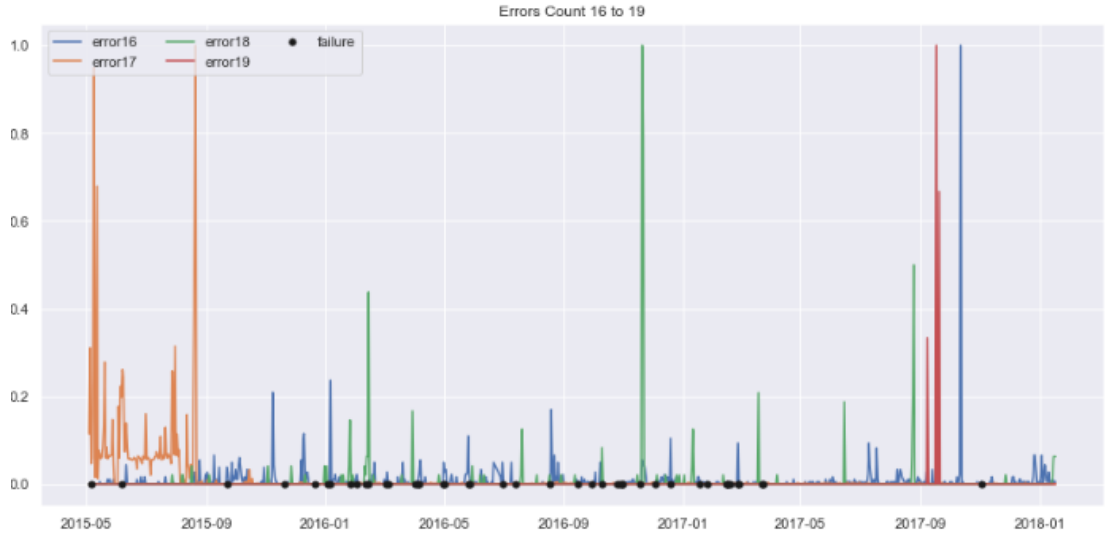
Two different zooms

In order to see patterns, we thought that we needed to see the same data with two different zooms. That consists in visualizing the features in small groups first (Zooming in). In order to allow easy comparison between features we decided to display graphs with 4 successive features and the machine failures.

That would allow us to correlate some errors together and hopefully see patterns.

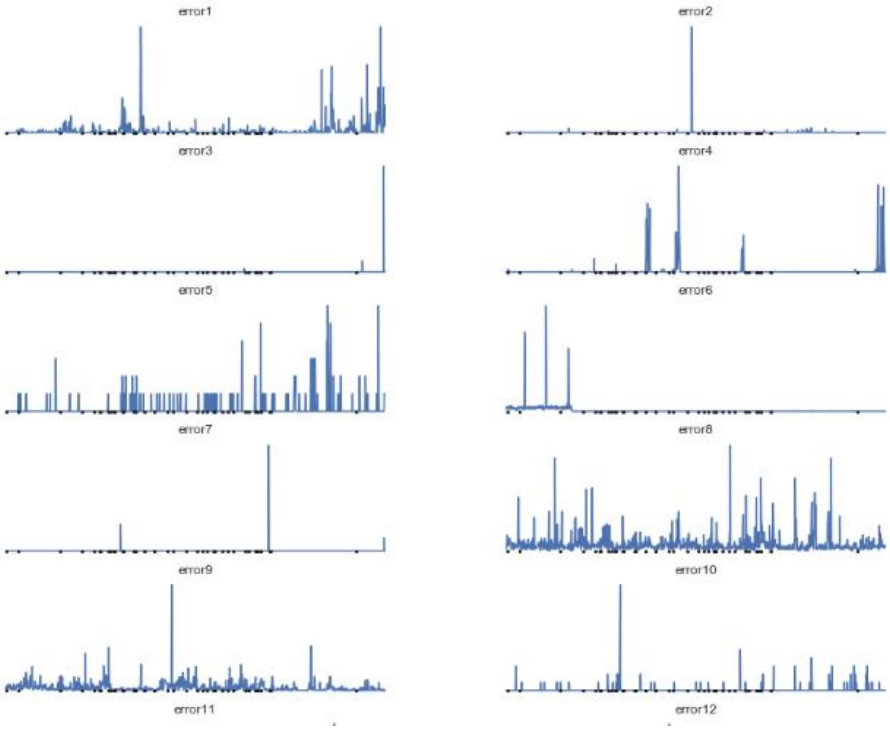
The errors are between 0 and 1(scaled) and the failures appear in the X-axis as big black points. This was done on purpose to be able to see links between errors and machine failures.

An example of this visualization is presented below:



This graph shows that the error17 has apparently no link with the failures as most of the failures appear when it is at 0. We can also see that when a failure happens, one of the errors is at a peak so the link between errors and machine failure is validated but the error peak is different at every failure.

Then for the second type of visualization we choose to zoom out and all the errors were displayed separately in a grid to have the global view and see if a particular error has a greater link with the machine failures. The grid allows us to have many error graphs on the same screen (see image below). We can also compare the errors between one another more easily in a grid.



These two visualizations would give use good intuition for using the model properly. The intuition results are described below.

We noticed for example that there are three types of errors:

- Peaks: contains one or at most two peaks of errors throughout the time period (errors: 2, 3, 7, 19)
- Short-lived: has activity during a limited amount of time - weeks or at most 6 months (errors: 6, 17, 22, 23, 24, 25, 26)
- Constant Activity: has activity throughout all the time period (errors: 1, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 20, 21)

It appears that the failures are not directly linked to a specific error count (all failures are not happening when we have peaks at error21 for example).

However, we can clearly see that some error counts peaks coincide with some failures (for example error5 and error8 seem to be correlated to the failures) so we can conclude that each failure is closely linked with a combination of error counts.

For trends we can see that the errors don't follow specific trends (increasing or decreasing over time).

For seasonality, apart from errors 5, 11 and 13 all error counts seem to be random.

Algorithms and Techniques

A possible solution to the selected problem could be in two steps:

1. Predicting the next days – Regression problem

2. Then depending on the type of anomaly using either a simple anomaly detector or a classifier – Classifier problem

If the anomaly is due to passing a threshold (upper-bound or lower-bound) then we can use a simple detector that would check when the data goes below or beyond the threshold. Otherwise, if the anomaly is not that simple, we could use a classifier that will then use all the features to predict if that day is with or without failure.

Another solution could be to use all the data (features + label) and try to predict them all at once. Considering this problem as a multivariate time series forecasting. The next features and labels would then be predicted at the same time.

I choose to go for the second type of solutions (multivariate time series forecasting) as it needs only one step (direct prediction) instead of two steps (regression + classification) for the first possible solution.

Solution

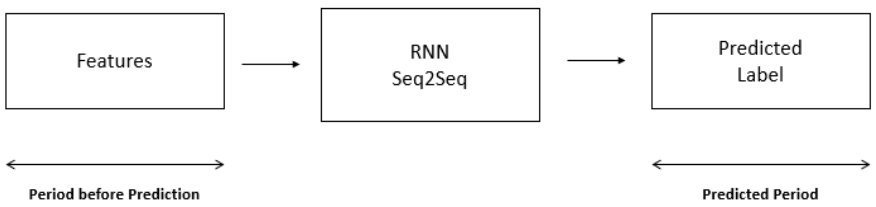
Our solution is to go for the approach with the least steps which will hopefully be the most effective approach. This means using only one algorithm that will predict the labels (or/and the features) directly.

We have to choose one algorithm for benchmark and another one that will be the selected model to compare to the benchmark. The Benchmark model will be described further in the Benchmark chapter below.

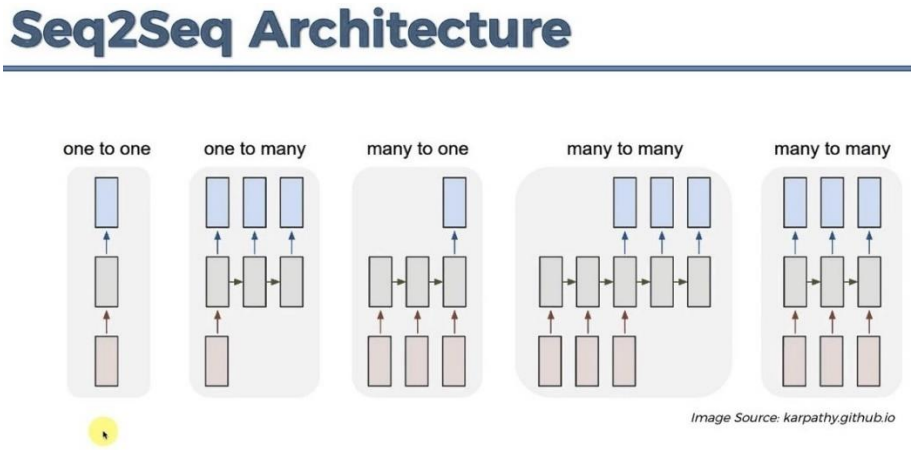
RNN Seq2Seq

Only one algorithm will be selected for the problem: Deep Learning-RNN-Seq2Seq. This algorithm is well suited for sequence prediction. And Multivariate Times Series can be also be considered as sequences.

The algorithm will have all the features as inputs, and it will predict the machine state: machine failure or not.



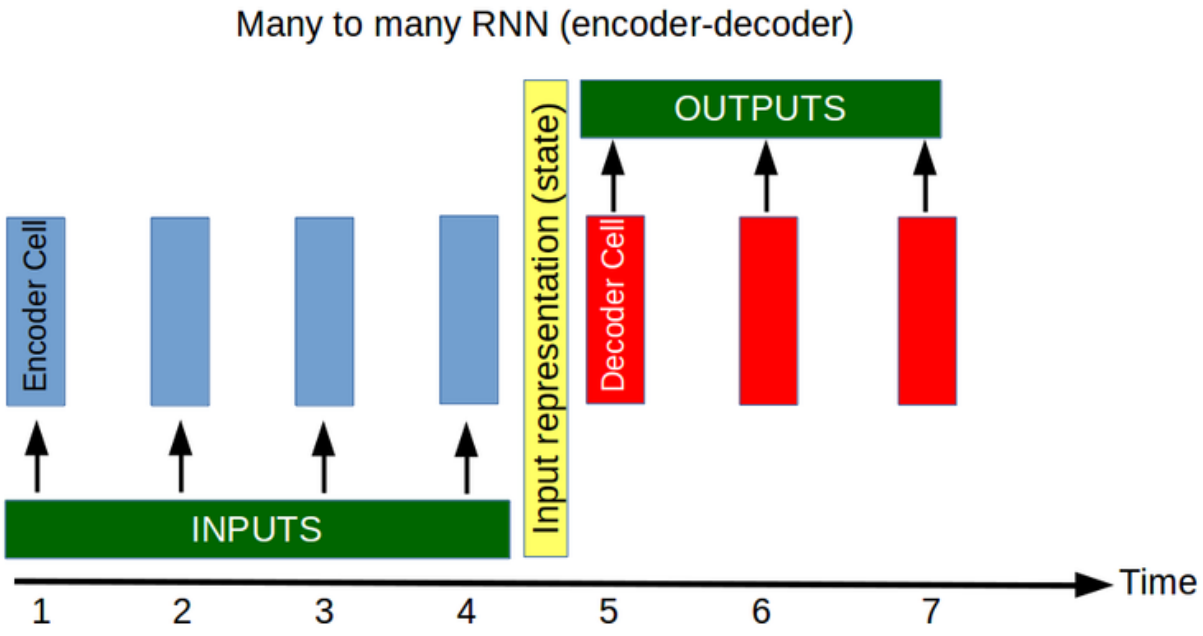
This algorithm has the advantage of taking data in input and predict a category directly without going through the two steps described above (Regression + Classification). It will directly take this problem as a classification problem.



Many variations exist for the Seq2Seq architecture, we choose to implement a flexible approach that allows to use any of these variations (many to one, one to one, one to many …).

RNNs Seq2Seq consists of two layers:

- Encoder RNN: Process input sequence and returns it is internal state and its outputs (ignored here)
- Decoder RNN: Predicts next outputs given the encoder state (See architecture below which is: 4 inputs -> 3 outputs)



The implemented model will just take two parameters:

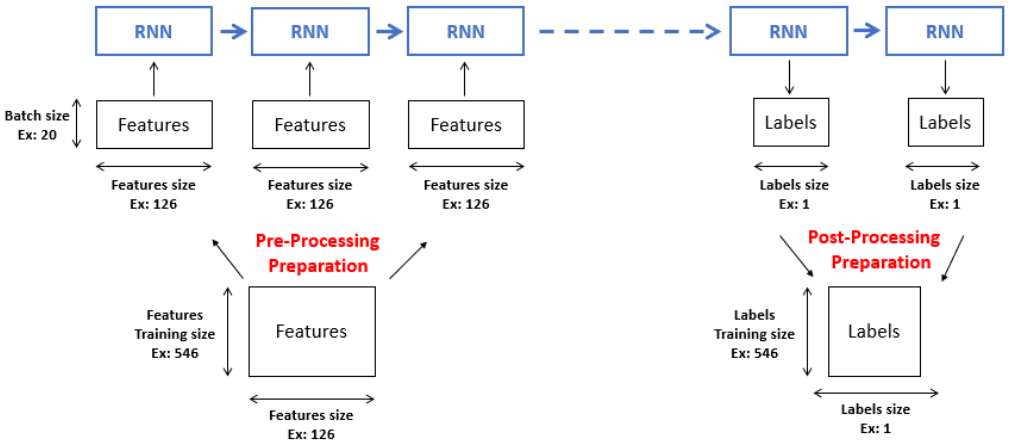
- input_sequence_length

- output_sequence_length

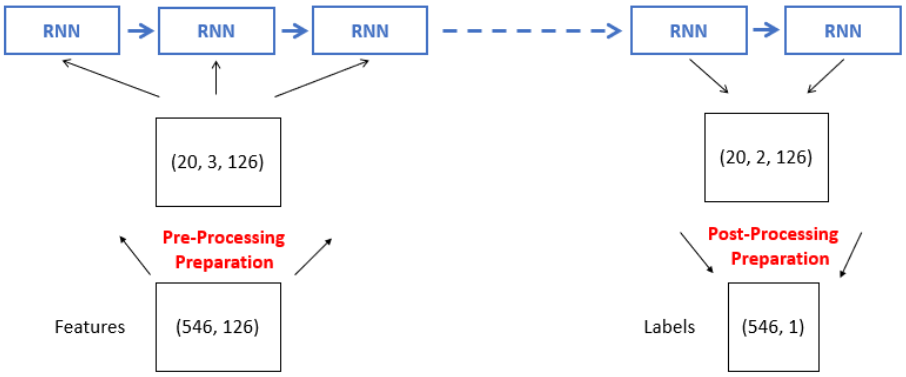
As an example, a one to one architecture is when input_sequence_length=1 and output_sequence_length=1. A one to many architecture is when input_sequence_length=1 and output_sequence_length is greater than one etc.

This flexible approach will however imply complex pre and post processing phases. The inputs and outputs must be transformed to be fed to the algorithm during training and transformed back to the initial shape when prediction is done.

A detailed example (RNNSeq2Seq- 3inputs->2outputs) is described in the following picture:



A simplified version of the example above is also depicted here:



The training phase requires batches of data. A batch is a subset of the training data. The batches will be used for training and the whole training set is used by the algorithm in an epoch. The training takes usually more than one epoch (the dataset is used many times by the algorithm).

In our solution, the dataset is quite small so the batches will be prepared with data augmentation. The same data will be reused to create a potentially infinite number of batches for the training (See Data Preprocessing chapter).

The evaluation technique for this model is imposed by the Kaggle competition. It is the AUC of the label that will still be used for measuring model correctness (See Metrics chapter).

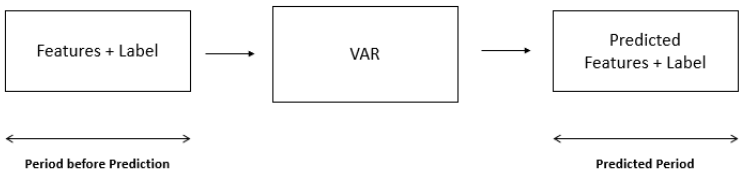
Benchmark

The benchmark model is a well-known algorithm for multivariate time series forecasting: Vector AutoRegression aka VAR.

This model is presented as the most successful and easy to use model for Multivariate Time Series (see first paragraph of this article: https://faculty.washington.edu/ezivot/econ584/notes/varModels.pdf).

This model will serve as a baseline for our solution (See description of selected model above) which is more flexible but also require more tuning.

Here is a diagram describing the way we will use the VAR algorithm:



The specificity of the VAR algorithm is that it gets in input a list of signals for a time period (say for example 100 days) and that it will predict all these signals for a specified period (example: 5 days). This algorithm then needs the features and labels in input not like traditional machine learning algorithm where an algorithm needs features and predicts labels.

III. Methodology

Data Preprocessing

Dates

All the dates (which were features in the initial dataframe) were transformed in indexes. The datasets then had DateTimeIndexes. That gives us 130 columns features (initially: 131 columns) and 1 column label (initially: 2 columns).

For the RNN Seq2Seq the pandas dataframes were converted to numpy arrays so the Date Time Index was lost.

Nan values

All Nan values were filled with zeros (fillna method on pandas dataframe).

Constant feature columns

After exploration we noticed that several columns were constant (all values at 0). It was all the error19 tick values (4 columns).

We then decided to remove them in order to create the main training and testing datasets (supposed to be used for VAR and RNN Seq2Seq). The remaining features in the training and testing dataset is 126.

But after applying the VAR to this data, we had an error saying that some columns were constant. We then applied a script that removed the constant columns. The resulting pandas dataframe had 118 columns. We used it for VAR only.

The Seq2Seq algorithm works with constant columns so we kept the 126 feature columns (all but the dates and the error19 tick columns) for the Seq2Seq algorithm.

Scaling

We used a scaler before applying the data to the VAR and Seq2Seq algorithms.

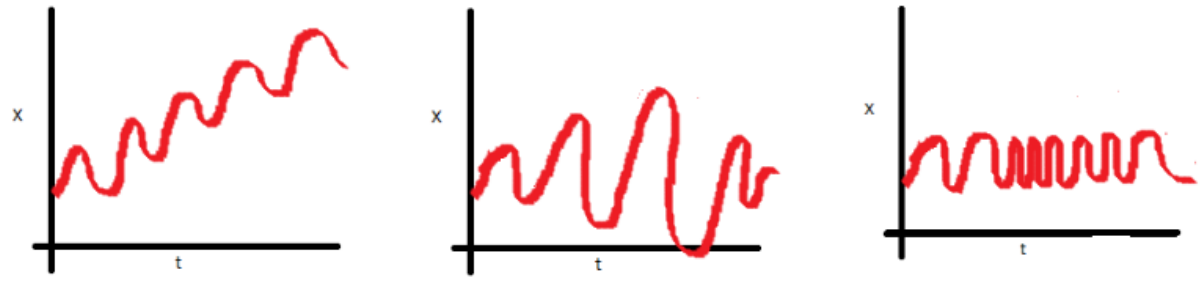
The scaler we chose is the MinMax scaler. It keeps the values between 0 and 1. This scaler was selected as it is the most used scaler.

Stationarity

The VAR algorithm only works with stationary data. We then had to check if the data we have is stationary.

Stationarity is one of the most important concepts when working with time series data. A stationary series is one in which the properties – mean, variance and covariance, do not vary with time.

Let us understand this using an intuitive example. Consider the three plots shown below:

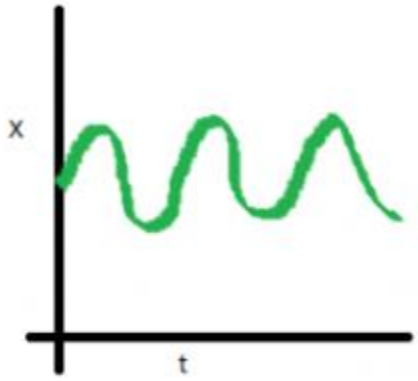


In the first plot, we can clearly see that the mean varies (increases) with time which results in an upward trend. Thus, this is a non-stationary series. For a series to be classified as stationary, it should not exhibit a trend.

Moving on to the second plot, we certainly do not see a trend in the series, but the variance of the series is a function of time. As mentioned previously, a stationary series must have a constant variance.

If you look at the third plot, the spread becomes closer as the time increases, which implies that the covariance is a function of time.

The three examples shown above represent non-stationary time series. Now let's look at another plot:



In this case, the mean, variance and covariance are constant with time. This is what a stationary time series looks like.

Predicting future values using the latter plot would be easier. Indeed, most statistical models require the series to be stationary to make effective and precise predictions.

The ticks represent the first, last, mean, stddev of the time of the day when the errors occurred. This data is repeating every day so we clearly expect a high correlation here.

For errors count, we need to see if they are correlated and we will apply a specific test for that: The Coint Johansen test.

According to this source (https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/), if this test returns values less than 1 in modulus, the features can be considered stationary.

We applied this test for a lag (input time period) of 1 as it was done in the source. This considers the stationarity with a lag of 1 day.

As the test only accepts 12 values for each call, we split the data in two subsets (columns 1 to 12 and columns 13 to 24). We skipped the remaining columns (25 and 26) for simplicity.

Pre-processing of 2D splits (train, validation and test data)

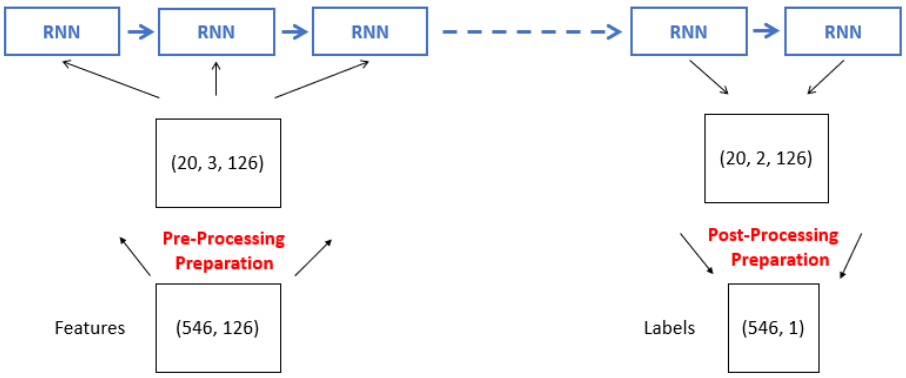
The Seq2Seq algorithm needs three splits:

- Seq2Seq Training data: to train the model (will be the source data of the generator)
- Seq2Seq Validation data: to check for overfitting/underfitting
- Seq2Seq Testing data: to check the model results

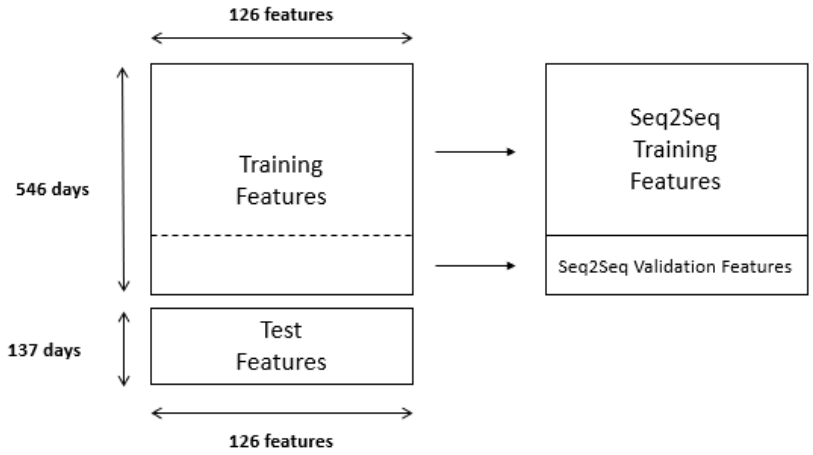
Although we have two datasets that are already prepared:

- Training dataset (546 rows)
- Testing dataset (137 rows)

We need to make sure we have three datasets with the right number of rows/days in order to feed the Seq2Seq with the right dimensions. We need an example to understand this problem: If we have a Seq2Seq with an input sequence length of 3 and an output sequence length of 2 (see image below).



We need to be able to generate a feature training (lower left side of image) dataset that has a multiple of 3 rows. If the training dataset size is not a multiple of 3, then it will be impossible to generate a dataset of shape (X, **3**, 126). Same thing for the validation data which will be taken from the training dataset:



The pre-processing of 2D splits consists in first splitting the training dataset in two datasets:

- Seq2Seq Training data
- Seq2Seq Validation data

The split will be done according to a validation split value (ex: 0.1). The Seq2Seq Validation data size will be for example 0.1 * Seq2Seq Training data size which is 54(546*0.1). the remaining Seq2Seq Training data size will be 546-54=492.

The second step is to make sure these two datasets are multiple of 3 (input sequence length) so that they can be reshaped to (X, 3, 126). In this example it works. But in the case where the training or the validation data is not a multiple of 3, we need to remove the extra data to have a multiple of 3. And the resulting training dataset, validation datasets and testing datasets must be consecutive so possible data shifts are also done on the data.

The result of this pre-processing is final training, validation and testing dataset that will serve as an input for the generator and all 3D pre-processing functions.

Generator (Data Augmentation)

The Seq2Seq is a Deep Learning algorithm. Deep Learning algorithms need a lot of data to be able to converge to a good model.

The generator function is a function that returns a batch of data at every call during the training phase (also called fitting). The trick here is to return different data at every call (potentially infinite data) even though the dataset is very small and limited (only 546 days/rows of data).

This function is implemented as follows:

1. Create a random value between 0 and (546-size of data to return)
2. Get the data from the datasets (features and label) from this random value
3. Return the data (features and label)

This returns different chunks of data from the same dataset at every call.

The Seq2Seq algorithm needs many inputs for training:

- Encoder inputs: feature training data
- Decoder inputs: zeros here (see explanation below)
- Decoder outputs: label training data

In machine translation applications (see "A ten-minute introduction to sequence-to-sequence learning in keras") something called teacher forcing is used. In teacher forcing, the input to the decoder during training is the target sequence shifted by 1.

This supposedly helps the decoder learn and is an effective method for machine translation. However, if the input to the decoder is 0, it forces the model to really memorize the values that are fed to the encoder since it has nothing else to work on. In some sense, teacher forcing might artificially induce vanishing gradients.

The generator will generate encoder inputs and decoder inputs and outputs all at the same time. It then needs the features and labels as inputs. All the resulting data will also be reshaped to have the right dimensions for training (example: (20, 3, 126) for encoder inputs, (20, 1, 1) for decoder inputs and (20, 1, 1) for decoder outputs).

Pre-processing of 3D validation/inference feature data

The Seq2Seq training data is fed by the generator function. But it is highly recommended to also use validation data to make sure the training is not underfitting/overfitting. So, the training phase will need 2 sets of data:

- Training data
- Validation data

Each consisting of the following 3:

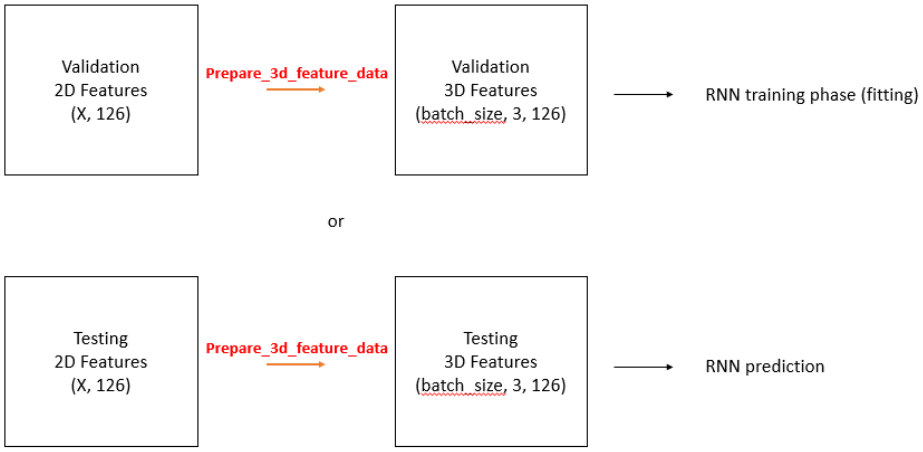
- Encoder inputs: (batch_size, input_sequence_length, 126)
- Decoder inputs: (batch_size, output_sequence_length, 1)
- Decoder outputs: (batch_size, output_sequence_length, 1)

We then need to create these 3 datasets for training and for validation. But training is already managed by the generator. So, we finally need to generate them for validation.

A function named “prepare_3d_feature_data” has been created to generate the inputs: Encoder inputs and Decoder inputs.

This function is also needed to generate data to feed the algorithm for predicting. If we have a test dataset of shape (Y, 126), we need to use the algorithm to do the prediction for this dataset, we will need to transform this dataset into Encoder inputs and decoder inputs to feed the algorithm. The same "prepare_3d_feature_data" will be used for that.

The following picture summarizes the objective of the "prepare_3D_feature_data":



The image above takes as an example an RNNSeq2Seq- 3inputs->2outputs.

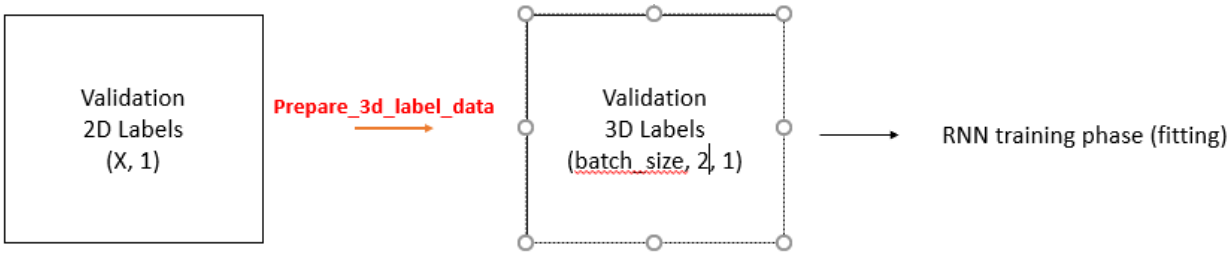
There is another function that has to generate the outputs (Decoder outputs) that is described below (3D validation label data).

Pre- processing of 3D validation label data

As it is explained above, the algorithm needs Decoder outputs for training and validation. The training decoder output is generated by the generator, so we now need decoder outputs for validation.

A function named "prepare_3d_label_data" was created for that. It will use the label data as an input and reshape it with some transformations (appending zeros if not enough data) in order to have the right shape ((batch_size, output_sequence_length, 1)).

The following picture summarizes the objective of the "prepare_3d_label_data":



The image above takes as an example an RNNSeq2Seq- 3inputs->2outputs.

Pre-processing and post-processing for the prediction step

The prediction phase is managed using a "predict" function. The role of this function is to apply the "prepare_3d_feature_data" described above to any dataset and use it to predict the machine failure for this dataset.

Of course, here we will use the testing dataset but any dataset with the same dimensions will work.

This function is implemented as follows:

1. Prepare 3d features
2. Use the features and predict using the Seq2Seq algorithm
3. Reshape the results to a 2D array

The resulting 2D array can be compared to the expected results.

Implementation

VAR

The VAR algorithm is available in a library called "statsmodels". We imported the library and used the VAR algorithm as is.

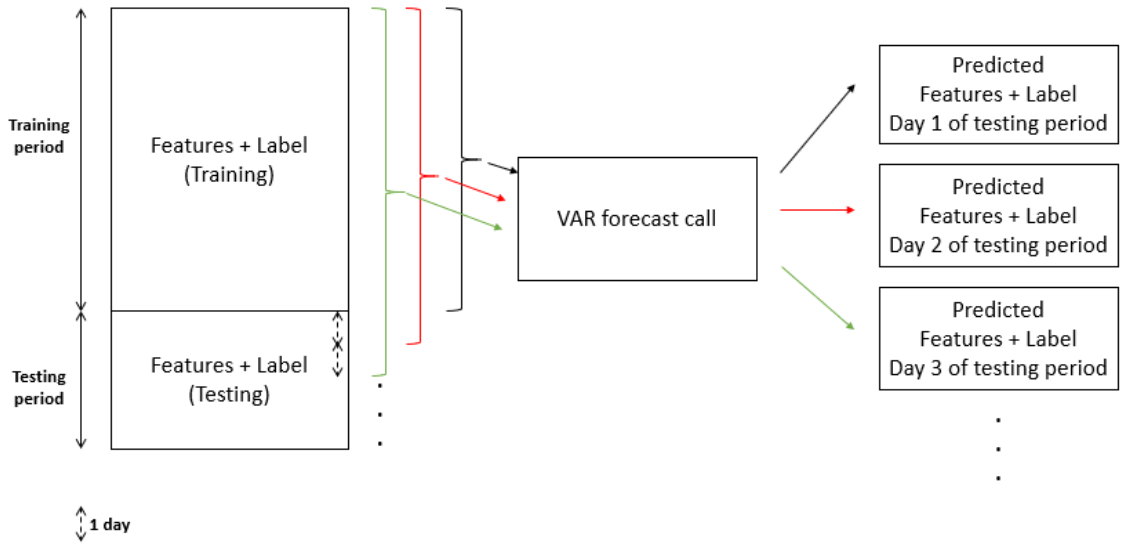
The var algorithm expects many parameters but we focused on the following that we noted as the most important:

- The training dataset is needed at the VAR class instantiation
- Then the following 2 params during the algorithm fit call:
    - maxlags: the maximum number of steps (in the past) to consider for the calculation of the forecast/prediction. Example: 10 - take the last 10 days/steps values to do the prediction
    - trend: The type of the last element of the equation to calculate the forecast. The VAR equation uses the past elements of all features to calculate the next features and it adds an extra parameter (either a constant or a trend or both). Possible values for this parameter are: constant, constant and trend, constant linear and quadratic trend and eventually no constant.
- At last the number of steps to predict is needed for the forecast call. Example: 30 - forecast the next 30 days/steps.

We tried to use the VAR algorithm to predict all the testing set at once (137 days in one forecast function call) and get the AUC but the results were below 40%. The resulting label prediction was a few peaks for the first 3 days followed be a slowly decreasing line that always remained below 0.05. The prediction was 0 for almost all the testing data.

To improve these results, we decided to use the VAR algorithm to predict only one day. The algorithm would be good for predicting the next day only. In order to evaluate this

algorithm with regards to the testing dataset (features + labels), the idea was to generate a prediction dataset (predicted features + labels for the period corresponding to the testing dataset) using the testing dataset (See image below for clarification).



The predicted dataset labels will then be used to evaluate the AUC (Predicted labels compared to the testing labels).

Here we will not split the training set and the testing set. All the data will be used by the VAR to predict the next day. The predicted days will correspond to the dates in the testing dataset. The testing dataset is indeed the last 137 days of the whole dataset. So we will predict the last 137 days (prediction dataset) and calculate the AUC against the real last 137 days (last 137 rows of the whole dataset).

The VAR forecast function will then be called 137 times. Each time, the input will be all the features and label before the day to be predicted.

The VAR algorithms does not accept constant columns (features) so we needed to remove them. The training and testing datasets had finally 117 features + 1 label (118 columns in total).

Seq2Seq

The RNN seq2Seq algorithm was implemented as specified.

It was implemented using the Keras library using TensorFlow as a backend.

The code is self-explanatory so there is no need to give more details. However, it can be good to know the different steps of the implementation:

1. Hyperparameters constants
2. Data Pre-Processing
3. Encoder implementation
4. Decoder implementation
5. Model compilation
6. Model training/fitting
7. Loss evaluation
8. Prediction

Refinement

VAR

The refinement for the VAR consisted in testing the “trend” parameter and keep the one with the best results.

Seq2Seq

For the Seq2Seq model a huge hyperparameter tuning phase was done. I spent nearly 80% of the time tuning and 20% of the time implementing the model.

Every time a parameter is changed, the following steps were applied:

1. Training
2. Displaying the training and validation loss
3. Predicting for the test dataset
4. Displaying the AUC result on the testing dataset

Then overfitting/underfitting or bad AUC would lead us to change the parameters again to improve.

The whole tuning went through the following steps:

- Fixing an architecture (Example: 3 inputs -> 1 output)
- Fixing some hyperparameters:
    - Validation split: 0.1 (we kept the minimum as the dataset is already small)
    - Loss function: binary cross entropy (as it is a classification problem)
    - Input sequence length: 3
    - Output sequence length: 1
- Tweaking the following parameters (classed by decreasing order of importance):
    - Learning rate
    - Optimizer
    - Regularizers
    - Initializers (of initial weights)
    - Dropout
    - Number of layers
    - Neuros per layer
    - Activation function
    - Batch Size
    - Steps per epoch
    - Epochs

The objective of this step is to have the best hyperparameters for this architecture (3->1) and then change the input and output sequence lengths (architecture) to see if we can have even better results.

- Tweaking the input and output sequence lengths:
    - Best result is with input sequence 1 and output sequence 1

The conclusion of this tuning led to the implementation we have in the provided jupyter notebook.

The next step led to results that were not strong enough to justify a switch.

- Tweaking the loss metric
    - Mean Squared Error (MSE) was giving similar results than the Binary Cross entropy and with slightly better quality (more peaks in the resulting prediction) but we choose to keep the Binary Cross entropy as it is a classification metric. The MSE is used for regression problems and is not suitable to our problem.

The optimal parameters can be checked in the jupyter notebook.

IV. Results

Model Evaluation and Validation

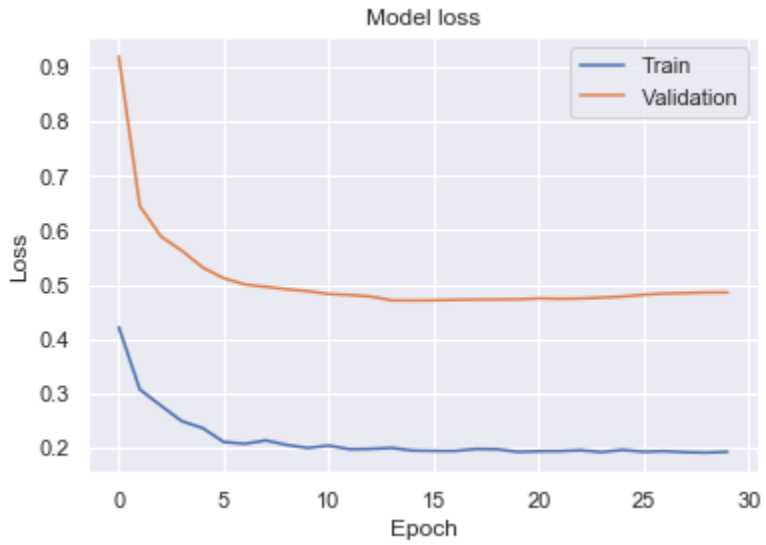
The chosen model results from a substantial tuning phase. As explained above, the evaluation and validation were applied each time a parameter was changed.

The evaluation was done by displaying the loss of the training and validation data. This allows us to see if the model generalizes well (no underfitting nor overfitting).

The final model is below expectations as the binary cross entropy is quite high for the validation data (around 0.48) and the model had a lot of trouble generalizing.

We had to keep a small number of epochs otherwise the model started to overfit. Keeping a small number of epochs means having a short training and the validation and training loss cannot be reduced to low values.

The final loss graph is depicted below:



We can clearly see that the training loss decreases steadily, and that the validation loss decreases as well. At 15 epochs the training and validation losses seem to stabilize.

We see no clear signs of overfitting (except a tiny increase after 25 epochs but this can be considered as noise). The training was stopped before the overfitting is clearly visible.

However, tuning all hyperparameters did not allow the validation loss to reach values lower than 0.48. The gap between the training loss and validation loss can be a sign of underfitting (depending on the scales). **We will consider it underfitting** in this case as both training and validation losses (binary cross entropy) are still high after 30 epochs (0.19 and 0.48).

Good binary cross entropies would be at 0.1 or less. Here the result is clearly indicating that the model cannot learn well from the data.

The training was stopped at 30 epochs, just before the model starts to overfit.

The model was not trained on a sample but on the whole dataset. There is no possible variation of the inputs to play with here.

Advantages & Drawbacks of each model

Benchmark: VAR

The implemented VAR algorithm has many advantages:

- **Model simplicity**: No need for feature preparations.
- **Prediction of label and features:** Pure time-series prediction.
- **Well known method:** Old method with very well-known formulas and improvements (statsmodels library).
- **Deterministic:** Running the algorithm many times with the same data gives the same result every time.
- **Not Data hungry:** Needs very little data to calculate predictions.

But our VAR implementation has also some drawbacks:

- **High probability of divergence**: Predicted features are used to predict label instead of real features (this needs to be improved with feature preparation).
- **Poor results for long predictions**: When deciding to predict many days at once, the results were very poor. So, we decided to use this model to predict one day at most. The cost was running the whole algorithm for every day predicted (Performance cost).
- **Performance**: The implemented VAR needs to be running for every day predicted which is very costly. The model is not a machine learning algorithm, so it processes all data to predict one day (No concepts of training and inference). It is not scalable. As an example, here it takes 5 to 10 minutes to predict 137 days.

Solution: Seq2Seq

The implemented Seq2Seq algorithm has many advantages:

- **Performance**: Can predict many days given only the previous X days (depending on the input sequence). Scalable. Can predict effectively many days in advance. Training and Inference are separated (Inference is really fast - less than 5 seconds to predict 137 days here).
- **High probability of convergence**: The model always uses the real features to predict labels.

- **Designed for long predictions**: The model can learn to predict many days in advance (output sequence of many days).
- **Recent method but well documented:** Many libraries implementing it (Keras, Tensorflow, Pytorch, MXNet ...) and other libraries facilitating hyperparameters search (SMAC3 - not implemented here due to lack of time).
- **Prediction flexibility**: Can be used to predict time series only or labels as a classification problem (that is what we chose here).

We can clearly see here that the Seq2Seq model covers all the drawbacks of the VAR algorithm and it also shares some advantages with the VAR algorithm. But we will see that it also has drawbacks that are covered by the VAR algorithm (See next).

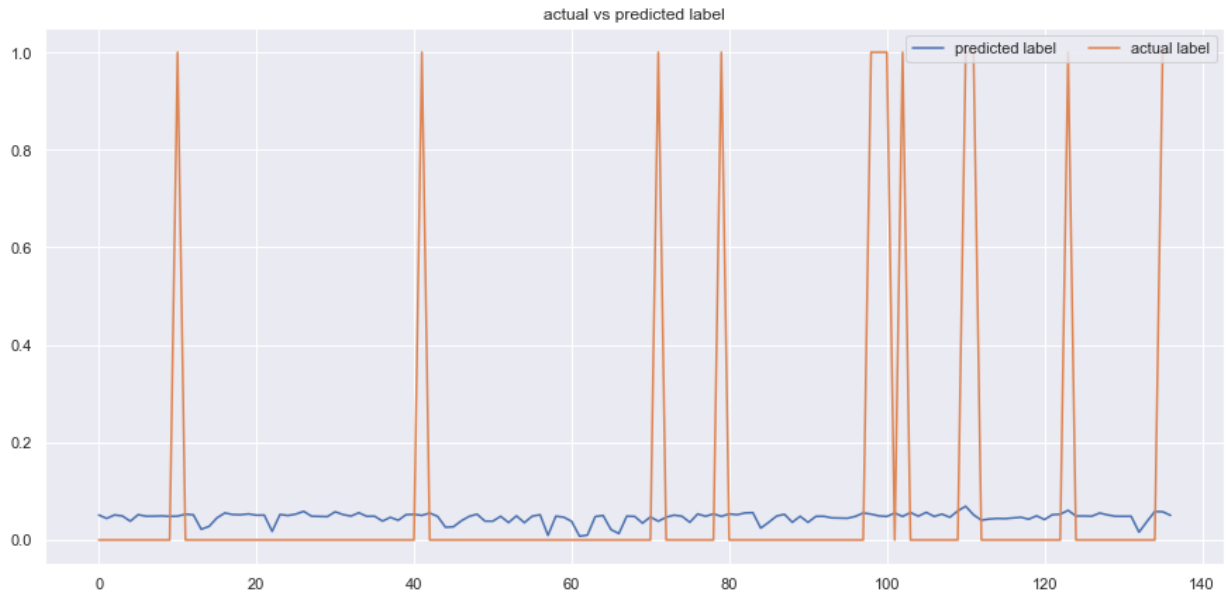
Seq2Seq drawbacks are as follows:

- **Model complexity**: Many hyperparameters to tweak and heavy data preparation needed.
- **Non-Deterministic**: Running the algorithm many times with the same data gives slightly different results every time.
- **Data Hungry**: Needs a lot of data to converge to a good model.

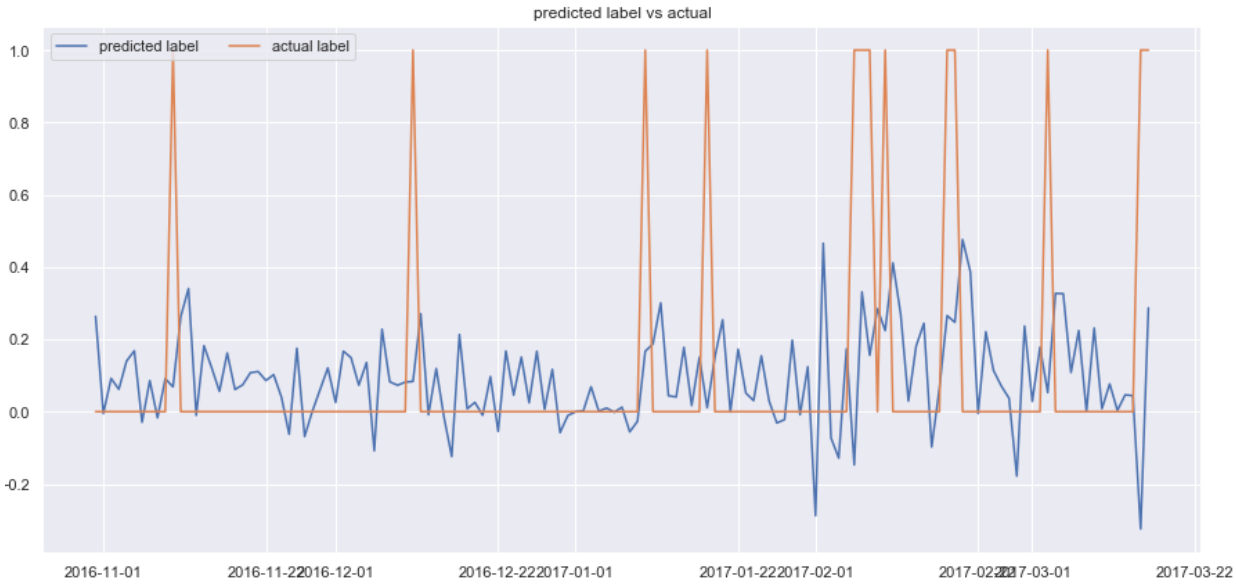
Justification

The results in the test prediction are quantitatively higher that the benchmark. The improvement is significant (+ 9%) but it also required significant time and effort.

Qualitatively, the results are not satisfactory as the shape of the prediction on the test data is relatively flat. Meaning that we cannot see clear peaks around the machine failures (see image below).



In the VAR algorithm we can clearly see peaks that are higher near the machine failures (see image below).



In theory, the Seq2Seq model seems to be the best of both models as it has many advantages of the VAR but it comes with a cost which is complexity and non-determinism.

Even though the quantitative comparison is in favor of the Seq2Seq algorithm, the qualitative comparison is very explicitly going in favor of the VAR algorithm. Compared to the Seq2Seq model, the VAR model clearly shows higher peaks near the actual failure days. Instead, the Seq2Seq model tried to have the best possible binary cross entropy loss even if that means having an almost flat prediction.

The Seq2Seq model needs more data and more tuning to be able to give results that are qualitatively correct. I tried to use another loss technique: Mean Squared Error (MSE) in order to have better qualitative results but the AUC was lower, and the quality of the results were just slightly better. I then choose to keep the Binary Cross entropy loss as it is a categorical loss. The MSE is a distance indicator that should be used for pure time series prediction (regression) and not for classification.

The poor results (underfitting) of the Seq2Seq model are most likely due to the small dataset size and the high dimensionality. I have no doubt that more data and more feature engineering (reducing the number of features/combining features/creating features ...) will give substantially better results for the Seq2Seq model.

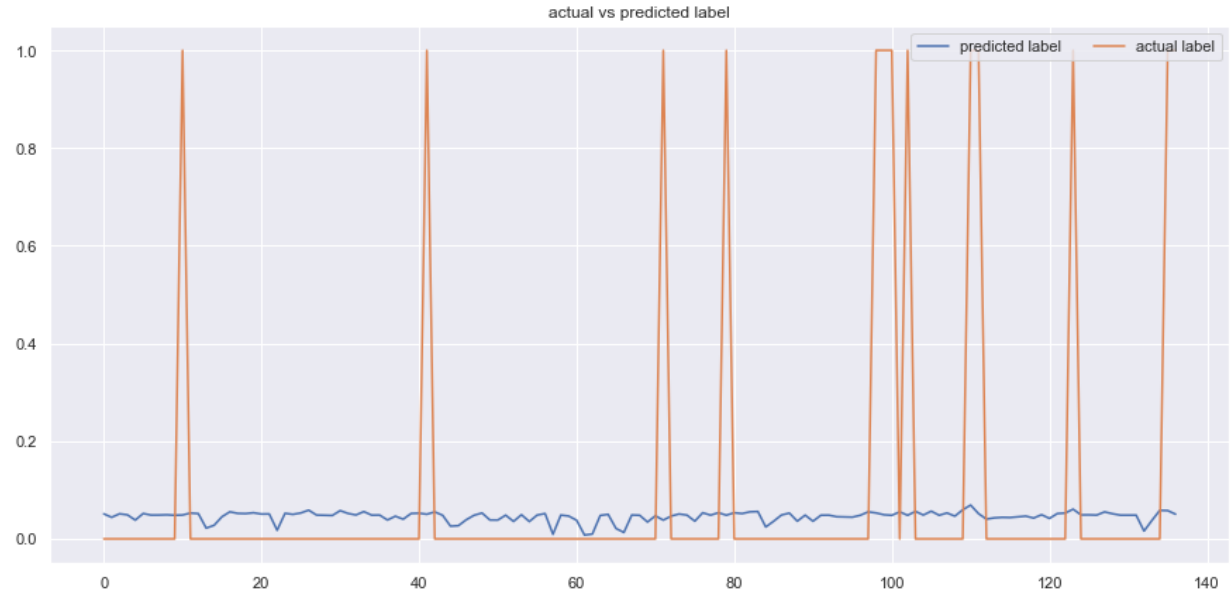
As an indication, the leaderboard of the Kaggle competition (https://www.kaggle.com/c/predictive-maintenance1/leaderboard) shows that a result of 0.652 would be at the third position.

My AUC was not calculated on the same data so we cannot compare but that shows an indication of the difficulty of this problem and the corresponding dataset.

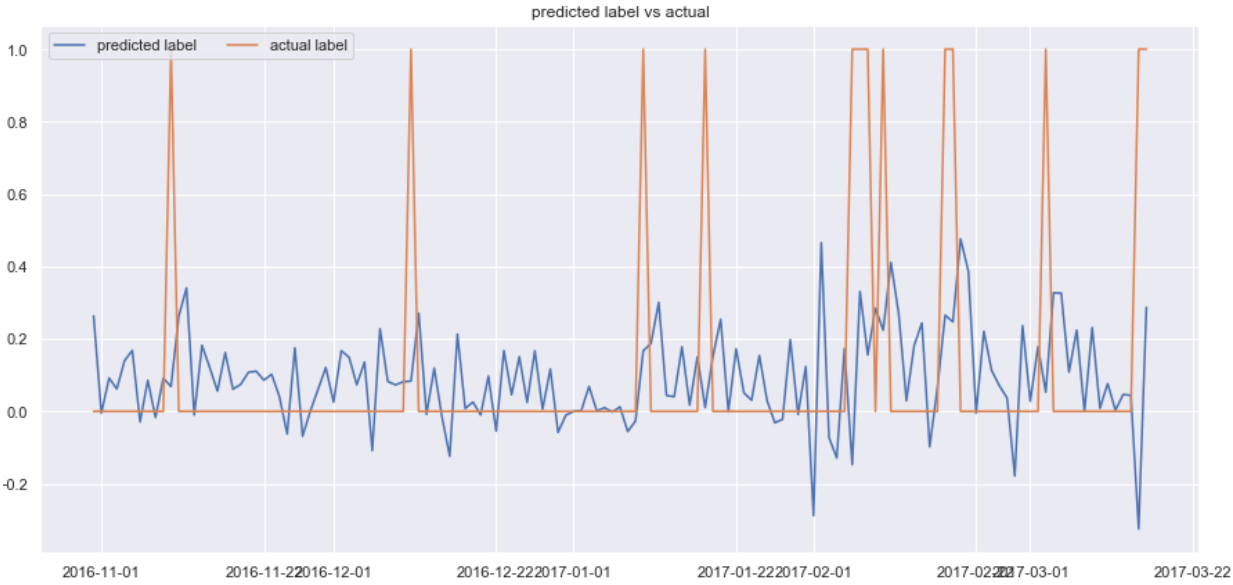
V. Conclusion

Free Form Visualization

Seq2Seq prediction on the test dataset:



VAR prediction on the same test dataset:



The results are self-explanatory. The VAR is qualitatively more suited to be used to predict a machine failure as the Seq2Seq algorithm.

Reflection

The interesting aspects of this project is the **definition of a benchmark and the challenge to overcome the benchmark result using another model**.

Another interesting point is the **use of a Non-Machine Learning algorithm** and a pure statistics model (VAR) and the comparison to the Machine Learning alternative. We clearly see here that the statistics model is not very flexible and designed specifically for time-series problems. The difference with the Seq2Seq algorithm is that it is more flexible and can be used for totally different use cases (Machine Translation …).

The **data preparation** for the selected model was quite a challenge here but the most difficult part was the **hyperparameter search**.

It was difficult but also very rewarding as I had to check all the possible hyperparameters and their meaning and effect on the data.

The result of this capstone project is an underfitting model but in order to try to escape underfitting I had to use many hyperparameters and see the effects on the result.

The model could not generalize well and had an inclination towards **overfitting and underfitting**. I think that the **very small dataset** (683 rows/days in total) was the **main reason for that**.

Improvement

Three directions could be considered for improving this project:

- **More data**: simplest way to improve the results would be having more data and thus more room for hyperparameter tuning.
- **Features Engineering**:
    - Creating new meaningful features: For example, adding all the error counts and creating a feature that we could name “totalerrors”. That feature might be a clear indicator of a machine failure.
    - Simplification of the features: using bucketing for all the errors and ticks in order to simplify the features and facilitate the model generalization. For example, transforming the ticks from a value between 0 to 86400 (number of seconds in a day) to categories like early morning, morning, afternoon, late afternoon and night.
- **Architecture change:** Change the architecture and choose a simpler model as for example a Multi-Layer Perceptron (MLP) or a Random Forest.

The architecture change should be used only if more data and features engineering bring no further improvement to the Seq2Seq.