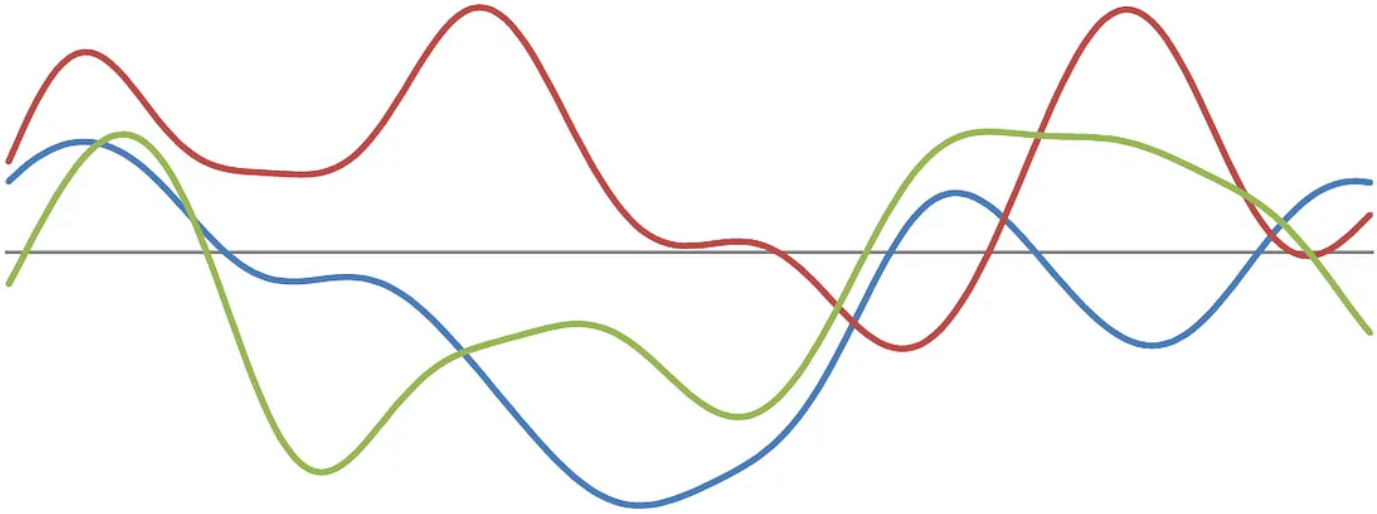
MACHINE LEARNING APPROACHES FOR TIME SERIES DATA

ML Approaches for Time Series

Pablo Ruiz · Follow
Published in Towards Data Science
14 min read · May 19, 2019

Listen Share

Modeling time series with non conventional models



In this post I play around with some Machine Learning techniques to analyze time series data and explore their potential use in this case of scenarios.

In this first post only the first point of the index is developed. The rest have a separate post which can be accessed from the index.

Note: This work was done by the beginning of 2017 so it is very likely that some libraries have been updated.

Index

1 — Data Creation, Windows and Baseline Models

1.1 — Data Creation

In this work we will go through the analysis of **non-evenly spaced time series** data. We will create synthetic data of 3 random variables *x1*, *x2* and *x3*, and adding some noise to the linear combination of some of the lags of these variables we will determine y, the response.

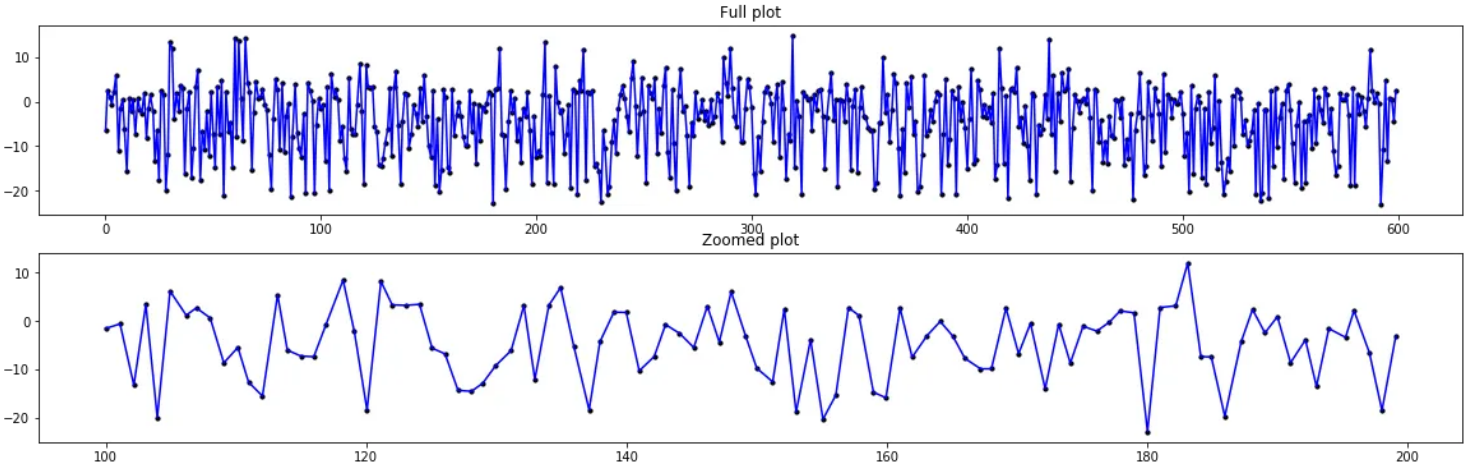
This way we can make sure that the function is not 100% predictable, the response depends on the predictors, and that there is a **time dependency** caused by the effect of previous **lags of the predictors** on the response.

This python script will create windows given a time series data in order to frame the problem in a way where we can provide our models the information the most complete possible.

Let's see then, in the first place, which is the data we have and what treatment we are going to apply.

```
N = 600

t = np.arange(0, N, 1).reshape(-1,1)
t = np.array([t[i] + np.random.rand(1)/4 for i in range(len(t))])
t = np.array([t[i] - np.random.rand(1)/7 for i in range(len(t))])
t = np.array(np.round(t, 2))

x1 = np.round((np.random.random(N) * 5).reshape(-1,1), 2)
x2 = np.round((np.random.random(N) * 5).reshape(-1,1), 2)
x3 = np.round((np.random.random(N) * 5).reshape(-1,1), 2)

n = np.round((np.random.random(N) * 2).reshape(-1,1), 2)

y = np.array([((np.log(np.abs(2 + x1[t])) - x2[t-1]**2) + 0.02*x3[t-
```

```
3]*np.exp(x1[t-1])) for t in range(len(t))])
y = np.round(y+n, 2)
```



Then, we have a function y which is the response of 3 independent random variables and with an added noise. Also, the response is directly **correlated with lags of the independent variables**, and not only with their values at a given point. This way we ensure **time dependency**, and we force our models to be able to identify this behavior.

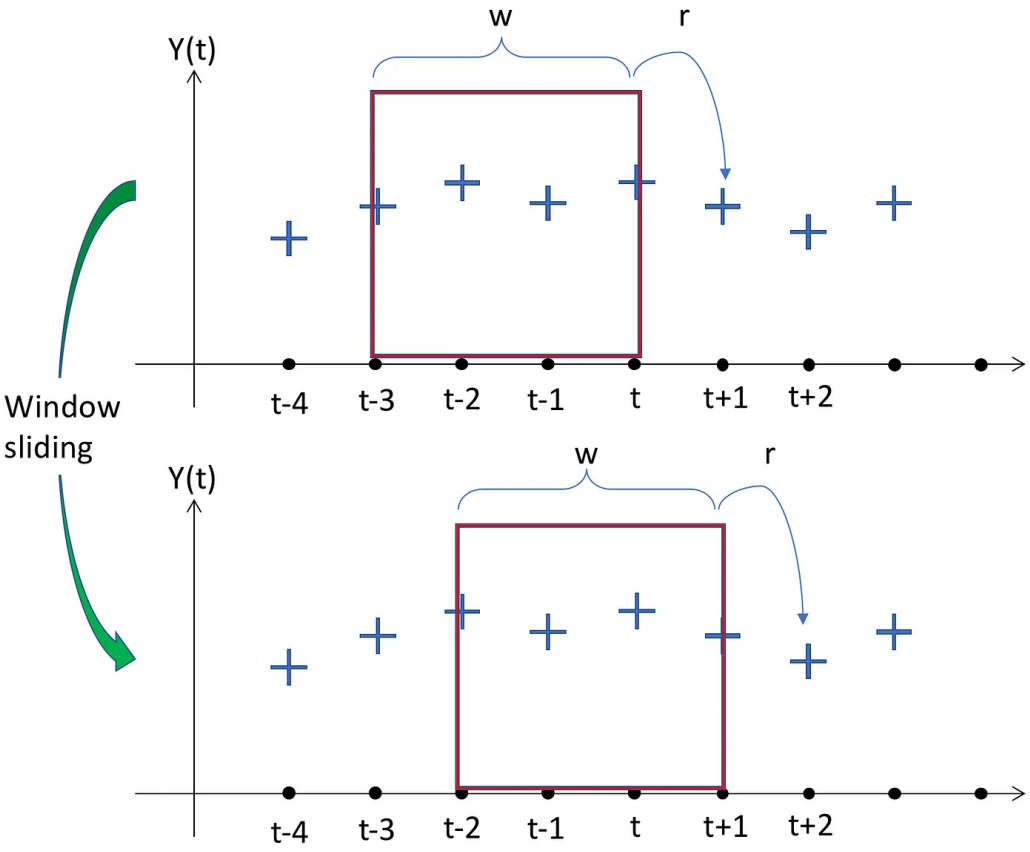
Also, the timestamps are **not evenly spaced**. This way, we reinforce the idea that we want our models to understand time dependency, as they can not just treat the series by number of observations (rows).

We have included exponentials and logarithmic operators with the intention to **induce high non-linearities** in the data.

1.2 — Framing by Windows

The approach followed for all the models of this work is to reshape the information we have by fixed windows that will give the model the most complete information possible at a given time point from the recent past, in order to achieve an accurate prediction. In addition, we will check how giving previous values of the response itself as an independent variable affects the models.

Let's see how we are going to do it:



The picture shows only the axes of time and the response. Remember that, in our case, there is 3 more variables which are the responsible for the values of the *t*.

The picture in the top shows a windows of a chosen (and fixed) size w, which in this case is 4. This means that, the model is going to map the information contained in that windows, with the prediction at point which is at *t+1*. There is an r in the size of the **r**esponse, because we could want to predict several time steps in the past. This would be a **many-to-many** relationship. For simplicity and easier visualization, we will work with `r=1`.

We can see now the effect of **Sliding Window**. The next pair of inputs-outputs that the model would have for finding the mapping function is obtained by moving the window one time step to the future, and proceed the same as we did at the previous step.

Ok then. How do we apply this to out current dataset? Let's look at what we need, and build our helper functions.

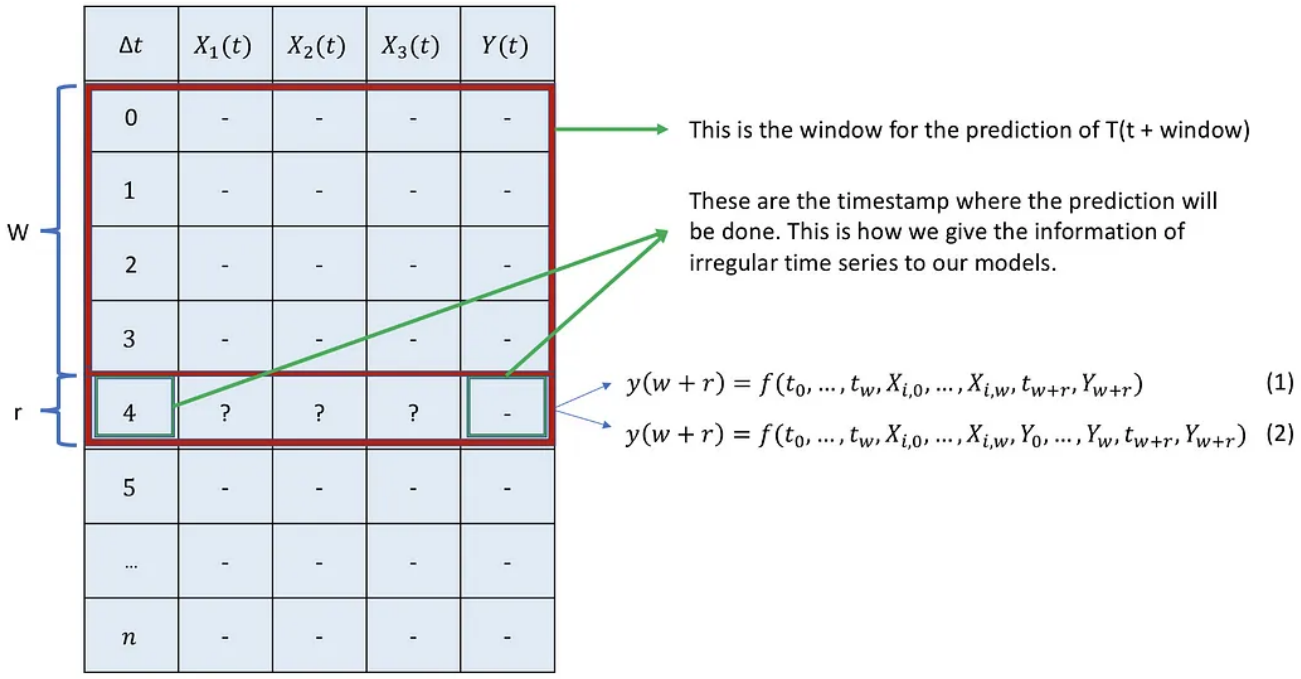
But first, we don't want time to be absolutes values, we are more interesting in knowing which is the **elapsed time between observations** (remember the data is NOT evenly spaced!). Therefore, let's create a Δt and take a look at our data.

```
dataset = pd.DataFrame(np.concatenate((t, x1, x2, x3, y), axis=1),
                       columns=['t', 'x1', 'x2', 'x3', 'y'])

deltaT = np.array([(dataset.t[i + 1] - dataset.t[i]) for i in
range(len(dataset)-1)])
deltaT = np.concatenate((np.array([0]), deltaT))

dataset.insert(1, '△t', deltaT)
dataset.head(3)
```

	t	Δt	x1	x2	x3	y
0	0.19	0.00	0.11	0.72	2.39	-6.52
1	1.01	0.82	3.39	1.02	0.09	2.55
2	2.21	1.20	0.17	1.98	3.76	1.11

Now that we know how our dataset will look like that, let's recreate what we want our helper function to do over a scheme of a table.

For a window of size 4:



What our function will do is to flatten all the information contained in the window, which is all the values inside the W window, and the timestamp(s) of when we want to make the prediction.

This way, we could have 2 different equation to model our system, depending if we include previous values of the responses as new predictors.

The result that the function have to return should look like:

#	$\Delta t(0)$	$\Delta t(1)$	$\Delta t(2)$	$\Delta t(3)$	$X_1(0)$	$X_1(1)$	$X_1(2)$	$X_1(3)$	...	$\Delta t(4)$	$Y(4)$
0	0	-	-	-	-	-	-	-	-	-	-
1	0	-	-	-	-	-	-	-	-	-	-
...	0	...	...	...	...	...	...	...	...	...	...
$n-(w+r)+1$	0	-	-	-	-	-	-	-	-	-	-

We will be able to create `l = n - (w+r) +1` windows, because we lose the first rows, as we don't have previous information for the first value of *Y(0)*.

All the lags we have mentioned acts like new predictors for the model (in this visualization the previous values of *Y* are not included, they will follow the same as the *Xi*. Then, the timestamps (elapsed) where we want the prediction to be done *Δt(4)*, and the corresponding value of what the prediction should be *Y(4)*. Note that all the first *Δt(0)* are initialized to *0* as we want to standardize every window to be in the same ranges.

Here is the code created to achieve this process. There is a function WindowSlider from which we can create objects to construct different windows changing the parameters.

3 — Baseline Models

> *"Always do the simple thing first. Only apply intelligence when required" — Thad Starner*

Create Windows

```
w = 5
train_constructor = WindowSlider()
train_windows = train_constructor.collect_windows(trainset.iloc[:,1:],
                                                  previous_y=False)

test_constructor = WindowSlider()
test_windows = test_constructor.collect_windows(testset.iloc[:,1:],
                                                previous_y=False)

train_constructor_y_inc = WindowSlider()
train_windows_y_inc =
train_constructor_y_inc.collect_windows(trainset.iloc[:,1:],
                                                  previous_y=True)

test_constructor_y_inc = WindowSlider()
test_windows_y_inc =
test_constructor_y_inc.collect_windows(testset.iloc[:,1:],
                                                previous_y=True)

train_windows.head(3)
```

	Δt(1)	Δt(2)	Δt(3)	Δt(4)	Δt(5)	x1(1)	x1(2)	x1(3)	x1(4)	x1(5)	...	x2(3)	x2(4)	x2(5)	x3(1)	x3(2)	x3(3)	x3(4)	x3(5)	Δt(6)	y
0	0.0	0.78	1.98	2.80	3.80	3.74	1.82	1.02	4.75	1.52	...	0.66	0.60	3.80	3.48	2.90	2.79	1.38	3.38	4.93	-11.47
1	0.0	1.20	2.02	3.02	4.15	1.82	1.02	4.75	1.52	0.72	...	0.60	3.80	4.24	2.90	2.79	1.38	3.38	0.53	5.12	-15.55
2	0.0	0.82	1.82	2.95	3.92	1.02	4.75	1.52	0.72	0.55	...	3.80	4.24	0.88	2.79	1.38	3.38	0.53	1.62	4.95	1.66

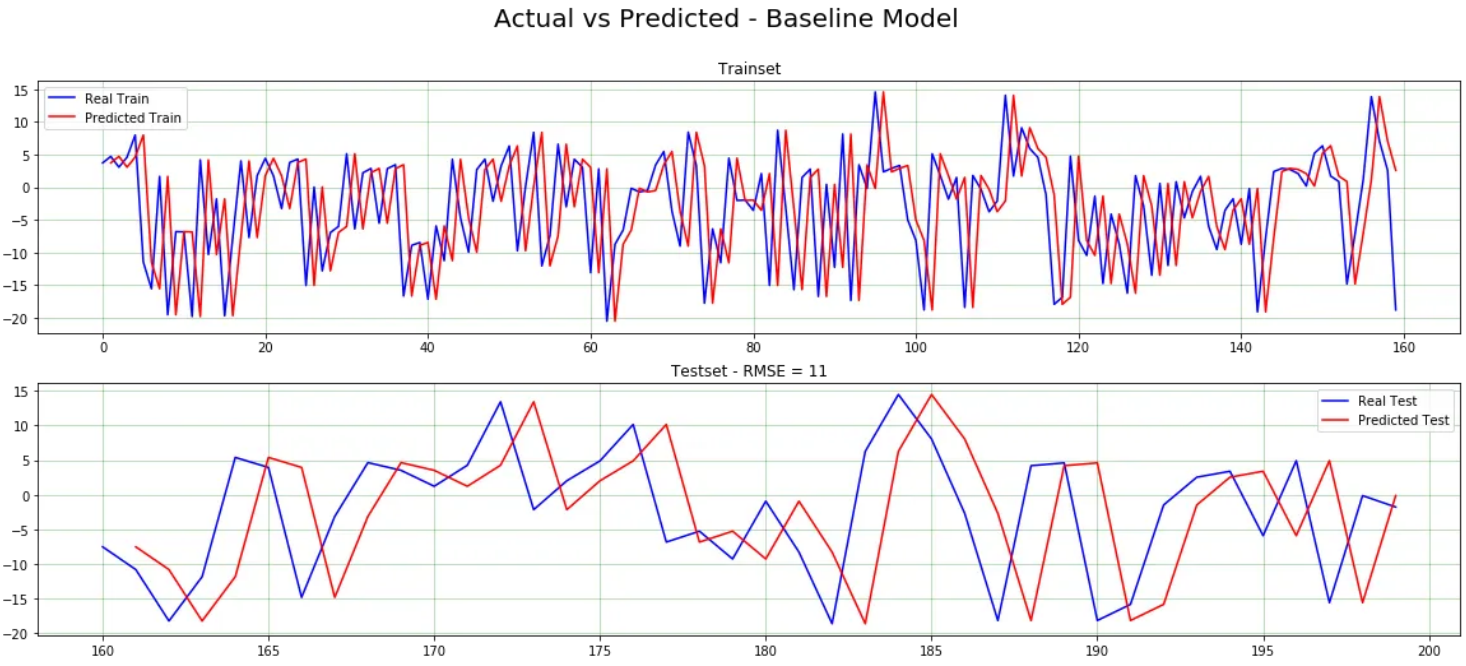
We can see how the windows brings for every prediction, the records of the (window_length) time steps in the past of the rest of the variables, and the accumulative sum of Δt.

Prediction = Current

We will first start with a simple model that will give the last value (the current one at each prediction point) as the prediction for the next timestamp.

```
# _________________ Y_pred = current Y _________________
bl_trainset = cp.deepcopy(trainset)
bl_testset = cp.deepcopy(testset)

bl_y = pd.DataFrame(bl_testset['y'])
bl_y_pred = bl_y.shift(periods=1)
```

```
bl_residuals = bl_y_pred - bl_y
bl_rmse = np.sqrt(np.sum(np.power(bl_residuals,2)) /
len(bl_residuals))
print('RMSE = %.2f' % bl_rmse)
print('Time to train = 0 seconds')

## RMSE = 11.28
```



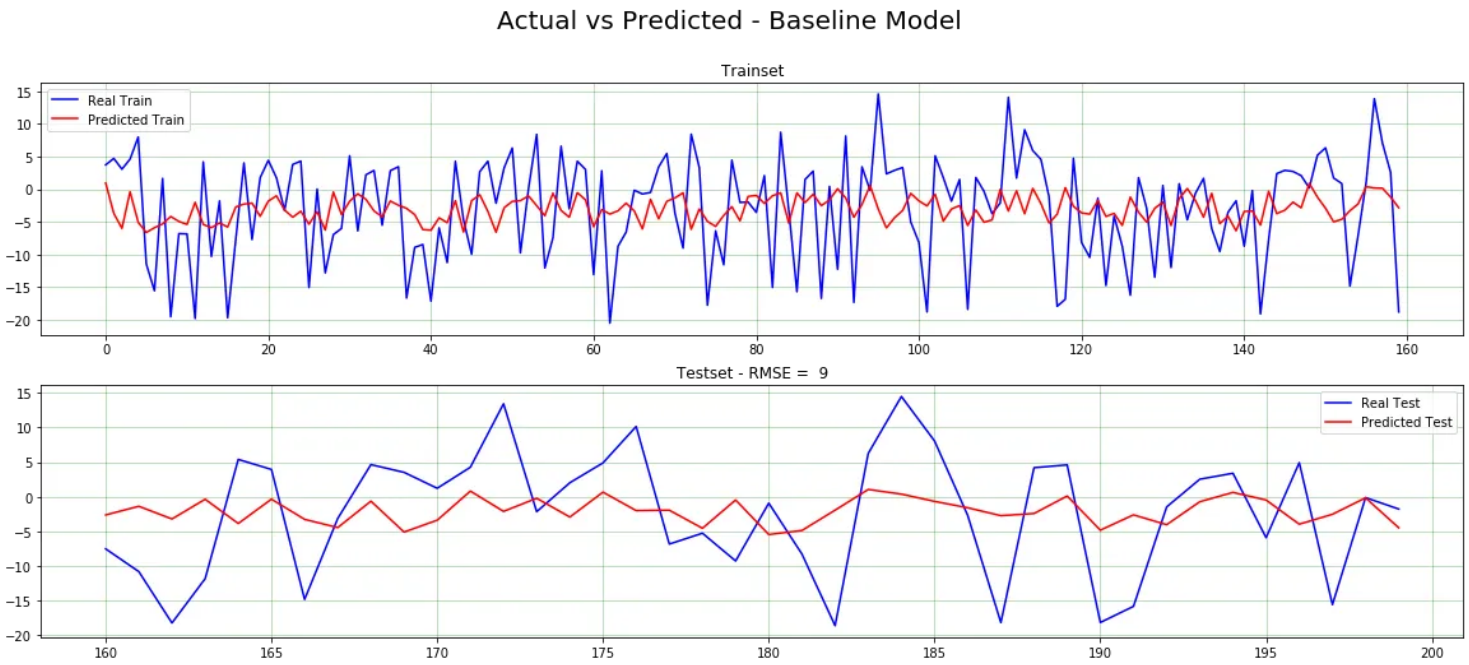
Conclusion We have already a value to compare our coming results with. We have applied the simple rule of given my current value as the prediction. For time series where the value of the response is more stable (a.k.a stationary), this method can sometimes perfoms better than a ML algorithm surprisingly. In this case, the zig-zag of the data is notorious, leading to a poor predicting power.

Multiple Linear Regression

Our next approach will be to build a multiple linear regression model

```
# _____________ MULTIPLE LINEAR REGRESSION _____________ #

from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
lr_model.fit(trainset.iloc[:,:-1], trainset.iloc[:,-1])

t0 = time.time()
lr_y = testset['y'].values
lr_y_fit = lr_model.predict(trainset.iloc[:,:-1])
```

```
lr_y_pred = lr_model.predict(testset.iloc[:,:-1])
tF = time.time()

lr_residuals = lr_y_pred - lr_y
lr_rmse = np.sqrt(np.sum(np.power(lr_residuals,2)) /
len(lr_residuals))
print('RMSE = %.2f' % lr_rmse)
print('Time to train = %.2f seconds' % (tF - t0))

## RMSE = 8.61
## Time to train = 0.00 seconds
```



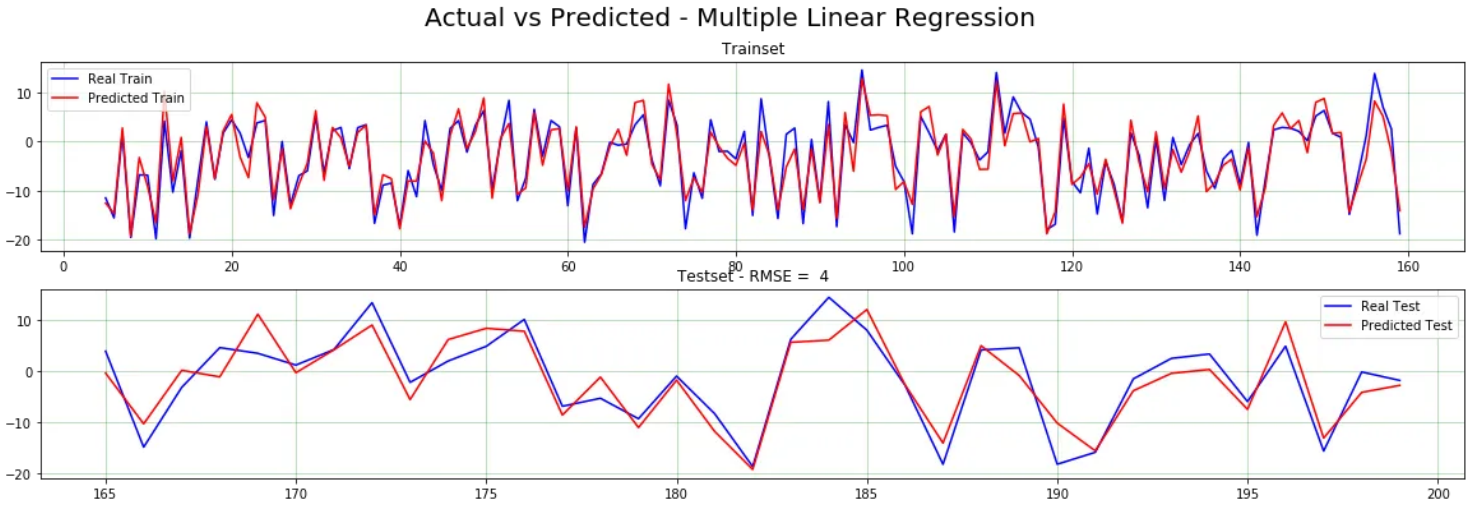
Conclusion

We can see how the multiple linear regression models are not able to capture how the response behaves. This is likely because the non-linearities in the relationship between the response and the independent variables. Also, it is the lags of these variables that affect the response at a given time. Therefore, the values are in different rows for the model that can not find to map this relationship.

I am curious to check now the assumptions we have made when explaining the construction of the windows. We said that we wanted to build a complete information set for every prediction point. Therefore, the prediction power should increase after the construction of the windows… Let's go for it!

MLR with the Windows

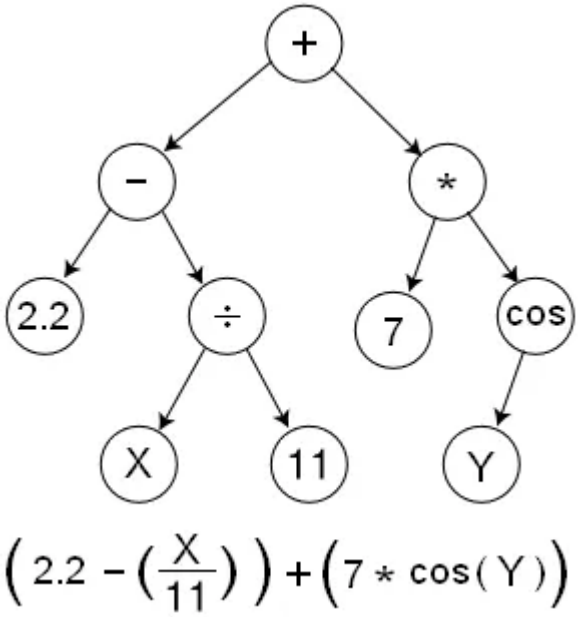
```
# ___________ MULTIPLE LINEAR REGRESSION ON WINDOWS ___________

from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
lr_model.fit(train_windows.iloc[:,:-1], train_windows.iloc[:,-1])

t0 = time.time()
lr_y = test_windows['y'].values
lr_y_fit = lr_model.predict(train_windows.iloc[:,:-1])
lr_y_pred = lr_model.predict(test_windows.iloc[:,:-1])
tF = time.time()

lr_residuals = lr_y_pred - lr_y
lr_rmse = np.sqrt(np.sum(np.power(lr_residuals,2)) /
len(lr_residuals))
print('RMSE = %.2f' % lr_rmse)
print('Time to train = %.2f seconds' % (tF - t0))

## RMSE = 3.84
## Time to train = 0.00 seconds
```



Wow! That was definitely a big improvement.

Now we have a very powerful model to defeat. It seems that with the new windows, the model is able to find the relationship between a whole window information and the response.

2 — Symbolic Regression



Symbolic Regression is a type of regression analysis that searches the **space of mathematical expressions** to find the model that best fits a given dataset.

The base of symbolic regression is **genetic programming** and, therefore, it is an evolutionary algorithm (a.k.a. Genetic Algorithm — GA)

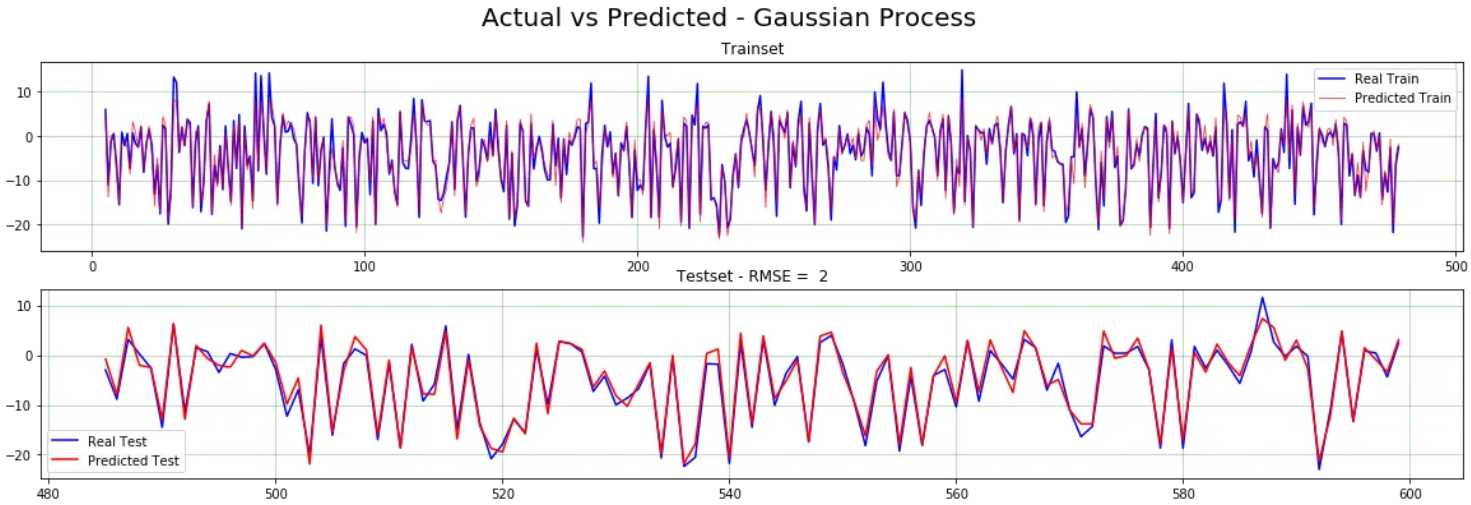
Summarizing in few words how the algorithm works, first we need to understand that a mathematical expression can be represented as a **tree structure**, like the figure above.

This way, the algorithm will start with a big population of trees at the first generation that will be measured according to a **fitness** function, in our case the RMSE. The best individuals of each generation are then **cross between them** and also some **mutations** are applied to include exploration and randomness. This iterative algorithms finish when a stopping criterion is met.

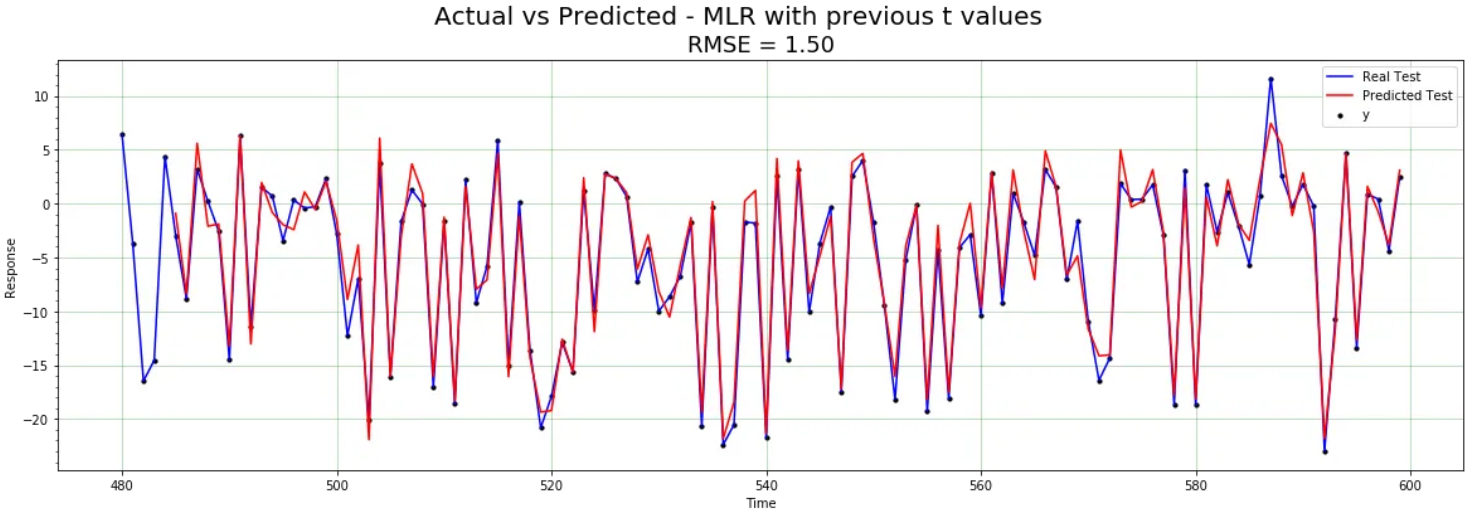
This video is a fantastic explanation of Genetic Programming.

Model

```
     |    Population Average    |             Best Individual              |
---- ------------------------- ---------------------------------------- ----------
 Gen   Length          Fitness   Length          Fitness      OOB Fitness  Time Left
   0    26.79 578184.1495185517        7 6.300696236990088              N/A     17.50s
   1     8.39 32.78869455486322       13 6.0620084131468435              N/A     30.72s
   2     6.64 14.092222825785154        5 4.831344822946351              N/A     34.69s
   3     5.87 43.861161532761685       15 4.328433654089277              N/A     36.06s
   4     6.45 12.174441144713375       15 3.9834799650028305              N/A     36.68s
   5     7.54 14.511781014471689       19 3.3604771904252564              N/A     36.77s
   6     8.37 19.90468507004755       19 3.3604771904252564              N/A     36.14s
   7     9.48 12.322632849663362       11 2.9960801408717885              N/A     35.17s
   8    10.64 101.18075733447115       15 2.441365364320807              N/A     34.00s
   9    11.42 16.058894251794147       15 2.880073537072859              N/A     32.77s
  10    12.43 259.8682786212403       19 2.5676192536236657              N/A     31.26s
  11    13.81 87.50369894635908       19 2.4459142301095995              N/A     29.77s
  12    14.96 28.08493680658899       15 2.441365364320807              N/A     28.16s
  13     16.6 10.924450593964929       15 2.441365364320807              N/A     26.28s
  14    19.21 744.6264199360863       29 2.4357716185359655              N/A     24.41s
  15    22.13 35.31884757869443       33 2.3144262470748544              N/A     22.48s
  16    26.15 31.52191683473308       23 2.049002644898246              N/A     20.39s
  17    29.64 182.71463779935087       23 2.049002644898246              N/A     18.23s
  18    32.72 12.420903946796189       39 2.049002644898246              N/A     16.07s
  19    36.09 9.579264647440674       41 2.049002644898246              N/A     13.78s
  20    40.66 6.947404374762276       17 2.003898474910186              N/A     11.31s
  21    45.46 48.644880476407835       43 1.8602649647615654              N/A      8.72s
  22    49.24 10.68522286423596       47 1.8602649647615654              N/A      5.95s
  23    52.92 11.66636402997101       15 1.7524655242077778              N/A      3.05s
  24    46.74 966.2706311897801       15 1.7524655242077778              N/A      0.00s
RMSE = 1.528348
Time to train 78.00
sub(X17, add(sub(add(mul(X14, X0), div(X14, X0)), X9), mul(X14, X14)))
```



```
    |    Population Average    |             Best Individual              |
---- ------------------------- ---------------------------------------- ----------
 Gen   Length          Fitness   Length          Fitness      OOB Fitness  Time Left
   0    25.99 3271183.2164324736        5 6.475140663977481              N/A     17.70s
   1     8.07 59.44971685522494        5 6.475140663977481              N/A     36.12s
   2     6.78 1636.586267450486        5 5.992479936509899              N/A     45.83s
   3     5.11 61.03154929550919        5 4.660101982428234              N/A     47.48s
   4      5.2 625.100695129756        5 4.660101982428234              N/A     46.50s
   5      5.9 55.27955908754312        7 4.074152620968852              N/A     44.35s
   6     6.41 13.959229475631034        9 2.553316055407164              N/A     41.82s
   7     6.97 45.00061286113768       11 2.9960801408717885              N/A     39.39s
   8     7.85 13.809865042359723       11 2.9960801408717885              N/A     37.22s
   9      8.5 28.522986336638432       19 2.85185627779184              N/A     34.95s
  10     9.09 1334.188547025256       11 2.7522846307634716              N/A     32.81s
  11     9.88 18.435564412866857       13 2.701814049466533              N/A     30.87s
  12    10.46 23.44205920517087       13 2.701814049466533              N/A     28.73s
  13    11.25 17.7941129717295       17 2.441365364320807              N/A     26.70s
  14    11.89 14.755499986297972       17 2.441365364320807              N/A     24.51s
  15     12.5 52369.954852698225       17 2.434215636639227              N/A     22.26s
  16    12.95 67.3822908770424       17 2.4342156366392267              N/A     20.02s
  17    13.64 198.03000767461657       17 2.4342156366392267              N/A     17.67s
  18    13.86 44.28558842472862       17 2.4342156366392267              N/A     15.39s
  19    14.07 11.416910833054617       13 2.0178564600089897              N/A     12.93s
  20    14.45 110.94072364607058       13 2.0178564600089897              N/A     10.45s
  21    14.94 8.054435540531596       51 1.7635980683363808              N/A      7.91s
  22    15.23 351105.04694917967       47 1.7636061447335514              N/A      5.31s
  23     15.5 65.5972986982092       39 1.7636159587769844              N/A      2.68s
  24    16.09 1017.8265569630215       37 1.763597129300764              N/A      0.00s
RMSE = 1.502650
Time to train 67.49
add(mul(sub(X3, X14), X14), sub(sub(sub(add(sub(sub(X9, X14), sub(X0, X14)), sub(X17, X14)), div(add(sub(X9, X14), su
b(X3, X14)), mul(div(X0, X5), add(X3, X22)))), X14), X14))
```



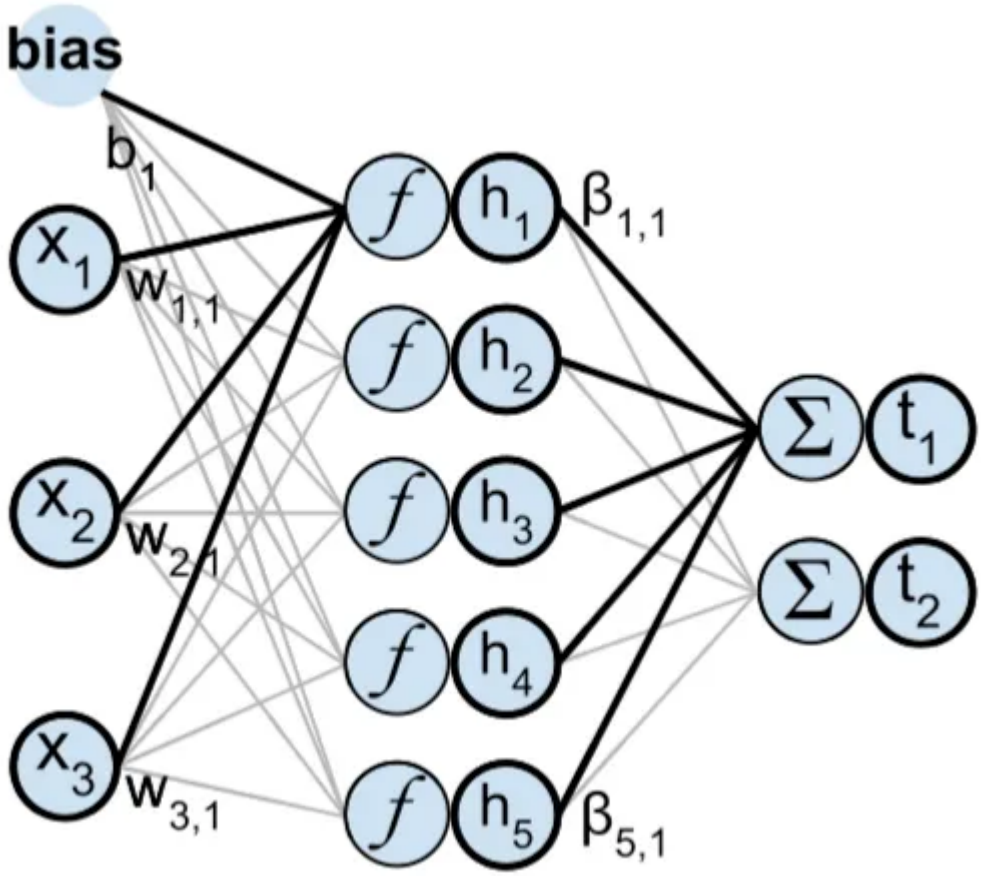
Conclusion

We have seen that Symbolic Regression performs incredible good with almost a perfect fit in the validation data.

Surprisingly, I have achieved the best accuracy by including only the four most simple operators (addition, subtraction, multiplication and division), with the drawback of more training time.

I encourage you to try different parameters of the model and improve that results!

3 — Extreme Learning Machines



Extreme Learning Machines are an important emergent machine learning techniques. The main aspects of these techniques is that they **do not need a learning process** to calculate the parameters of the models.

Essentially, an EML is a Single-Layer Feed-Forward Neural Network (SLFN). ELM theory show that the value of the weight of this hidden layer need not to be tuned, and be therefore independent of the training data.

The **universal approximation property** implies that an EML can solve any regression problem with a desired accuracy, if it has enough hidden neurons and training data to learn parameters for all the hidden neurons.

EMLs also benefit from model structure and regularization, which reduces the negatives effects of random initialization and overfitting.

Given a set of N training samples `(x, t)`. A SLFN with L hidden neuron outputs are:

$$\sum_{j=1}^{L} \boldsymbol{\beta}_j \phi(\mathbf{w}_j \mathbf{x}_i + b_j), \; i \in [\![1, N]\!], \qquad (1)$$

The relation between the target and the inputs and outputs of the network are:

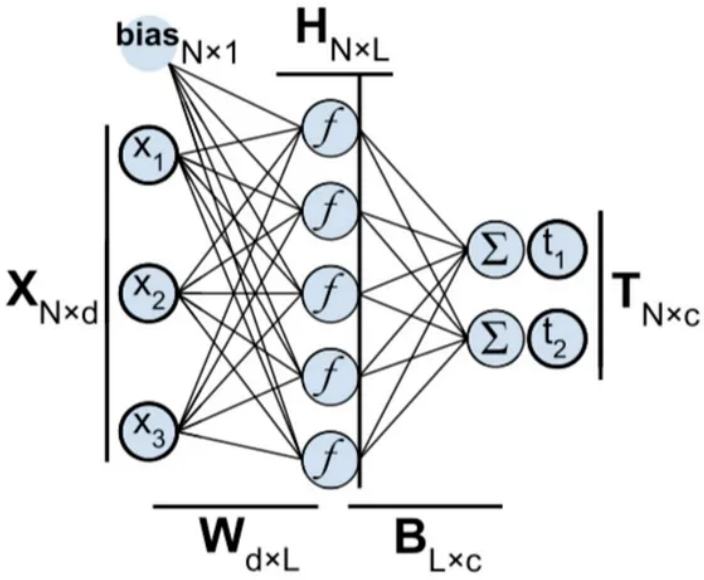
$$\mathbf{y}_i = \sum_{j=1}^{L} \boldsymbol{\beta}_j \phi(\mathbf{w}_j \mathbf{x}_i + b_j) = \mathbf{t}_i + \epsilon_i, \; i \in [\![1, N]\!], \qquad (2)$$

The hidden neurons are transforming the input data into a different representation in two steps. First, the data is projected into the hidden layer through the weights and biases of the input layer, and then applying to the result of that a non-linear activation function.

Practically, ELMs are solved as common Neural Networks in their matrix form. The matrix form is represented here:

$$\mathbf{H} = \begin{bmatrix} \phi(\mathbf{w}_1\mathbf{x}_1 + b_1) & \cdots & \phi(\mathbf{w}_L\mathbf{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{w}_1\mathbf{x}_N + b_1) & \cdots & \phi(\mathbf{w}_L\mathbf{x}_N + b_L) \end{bmatrix}, \quad (3)$$

$$\boldsymbol{\beta} = \left(\boldsymbol{\beta}_1^T \cdots \boldsymbol{\beta}_L^T\right)^T, \ \mathbf{T} = \left(\mathbf{y}_1^T \cdots \mathbf{y}_N^T\right)^T. \quad (4)$$



And here is where the important part of this method comes. Given that T is the target we want to reach, a unique solution a the system with least squared error cam be found using **Moore-Penrose generalized inverse.** Therefore, we can compute in one single operation the values of the weights of the hidden layer that will result in the solution

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T} \quad (8)$$

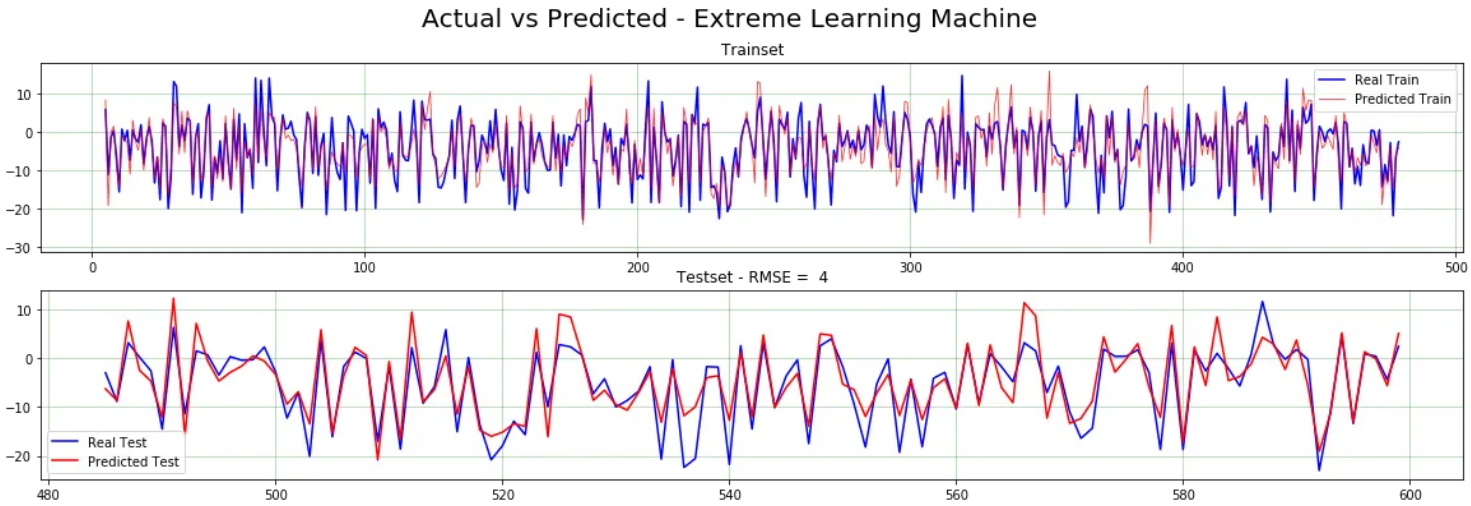
This pseudoinverse in calculated using the Singular Value Decomposition

In this article there is a well documented description in detail about how EML works, and a package for High-Performance Toolbox for EML and implementation in MATLAB and Python.

Model

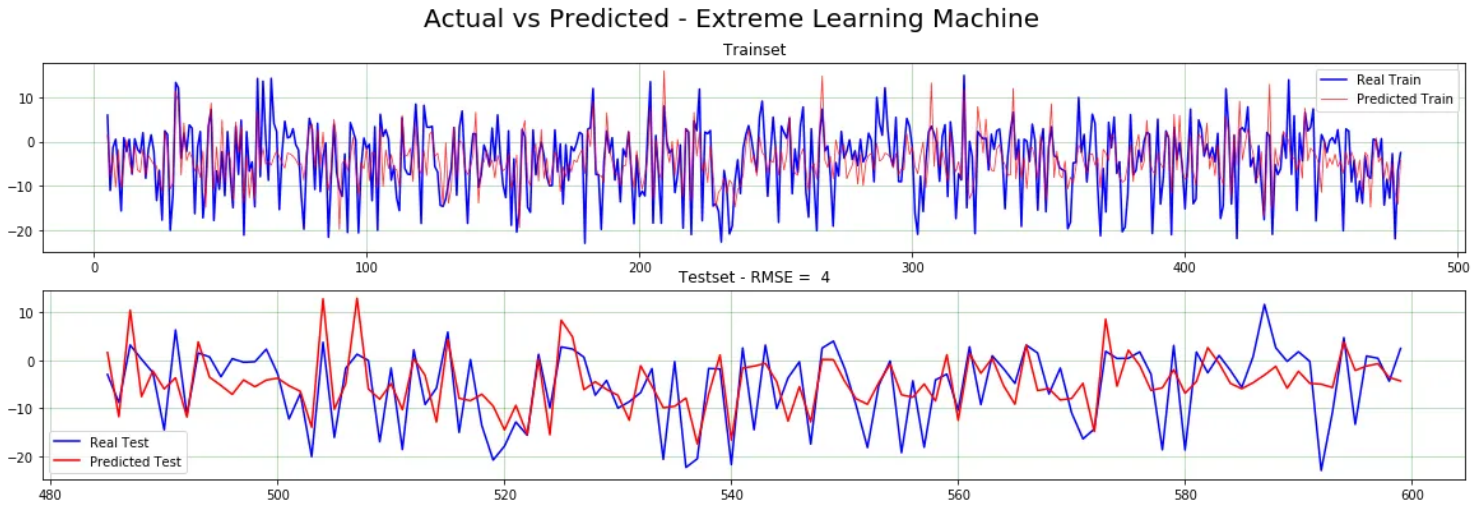
Without taking into account previous values of y as features

```
RMSE = 3.77
Time to train 0.12
```



Taking into account previous values of y as features

```
RMSE = 6.37
Time to train 0.00
```



Conclusion

We can see how EMLs have a great predicting power over our data. Also, it got way worse results where we included previous values of the response as predictors.

Definitely EMLs are models to keep on exploring, this was a quick implementation that already shows their great power, they were able to compute that accuracy with simple matrix inversion and few more operations.

Online Learning

The absolutely greatest advantage of EMLs is that they are very *cheap* computationally for implementing online models. In this article there is more information about the update and downdate operations.

In few lines, we could say that the model becomes adaptative, and if the prediction error goes beyond a stablished threshold, this particular data point is incorporated in the SVD so the model does not required an expensive complete retraining. This way the model can adapts and learn from changes that could happen to the process.

4 — Gaussian Processes

This post is part of a series of posts. The starting post van found here

A Gaussian Processes is a collection of random variables such that every finite collection of those random variables has a multivariate normal distribution, which means that every possible linear combination of them is normally distributed. (Gaussian processes can be seen as an infinite-dimensional generalization of multivariate normal distributions).

The distribution of the GP is the **joint distribution** of all those random variables. In few words, GPs use the **kernel function** that determines the similarity between point to predict the value for an unseen point.

This video is a fantastic quick intro to Gaussian Process to predict CO2 levels.
This book is the main guide to Gaussian Processes.

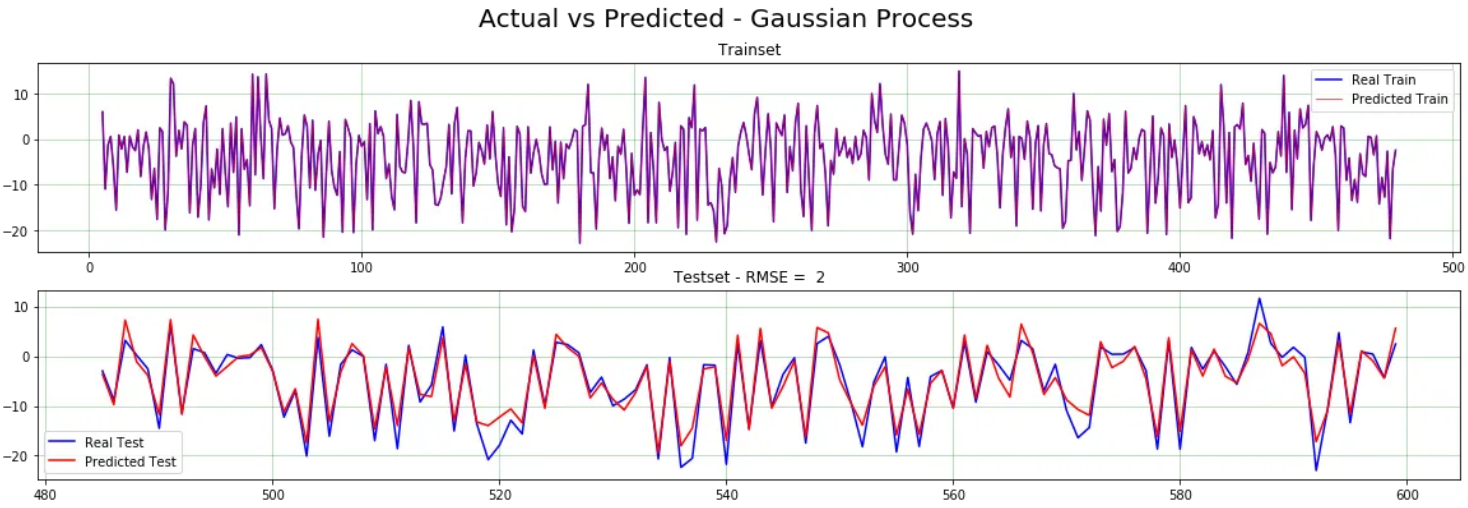
One clear **advantage** of GP is the fact that we obtain a **standard deviation** at every prediction, that can be easily used to perform a confidence interval around the predictions.

Model

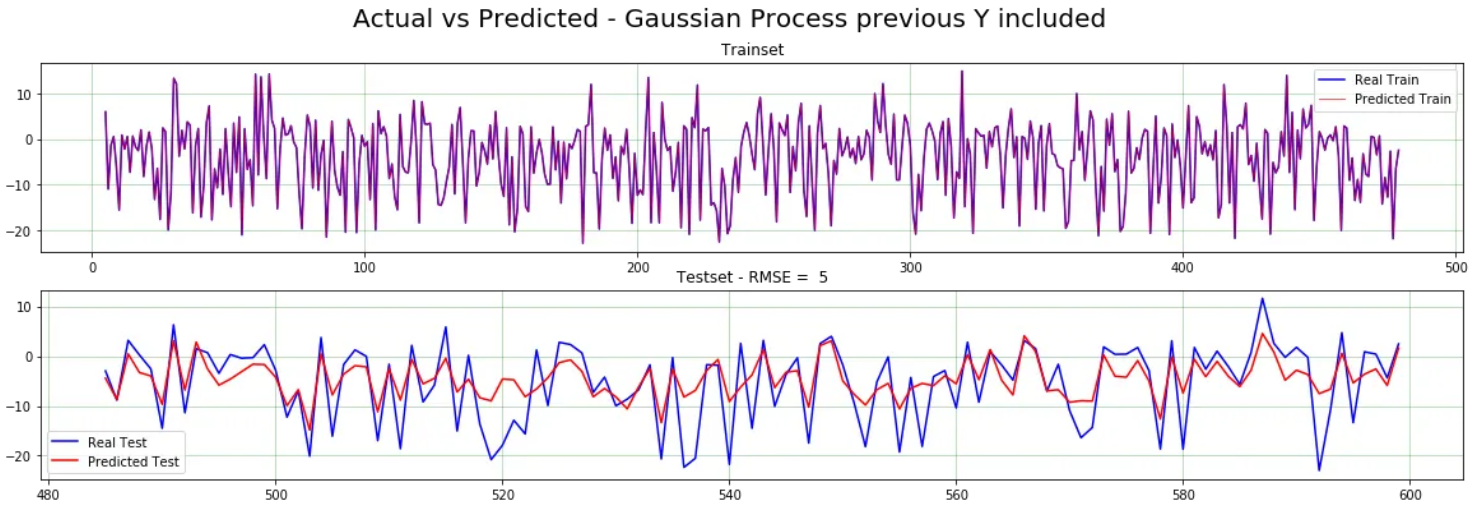
A very simple CNN to play around:

Without taking into account previous values of y as features

```
RMSE = 2.320005
Time to train 4.17
```



Taking into account previous values of y as features



The performance in the validation is far worse that if we don't show the model previous values of the response.

Conclusion

We have seen how Gaussian Processes are another wonderful approach with a high predicting power. This model also get worse results where introducing previous values of the response as predictors.

The main point, is that we can play with an almost **infinite combination of tuned kernels** to look for the combination of random variables which their joint distribution better fits our model. I encourage you to try your own kernel and improve those results!

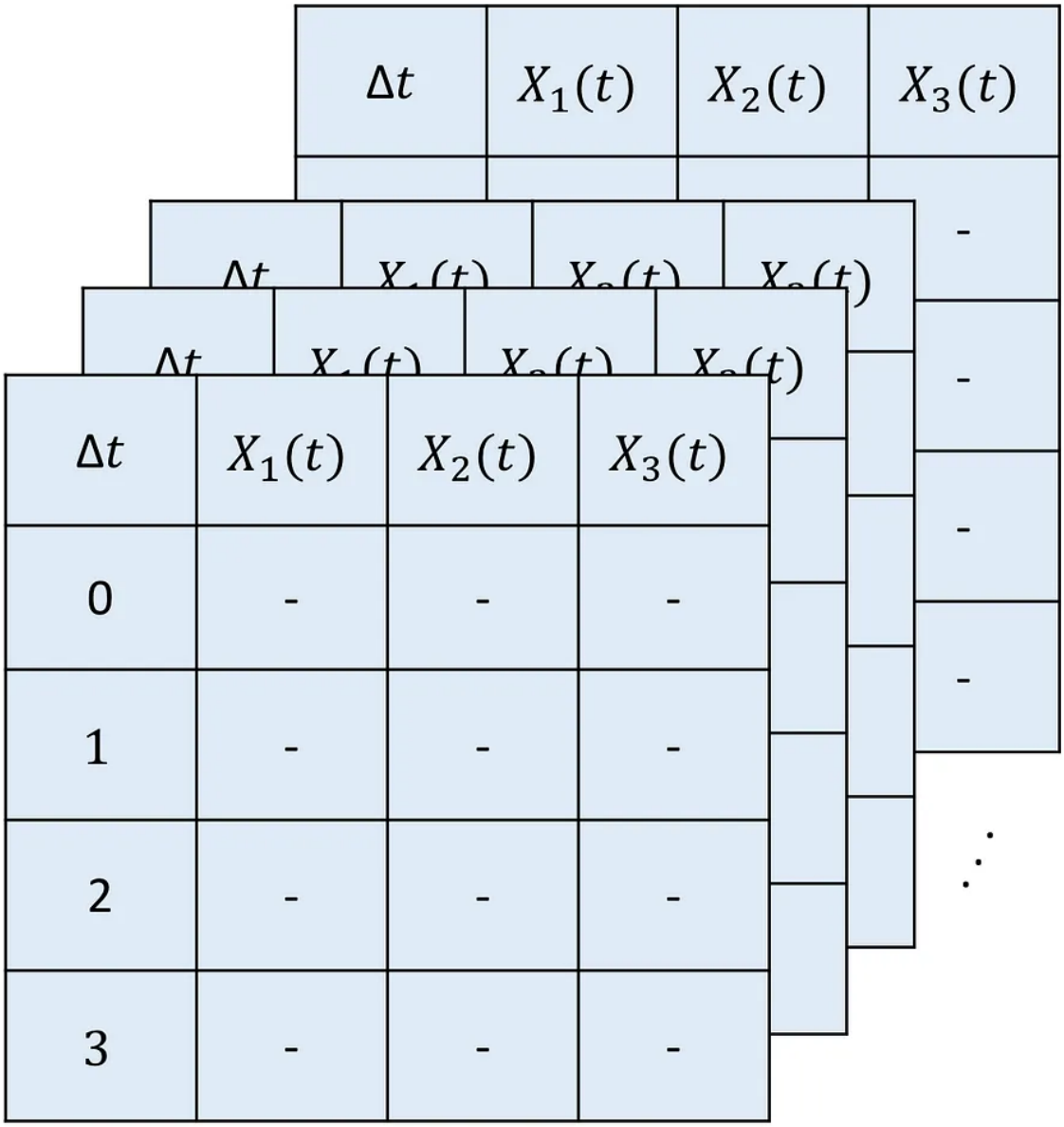
5 — Convolutional NN

The idea is that the window of previous values defines as a ***picture*** the state of the process at a given time.

Therefore, we use a parallelism to *image recognition*, as we want to find patterns that map "pictures" to the response value. We include in our `timeseries.py` a new function, `WindowsToPictures()`.

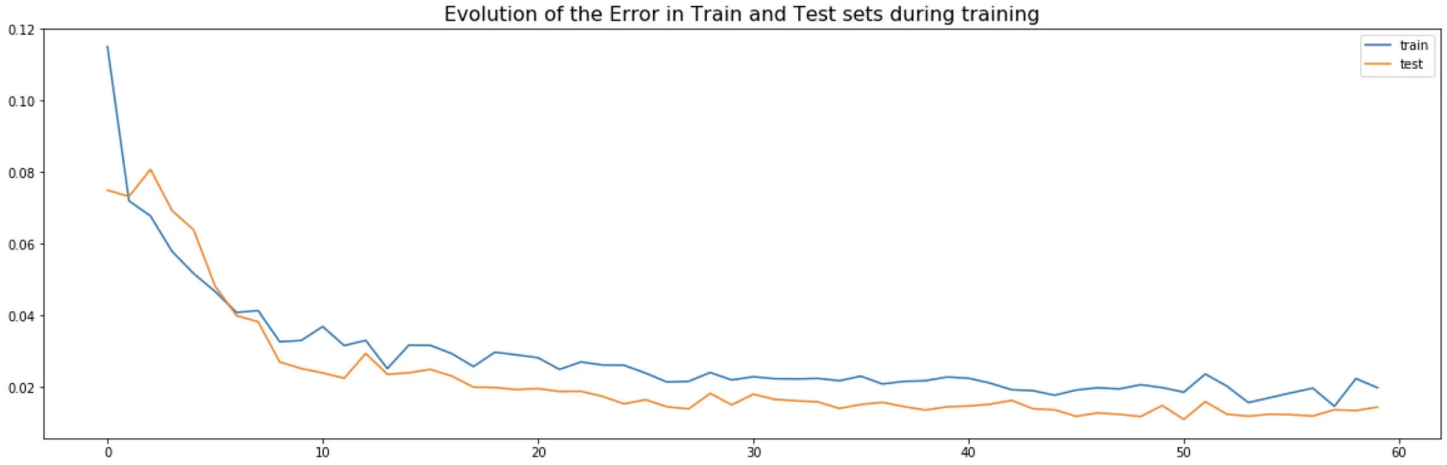
This functions takes the windows that we have been using as inputs, and creates a picture with all previous values of a *window length* of all the columns, for each value in the response.

If we remember the reshaping to windows in the chapter 1, this time, the windows won't be flatten. Instead, they will be stacked in the third dimension of a 3D tensor, where each *slice* will be a picture to map with a unique response value. Here is an illustration:

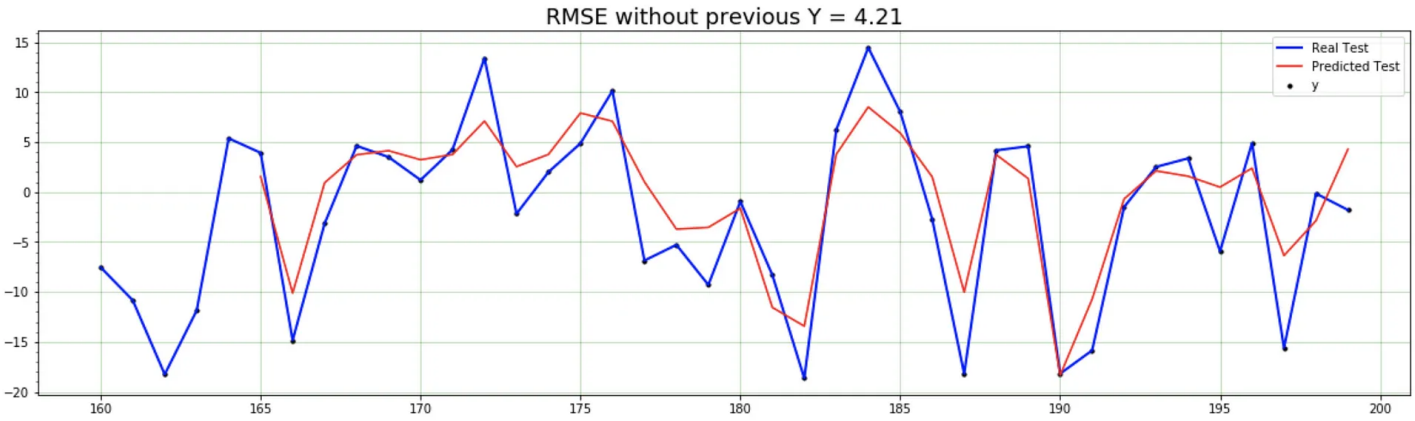


Model

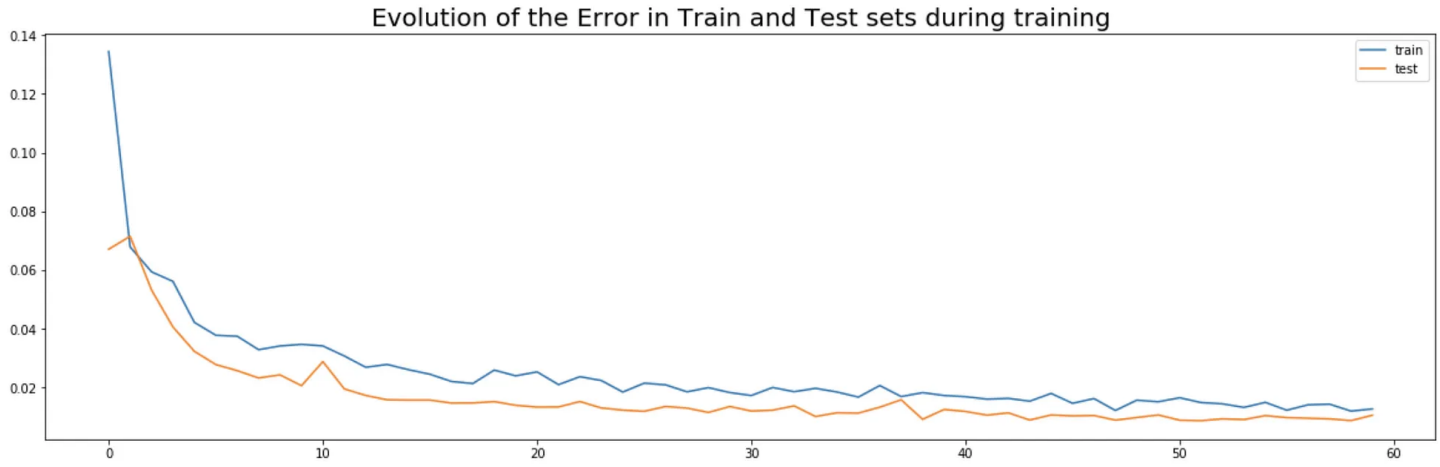
Without taking into account previous values of y as features



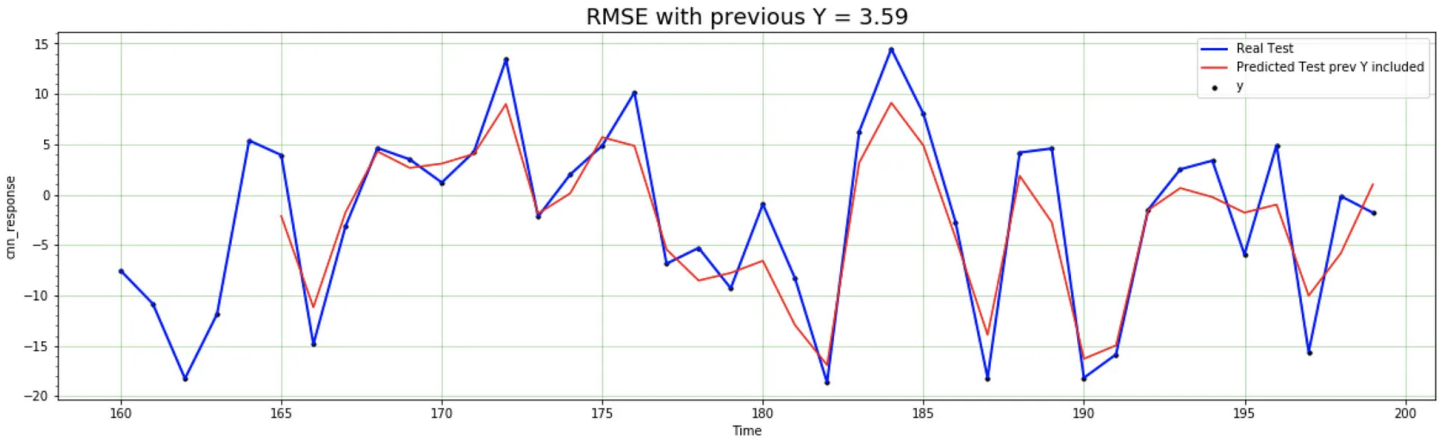

```
RMSE = 4.21
```



Taking into account previous values of y as features




```
RMSE = 3.59
```



Conclusion

Taking the previous state of a process as a picture of the process for every time step seems like a reasonable approach for multivariate time-series forecasting.

This approach allows to frame the problem to whatever king of problem, such as financial time-series forecasting, temperature/weather prediction, process variables monitoring…

I still would like to think about new ways of creating the windows and the pictures to improve the results, but for me it looks like a robust module agains overfitting as we can see in the peaks, it never tends to go beyond the real values.

I would love to know what you come up with to improve it!

Machine Learning | Time Series Analysis | Time Series Forecasting | Ml Time Series

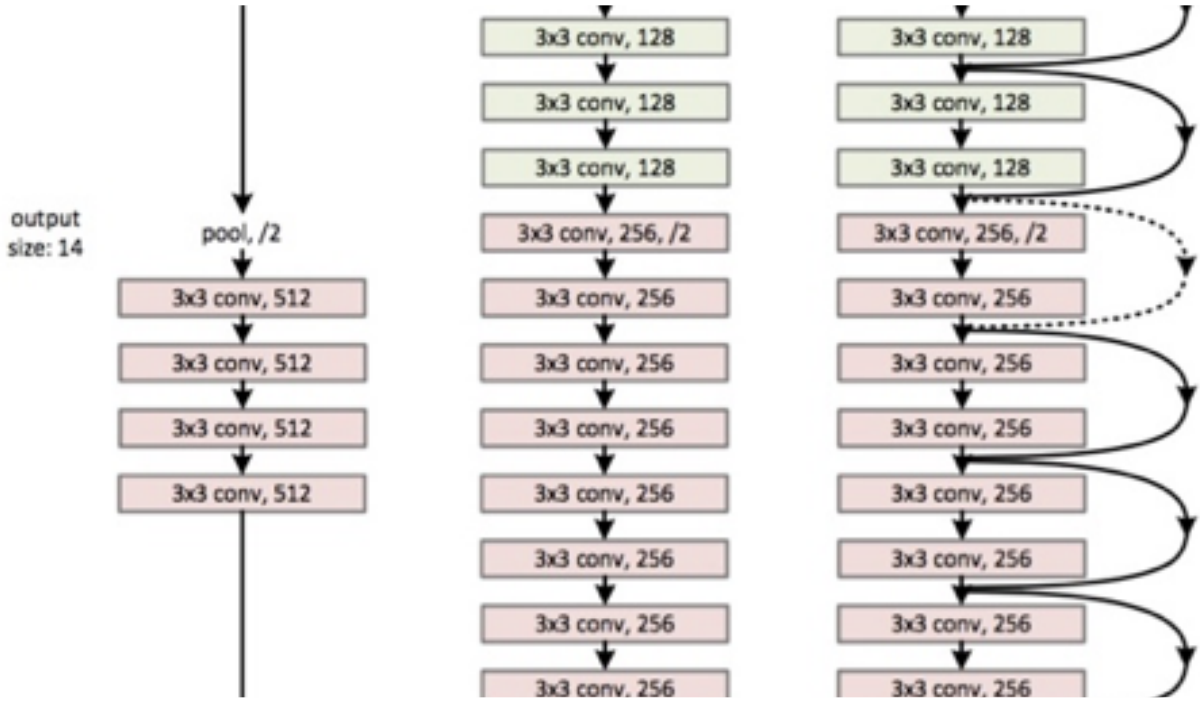
Follow

Written by Pablo Ruiz

757 Followers · Writer for Towards Data Science

Machine Learning @ Twilio & DL Research Collaborator @ Harvard

More from Pablo Ruiz and Towards Data Science



Pablo Ruiz in Towards Data Science

Understanding and visualizing ResNets

This post be found in PDF here.

9 min read · Oct 8, 2018

1.2K 14

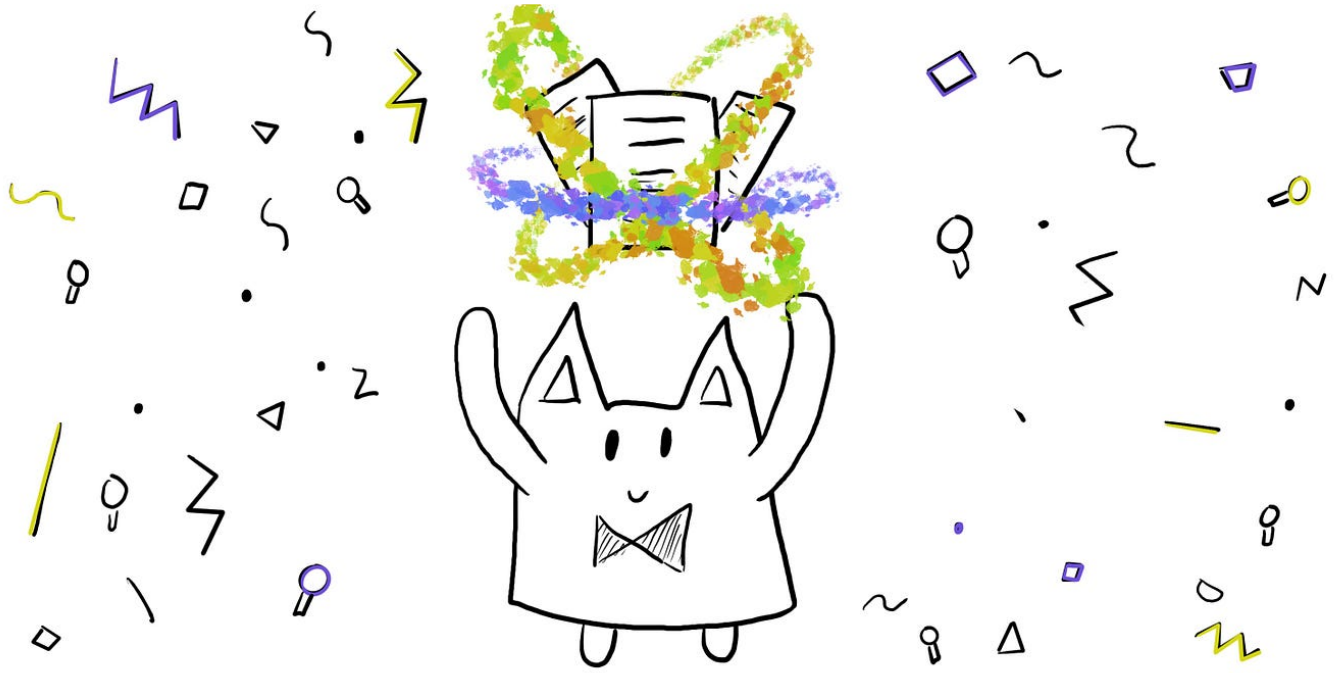
Marco Peixeiro in Towards Data Science

TimeGPT: The First Foundation Model for Time Series Forecasting

Explore the first generative pre-trained forecasting model and apply it in a project with Python

· 12 min read · Oct 24

2.1K 18

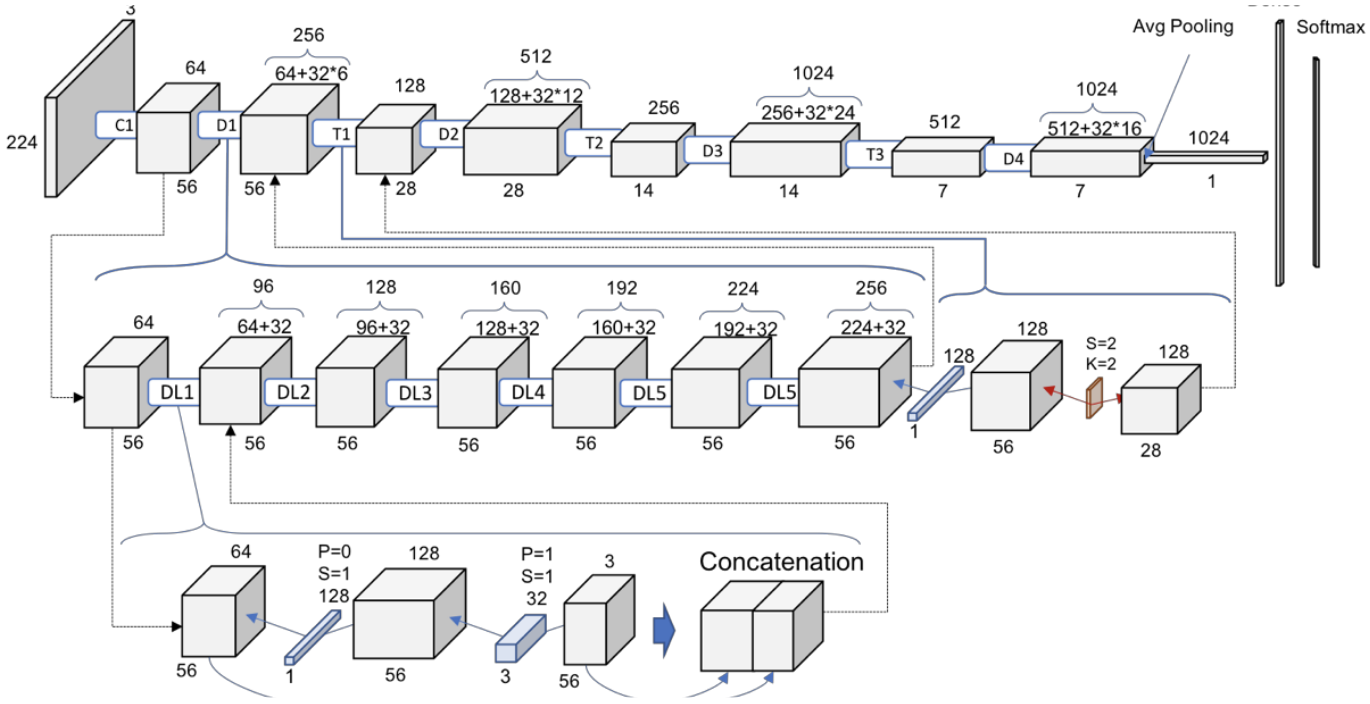


Adrian H. Raudaschl in Towards Data Science

Forget RAG, the Future is RAG-Fusion

The Next Frontier of Search: Retrieval Augmented Generation meets Reciprocal Rank Fusion and Generated Queries

· 10 min read · Oct 5



Pablo Ruiz in Towards Data Science

Understanding and visualizing DenseNets

This post be found in PDF here.

7 min read · Oct 10, 2018

716 9

See all from Pablo Ruiz

See all from Towards Data Science

Recommended from Medium

Cem ÖZÇELİK in MLearning.ai

Multivariate Time Series Forecasting

Part I : Exploratory Data Analysis & Time Series Analysis

27 min read · Jul 2

92

Marco Peixeiro in Towards Data Science

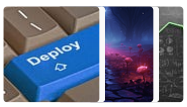
TimeGPT: The First Foundation Model for Time Series Forecasting

Explore the first generative pre-trained forecasting model and apply it in a project with Python

· 12 min read · Oct 24

2.1K 18

Lists



Predictive Modeling w/ Python

20 stories · 590 saves

Practical Guides to Machine Learning

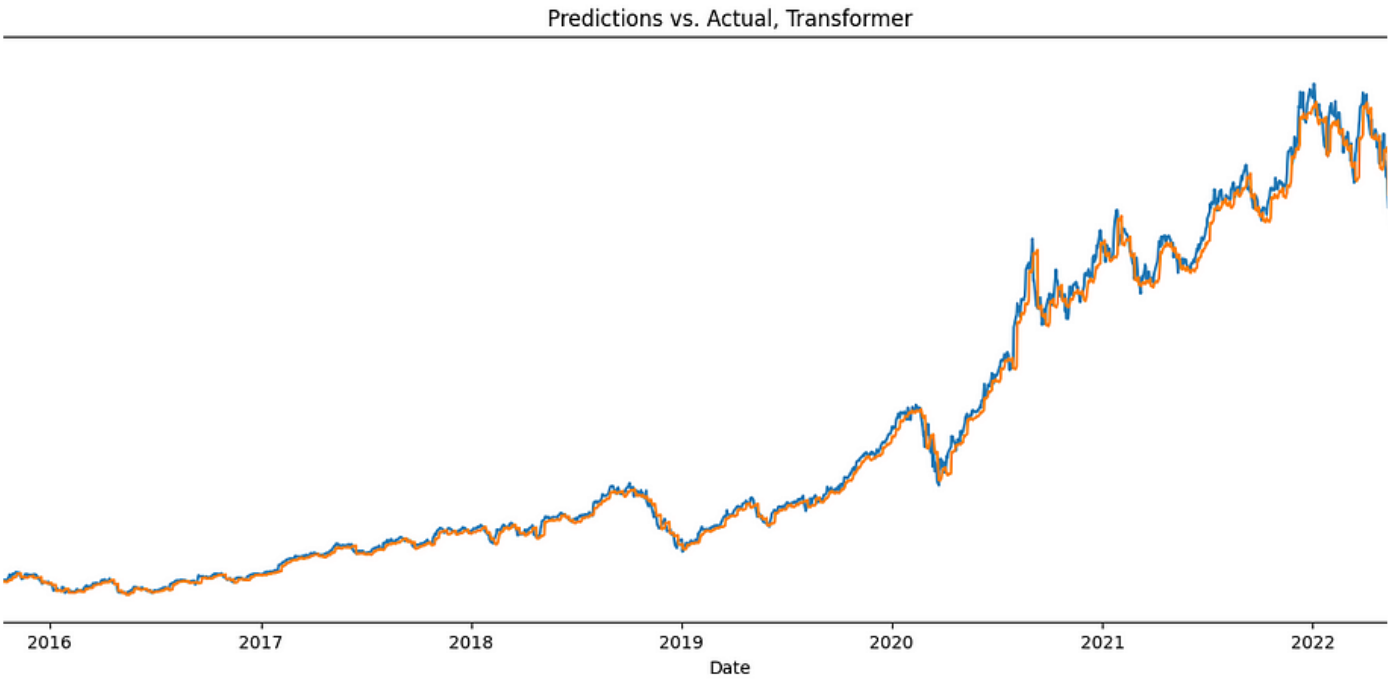
10 stories · 670 saves

Natural Language Processing

842 stories · 392 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 195 saves



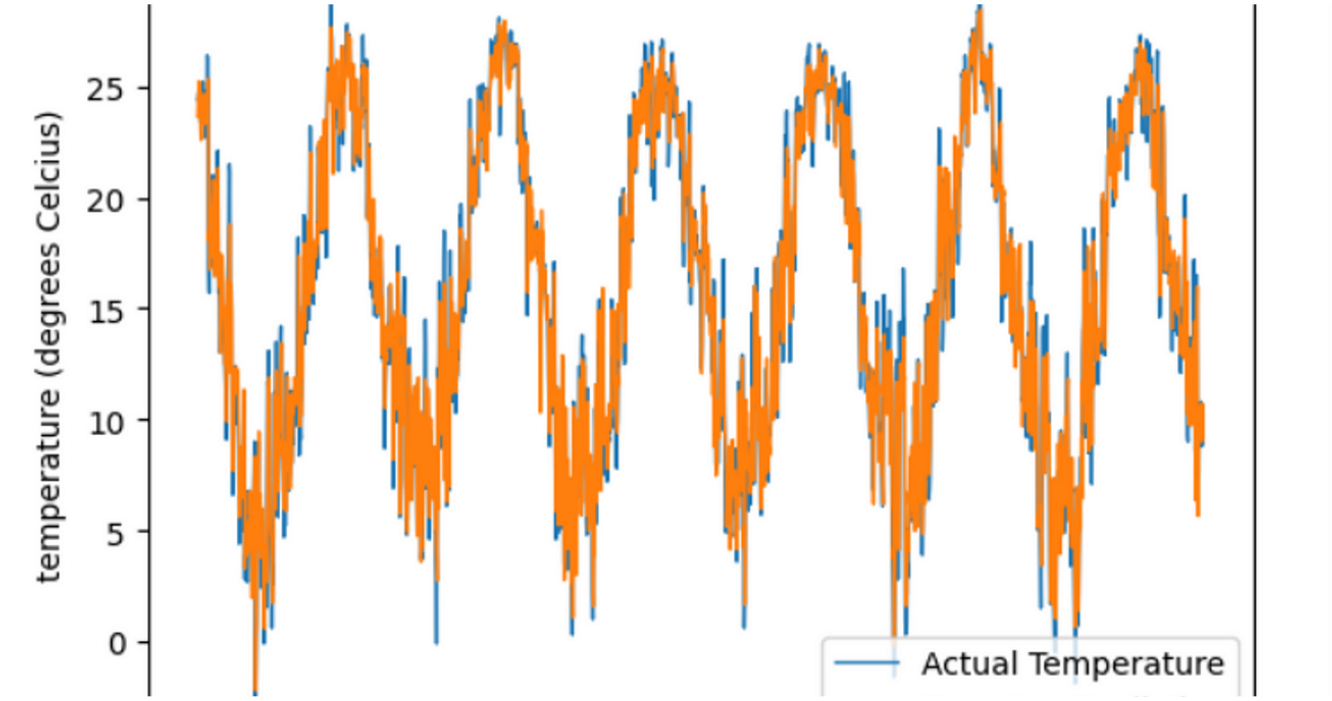
Michael May

Transformers vs. LSTM for Stock Price Time Series Prediction

Can a transformer architecture simplify our model and get better results?

9 min read · Jun 24

90 10




Can Ozdogar

Time-series Forecasting using LSTM (PyTorch implementation)

Exploring implementation of long short-term memory network using PyTorch and weather dataset

8 min read · Sep 9

53 1

Luís Fernando Torres in LatinXinAI

How I Deployed a Machine Learning Model for the First Time

Going beyond Jupyter Notebooks 🚀

13 min read · Jul 18

910 8

Vaibhav Rastogi

Decomposition in time series analysis

Decomposition in time series analysis is a method used to separate a time series into three distinct components: trend, seasonality, and...

2 min read · Aug 13

See more recommendations