



UiT The Arctic University of Norway

## DTE-2501 AI Methods and Applications

*Creating a dataset for Time Series Prediction using Sliding Window*

Andreas Dyrøy Jansson

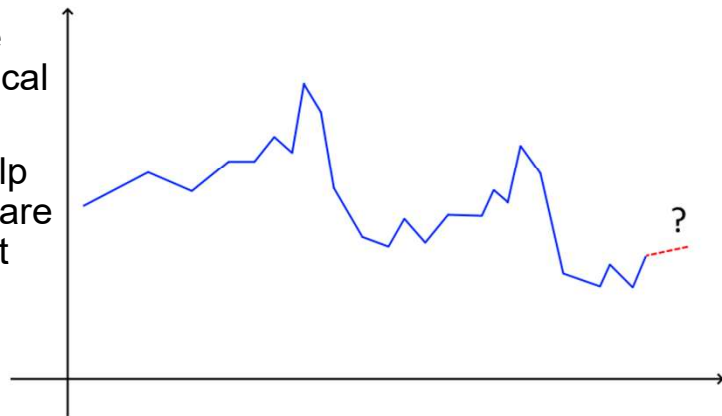
PhD Candidate

C3190

[andreas.d.jansson@uit.no](mailto:andreas.d.jansson@uit.no)

## Time series prediction – the short version

- We want to predict the future based on historical data
- AI (regression) can help us – but we must prepare the data so that we get an “input->output” dataset
  - One approach is the “sliding window”

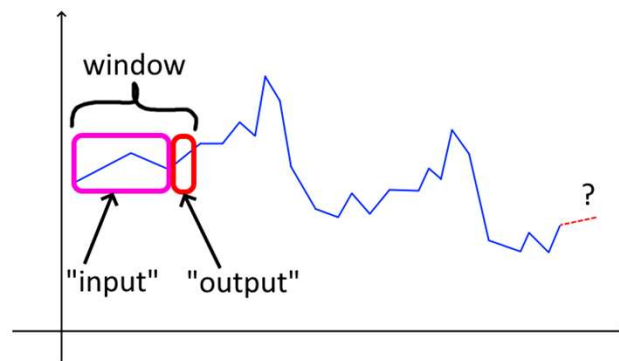


Time series prediction is simply that we want to predict some future state based on historical records. We can do this using regressions, which you may remember from mathematics. You know that we mainly have two types of models in AI and machine learning: classification and regression. For classification, the model is trained on some data, and it predicts a discrete value, corresponding to a class. Regression models are basically the same, but instead they output continuous values. In order to train any machine learning model, we must feed it data on the correct format, for supervised learning models, this means a set of input values  $X$ , and the

expected output  $Y$  associated with each  $X$  value. For time series, which is basically a series of observations, we can use a technique called sliding window, which we will take a closer look at now.

## The sliding window concept

- What the name implies – we “slide” a “window” across our data
- A range of data points becomes the input,  $X$ , and the data point immediately after becomes the expected output,  $Y$
- Our supervised model can then be trained on the  $X$ - $Y$  dataset



The name sliding window comes from the fact that we basically create a window in time, and slide this window across our time series data set. The window has a certain length in time, and this range of data points becomes the input. The trick here is to make the next data point immediately after the points captured by the window, the expected output  $Y$ . We can then train our model on the  $X$ - $Y$  pairs of data.

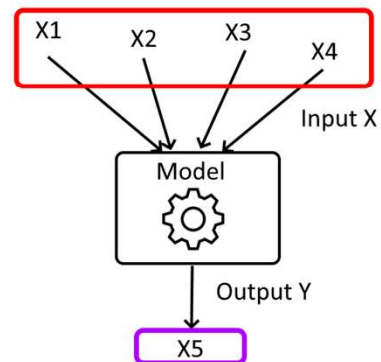
## Sliding window algorithm

*Given a series of observations  $S$  on the form  $[x_1, x_2, x_3, x_4, \dots, x_n]$   
 $X, Y = []$   
window size  $W$  is some small number  $< n$   
FOR  $i < n - W$  DO  
     $X.append(S[i:i + w])$   
     $Y.append(S[i + W])$*

The basic algorithm for a sliding window parser looks like this. Let's say that we have a series of observations like  $x_1, x_2, x_3, x_4$ , all the way to  $x_n$ . We start by initializing two empty lists  $X$  and  $Y$ , which will hold our transformed values. We pick the size of the window  $W$ , which is a small number. Then, for  $i$  less than  $n$  minus  $W$ , we simply append the  $i$  to  $i + w$  datapoints to the list  $X$ , and the  $i + w$ th data point to the list  $Y$ . We keep going until we reach the end, and the very last data point becomes the last expected output.

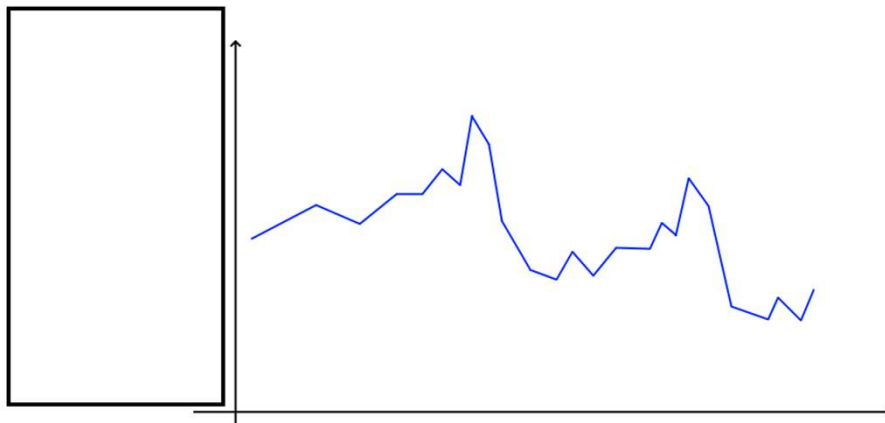
## The basics

- For example, let's choose a window size of 4
- Then, the first 4 data points become the input X, and the 5<sup>th</sup> is the expected output Y
- We move the “window” one step ahead, and repeat
  - X = X2, X3, X4, X5
  - Y = X6
  - Etc.



To show the basic principle in practice, let's say that we choose a window size of 4. Then, the first 4 data points in the time series become the input X, and the 5<sup>th</sup> data point is the expected output, based on the 4 preceding points. We move the window one step ahead, and repeat the procedure, so that the next pair of input and output values becomes X equals x2, x3, x4, x5, and x6 as Y.

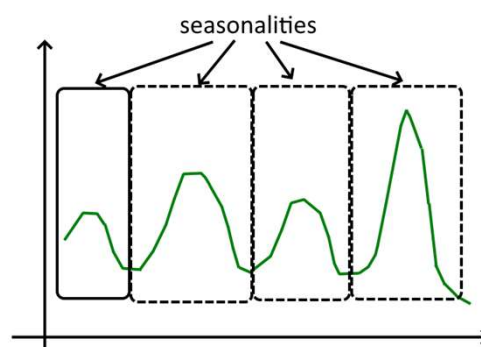
In practice:



This animation shows how the window moves across the time series, and puts each x and y value in our training set.

## How to choose window size?

- Plot the data using your favorite data plotter
- If you notice seasonality of a certain length, this is a good place to start
- A smaller window is better at generalizing, but may miss out on long-term trends
- A larger window can capture trends better, but may cause the model to overfit



Our next question is how do we determine the window size? One simple way is to examine the data visually. Plot the data using your favorite plotter, and look for any repeating patterns or seasonality. If you notice seasonality of a certain length, this is a good initial value for  $W$  to start with. Fine tuning can be done by testing the model on different window lengths until you achieve satisfactory performance. In general, we can say that a smaller window is better at generalizing, but may fail to capture longer trends in the data. Likewise, a larger window is more likely to capture long-term trends, but is also be more likely to overfit.



- Using our data set, we can train a simple learner
  - For example, a regression tree:



```
from sklearn.tree import DecisionTreeRegressor # pip install scikit-learn

# window size is a small number, depending on the length
# of our time series, and if there are any seasonalities
dataset = [2.1, 1.3, 1.2, 1.7, 0.9, 0.8, 1.8]

def make_dataset(window_size = 4):
    x, y = [], []
    # read and format the data using the sliding window algorithm...
    # the result should be:
    # x = [[2.1, 1.3, 1.2, 1.7], [1.3, 1.2, 1.7, 0.9], [1.2, 1.7, 0.9, 0.8]]
    # y = [0.9, 0.8, 1.8]
    return x, y

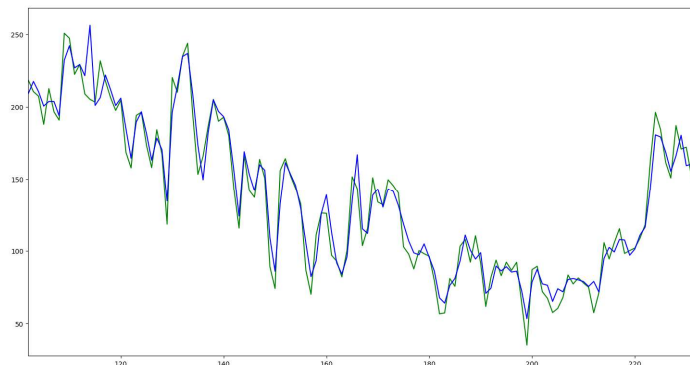
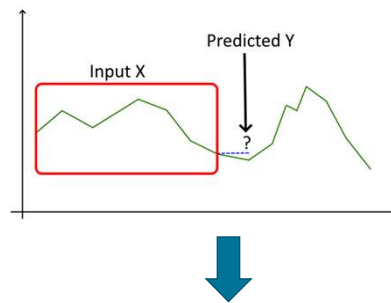
X, Y = make_dataset() # make the dataset using sliding window
model = DecisionTreeRegressor(random_state = 0) # create the model
model.fit(X, Y) # train the model on the input X and the expected output Y
prediction = model.predict([[1.7, 0.9, 0.8, 1.8]]) # predict the next value based on a given input
```

When we have parsed the data file and created a test set, we can train the model. For example, we can train a simple regression tree, like so. Let's say that we have the following data: 2.1, 1.3, 1.2, 1.7, 0.9, 0.8, and 1.8. We see from the plot that there may be seasonality of length 4, so we pick 4 as the window size. We then slide the window across the data, and end up with these pairs of X and Y data. We use the regression tree from scikit, and we set the random state to 0. This is just a seed for the internal random generator, which means that we should get the same result every time we train the model.

We train the model using the `fit()` function, and pass in the data we just formatted. Then, we can pass in the last four data points, and predict the future value.

## Plot the data

- To test our model, we can plot its predictions alongside the original data
- In this example, the original data is in green, and the prediction is blue



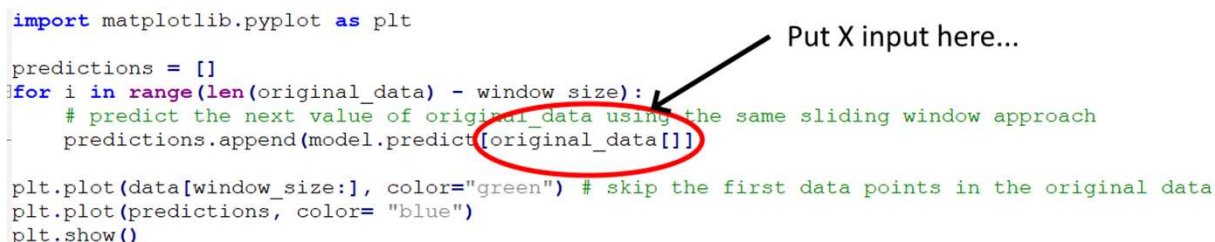
To show the accuracy of our model visually, we can plot the predicted values alongside the original data. The first data points are the input, and the model tries to predict the next value based on these points. Here we have a much longer time series, and the original data is shown in green. The blue line shows each predicted output of a regression model, based on the previous data points, repeated over the whole original data set.

## Plotting the predictions...

```
import matplotlib.pyplot as plt

predictions = []
for i in range(len(original_data) - window_size):
    # predict the next value of original_data using the same sliding window approach
    predictions.append(model.predict(original_data[i]))

plt.plot(data[window_size:], color="green") # skip the first data points in the original data
plt.plot(predictions, color="blue")
plt.show()
```



- Note: When we plot the predictions alongside the original data, we must offset the original data by “window\_size”!
  - Remember, we need at least “window\_size” number of data points in order to predict the next value

To plot the predictions in Python, we can use matplotlib. Here is an example of how this can be done. Here we use the same sliding window approach on our test set, and make a prediction. We save each prediction in a list, and plot it alongside the original data.

Remember that in order for both graphs to line up, we must offset the original data by the same length as the window size. This is because we needed window size number of data points to predict the next value, so the first value in our predicted time series may be the 4<sup>th</sup> or 7<sup>th</sup> element in the original data set.