Deep Learning Project

1. Introduction

Healthware Group is a global health innovation and technology leader providing transformational advisory and technology services for commercial, medical, and R&D operations of life-sciences and digital health companies, combined with design and development of digital medicines and digital therapeutics products.

In the company's vision digital technology & innovation are the driving forces behind the transformation in healthcare, leading to a world of increasingly relevant, human-sized, solutions to health challenges.

a) Subject of the project

The main subject of the project is Parkinson disease. Parkinson disease, the second most common neurological disorder that causes significant disability, reduces the quality of life and has no cure.

b) Information about the data

The dataset contains information about 40 examined patients. They can be grouped as follows:

- 24 with Parkinson's Disease
- 3 with advanced Parkinson's Disease
- 13 without Parkinson's Disease

Obtained data comes from wearable activity trackers. It provides information such as:

- accelerometer readings In the three axes (x, y, z)
- identification of a patient
- heart rate
- date and timestamp
-

The data is fully in a time series format. It contains information about the time of the measurements. Its characteristic is an uniform granularity - aggregation by mean at 1 second and resampling by mean at 10 seconds.
The data is also noisy and it contains missing values in a significant amount of records.

c) Goal of the project

The main goal of the project is to verify if provided data can be treated with deep learning approaches for time series.
The company would like to use it in the future in order to identify OFF symptoms perceived by the study subject with reasonable accuracy from real-world data collected. OFF periods are times when Parkinson's disease (PD) medication, namely levodopa, is not working optimally. As a result, symptoms return.

2. Project tasks

a) Task 1 - Next Value Prediction

The first task is focused on prediction of the next value for each of the three time series (X, Y, Z). Every time series is recorded each 10 seconds.
Provided data consists of two .csv files, the first one contains training data and the second one contains data that will be used for testing of the created deep learning model. Both files contain information about accelerometer readings In the three axes (x, y, z).

The goal is to predict the next value of the sequence for each time series.
The task is an example of supervised machine learning according to the need of data conversion that involves rolling window technique.
Additionally, the task is also about checking how the rolling window parameters affect the result based on one of the provided time series (X, Y, Z).

b) Task 2 - Anomaly Detection Task

The data for the task is also given from wearable devices. It contains mentioned earlier features listed below:

- accelerometer readings In the three axes (x, y, z)
- identification of a patient
- heart rate
- date and timestamp

Data are collected by patients with and without Parkinson' Disease. The goal is to identify anomalous events, i.e.tremors, in a set of observations.

The training set is composed of control patients, i.e. volunteers without Parkinson's Disease. Granularity of the data is defined by aggregation by mean at 1 second (each 1 second there is a record). Missing values from the heart rate attribute are labeled with -1.

The test set is composed of patients with Parkinson's Disease. Granularity of the data is defined by aggregation by mean at 10 second (each 10 second there is a record).

3. Implementation of the Next Value Prediction task (task 1)

a) Data preprocessing and formatting

At first, the most basic check was done - if there are missing values in the dataset. It was very important from the prediction point of view because unidentified missing data among the dataset can cause corruption in the predictive process and decrease prediction accuracy.
As it turned out, there was no missing data among the given data.

```
1 # checking empty values
2 print(train_data.isnull().sum())
3 print(test_data.isnull().sum())
```

```
x    0
y    0
z    0
dtype: int64
x    0
y    0
z    0
dtype: int64
```

Figure 1. Checking missing values in the dataset

After a brief examination of the .csv files content, we discovered that data values are both positive and negative and their range is wide (X - from -1125 to 1039, Y - from -1019 to 1046, Z - from -1001to 1032).
This leads to the next step of data preprocessing - normalization process.

Normalization is a rescaling of the data from the original range so that all values are within the range of 0 and 1. In the project it was done by using the *scikit-learn* object *MinMaxScaler*.
The scaler requires data to be provided as a matrix of rows and columns. Our data is loaded as a Pandas Series. It must then be reshaped into a matrix of one column with many rows and then the normalization process is applied for each of the timeseries X, Y, Z (as in the function below).

```
# normalization function
def normalize(data):
  result = data
  for i in ['x','y','z']:
    values = data[i].values
    values = values.reshape((len(values), 1))
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaler = scaler.fit(values)
    normalized = scaler.transform(values)
    result[i] = normalized
  return result
```

Figure 2. Normalization function

The next step of formatting the data was about creating an additional column with time data to reflect the measurement frequency of recorded data (every time series is recorded each 10 seconds). As there was no such column previously it had to be created manually as for time series data, it's conventional to represent the time component in the index of a Series or DataFrame so manipulations can be performed with respect to the time element.

```
# creating dataindex column
date_today = datetime.now()
data_index_train = pd.DataFrame({'timestamp':pd.date_range(date_today, periods=len(norm_train_data), freq='10S')})
data_index_test = pd.DataFrame({'timestamp':pd.date_range(date_today, periods=len(norm_test_data), freq='10S')})
```

Figure 3. Creation of dataindex column

Then, re-structuring the dataset with a set of features/input variables (x) and the output variable (y) occurred. In this step the time series data is phrased as supervised learning. It was done by using the value at the previous time step to predict the value at the next time-step. Representation of a rolling window was used, as the window of inputs and expected outputs is shifted forward through time to create new "samples" for a supervised learning model. The *shift() function* and *rolling() function* in Pandas were used to automatically create new framings of time series problems given the desired shift length and window size.
Lag Periods (summary of values over a fixed window of prior time steps) by Window Moving Average were used in order to smooth out the values and impacts of outliers and fluctuations. Default window_size = 2 and default shift = 1.

```
def lag_features(data, col, shift=1, window_size=2):
  result = data
  result['%s_lag1' % col] = result[col].shift(shift)
  window = result['%s_lag1' % col].rolling(window=window_size)
  means = window.mean()
  result['%s_mean' % col] = means
  return result, result
```

Figure 4. Function for conversion time series data into supervised learning

b) Data identification

There are three important concepts about time series data - **trend** (shows the general tendency of the data to increase or decrease during a long period of time), **seasonality** (characteristic of a time series in which the data experiences regular and predictable changes) and **stationarity** (stationary if its statistical properties such as mean, variance remain constant over time).
In order to get information about the above concepts we used the *seasonal_decompose function.* There are two options for Seasonal Decompose:

- Additive (when trend is more linear, seasonality and trend seem to be constant)
- Multiplicative (trend is more non-linear)

Our data is more like a multiplicative option (as the trend is non-linear), so the seasonal decompose results revealed horizontal trends and lack of certain seasonality in all time series.

The next step was a check for stationarity by using Dickey-Fuller Function. It returns a p-value, if p<0.5: data is stationary , p>0.5: data is not stationary. It is a very important part because if the data is non-stationary then we cannot build a model on it so we have to make the time series stationary. Using non-stationary time series data in models produces unreliable and spurious results and leads to poor understanding and forecasting.
After performing the test it turned out that all data is stationary, so we can move on to building a model.

c) Building a model

Before building a model, the original data was divided into three smaller data frames for each of the time series X, Y, Z.
First step was to separate the data to x_train, y_train, x_test and y_test - Input(x) and Output(y).

```
sx_train_X, sy_train_X = x_train.drop(['x'], axis=1), x_train['x']
sx_test_X, sy_test_X = x_test.drop(['x'], axis=1), x_test['x']
```

Figure 5. Separating the data to Input(x) and Output (y)

As a result of this operation for time series X, divided data looks as below.

timestamp	x_lag1	x_mean
2022-01-21 12:38:56.392034	NaN	NaN
2022-01-21 12:39:06.392034	0.813943	NaN
2022-01-21 12:39:16.392034	0.814414	0.814178
2022-01-21 12:39:26.392034	0.803109	0.808761
2022-01-21 12:39:36.392034	0.757890	0.780499
...	...	...
2022-02-07 16:46:36.392034	0.997174	0.997174
2022-02-07 16:46:46.392034	0.996703	0.996938
2022-02-07 16:46:56.392034	0.996703	0.996703
2022-02-07 16:47:06.392034	0.996703	0.996703
2022-02-07 16:47:16.392034	0.996703	0.996703

148371 rows × 2 columns

Figure 6. Data for time series X after separating the data to Input(x) and Output (y)

After separating the data, we used the XGBoost Model library and fit the model with the training set. XGBoost is an efficient implementation of gradient boosting for classification and regression problems. XGBoost can also be used for time series forecasting, although it requires that the time series dataset be transformed into a supervised learning problem first.
The number of trees (or rounds) in an XGBoost model is specified to the XGBRegressor class and it was set to 1000 (after experimenting with different values, this one has the best results). The evaluation metric for data validation is set to Mean Absolute Error. The *early_stopping_rounds*=50 in order to stop if 50 consecutive rounds without decrease of error and avoid overfitting of the model.

```
model_x = xgb.XGBRegressor(n_estimators=1000)
model_x.fit(sx_train_X, sy_train_X,
        eval_set=[(sx_train_X, sy_train_X), (sx_test_X, sy_test_X)], eval_metric="mae",
        early_stopping_rounds=50, #stop if 50 consequent rounds without decrease of error
        verbose=True)
```

Figure 7. XGBoost model for X time series

d) Results for time series X

After training the model, we applied it to the test set and evaluated its performance. The performance measure is mean absolute error (MAE). MAE is calculated based on the average of the forecast error values and the error values are converted to be positive.

```
preds_x = pd.DataFrame(model_x.predict(sx_test_X))
```

```
print("Value of MAE for X:")
round(mean_absolute_error(sy_test_X, preds_x) * 100,2)
```

```
Value of MAE for X:
3.99
```

Figure 8. Prediction and evaluation of the model performance

MAE value obtained for time series X model was ~ 4%, which can be considered as a good result.
As visualizing all data for model performance for time series X is not very clear, we decided to visualize some interesting parts of it which can present interesting conclusions.

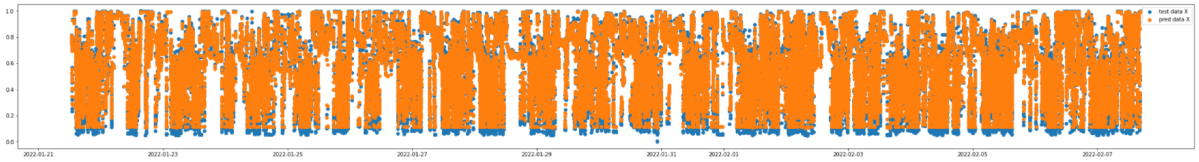


Figure 9. Visualization of model performance for the whole time series X

Analysis of the visualization of comparison of test data (blue) and predicted data (orange) for the first 150 records from time series X showed that the model had some troubles at the very beginning, but later as long as data is more or less constant it performed very well. The biggest issue is when data varies and its value starts to change rapidly, then the model produces poorly precise predictions.

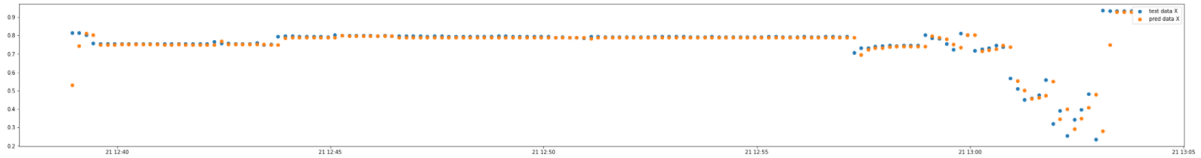


Figure 10. Visualization of model performance for first 150 elements of time series X

Figure 11. Visualization of model performance for 400:600 elements of time series X

The MAE loss of the XGBoost model for each epoch on the training (blue) and test (orange) datasets was also measured and visualized. Results can be considered as a good fit because training and validation loss decreases to a point of stability with a minimal generalization gap.

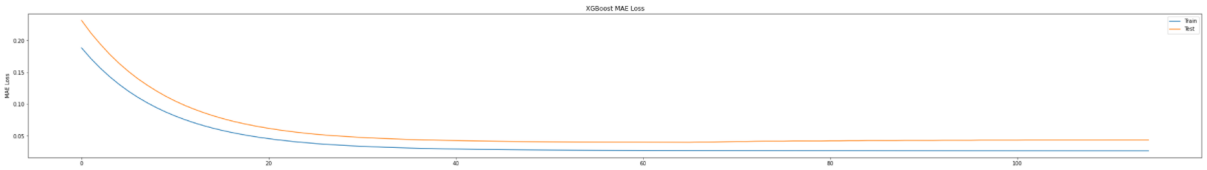


Figure 12. Visualization of model's MAE loss for time series X

After choosing the most optimal XGBoost model configuration, the model was used to make a prediction on new data. This process is about predicting beyond the training dataset. The procedure is identical to making a prediction during the evaluation of the model: as we always want to evaluate a model using the same procedure that we expect to use when the model is used to make predictions on new data.

We fitted the model on all available data and made a multi-step prediction beyond the end of the dataset. We performed a few experiments with the number of elements to predict, we tested the model against sizes - [10,30,50], which means that 10/30/50 new pieces of data will be predicted based on the last 25 elements from the provided dataset. Results were visualized and measured by using MAE. The value of MAE for 10 predicted elements was at ~7.3%, for 30 predicted elements was at ~ 13.5% and for 50 predicted elements was at 9.5%.

Our first assumption was about increasing MAE with the number of predicted elements (and it can be observed during change from 10 to 30), but then we noticed that with 50 predicted elements the MAE value decreased. The explanation behind this is in the used method, which takes *n* predicted elements and adds them to the input elements in case when input elements < elements to predict. With 50 predicted elements we obtained smaller MAE than with 30 elements because input elements at the end consisted of 25 original pieces of data + 25 predicted data. Then it can be observed that the MAE value stabilizes.

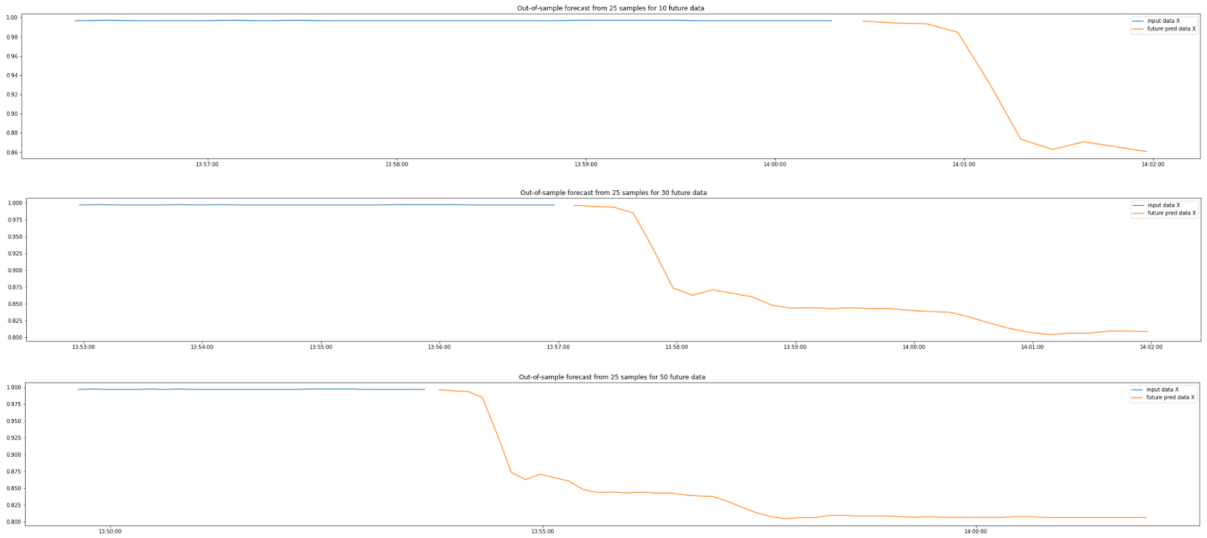


Figure 13. Out-of-sample prediction for sample size = 25 and number of new data = 10,30,50 for time series X

e) Results for time series Y

The steps for time series Y were analogical to those for time series X. At first the MAE value was measured at 4.31%, which is slightly greater than for time series X, but still it can be considered as a good result.
Visualizing all data for model performance for time series Y was also very unreadable and we decided to visualize some interesting parts of it which can present interesting conclusions.
Below the analysis of the visualization of comparison of test data (blue) and predicted data (orange) for the first 250 records from time series Y is presented. Conclusions are similar to those for time series X - the model had troubles at the very beginning, but later as long as data is more or less constant it performed very well.

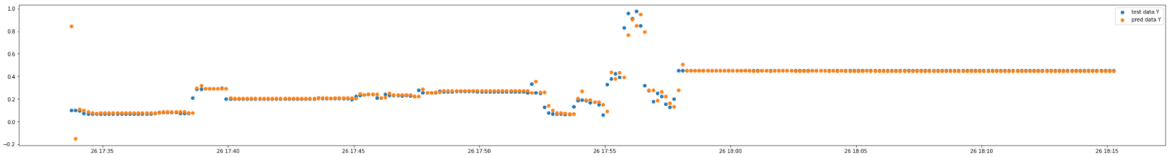


Figure 14. Visualization of model performance for first 250 elements of time series Y

After analysis of the visualization of the MAE loss of the XGBoost model it can be observed that the loss value decreases more rapidly than for time series X, but overall results can be considered as a good fit.

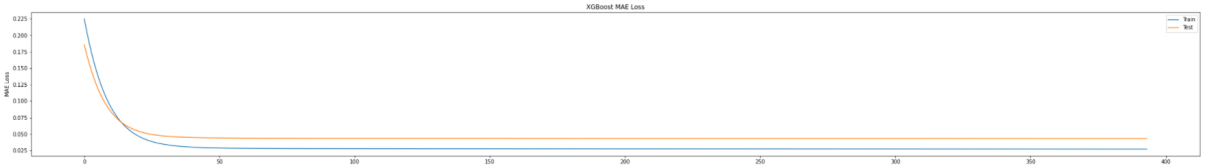
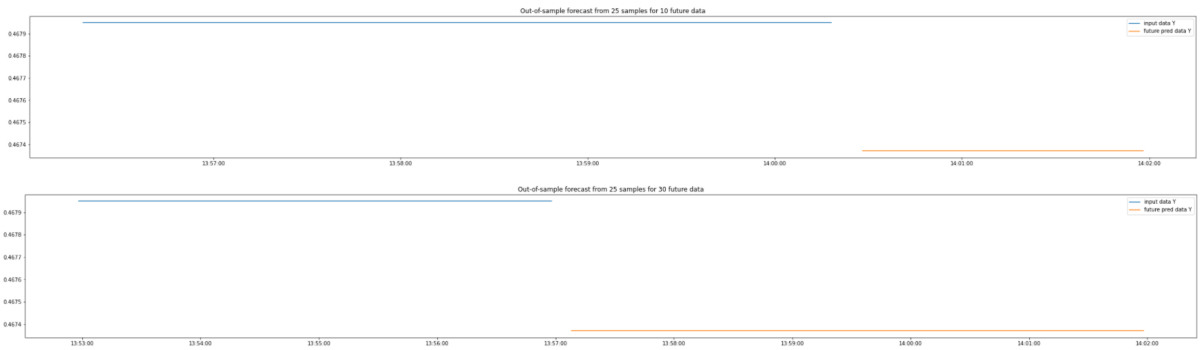


Figure 15. Visualization of model's MAE loss for time series Y

Out-of-sample predictions were also performed for the model against sizes - [10, 30, 50] of predicted elements out of 25 pieces of original data. Obtained MAE values were as follows: 0.06% for 10 future data, 0.05% for 30 future data and 0.03% for 50 future data. Conclusions were similar to those from experiment with time series X - MAE decreases when in the input sample is more predicted data. The reason why obtained MAE values for time series Y are so small may be the fact that the last 25 elements from original time series Y dataset are quite similar and differences between them are insignificant (if any). That is why future prediction is so precise and stable.



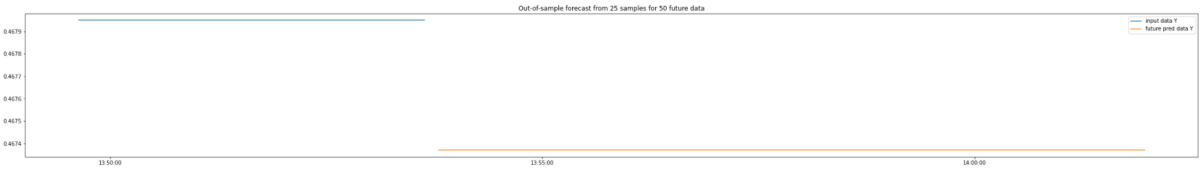


Figure 16. Out-of-sample prediction for sample size = 25 and number of new data = 10,30,50 for time series Y

f) Results for time series Z

The steps for time series Z were analogical to those for time series X and Y. The MAE value for the model with time series Z was 3.96%, which is better than for both previous time series.
Analysis of the visualization of comparison of test data (blue) and predicted data (orange) for the first 250 records from time series Z emphasized the same issues as previously - troubles with fitting at the very beginning and during rapid values changes.

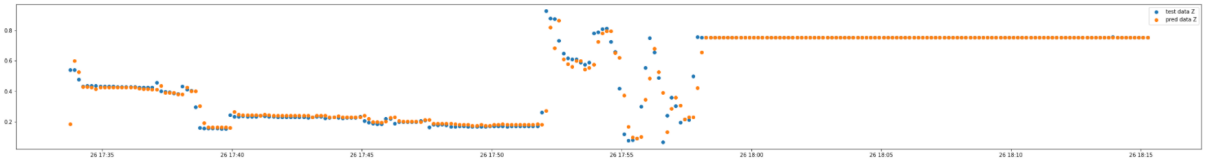


Figure 17. Visualization of model performance for first 250 elements of time series Z

Analysis of the visualization of the MAE loss for the model confirms obtained earlier the smallest MAE value for time series Z. On the chart the smallest generalization gap can be observed.

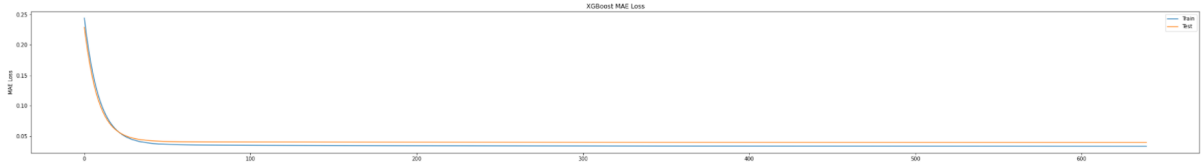
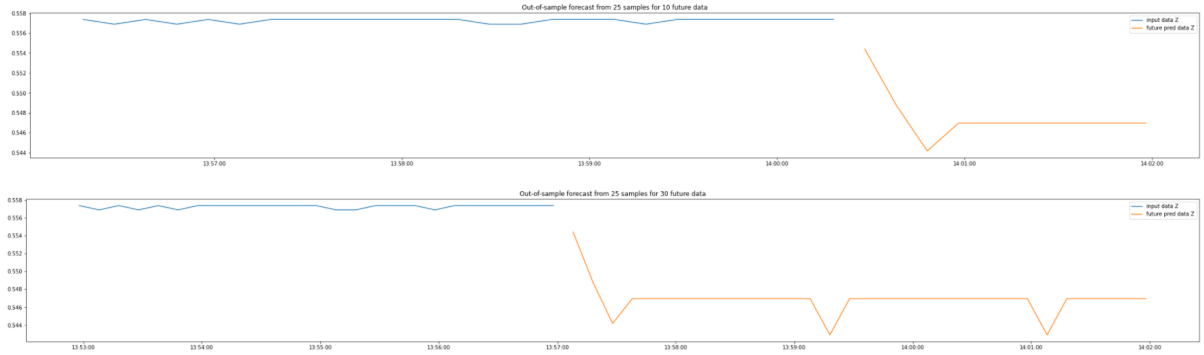


Figure 18. Visualization of model's MAE loss for time series Z

Out-of-sample predictions were processed similarly to the previous ones. Obtained MAE results were as follows: 1% for 10 future data, 0.9% for 30 future data, 0.6% for 50 future data. Conclusions are analogical as for out-of-sample predictions for time series X and Y.



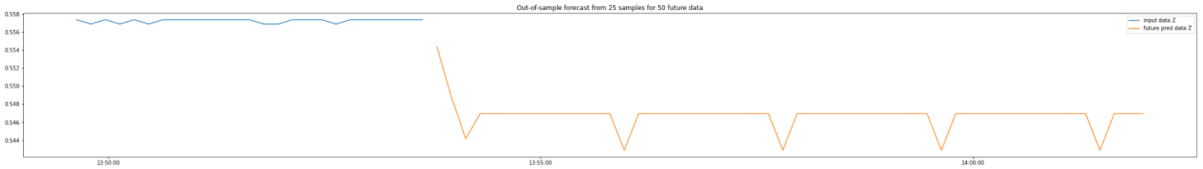


Figure 19. Out-of-sample prediction for sample size = 25 and number of new data = 10,30,50 for time series Z

g) How do the *window_size* and the *window_shift* affect the result for time series X?

In order to examine how the *window_size* and the *window_shift* affect the result for time series X a few experiments were conducted.
As the default *window_size* = *2* and default *window_shift* = *1* (which gave MAE = ~ 4%) in experiments both factors were modified as follows:

1) *window_size* = *4* and *window_shift* = *2,* MAE = 5.55%
2) *window_size* = *6* and *window_shift* = *4,* MAE = 7.33%
3) *window_size* = *10* and *window_shift* = *2,* MAE = 5.62%
4) *window_size* = *10* and *window_shift* = *4,* MAE = 7.55%

Every time MAE values were greater than for the original model, so it can be said that increasing *window_size* and *window_shift* worsen model's results in general.
It can be also seen that the size of *window_shift* is very significant in case of model's precision as even when we set big window (10) but small shift (2) we obtained decent results (MAE = 5.62%) in contrary to the case when we set big window (10) with shift = 4 (MAE = 7.55%). Moreover it can be seen that the result of increasing both factors is also overfitting of the model (plot of training loss continues to decrease with experience, plot of validation loss decreases to a point and begins increasing again).

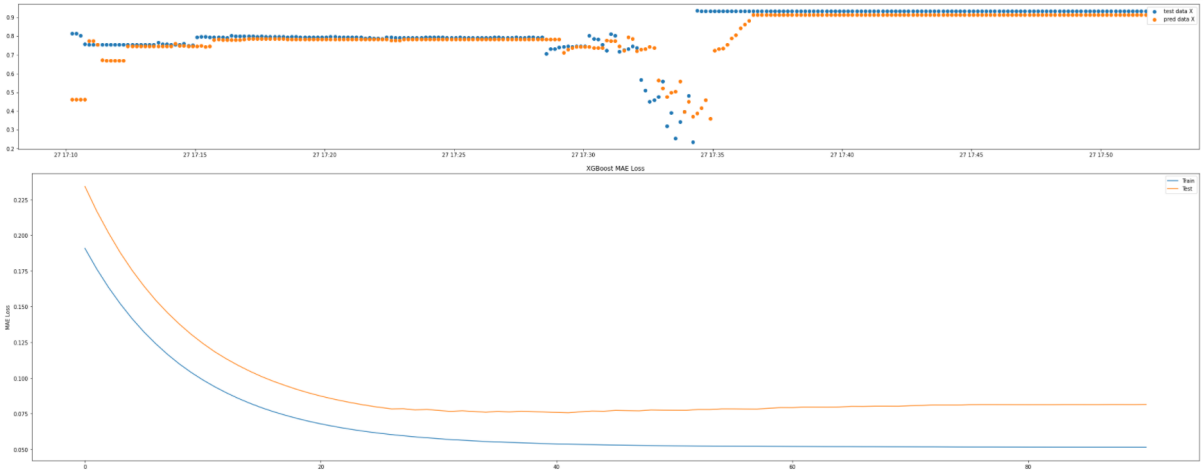


Figure 20.Results for time series X model for windows_size=10 and shift = 4

h) Conclusions for the Next Value Prediction task

The main topic of this task was about forecasting on provided time series X, Y, Z. In order to do that XGBoost Model library was used. Created model evaluated by the Mean Absolute Value metric obtained good results smaller than 10%. In general, it has some troubles at the very beginning, but later as long as data is more or less constant it performs very well. The biggest issue is when data varies and its value

starts to change rapidly, then the model produces poorly precise predictions.
Visualization of the training loss and validation loss for each time series showed that the model produces a good fit.
Out-of-sample predictions for the time series formulate a conclusion that the more already predicted and stale values in an input sample the better outcome is produced.
Examination of an impact of *window_size* and the *window_shift* for time series X emphasized that increasing width of both features affects negatively on the model performance and causes overfitting.