Faculty of Information Technology and Electrical Engineering

DT8807 - Advanced Topics in Deep Learning with Python

Correlated multivariate time series imputation

Author:
Luka Grgičević

June, 2023

Table of Contents

1 Introduction

An automatic identification system (AIS) is well established automatic tracking system used to provide relevant information to identify vessels, their class, size, and data about previous speeds and positions, among others. Each vessel transmits its information via radio signals using the National Marine Electronics Association (NMAE) standard for marine communication, which is picked up by terrestrial or satellite receivers. Today, historical and real-time AIS data is processed by machine learning algorithms for the purpose of saving fuel [8], predicting vessels trajectories, estimated time of arrivals to destination ports [7], analyzing the port's anchoring areas and finding open-seas patterns in ship routes, obtaining information about the specific commercial vessel trading patterns, optimizing search and rescue operations [13] and even identifying, a rather accurate, vessel dynamic model [16]. Data processed in this way is today called a dynamic AIS [12].

AIS data is subject to noise due to environmental influences and delays. It comes in sampling time that is irregular because of different navigational status vessels are broadcasting. For example, a vessel that is anchored only has to transmit its data every 3 minutes, unlike ships operating at speeds larger than 23 knots which have to transmit every 2 seconds [10]. Dynamic AIS requires processing missing and irregularly sampled data. This is equivalent to correlated multivariate time series data imputation from the estimated functions that represent the vessel model in three degrees of freedom. On a vessel control system level, where the onboard sensor data is gathered, such data extrapolation and function interpolation to obtain better models of vessel motions are equivalent to state estimation and filtering.

The author refers the reader to the community actively engaged in dealing with corrupted and irregularly sampled time series [1]. In this report, data imputation is done using a variant of the recurrent neural network model, presented in chapter 3.1 [1]. The main reason is domain knowledge about the model's behaviour. More specifically, the rate of change of time series columns is rather small, there is high autocorrelation in the time series, and there is a high correlation between a positional pair and a speed-course pair.

2 Literature review

There exists a large amount of research literature on time series data imputation because it spans different disciplines. Widely used model-free machine learning methods in classification, prediction, imputation and clustering of time series imply self-attention architectures [4], recurrent neural networks (RNN) architectures [1][15], graph neural network architectures [17], generative adversarial network architectures [9], transformers [14], long short-term memory (LSTM) architectures [5] and probabilistic methods [2], to name a few. Recent research shows LSTM neural networks have better performance than RNN because of their issue with the inability to capture long-term information [6]. However, by adding *leaky* gates in RNNs *hidden* units that adaptively remember and forget the information, this issue might be solved [3].

3 Methodology

3.1 Data

The data consists of position in latitude and longitude, speed over ground and course over ground. It is obtained from The Norwegian Coastal Administration[2] AIS terrestrial network and does not include data on fishing vessels under 15 meters and recreational craft under 45 meters. The number of timestamps per vessel varies between 500 to several thousand and for sampling times one might make 3 clusters around 2, 10 and 15 seconds, which correspond to International Maritime

[1]Link to the PyPOTS community: https://pypots.com/
[2]Link to the Kysteverket: https://kystverket.no/en/navigation-and-monitoring/ais/access-to-ais-data/

Organization (IMO) specification on navigational status, with small delays. The dataset has been prepared for training in the following manner:

- Removal of unnecessary columns in obtained raw .csv files.
- Uploading 40 parquet[3] files on High Performance Computing Group (IDUN) cluster [11], where each file contains timestamps from 200 to 300 vessels.
- Choosing vessels that contain information longer than 500 samples
- Grouping data in data frames[4] for each vessel separately for preprocessing.
- Resampling of time series at the smallest cluster (3 seconds), taking the mean value in time buckets and making groups of 6000 timestamps per vessel. The total number of timestamps is 12 876 000.
- Wrapping course angles from {0,360} to {-180,180} degrees to prepare for scaling to {-1,1}.
- Taking 20% of the total data for testing and then 20% from the remaining 80% for validating. Each group is sorted by vessel's unique maritime mobile service identities (MMSI).
- Making a 3-dimensional NumPy arrays[5] for all three sets such that the first dimension is the length of the MMSI list, the second is the number of timestamps per vessel and third is the number of correlated data columns.

Two scenarios were tested and the results of imputation performance were compared. In the first, we left only longitude and latitude and in the second speed over ground and course were included.

3.2 Model

By using RNN for vessel position time series imputation, one should be able to train the model that doesn't require long-term memory to impute values. The vanishing gradient problem that affects RNNs shouldn't be of concern if we don't initialize numerous hidden cells.

In this subchapter, a model used to obtain the results is presented in detail [1].The bidirectional recurrent imputation for time series (BRITS) instantiates the dynamical system as a bidirectional RNN where the mean imputation result is taken as final. The model consists of a recurrent layer and a regression layer. Missing values are treated as nodes in RNN graph, which are updated during backpropagation which runs counterclockwise in Figure 1. Measurements $x_{t\pm k}$ which are taken irregularly and consist of missing values are fed into the model consecutively. The time series of the measurements are complemented with a masking vector $m_t = 0, 1$ and time gaps vector δ_t. Value x_{t-1} is combined with the previous imputation estimates $\hat{x}_t$ to construct the *complement* variable x^c_{t-1}.

$$x^c_t = m_t * x_t + (1 - m_t) * \hat{x}_t \tag{1}$$

From the complement vector x^c_t, a correlation between time series $\hat{z}_t$ is captured using Equation (2). W_z and b_z are neural network parameters and a bias vector.

$$\hat{z}_t = W_z * x^c_t + b_z \tag{2}$$

Next, a weighted sum combination of estimates obtained by correlation $\hat{z}_t$ and auto-correlation $\hat{x}_t$ is done. The weight $\beta_t \in [0, 1]$ is updated using Equation (4), where the delay vector γ_t is concatenated with the masking vector m_t to feed into the network parameters W_β with a

[3]Apache Parquet: https://parquet.apache.org/
[4]Pandas package: https://pandas.pydata.org/
[5]NumPy package: https://numpy.org/

bias b_β. Mixing by concatenation is done because the x_t^c can be made of true measurements or autocorrelated estimates.

$$\hat{c}_t = \beta_t \odot \hat{z}_t + (1 - \beta_t) \odot \hat{x}_t \tag{3}$$

$$\beta_t = \sigma(W_\beta * [\gamma_t \circ m_t] + b_\beta) \tag{4}$$

Therefore, the sigmoid activation gates are modulated by the decay factor vector γ_t which is governed by Equation (5). It is clear that having a bigger gap between the measurements reduces the temporal decay factor.

$$\gamma_t = exp\{-max(0, W_\gamma * \delta_t + b_\gamma)\} \tag{5}$$

Once the combination vector $\hat{c}_t$ is obtained, we place them instead of missing values in the measurement vector x_t. The new combination vector x_t^c is obtained using the mask m_t.

$$c_t^c = m_t * x_t + (1 - m_t) * \hat{c}_t \tag{6}$$

Next, the hidden states h_t which keep the memory are obtained using Equation (7).

$$h_t = \sigma(W_h * [h_{t-1} \odot \gamma_t] + U_h * [c_t^c \circ m_t] + b_h) \tag{7}$$

The final layer is a fully connected regression layer to predict the next time stamp $\hat{x}_t$.

$$\hat{x}_t = W_x * h_{t-1} + b_x \tag{8}$$

Here we complete one cycle and return to Equation (1) to form another complement vector x_t^c using estimates from the previous step where the measurements are missing. Figure 1 visualizes this process. At the end of the cycle, an estimate $\hat{x}_{t-1}$ is made for the values at the time stamp t.

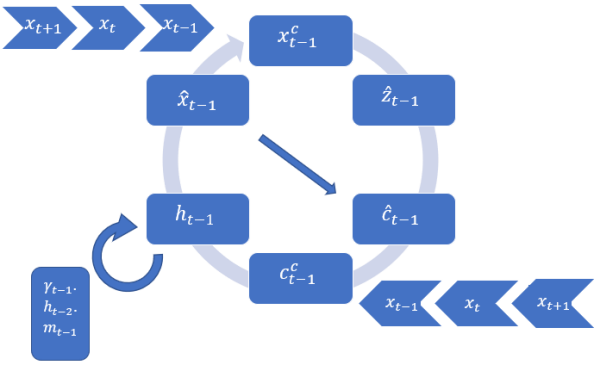

Figure 1: BRITS model representation

The reconstruction loss function is a combined mean absolute error between measurements x_t and the three vectors $\hat{x}_t$, $\hat{c}_t$ and $\hat{z}_t$ only where the masking vector m_t is equal to 1.

$$\mathcal{L} = \sum_n (|x_t - \hat{x}_t| + |x_t - \hat{c}_t| + |x_t - \hat{z}_t|)/n \tag{9}$$

To evaluate the imputation performance and compute mean validation loss, there are 20 per cent randomly eliminated values from the observed measurement which are used as the ground truth. The model isn't trained on these values and the number can be easily increased if the total number

of observations is bigger. Consistency loss is a discrepancy measure between forward and backward pass in value imputation. The final value is taken as a mean of two passes. Equation (9) is also used as a training loss metric in addition to the cumulative consistency losses.

4 Results

In the first scenario, the BRITS was trained for 200 epochs, adaptive moment estimation (ADAM) optimizer [6] with a learning rate set to 0.01 and 150 hidden states on longitude and latitude data columns only, with around 210 vessels with 6000 sparse timestamps each. One epoch consists of forward and backward passes through the weights. In Figure (2) consistency, validation (imputation) and training loss are visualized.

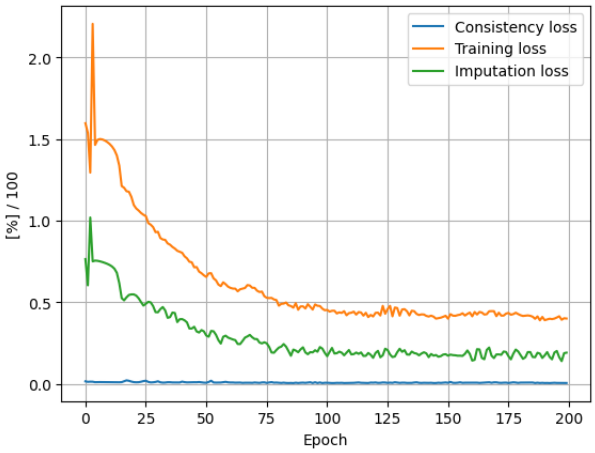

Figure 2: BRITS model representation

In the second scenario, the BRITS model was trained in the same setting as the first, except for two additional columns that were added. Course and speed over ground. The losses over epochs are plotted in Figure (3).

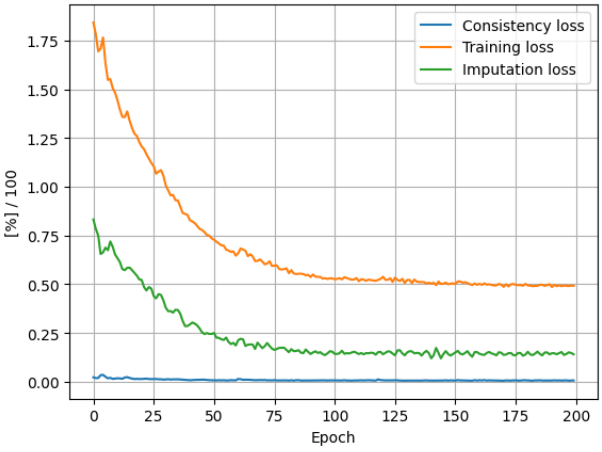

Figure 3: BRITS model representation

Slightly higher training loss and lower validation loss, as well as lower oscillations from epoch to epoch in the second trial, could be observed. It was to be expected that the second scenario should have provided better imputation results since the additional columns are clearly enabling better insight into vessel behaviour.

[6] Adam: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

5 Conclusion

During the training, the model allocated around 50 gigabytes of memory in the IDUN cluster and computational time was around one hour on a single graphics processing unit. Due to the overloaded schedule[7], there was no time to repeat the training under different conditions in order to improve the results or to obtain the standard deviation for the repeated measurements of model performance.

The correlation between position, speed and heading should have been better captured, particularly because individual column connections are related to the movement of the rigid body. Therefore, the goal of future work is to investigate potential improvements in the dataset as well as network architecture. The dataset could include a vessel length column which is constant throughout 6000 timestamps. Maybe the model manages to recognize that the reason behind different dynamics is in the vessel length. The rest of the influences, such as vessel mass distribution, hydrodynamical parameters, environmental conditions etc are out of reach naturally.

The network architecture might be improved by perhaps finding a better way to model the correlation $\hat{z}_t$. For any motion of a rigid body, inertia has a big impact on the future trajectory, so the model should be able to recognize that past values dataset columns are crucial for estimating the current value in one of them.

In order to replicate the results a python[8] script is provided in the appendix[9]. The author is involved in the preparation of the AIS dataset to be published in time series database (TSDB)[10] for other researchers to use. The dataset and the preprocessing script will be publicly available.

[7] Website of the workload manager: https://slurm.schedmd.com/documentation.html
[8] https://www.python.org/
[9] To access the IDUN cluster one has to be affiliated with institutions involved in the project
[10] Repository: https://github.com/WenjieDu/TSDB

Bibliography

[1] Wei Cao et al. *BRITS: Bidirectional Recurrent Imputation for Time Series*. 2018. arXiv: 1805.10572 [cs.LG].

[2] Xinyu Chen and Lijun Sun. 'Bayesian Temporal Factorization for Multidimensional Time Series Prediction'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/tpami.2021.3066551. URL: https://doi.org/10.1109%2Ftpami.2021.3066551.

[3] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[4] Wenjie Du, David Côté and Yan Liu. 'SAITS: Self-attention-based imputation for time series'. In: *Expert Systems with Applications* 219 (Jan. 2023), p. 119619. DOI: 10.1016/j.eswa.2023.119619. URL: https://arxiv.org/abs/2202.08516.

[5] F.A. Gers, J. Schmidhuber and F. Cummins. 'Learning to forget: continual prediction with LSTM'. In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*. Vol. 2. 1999, 850–855 vol.2. DOI: 10.1049/cp:19991218.

[6] Mesut Guven and Fatih Uysal. 'Time Series Forecasting Performance of the Novel Deep Learning Algorithms on Stack Overflow Website Data'. In: *Applied Sciences* 13.8 (2023). ISSN: 2076-3417. DOI: 10.3390/app13084781. URL: https://www.mdpi.com/2076-3417/13/8/4781.

[7] Ibadurrahman et al. 'Long-Term Ship Position Prediction Using Automatic Identification System (AIS) Data and End-to-End Deep Learning'. In: *Sensors* 21.21 (2021). ISSN: 1424-8220. DOI: 10.3390/s21217169. URL: https://www.mdpi.com/1424-8220/21/21/7169.

[8] Q Liang, H A Tvete and H W Brinks. 'Prediction of vessel propulsion power from machine learning models based on synchronized AIS-, ship performance measurements and ECMWF weather data'. In: *IOP Conference Series: Materials Science and Engineering* 929.1 (Nov. 2020), p. 012012. DOI: 10.1088/1757-899X/929/1/012012. URL: https://dx.doi.org/10.1088/1757-899X/929/1/012012.

[9] Yonghong Luo et al. 'Multivariate Time Series Imputation with Generative Adversarial Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/96b9bff013acedfb1d140579e2fbeb63-Paper.pdf.

[10] International Maritime Organization. *IMO Resolution A.1106(29) – Revised Guidelines for fhe Onboard Operational Use of Shipborne Automatic Identification Systems (AIS) – (Adopted on 2 December 2015)*. URL: https://imorules.com/IMORES_A1106_29.html (visited on 6th June 2023).

[11] Magnus Själander et al. 'EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure'. In: *arXiv:1912.05848 [cs]* (Dec. 2019). arXiv: 1912.05848 [cs].

[12] Inc. Spire Global. *Dynamic AIS*. URL: https://spire.com/maritime/solutions/dynamic-ais/ (visited on 6th June 2023).

[13] Iraklis Varlamis, Konstantinos Tserpes and Christos Sardianos. 'Detecting Search and Rescue Missions from AIS Data'. In: June 2018. DOI: 10.1109/ICDEW.2018.00017.

[14] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[15] Jinsung Yoon, William Zame and Mihaela Schaar. 'Estimating Missing Data in Temporal Data Streams Using Multi-Directional Recurrent Neural Networks'. In: *IEEE Transactions on Biomedical Engineering* PP (Nov. 2017). DOI: 10.1109/TBME.2018.2874712.

[16] Mingyang Zhang et al. 'A deep learning method for the prediction of 6-DoF ship motions in real conditions'. In: *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* 0.0 (0), p. 14750902231157852. DOI: 10.1177/14750902231157852. eprint: https://doi.org/10.1177/14750902231157852. URL: https://doi.org/10.1177/14750902231157852.

[17] Xiang Zhang et al. 'GRAPH-GUIDED NETWORK FOR IRREGULARLY SAMPLED MULTIVARIATE TIME SERIES'. In: Feb. 2022.

Abbreviations

ADAM adaptive moment estimation.

AIS automatic identification system.

BRITS bidirectional recurrent imputation for time series.

IDUN High Performance Computing Group.

IMO International Maritime Organization.

LSTM long short-term memory.

MMSI maritime mobile service identities.

NMAE National Marine Electronics Association.

RNN recurrent neural networks.

TSDB time series database.

APPENDIX

Listing 1: Schell script for SLURM job scheduler at IDUN

```
#!/bin/sh
#SBATCH --partition=GPUQ
#SBATCH --time=0-05:00:00
#SBATCH --nodes=2                # 2 compute nodes
#SBATCH --mem=50G               # 50GB
#SBATCH --job-name="Time series imputation"
#SBATCH --gres=gpu:1
#SBATCH --output=Imputation-gpu.out
#SBATCH --mail-user=luka.grgicevic@ntnu.no
#SBATCH --mail-type=ALL

#module load Python/3.10.8-GCCcore-12.2.0
#module load cuDNN/8.4.1.50-CUDA-11.7.0

WORKDIR=${SLURM_SUBMIT_DIR}
cd ${WORKDIR}
echo "We are running from this directory: $SLURM_SUBMIT_DIR"
echo "The name of the job is: $SLURM_JOB_NAME"
echo "The job ID is $SLURM_JOB_ID"
echo "The job was run on these nodes: $SLURM_JOB_NODELIST"
echo "Number of nodes: $SLURM_JOB_NUM_NODES"
echo "We are using $SLURM_CPUS_ON_NODE cores"
echo "We are using $SLURM_CPUS_ON_NODE cores per node"
echo "Total of $SLURM_NTASKS cores"

python ./Imputation.py
uname -a
```

Listing 2: Python script for dataset preprocessing and model training, Imputation.py

```
# We have 40 parquet files uploaded on the server
#pip install https://github.com/WenjieDu/PyPOTS/archive/main.zip
#pip install pyarrow

from pypots.data.utils import mcar, masked_fill
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import torch
import os
import zipfile
import glob
import pyarrow as pa
import pyarrow.parquet as pq
from pypots.imputation import BRITS
from pypots.optim import Adam
import random as r

path_to_parquets = '/cluster/work/lukagr/AIS/parquets'

filenames= os.listdir (path_to_parquets) # get all files' and folders' names in the current directory

mmsis = list()
s = list()
a = list()
ships = dict()

#Looping through the files

for p in filenames:

    path = path_to_parquets + '/' + p
    df = pd.read_parquet(path, engine='pyarrow')

    #setting datetime as index as float

    datetime_series = pd.to_datetime(df["date_time_utc"])
    datetime_index = pd.DatetimeIndex(datetime_series.values)
    df = df.set_index(datetime_index)
    df['date_time_utc']= pd.to_numeric(datetime_series.values) / 10 ** 9

    #listing vessels with according to their mmsi numbers
    vessels = list(df["mmsi"].unique())

    for k in vessels:

        #Dataframe per mmsi group
        g = df.groupby(by = "mmsi").get_group(k)
        #Take only part of the group where the vessel is moving and data is bigger than 2000
        g = g.loc[g['sog'] > 0.2 ].iloc[0:2000]

        if len(g) == 2000:
        #resample into 3s buckets and take the mean value inside, then cut at 6000  timestampt
         g = g.resample('3S').mean(numeric_only=True).iloc[0:6000]

        if len(g) == 6000:

         #Repair mmsi and length columns
         g.loc[:,['mmsi']] = g.loc[:,['mmsi']].ffill()
         g['mmsi'] =  g['mmsi'].values.astype(int)
         g.loc[:,['length']] = g.loc[:,['length']].ffill()

        #Change the index with reseted time so each vessel starts from t=0
         datetime_series = pd.to_datetime(g.index)
         t = np.amin(datetime_series.values).astype(float)
         g["time"] = ( datetime_series.values.astype(float) - t ) / 10 ** 9

        #Wrap the heading and course so it doesn't influence the scaler later on
         g["heading"] = g["true_heading"].apply(lambda x: np.mod(x - 180.0, 360.0) - 180.0)
         g["course"] = g["cog"].apply(lambda x: np.mod(x - 180.0, 360.0) - 180.0)

         # Take only one trajectory per vessel mmsi
         m = g["mmsi"][0]

         if m not in mmsis:
            s.append(g.reset_index(drop=True).drop(columns = ["nav_status", "message_nr","imo_nr",
```

```
            "date_time_utc", "true_heading","cog"]))
            a.append(pd.DataFrame({"Length":[g["length"][0]], "MMSI": [g["mmsi"][0]]}))

        mmsis.append(m)

# Grouping all in one dataframe and making target column as length for eventual classification
ships['X'] = pd.concat(s)
ships['y'] = pd.concat(a)
ships['y'].set_index("MMSI", inplace=True)

#print(len(mmsis))


def AIS_prepare(artificially_missing_rate: float = 0.2):
    """Generate a fully-prepared AIS dataset for model testing.

    Parameters
    ----------
    artificially_missing_rate :
        The rate of artificially missing values to generate for model evaluation.
        This ratio is calculated based on the number of observed values, i.e. if artificially_missing_rate = 0.1,
        then 10% of the observed values will be randomly masked as missing data and hold out for model evaluation.

    Returns
    -------
    data: dict,
        A dictionary containing the generated AIS dataset.

    """
    assert (
        0 <= artificially_missing_rate < 1
    ), "artificially_missing_rate_must_be_in_[0,1)"

    X = ships['X']
    y = ships['y']

    #print (X.shape)

    # Sorting by time and mmsi is necessary in order to find and
    #link the vessel data after training
    ships_mmsi = X["mmsi"].unique()
    #print (len(ships_mmsi))

    train_set_ids, test_set_ids = train_test_split(ships_mmsi, test_size=0.2)
    train_set_ids, val_set_ids = train_test_split(train_set_ids, test_size=0.2)

    train_set_ids.sort()
    val_set_ids.sort()
    test_set_ids.sort()

    train_set = X[X["mmsi"].isin(train_set_ids)].sort_values(["mmsi", "time"])
    val_set = X[X["mmsi"].isin(val_set_ids)].sort_values(["mmsi", "time"])
    test_set = X[X["mmsi"].isin(test_set_ids)].sort_values(["mmsi", "time"])

    train_y = y[y.index.isin(train_set["mmsi"])].sort_index()
    val_y = y[y.index.isin(val_set["mmsi"])].sort_index()
    test_y = y[y.index.isin(test_set["mmsi"])].sort_index()
    train_y, val_y, test_y = train_y.to_numpy(), val_y.to_numpy(), test_y.to_numpy()

    #Here we don't need to drop speed, course or length columns
    train_set = train_set.drop(["time","mmsi","length","heading","course","sog"], axis=1)
    val_set = val_set.drop(["time","mmsi","length","heading","course","sog"], axis=1)
    test_set = test_set.drop(["time","mmsi","length","heading","course","sog"], axis=1)

    train_X, val_X, test_X = (
        train_set.to_numpy(),
        val_set.to_numpy(),
        test_set.to_numpy(),
    )

    scaler = StandardScaler()
    # normalization
    train_X = scaler.fit_transform(train_X)
    val_X = scaler.fit_transform(val_X)
    test_X = scaler.fit_transform(test_X)

    #Last dimension should be changed to 4 if adding two columns, 6000 is the number of timestamps per vessel
    train_X = train_X.reshape(len(train_set_ids), 6000, 2)
    test_X = test_X.reshape(len(test_set_ids), 6000, 2)
    val_X = val_X.reshape(len(val_set_ids), 6000, 2)

    data = {
        "n_classes": len(ships['y']["Length"].unique()),
        "n_steps": 6000,
        "n_features": train_X.shape[-1],
        "train_X": train_X,
        "train_y": train_y.flatten(),
        "val_X": val_X,
        "val_y": val_y.flatten(),
        "test_X": test_X,
        "test_y": test_y.flatten(),
        "scaler": scaler,
        "train_id": train_set_ids,
        "test_id": test_set_ids,
        "val_id": val_set_ids,

    }

    if artificially_missing_rate > 0:
        # mask values in the validation set as ground truth
        val_X_intact, val_X, val_X_missing_mask, val_X_indicating_mask = mcar(
            val_X, artificially_missing_rate
        )
        val_X = masked_fill(val_X, 1 - val_X_missing_mask, torch.nan)

        # mask values in the test set as ground truth
        test_X_intact, test_X, test_X_missing_mask, test_X_indicating_mask = mcar(
            test_X, artificially_missing_rate
        )
        test_X = masked_fill(test_X, 1 - test_X_missing_mask, torch.nan)

        data["val_X"] = val_X
        data["test_X"] = test_X
        data["test_X_intact"] = test_X_intact
        data["test_X_indicating_mask"] = test_X_indicating_mask
```

```
        data["val_X_intact"] = val_X_intact
        data["val_X_indicating_mask"] = val_X_indicating_mask

    return data

D = AIS_prepare()

#The details could be found in the authors Github repository and the references

brits = BRITS(
    n_steps = D["n_steps"],
    n_features = D["n_features"],
    batch_size=2000,
    optimizer=Adam(lr=1e-2),
    rnn_hidden_size=150,
    epochs=200, # here we set epochs=10 for a quick demo, you can set it to 100 or more for better performance
    patience=None, # here we set patience=5 to early stop the training if the evaluting
    #loss doesn't decrease for 5 epoches. You can leave it to defualt as None to disable early stopping.
    device=['cuda'], # just leave it to default, PyPOTS will automatically assign the best device for you.
                     # Set it to 'cpu' if you don't have CUDA devices.
                     #You can also set it to 'cuda:0' or 'cuda:1' if you have multiple CUDA devices.
    model_saving_strategy="best",
    num_workers=0,
    saving_path="results/imputation/brits", # set the path for saving tensorboard files
)

dataset_for_training = {
    "X": D['train_X'],
}

dataset_for_validating = {
    "X": D['val_X'],
    "X_intact": D['val_X_intact'],
    "indicating_mask": D['val_X_indicating_mask'],
}

dataset_for_testing = {
    "X": D['test_X'],
}

brits.fit(train_set=dataset_for_training, val_set=dataset_for_validating)

brits_imputation = brits.impute(dataset_for_testing)

#scaler = D['scaler']

#test = pd.DataFrame(scaler.inverse_transform(dataset_for_testing['X'][7]), columns = ["lon", "lat"])
#test_im = pd.DataFrame(scaler.inverse_transform(brits_imputation[7]), columns = ["lon", "lat"])
```