University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialization: MSc. Computational Engineering	Spring semester, 2022 Open
Author: Ali Tahir	.... Ali Tahir

Programme coordinator:
Øystein Arild

Supervisor:
UiS - Prof. Dan Sui

Title of master's thesis:
IMPACT OF DATA PRE-PROCESSING TECHNIQUES ON MACHINE LEARNING MODELS

Credits: 30

Keywords: Machine Learning, Recurrent neural network, K-Nearest Neighbors Algorithm, Artificial Neural Network.	Number of pages: 87 + supplemental material/other: 50 Stavanger, 15th July 2022

Ali Tahir

IMPACT OF DATA PRE-PROCESSING TECHNIQUES ON MACHINE LEARNING MODELS

Master Thesis Project for the degree of
MSc. in Computational Engineering

Stavanger, July 2022

University of Stavanger
Faculty of Science and Technology
Department of Energy Resources

Universitetet
i Stavanger

Abstract

The Volve dataset, which contains the time series values of different sensors that have been used at the Volve drilling site contains many flaws which make it hard for machine learning models to learn from the dataset and provide useful insights and future predictions. Three flaws have been highlighted including missing data, different frequency rates, and too many attributes (high dimensional data). To solve the issues, present in time series data, a data preprocessing pipeline has been proposed which first removes the noise through the rolling mean. Then applies gap analysis to remove the columns whose gaps can not be filled with data imputation methods. After that gap has been filled by the KNN imputer which imputes the missing values in the data. After that data resampling has been applied to make the sampling rate consistent as the time series prediction model takes a constant sampling rate. For hyper-parameter tuning of the resampling method AIC and BIC value has been created on a grid of hyper-parameters. After resampling, top parameters were selected on basis of Pearson correlation, after which AIC and BIC have been used to select the most relevant 3 parameters. These 3 parameters has then be used to train three models that are: RNN + MLP, LSTM + MLP, and LSTM + RNN + MLP. On basis of mean absolute error (MAE) best model has been selected which is RNN + MLP.

Acknowledgments

I, first of all, would like to start by being grateful to God Almighty, for giving me all the strength required in the completion of my studies.

This thesis study for the completion owes lot of gratitude and appreciation to number of individuals for their selfless contribution, especially my supervisor Prof. Dan Sui for the guidance and supervising this research by arranging weekly meetings, productive discussion and sharing knowledge.

I would like to show my deepest gratitude to University of Stavanger for giving me platform and all the professors who were part of this journey.

Finally, I would like to thank my family and peers for their complete support.

Ali Tahir

List of Abbreviations

ANN	Artificial Neural Network
BHA	Bottom Hole Assembly
CSV	Comma-Separated Value
DWOB	Downhole Weight on Bit
DTQ	Downhole Torque
ECD	Equivalen Circulating Density
HPHT	High Pressure High Temperature
ID	Inside Diameter
IQR	Interquartile Range
KNN	K Nearest Neighbors
LWD	Logging While Drilling
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi Layer Perceptron
MSE	Mechanical Specific Energy
MWD	Measuring While Drilling
NN	Neural Network
OD	Outside Diameter
RF	Random Forest
RNN	Recurrent Neural Network
ROP	Rate of Penetration
RPM	Revolutions per minute
SPP	Standpipe Pressure
STQ	Surface Torque
SWOB	Surface Weight on Bit
WOB	Weight on Bit
WDP	Wired Drill Pipe

Table of Contents

Abstract ii

Acknowledgments iii

List of Abbreviations iv

List of Figures viii

List of Tables ix

1 Introduction 1

1.1 Data Pre-Processing and Machine Learning 1

1.2 Literature Review 3

1.2.1 Missing Data Imputation 3

1.2.2 Volve Dataset 5

1.3 Motivation and Problem Statement 6

1.4 Objective 7

1.5 Thesis Organization 7

2 Background 8

2.1 Data Analysis 8

2.1.1 Data cleaning 8

2.1.2 Data Transformation 9

2.2 Data Imputation 9

2.3 Data Resampling 10

2.4 Feature Selection 10

2.4.1 AIC & BIC 11
2.5 Time Series Models 12
2.5.1 Time Series Analysis 12
2.5.2 Artificial Neural Networks 13
2.5.3 Long Short Term Memory 13
2.5.4 Recurrent Neural Networks 14
2.5.5 Mean Absolute Error 15

3 Mathodology **18**
3.1 Machine Learning Pipeline 18
3.1.1 Missing Data Imputation 20
3.1.2 Data Resampling 21
3.1.3 Feature Selection 22

4 Results and Discussion **27**
4.1 Dataset 27
4.2 Simulation Environment 27
4.3 KNN-Imputation 28
4.4 Resampling Hyperparameter Tuning 28
4.5 Feature Selection 30
4.6 Model Selection 33

5 Conclusion and Future Work **35**
5.1 Conclusion 35
5.2 Future Work 35

References **37**

Appendices **38**

Appendix A Parameter Combinations **40**

Appendix B Python Code **43**

B.1 Resampling Method Optimization 43
B.2 Parameter Selection 73

List of Figures

1.1 Data faults and depth v/s inclination graph 3

2.1 Operations of LSTM [1] 14

2.2 Architecture of LSTM [1] 15

2.3 Gates in LSTM 16

2.4 Architecture of RNN [2] 17

3.1 Flowchart for feature selection (left) and Data Resampling hyper-parameter tuning (right) via AIC and BIC 19

3.2 Model architecture of LSTM + MLP 24

3.3 Model architecture of RNN + MLP 25

3.4 Model architecture of LSTM + RNN + MLP 26

4.1 Graphs before (top) and after (bottom) KNN imputation 30

4.2 Heatmap of pearson correlation 31

4.3 AIC (top) and BIC (bottom) graphs for different parameter combinations 32

4.4 AIC (top) and BIC (bottom) graphs for different parameter combinations 34

List of Tables

2.1 Possible combination of 4 variables for applying AIC and BIC 12

4.1 Results for resampling method selection and hyper-parameter tuning 29

4.2 Pearson Correlation of parameters with dependent variables 30

4.3 Mean Absolute Error after training different models. 33

Chapter 1

Introduction

1.1 Data Pre-Processing and Machine Learning

Cleaning of dataset is most of the basic step in pre-processing for a machine learning pipeline. Pre-processing of data include [3]:

- Data cleaning
  - Missing values
  - Noisy Data
- Data integration
  - Data consolidation
  - Data propagation
  - Data virtualization
- Data reduction i.e. dimensionality reduction, feature selection
- Data transformation i.e. feature extraction

Data reduction and Data Transformation are the steps of feature engineering but these two techniques of pre-processing and feature engineering intersect with each other and are thus considered as the same step.

The next step in the pipeline formulation is selecting an ML model that needs to be trained on the pre-processed dataset. Selecting the perfect model is also necessary as there are different models that serve the same purpose i.e., classification can be done through Random Forest, Decision tree, and xgBoost algorithms but a specific model can be selected for a specific type of application with a literature study.

In pipeline formation, another thing that needs to be considered is hyper-parameter tuning. Hyper-parameter tuning is to tune parametric values such as learning rate, batch size, and the number of clusters. For example, in the case of the random forest algorithm, a few important parameters to be tuned are maximum depth, the minimum number of leaves, the number of estimators, and minimum number of sample splits, etc. In the case of deep learning models, hyper-parameters can be the number of iterations, learning rate, momentum, and optimizer. There are ranges of values that can be set in hyper-parameters and optimal values can help improve the model significantly.

The final step is evaluating the model, evaluation step tells us how our model performed and applies remedies in case of poor performance. Evaluation methods can tell us where our model is performing poorly i.e., the confusion matrix can tell us that for which class our model is performing poorly (if that is the case).

In petroleum data pre-processing is a need rather than a choice. There are several reasons for this statement. As it can be seen from Figure 1.1 there are several faults in the data. This is the same well, inclination vs measured depth plot. Red is from the time-based log, blue is from depth based log. There are many things wrong here:

- Even though identical variables have been plotted, a shift by 28m was given to the depth for the wells to match.
- 1 – the area where depth-based plot is just missing data
- 2 – depth-based data exists, but at a low frequency
- 3 – time-based data is noisy (can see two distinct paths). Additionally, depth stops to match
- 4 – discontinuity on time-based data

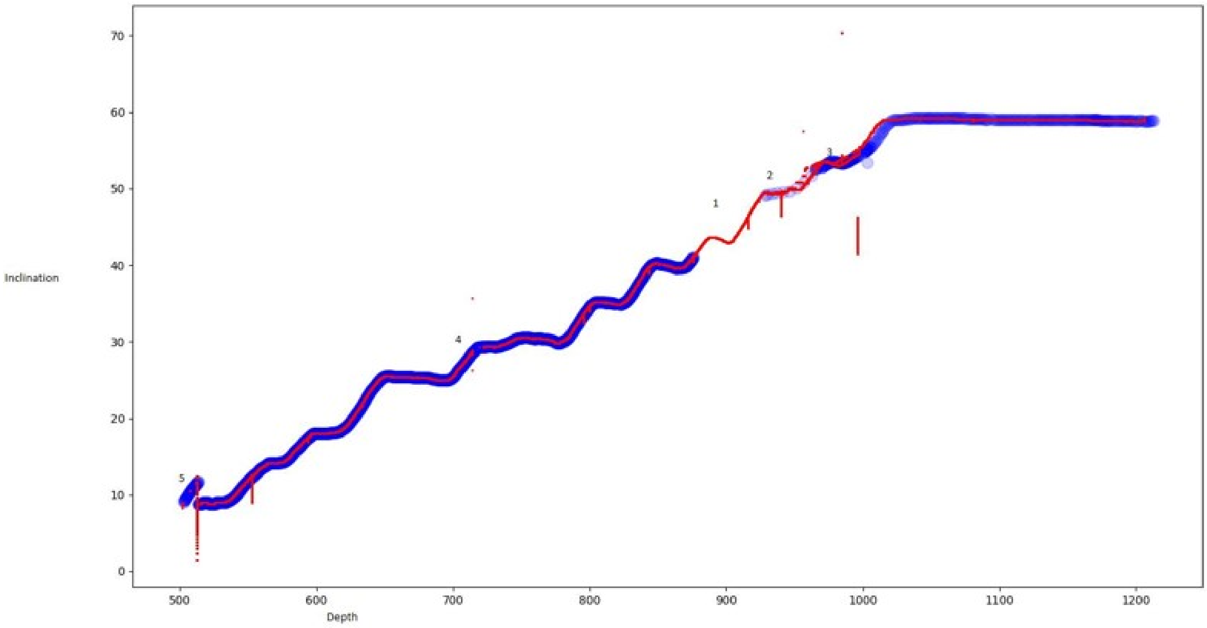

Figure 1.1: Data faults and depth v/s inclination graph

- 5 – artifacts at the start of the well, both datasets.
- Time-based data has “stalactites” hanging from the well path that needs to be filtered out (only thing easy to automate, short median filter)

For using this petroleum data in a time series prediction machine learning pipeline this data needs to be cleaned and processed.

1.2 Literature Review

1.2.1 Missing Data Imputation

Zhu et al. [4] considered data pre-processing a crucial task before model training and gave a review of different steps in data pre-processing. In this review, missing data imputation was also covered. These are the methods stated in the review for missing value imputation:

- Mean substitution
- Hot-deck substitution
- Regression substitution

- Conditional distribution-based substitution
- Multiple imputation

Zhang et al. [5] give an analysis that existing methods for replacing missing data entries use some deterministic or random imputation methods. Keeping this in mind they proposed clustering-based random imputation (CRI), which makes clusters of values having missing data and values that are complete. Then impute values on basis of the cluster which is near to missing data point. Their method proved effective in missing values imputation tasks.

Somasundaram and Nedunchezhian [6] stated that in real-world datasets missing entries, noise and inconsistencies are a common occurrence and there is a need to correct these problems. They applied three different methods of data imputations (Constant substitution, Mean attribute value substitution, and Random attribute value substitution). They compared the performance of each method using clustering methods. The data set used by them for evaluation was Wisconsin diagnostic breast cancer (WDBC) dataset [**?**].

Huang et al. [7] predicted the cost of an unseen project on basis of existing projects. They analyzed that feature selection affected their model performance significantly. To improve their pre-processing stage of data they applied a three-stage technique based on data imputation, data normalization, and feature selection. Their experiments showed a significant increase in accuracy when Z-Score normalization, kNN imputation, and mutual information-based feature weighting were used.

Xu et al. [8] observed that the existing method of value imputation shows poor performance when the number of missing values increases. They proposed a missing value imputation algorithm based on the evidence chain (MIAEC), that estimates missing data by first mining relevant evidence of missing data and then combining them to build an evidence chain. To further speed up the process map-reduce method was used, not only this the algorithm was made to run on distributed systems. Their method showed elevated performance than existing methods based on naive Bayes and mode imputation methods.

Nelwamondo and Marwala [9] gave a review of missing data imputation methods based on computational intelligence techniques. They highlighted that in literature there is a gap in the use of computational intelligence methods for missing data imputation. The researchers have mostly used easy methods because computational intelligence methods are complex. They highlighted three patterns of missing data that are, univariate pattern, monotone pattern, and arbitrary pattern. Every missing data pattern requires a different technique for data imputation.

Rja [10] and Rubin [11] showed three types of missing data patterns that affect the choice of the method. These data patterns are missing at random (missing data is unrelated to other missing variables but related to some observable variables), missing completely at random (missing data is unrelated to both other missing and observable variables), and missing not at random (missing data is related to other missing values).

Rimal [12] worked on proving that missing data size matters when applying the data imputation method. In this research, the R programming language was used to carry out the simulations. This research also highlights that it is not possible to find a method of data imputation before analyzing the dataset first.

An extensive study was done by Liu and Hauswirth [13]. They studies 118 missing data imputation methods and highlighted 9 influential factors and 12 selection criteria. They proposed a provenance meta-learning method to select the proper imputation method from the methods they chose.

From this literature, it is quite clear that there are many missing data imputation methods and the selection of a specific method depends highly on missing data patterns.

1.2.2 Volve Dataset

The Volve dataset is quite new and there have been only a few works present in the literature. This study is based on a series of works done by Tunkiel et al. [14].

Tunkiel et al. [15] identified that the existing research that claims to have achieved an R2 value of as high as 0.996 for the rate of penetration prediction has no independent datasets that can verify these claims. They worked on providing a benchmark dataset based on Equinor's public Volve dataset. They gathered data from seven wells with nearly 200,000 samples with 12 common attributes. This data can work as the benchmark to test existing studies and verify their claims. Also, this data can work as a base for many new research studies.

Tunkiel et al. [16] highlighted the problem of the machine learning model becoming obsolete due to equipment changes on the drilling site. The change happens due to the fast-changing pace of the drilling industry. To solve this problem, they proposed a training while drilling approach. The model is deployed on already working well where data is continuously coming from the sensors. They used recurrent neural networks (RNNs) to capture the dynamic and ever-changing nature of the data. This type of model learns from new features while keeping in mind the patterns learned from previous data.

Tunkiel et al. [17] noticed that the data-driven machine learning models are mostly black boxes and there is no way to find why a model is behaving erratically. TO uncover this mystery sensitivity analysis of data is necessary. Sensitivity analysis of data can help uncover these erratic behaviors which might be caused by overfitting. They used the approach of the one-at-a-time method to cover the hyperspace of potential inputs.

Tunkiel et al. [18] stated that the drilling data is generated continuously from various sensors and this data is huge. But there are several reasons because of which the data gets lost and there are many sensors i.e., gamma and inclination that lags by many meters. They found a solution to this by proposing a novel approach that provides a prediction for lagging data. They combined a trend-based prediction model with traditional artificial intelligence models to predict the lagging data.

Tunkiel et al. [19] worked on exploring Equinor's Volve dataset as the dataset is large and needs pre-processing before performing analysis. The main objective of this study was to overcome the basic obstacles of dealing with this data and to help the new studies happening in the field of oil drilling.

Based on these studies, this study aims to make changes to existing pipelines and apply hyper-parameter tuning to improve the already existing results.

1.3 Motivation and Problem Statement

On basis of the fact that petroleum data needs to be cleaned before processing, there is a need for a pipeline that pre-processes the data before feeding it to the machine learning model. This pipeline also needs to be tuned using hyper-parameter optimization to get the most efficient results out of it. To evaluate the results of hyper-parameter optimization, Akaike Information

Criterion (AIC) and Bayesian Information Criterion have been used. Not only pre-processing but model selection is also required in pipeline optimization. For this purpose, multiple time series prediction models have been considered and compared.

1.4 Objective

Objectives of this thesis include:

1. Building ML pipeline for time series analysis of petroleum data.
2. Designing pre-processing pipeline and optimizing it via AIC and BIC.
3. Machine learning model selection and optimization via AIC and BIC.

1.5 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 presents the mathematical background of AIC and BIC. It will also give basic ideas behind the steps of pre-processing pipeline and introduces the ML models for time-series prediction. All important definitions are given in this section. Chapter 3 discusses the Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM). This section will also explain the flow diagram of the ML pipeline. Chapter 4 discusses the dataset and simulation environment, and results. It also discusses the outcomes of the results. Chapter 5 concludes the thesis and states future recommendations.

Chapter 2

Background

2.1 Data Analysis

Data Pre-processing is to transform the raw data into a useful and efficient format. Steps that involve data pre-processing are:

2.1.1 Data cleaning

Data has many irrelevant and missing parts, that need to be dealt with. This included missing data handling and noise removal.

Missing data

This situation arises when some of the data is missing. There are several reasons that can cause the data to go missing i.e., faulty sensors. Missing data can be dealt with in 2 ways, either remove the missing data entries or even columns or fill the missing data with entries. To fill missing data there are several methods that include: filling entries with mean or most probable value.

Noisy Data

The data that can't be interpreted by the machine and is useless is called noisy data. A real-world dataset is affected by several components and one of them is noise [20]. It is an unavoidable problem and affects the data performance of machine learning models. There are two main causes of noise in data [21]:

- Implicit error (fault of measurement tools).
- Random error (human error by data collectors).

In petroleum datasets, the noise that is present in the data is implicit noise and it is caused by the sensors. It can be handled in several ways that are:

Binning: This method works on data that is sorted. This method smoothes that data by dividing the data into several segments and then replacing values in each segment by mean or on basis of boundary values.

Regression: In this method data is made smooth by fitting it into a regression model. Both linear (1 independent variable) and multiple (multiple independent variables) regression can be used.

Clustering: In this method similar data points fall into the same cluster and outliers are removed.

2.1.2 Data Transformation

This step transforms the data and converts it into a suitable form for machine learning models. There are several data transformation methods that include:

Normalization: This method scales the data into a defined range i.e., between 0 and 1, or between -1 and 1.

Attribute Selection: This method either selects the attributes from within given attributes or constructs new attributes on basis of existing ones.

2.2 Data Imputation

Missing data occurs in many statistical analyses. Missing data occurs due to implicit errors where the sensors are unable to provide any value for an entry. There is no concrete method that can help in imputing the missing values and the user needs to select from some existing methods to achieve this task. These existing methods apply imputation with various methods and these methods can sometimes be useful and sometimes that is not helping at all. Selecting the appropriate methods requires struggle and time.

There are several imputation methods that can help in imputing missing values in the dataset. There are two basic imputation methods:

- **Single Imputation:** In the single imputation method, each value is estimated separately. This method is simple as each value is calculated separately. This method includes imputing missing values with mean or regression. This imputation method is biased
- **Multiple Imputation:** This method imputes values simultaneously and then calculates errors to optimize the prediction. Sparse matrix completion methods like soft-impute is an example of multiple-imputation. This method is unbiased.

2.3 Data Resampling

Data sampling is the process of selecting observations at the time of data collection. While data resampling is to improve the already collected data. Resampling has many advantages in time series machine learning analysis. The basic reason is that time series prediction models require data to be evenly sampled. Thus calculating the sampling rate in time series data is a challenge and requires tuning.

2.4 Feature Selection

Feature engineering helps in extracting features from data. It is a crucial task as it affects the model accuracy directly. Plus there are many methods of feature engineering and selecting the method according to input data is a difficult task. Feature engineering is also important because it directly affects the accuracy of the ML model. If features extracted from the dataset read the pattern correctly then the model will give good performance. If the features extracted do not show any specific pattern then the performance will be poor. Feature selection also falls under the domain of feature engineering. If the dataset is large it causes the model to take more time and resources and may be due to some poor features model still doesn't show good performance. The pain point is to select which variables to use and which to skip, there are methods to determine that i.e. Z-score or R^2, and these are used to apply elimination one variable at a time. This variable selection method is tiresome and consumes a lot of time. In feature engineering feature selection is an important task from a high dimensional space of extracted features. The improvement can be done so the user doesn't have to look at the value and remove variables manually. Plus these techniques are applied when the model has been trained successfully and this causes the use of extra resources and time.

2.4.1 AIC & BIC

AIC and BIC are methods of selecting models by scoring them on the basis of their log-likelihood and complexity. AIC and BIC are used to compare statistical models on the basis of the number of free parameters and their values. It takes into account how much a variable contributes to solving a problem. For example, if we have a regression model which is being tested on the basis of four variables A1, A2, A3, and A4, AIC and BIC fit the regression model with different combinations of these variables and calculate the value using the formula:

$$AIC = -2(log\text{-}likelihood) + 2K \tag{2.1}$$

Here, K is the number of parameters used to fit the model, and log-likelihood checks if the model is a good fit or not.

$$BIC = -2(log\text{-}likelihood) + Klog(n) \tag{2.2}$$

Here, n is the number of values in the dataset (sample size), and k is the number of parameters.

The number of combinations for variables can be calculated by $2^N - 1$. Here N is the number of total variables. Combination of these values are shown in table 2.1.

As the number of variables is 4 so number of combinations will be $2^4 - 1 = 15$.

For each combination regression model is trained and AIC and BIC values are calculated using (2.1) and (2.2) respectively. A combination with the minimum value of AIC or BIC is

Table 2.1: Possible combination of 4 variables for applying AIC and BIC

A1	A2
A3	A4
A1, A2	A1, A3
A1, A4	A2, A3
A2, A4	A3, A4
A1, A2, A3	A1, A2, A4
A2, A3, A4	A1, A3, A4
A1, A2, A3, A4	

selected as the best model.

When comparing the two models BIC puts a higher penalty term than AIC.

2.5 Time Series Models

2.5.1 Time Series Analysis

A specific way of analyzing a sequence of data over time is called time series analysis. In time-series data, data points are at a constant interval in time rather than at random points. Time-series data analysis can be done to see a change in data over time and thus predict future values. Data trends and seasonal changes are also part of the data to see the trends. The time series dataset needs to be extensive in order to ensure the consistency and reliability of the prediction.

Time series datasets are used to understand the trends and patterns over time. Visualization of time series data shows seasonal trends and provides a deeper picture of data. For example, the sale of a specific product increases at a specific time of the year i.e., groceries at the start of each month. Some examples of time-series datasets are:

- Weather Data
- Stock prices data
- Brain Monitoring (EEG)

Time series models can not be generalized to all king of time-series datasets. However deep learning has shown promising results in generalizing the models that can work with most of the datasets. Some important deep learning time series analysis models are:

- Long Short Term Memory (LSTM)
- Recurrent Neural Networks (RNN)

2.5.2 Artificial Neural Networks

Artificial Neural Networks are the most basic type of deep learning model that can take tabular data and then classify it. It consists of neurons that try to mimic the behavior of the human brain. Each neuron is connected to other neurons to form a network. Each neuron has weights and biases, whose values change during the optimization of the model. A simple ANN is shown in **??**. It consists of fully connected layers where each neuron is connected to all the neurons in the next layer. At first, each value from independent variables is given as input. Weights and biases are selected at random at the start. On basis of initial values of weights and biases predictions are made. Then error is calculated on basis of predicted values and actual values. On basis of error, weights and biases are updated. This procedure repeats until convergence.

2

2.5.3 Long Short Term Memory

LSTM is the neural network that takes previous knowledge into account for future predictions. LSTM takes advantage of both long-term memory (LTM) and short-term memory (STM) to learn. It has two basic mechanisms:

- **Forgetting Mechanism:** Forgets all the information that is not relevant.
- **Saving Mechanism:** Saves the information that is relevant and can help in the future.

To carry out the above tasks LSTM takes advantage of gates. There are 4 kinds of gates:

- **Forget Gate:** LTM uses forget gate to forget unuseful information.
- **Learn gate** Event (current input) and STM are combined so that recently learned information can be applied to the current input.

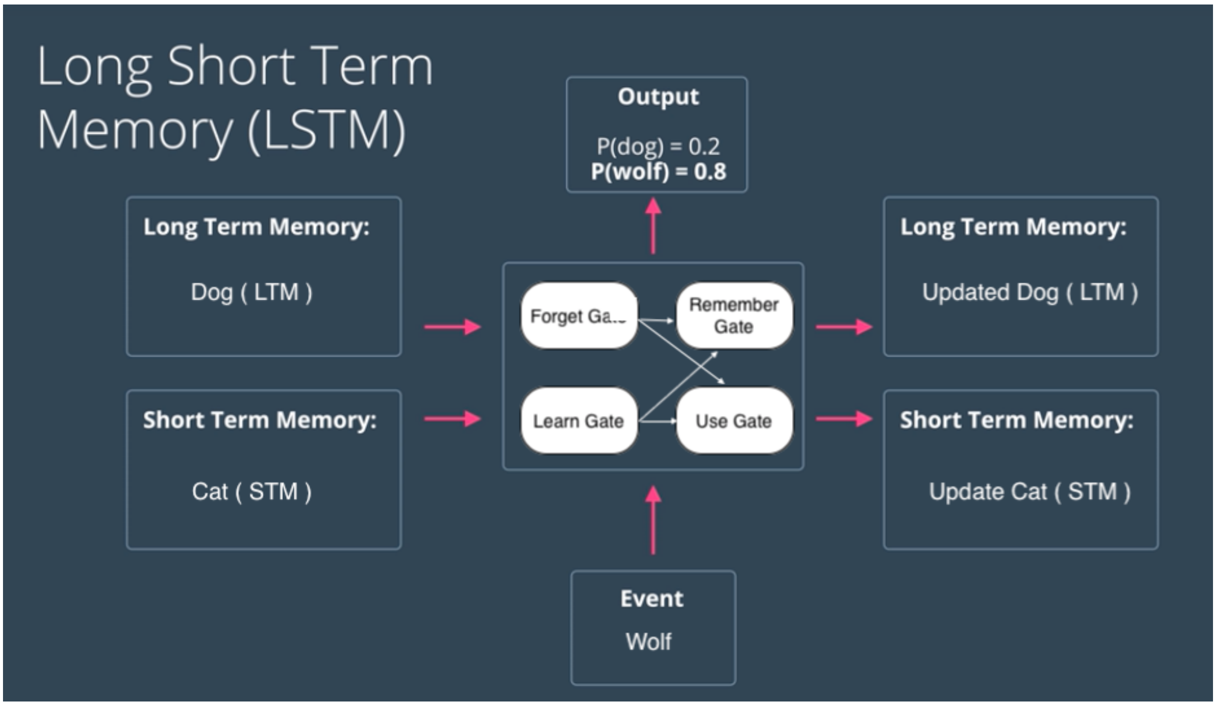

Figure 2.1: Operations of LSTM [1]

- **Remember gate:** LTM, STM and event are combined at the remembering gate to update the LTM.
- **Use gate:** LTM, STM, and event are combined to predict the current output and thus it updates STM.

Figure 2.1 shows the working of LTM and STM using the gates and how the values are updated. Figure 2.2 shows the basic architecture of LSTM.

This architecture can be divided for deep understanding. Figure 2.3 shows the architecture of each gate used in the LSTM model.

2.5.4 Recurrent Neural Networks

Artificial neural networks and convolution neural networks work best in the case of sequential image datasets respectively but fail for temporal data (dependency over time). Some examples of temporal datasets are speech datasets and stock price datasets etc.

A recurrent neural network (RNN) is similar to ANN with a small addition, which is it also takes previous input into account along with current information. The basic architecture of RNN is shown in Figure 2.4

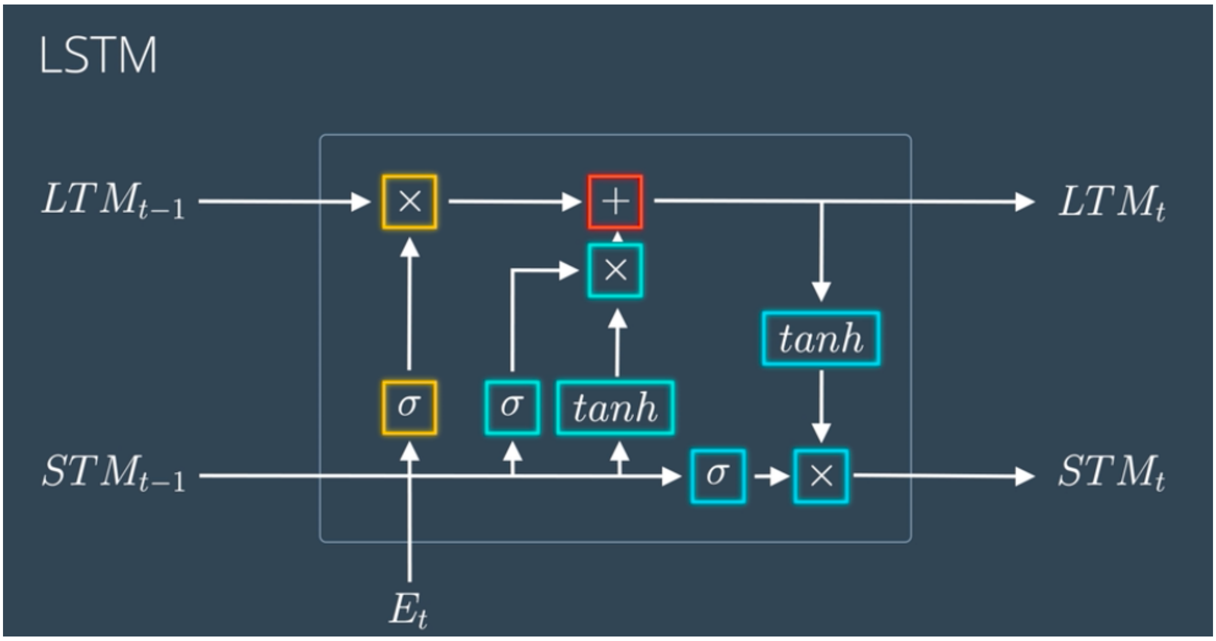

Figure 2.2: Architecture of LSTM [1]

2.5.5 Mean Absolute Error

Absolute error is the error in the measurements. Simply speaking it is the difference between the measured values and true values.

$$\delta x = |x_i - x| \tag{2.3}$$

An absolute sign is needed because sometimes the difference is negative.

Mean absolute error is the mean of all the absolute errors.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |x_i - x| \tag{2.4}$$

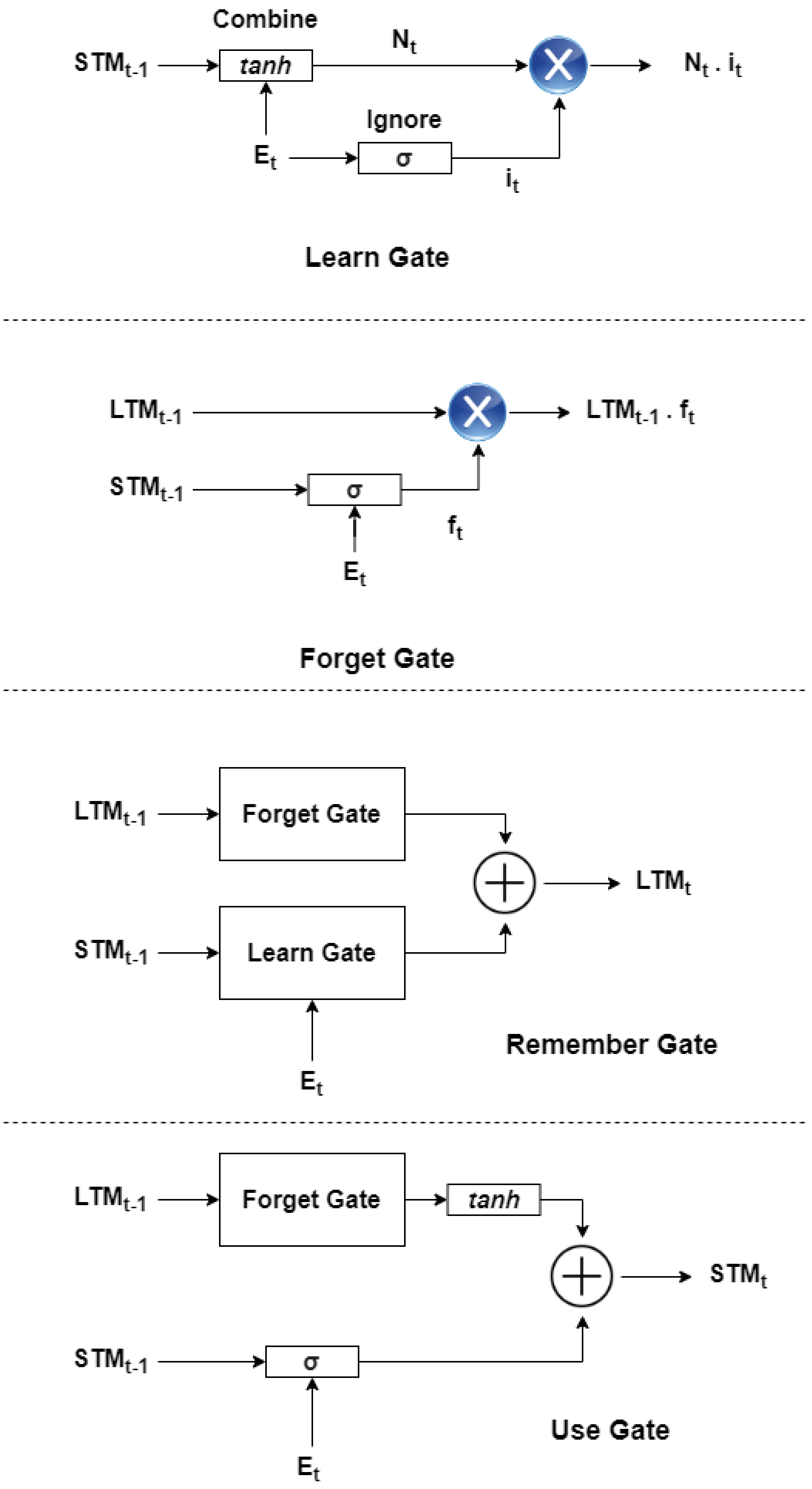

Figure 2.3: Gates in LSTM

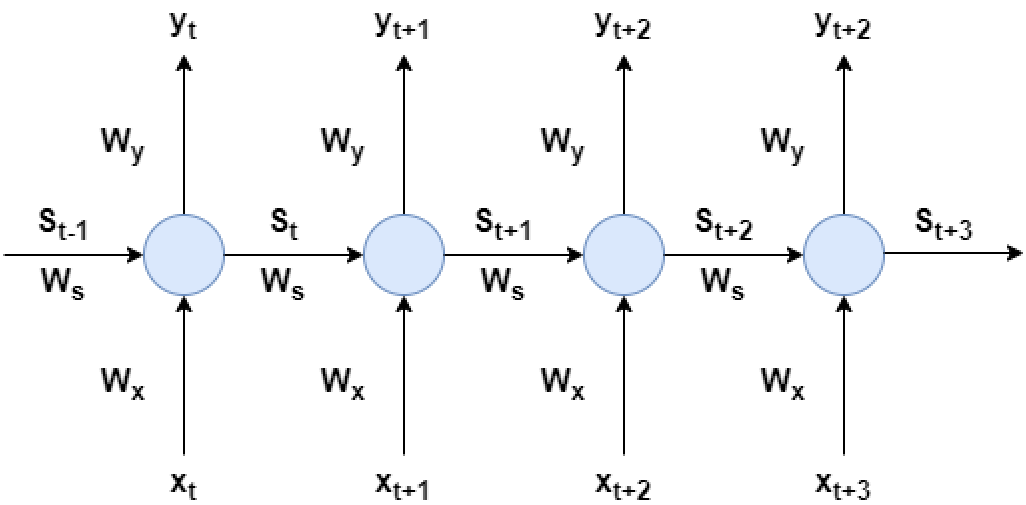

Figure 2.4: Architecture of RNN [2]

Chapter 3

Mathodology

3.1 Machine Learning Pipeline

Building a Machine Learning pipeline is the first step in solving a machine learning problem. Figure 3.1 shows the overall pipeline of the model.

As seen from the Figure 3.1, the first steps are to remove unwanted columns i.e., the indexing column. This also includes the column that has a high correlation with the other independent columns for example ROP and inverse ROP as inversely highly correlated.

The second step is to analyze the data and run the gap analysis on basis of the number of gaps and length of each gap [14]. There are total of 4 scenarios:

1. Few gaps + low overall percentage
    - sensor obstruction
    - temporary sensor failure.
2. Few gaps + high overall percentage
    - equipment setup change
    - permanent sensor failure
    - data corruption
3. Multiple gaps + low overall percentage

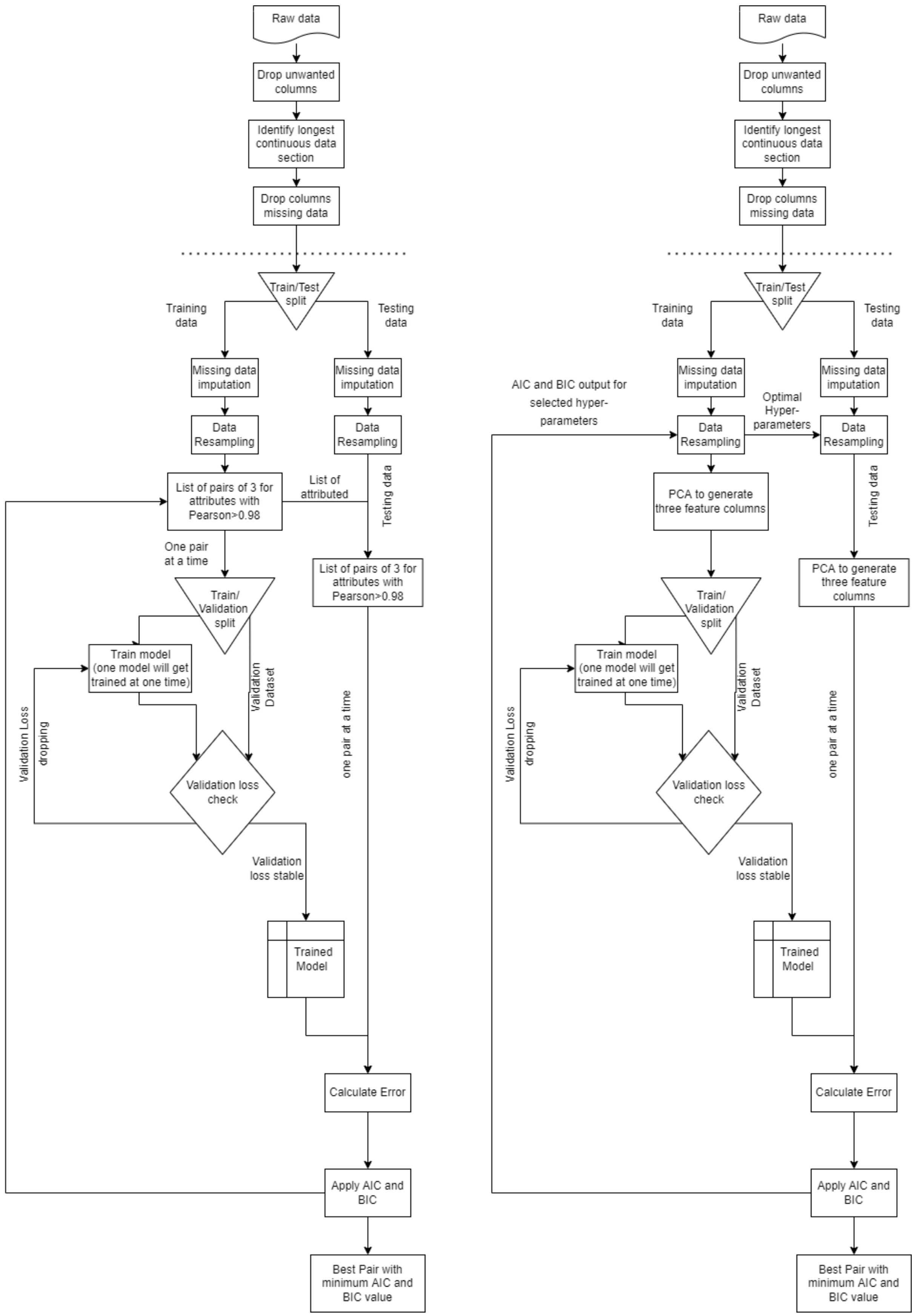

Figure 3.1: Flowchart for feature selection (left) and Data Resampling hyper-parameter tuning (right) via AIC and BIC

- uneven polling frequency
- sequential sensor use (MWD)

4. Multiple gaps + high overall percentage

- sensors with low polling frequency
- data received occasionally

If the gap sizes are too large then it means that data is actually missing and filling that data is of no use and it is best to remove those columns. To find the columns that need to be removed as they can not be filled, a gap coefficient was used [14]. The gap coefficient is given by:

$$GC_i = \frac{i}{GQ_i \cdot TL} \tag{3.1}$$

where GC is the gap coefficient, i is gap length, GQ_i is the number of gaps of length i, and TL is the total length. The gap coefficient for each attribute is calculated and if GC surpasses a threshold value then it means that those gaps can not be filled and the attributes need to be removed.

After that data is split into training data and testing data. Testing data is the one that needs to be used for final testing after the model has been trained.

3.1.1 Missing Data Imputation

The next step in data pre-processing is to impute the missing values. There are several missing data imputation methods but the K-nearest neighbors (KNN) data imputation method has proven to be generally effective. In the KNN data imputation method, k nearest neighbors are identified on basis of Euclidean distance for a specific missing value, and the mean value of the neighbors is selected to be imputed at that particular missing point. Euclidean distance is calculated by Equation 3.2. For data imputation, KNN imputer with 3 nearest neighbors has been used with Euclidean distance.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{3.2}$$

3.1.2 Data Resampling

Data Resampling is an important step as i=time series analysis models require evenly sampled data. Two types of resampling methods have been used and tuned. The methods are:

- Radius Neighbors Regressor
- KNN Regressor

For hyper-parameter optimization of data resampling methods following parameters were considered:

1. Resampling Weights
   - **Uniform:** All points in the neighbourhood are given equal weights.
   - **Distance:** All points are given weights on basis of the distance from the point under consideration. Closer points have more weight then points that are far.
2. Algorithm to compute NN
   - Ball Tree Algorithm
   - KD Tree Algorithm
3. Distance Metric
   - **1:** Manhattan Distance (Equation 3.3)
   - **2:** Euclidean Distance (Equation 3.2)
   - **3:** Minkowski Distance (Equation 3.4)

$$d(p,q) = \sum_{i=1}^{n} |q_i - p_i| \tag{3.3}$$

$$d(p,q) = \sqrt[r]{\sum_{i=1}^{n} (q_i - p_i)^r} \qquad where,\ r > 0 \tag{3.4}$$

3.1.3 Feature Selection

Feature selection is an important step of the pipeline as there are features that support the machine learning model and there are features that degrade the efficiency and accuracy of the model. Feature selection is the process of selecting the relevant features or generating new features for reducing the dimension of the data and fitting the model appropriately. There are several feature selection methods, some of them are:

Principal Component Analysis (PCA)

The principal component analysis is a statistical procedure, that summarizes the high dimensional data into lower dimensional data using the principal components of the data points.

PCA finds the lines, planes, and hyper-plane in k-dimensional space and approximates the data on basis of least square approximation. This is done by making the variance of the coordinates as large as possible on the line/plane.

The steps involvind the PCA are:

1. Suppose a matrix X with N rows and K columns. Plot each point on K-dimensional space after scaling to unit variance.

2. Subtract variable averages from the data to get a mean-centering vector, that represents a point in the data.

3. Origin is shifted to the mean-centering point.

4. First principal component (PC1) is found by fitting a line passing through the mean-centering point and is the best fit on basis of the least square. Each point can be projected onto this line to get a new value.

5. Second principal component (PC2) is found by passing a line perpendicular to the first component. This represents the second-largest variation in the data.

PC1 and PC2, combined together represent the whole data. For hyper-parameter tuning of the resampling step, PCA has been used to generate new features.

Pearson Correlation

Pearson correlation is the method that finds correlation between independent variables and dependent variables of the data by assigning the value between -1 and 1. Where, -1 means total negative correlation, 0 means no correlation and 1 means total positive correlation. Pearson correlation can be found by:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{3.5}$$

Pearson correlation can help in removing the independent variables that have a high correlation between them and can also help in identifying the columns that have a high correlation with the dependent variable.

AIC and BIC can be combined with Pearson correlation to find the best features. Top features with high correlation can be used to further reduce the number of training features by using AIC and BIC. After applying Pearson correlation, features are combined in pairs of 3, and then after running the model on a single combination, AIC and BIC value is calculated. Feature pairs with a minimum value of AIC and BIC are the best pair and can be used to get the best results. For the selection of best variables, Pearson with AIC and BIC has been used.

Time Series Prediction

Different models has been combined for generating the models that has been used for time series prediction. Time series prediction model has been combined with mylti-layer perceptron (MLP) model. Where, MLP is the fully connected feed forward, artificial neural network. The combinations are as follow:

1. LSTM + MLP (Figure 3.2)

2. RNN + MLP (Figure 3.3)

3. LSTM + RNN + MLP (Figure 3.4)

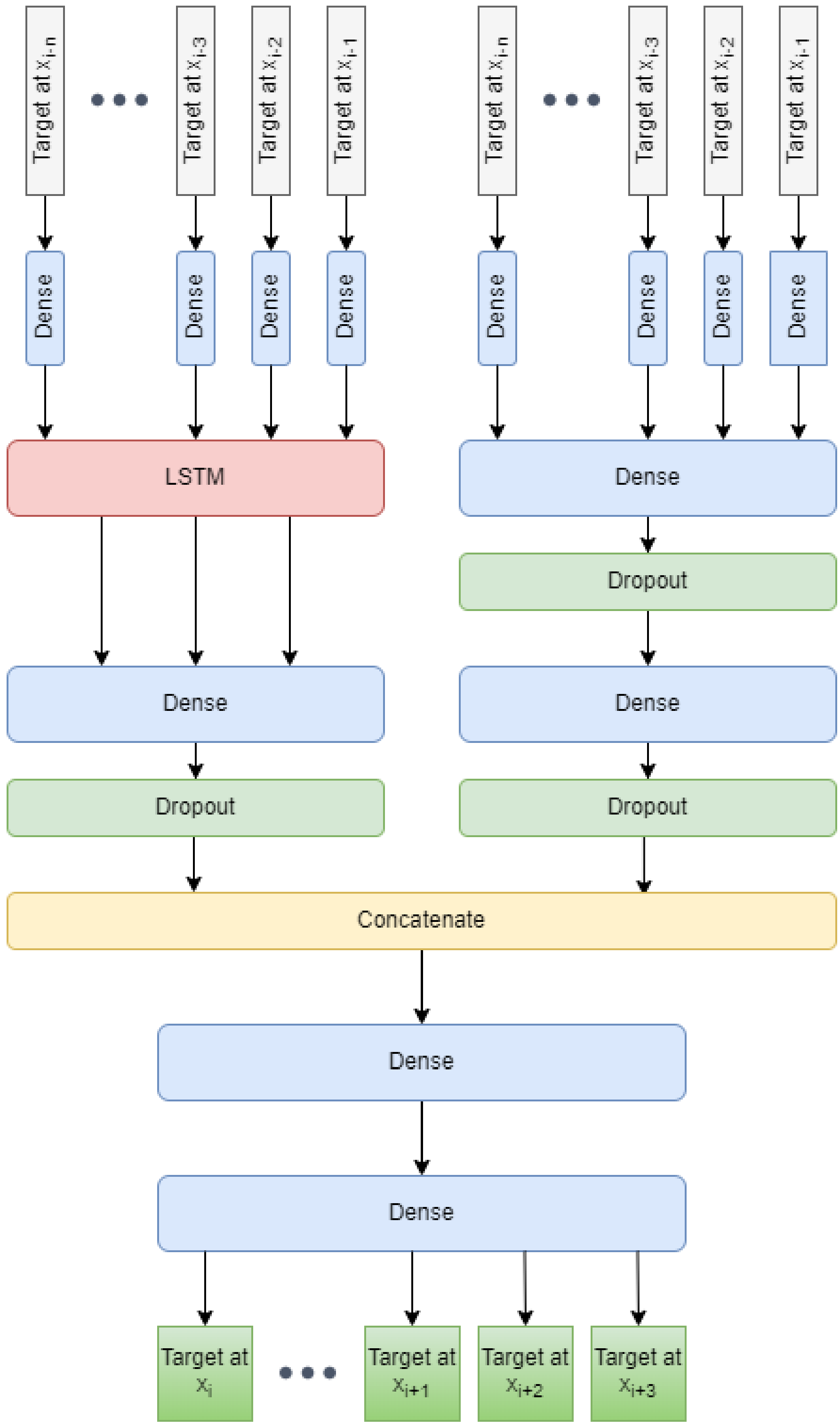

Figure 3.2: Model architecture of LSTM + MLP

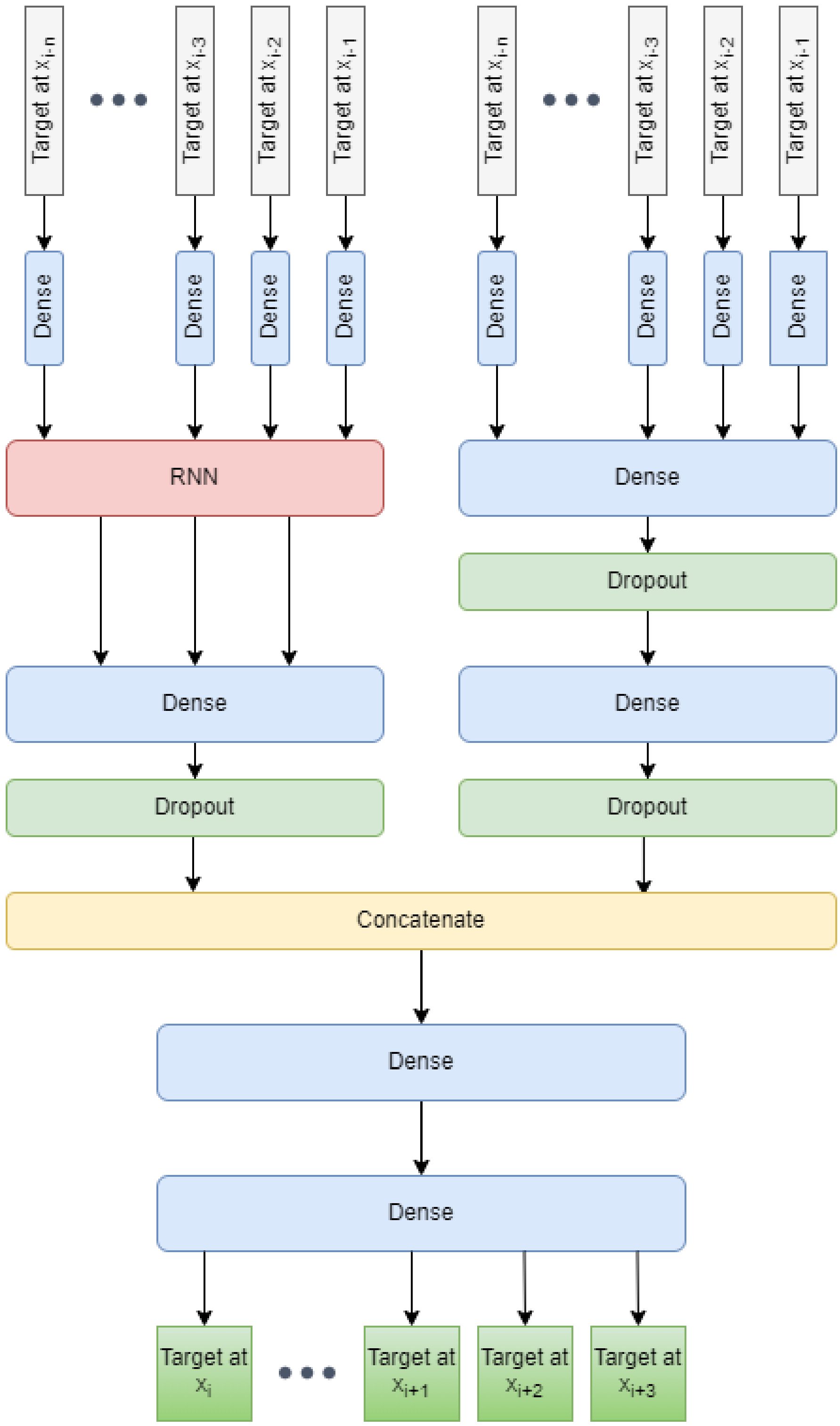

Figure 3.3: Model architecture of RNN + MLP

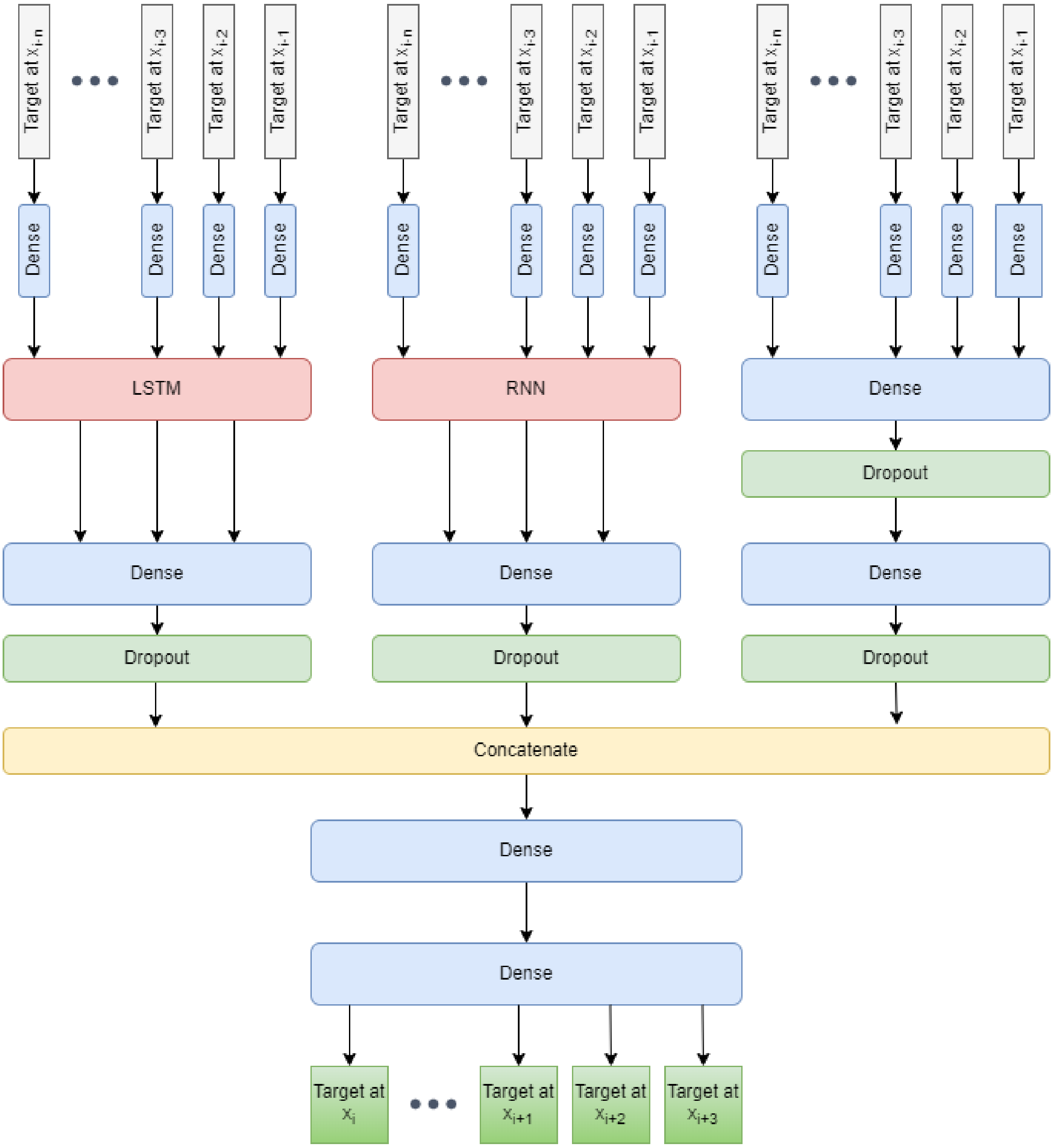

Figure 3.4: Model architecture of LSTM + RNN + MLP

Chapter 4

Results and Discussion

4.1 Dataset

The Volve dataset Equinor [1] was published in 2018. The fossils here are sandstone of the middle Jurassic age in the Hugin Formation. The depth of the Volve field is 2700m and 3100m where the seabed ends at 80m. The field was completely shut down by 2018 and the dataset was released to enhance the research in the oil and gas industry. During the whole operation, 56000 barrels per day were extracted from the oil field on average. The data that has been released has 40000 files of various kinds. The dataset has 75% and 80% data gaps or empty cells

4.2 Simulation Environment

All the simulations took place on Google Colab Jupyter notebooks with 12.75GB RAM and Tesla P100 16GB and Tesla P4 GPUs.

AT first KNN imputer was applied to the three nearest neighbors. This step does not need validation as it is a necessary step. Although it can be compared with other imputation models.

After that, the hyper-parameter tuning of the resampling step was done where two methods, KNN regressor, and Radius Neighbour regressor were compared on basis of their parameter values. For this stage, the feature selection step was not used and simply PCA was implemented. After applying PCA, the model used for the time series prediction was LSTM. AIC and BIC

[1] https://data.equinor.com/dataset/Volve

have been calculated after fitting the model for each combination of the resampling method and its hyper-parameters.

During the feature selection first, the Pearson correlation was applied. From Pearson correlation, parameters with more than 98% correlation with the output parameter were selected and all possible combinations of 3 parameters were made from them. These combinations of the parameters were used to train the model one by one. AIC and BIC values were calculated after each run and a combination with the minimum value of AIC and BIC was selected as the best. During feature selection resampling method was used which was best in the previous stage, and the model used was LSTM + MLP.

The last step is model selection, in this step different models were tested for the best resampling method and best parameters, In the end, the best model was selected on basis of mean absolute error.

4.3 KNN-Imputation

In KNN imputation the nearest neighbors are selected as 3. Each row is imputer separately on basis of the distance from the 3 nearest neighbors. Figure 4.1 shows the plots of some parameters before and after the KNN imputation.

As it can be seen from the Figure 4.1, the graphs are quite predictable when seen, and after imputation, the graphs are as expected.

4.4 Resampling Hyperparameter Tuning

As described in the previous section, two methods were compared for the resampling, and for each method, hyper-parameters has been tuned. Table 4.1 shows all the possible combinations that have been tested and their respective AIC, BIC, and mean absolute error value. As it can be seen from the table the minimum value has been obtained for radius neighbor regressor with uniform distance using ball tree algorithm and with Minkowski distance. For this combination, all the output evaluation matrices have minimum values.

Table 4.1: Results for resampling method selection and hyper-parameter tuning

Resampling Method	Weight	Algorithm	P4	AIC	BIC	MAE
radius	uniform	ball_tree	Manhatten	1393477.80	3716214.07	9.17
			Euclidean	1392074.95	2267638.43	7.57
			Minkowski	1392066.43	2267629.91	6.42
		kd_tree	Manhatten	1392075.74	2267639.22	7.68
			Euclidean	1392076.66	2267640.14	7.82
			Minkowski	1392076.05	2267639.53	7.73
	distance	ball_tree	Manhatten	1392077.07	2267640.54	7.88
			Euclidean	1392077.79	2267641.27	7.99
			Minkowski	1392079.65	2267643.13	8.28
		kd_tree	Manhatten	1392079.22	2267642.70	8.22
			Euclidean	1392078.76	2267642.24	8.14
			Minkowski	1392077.07	2267640.55	7.88
knn	uniform	ball_tree	Manhatten	1392068.59	2267632.07	6.69
			Euclidean	1392080.39	2267643.87	8.40
			Minkowski	1392073.25	2267636.73	7.32
		kd_tree	Manhatten	1392077.83	2267641.31	8.00
			Euclidean	1392076.82	2267640.30	7.84
			Minkowski	1392077.79	2267641.27	7.99
	distance	ball_tree	Manhatten	1392077.77	2267641.25	7.99
			Euclidean	1392077.91	2267641.38	8.01
			Minkowski	1392075.77	2267639.25	7.68
		kd_tree	Manhatten	1392078.59	2267642.07	8.11
			Euclidean	1392078.09	2267641.57	8.03
			Minkowski	1392079.04	2267642.52	8.19

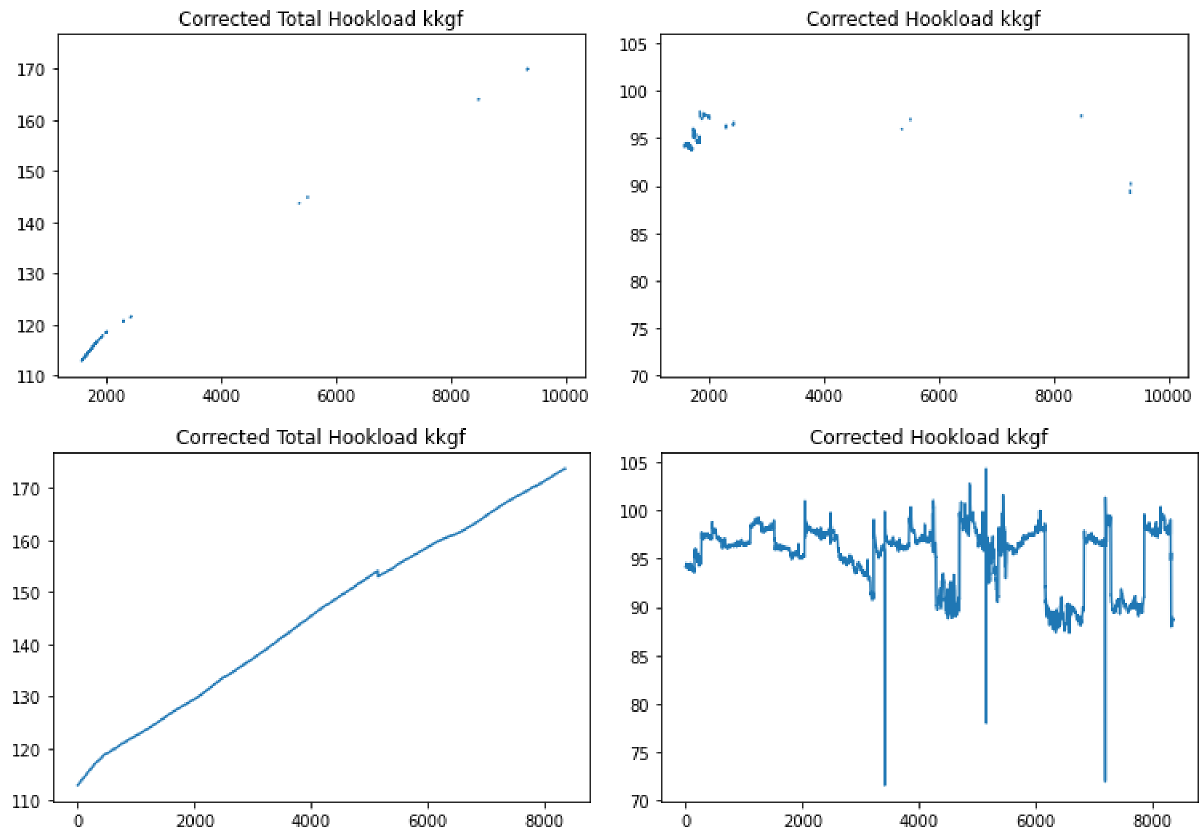

Figure 4.1: Graphs before (top) and after (bottom) KNN imputation

4.5 Feature Selection

AIC and BIC have been used for feature selection too. Here at first, the Pearson correlation matrix was found which is shown in Figure 4.2.

After Pearson correlation, parameters were extracted having a correlation value greater than 98% with the dependent variable as shown in Table 4.2.

From these parameters, all possible combinations of the three parameters have been made.

Table 4.2: Pearson Correlation of parameters with dependent variables

Parameter	Pearson Correlation
Measured Depth m	0.9813667349941119
Hole depth (MD) m	0.9813684472500676
Hole Depth (TVD) m	0.9826332567926808
Corrected Total Hookload kkgf	0.9827814199492585
Extrapolated Hole TVD m	0.9828458285022305
Bit Drill Time h	0.9854056047108368
RGX_RT unitless	0.9909452831190304
RHX_RT unitless	0.998696718883108

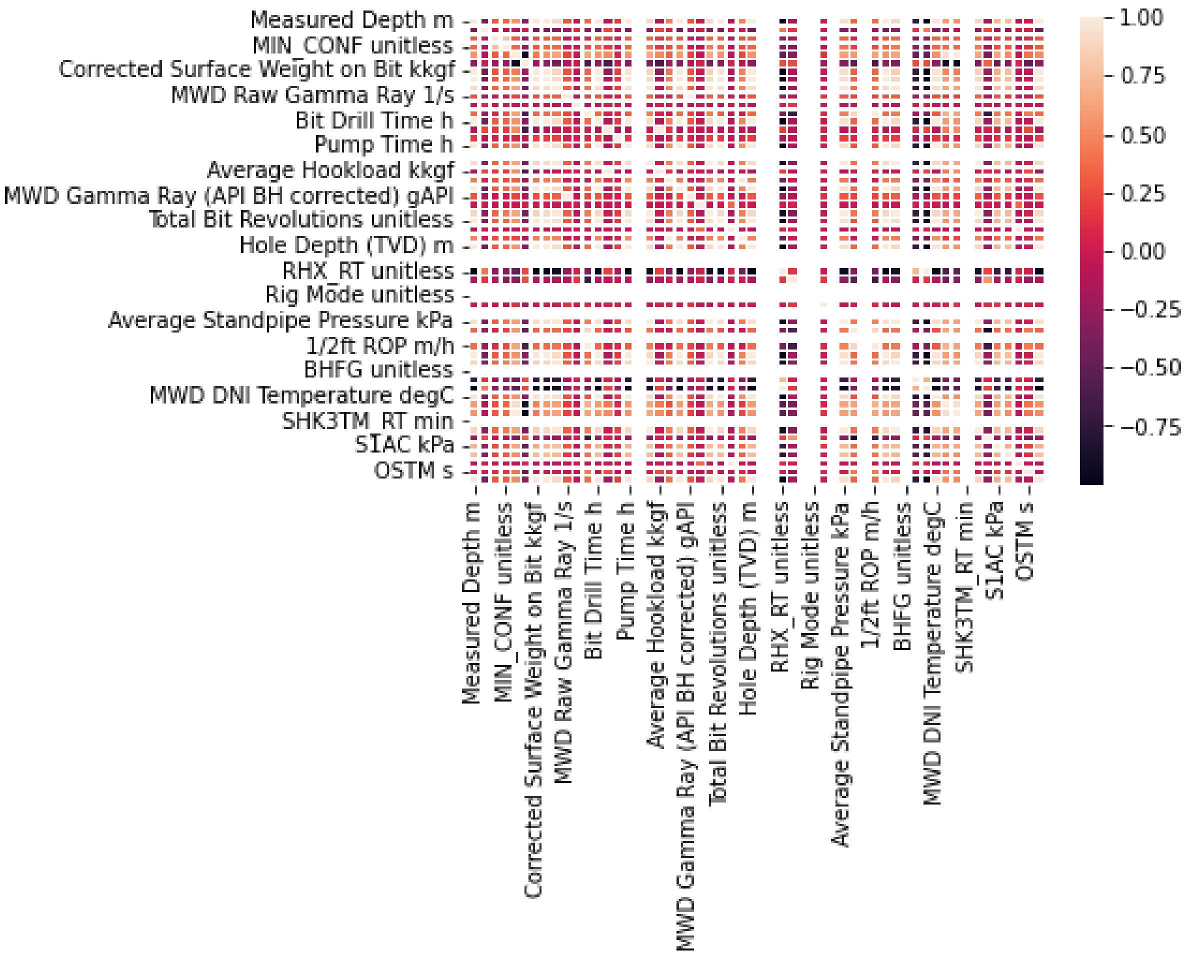

Figure 4.2: Heatmap of pearson correlation

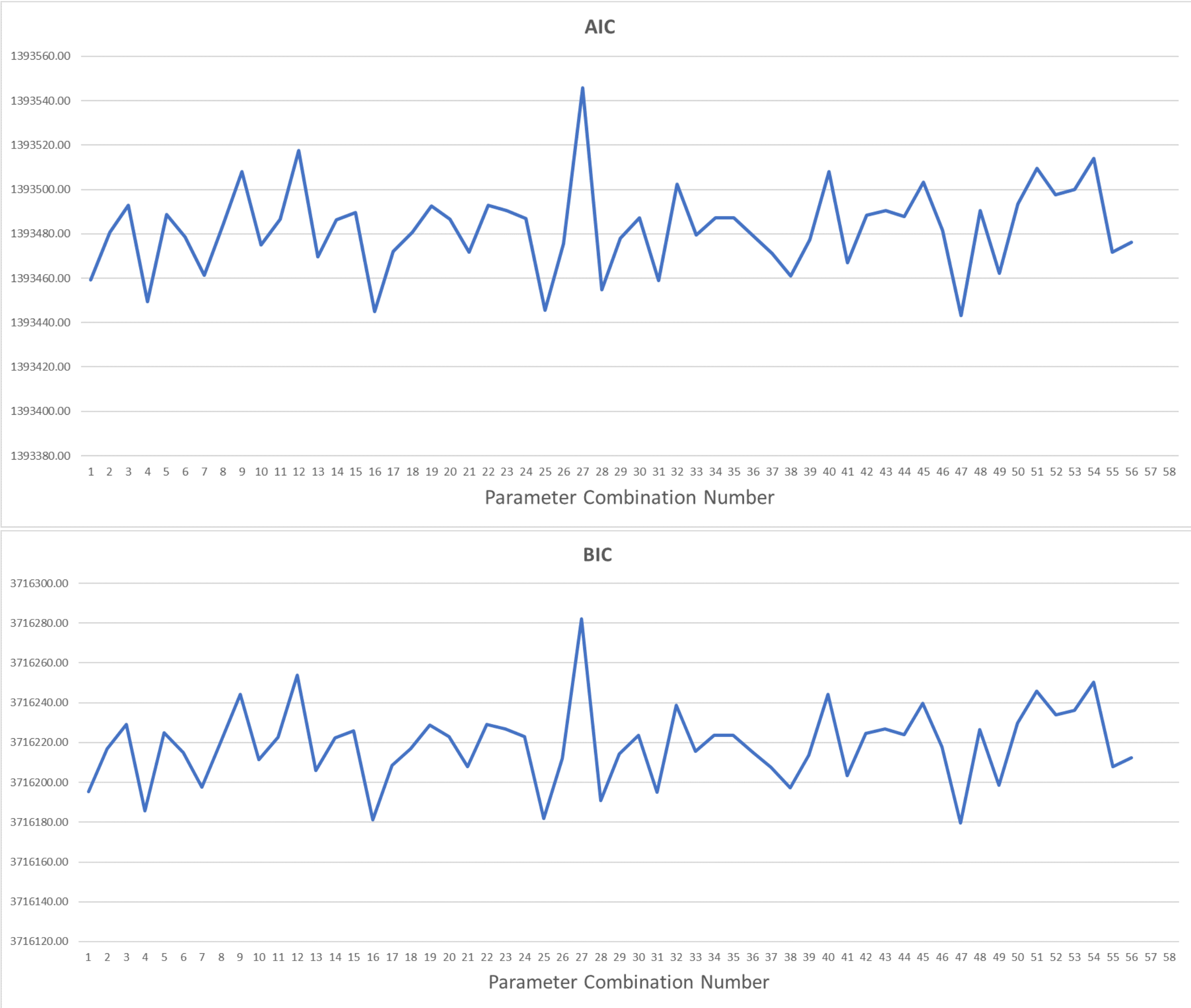

Figure 4.3: AIC (top) and BIC (bottom) graphs for different parameter combinations

All possible combinations can be seen in Appendix A. These parameters have been then used to train the model and AIC and BIC values have also been calculated for each combination. Figure 4.3 shows the graph of AIC and BIC values for each combination. From the graphs it is clear that the minimum AIC and BIC value has been achieved at combination number 46 which has the following parameters:

- Corrected Total Hookload kkgf
- Extrapolated Hole TVD m
- Bit Drill Time h

AIC and BIC value for this combination is 1393443.28 and 3716179.55 respectively.

Table 4.3: Mean Absolute Error after training different models.

Model	**MAE**
LSTM + MLP	8.8
RNN + MLP	8.71
LSTM + RNN + MLP	8.82

4.6 Model Selection

All the parameters have been tuned and fixed, models were selected from among the models that have been highlighted in the previous chapter. For each model mean absolute error (MAE) value has been noted and the model has a minimum MAE value that has been selected as the best. The results have been shown in Table 4.3.

For each model, the respective prediction curve can be seen in Figure 4.4. As it can be seen from the figure, RNN is showing the best results, the variance in the prediction line is less as compared to other models.

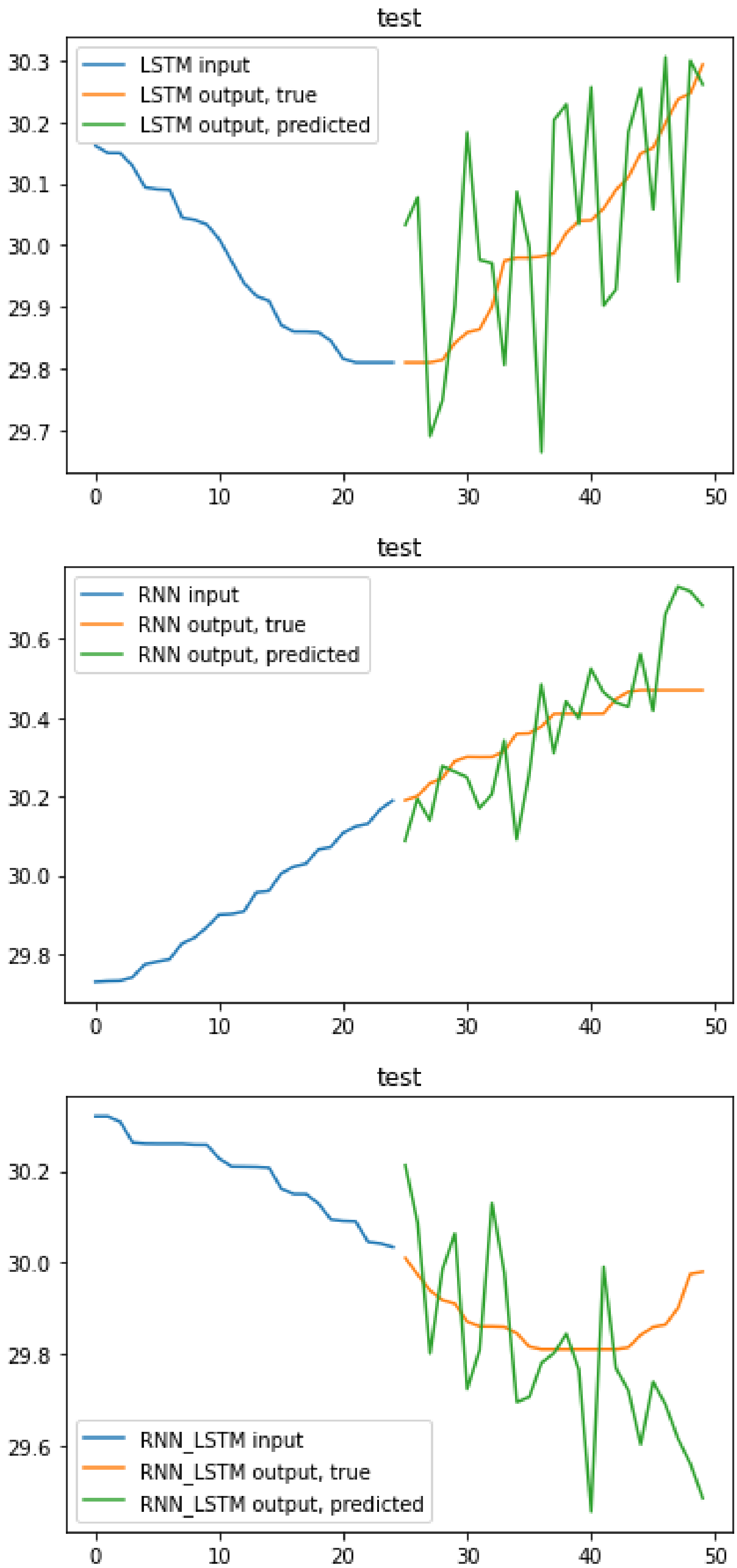

Figure 4.4: AIC (top) and BIC (bottom) graphs for different parameter combinations

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This research aims to build and improve an ML pipeline via hyper-parameter optimization. At first, an already existing pipeline has been studied and work has been done to improve it. This thesis aimed at improving missing data imputation, resampling, feature selection, and in the end model selection. Missing data imputation has been done by KNN imputer which imputed the missing values on basis of nearest neighbors. After that, the resampling method has been selected after tuning their hyper-parameters. For feature selection firstly Pearson correlation has been used to narrow down the top relevant features, after that all possible combinations of these features have been created as a group of three. AIC and BIC values have been used to select the most optimal set of parameters. After feature selection, three models were considered and compared on basis of mean absolute error (MAE). RNN + MLP surpasses all the other models on basis of MAE.

5.2 Future Work

In the future, these results can be further improved and tested by joining all the parameter selections as one. As in this research, parameter optimization for each step has been done separately.

References

[1] Gourav Singh. Understanding architecture of lstm, 2021.

[2] Gourav Singh. Recurrent neural networks for sequence learning, 2020.

[3] Sana Mushtaq. Data preprocessing in detail, 2019.

[4] Jinlin Zhu, Zhiqiang Ge, Zhihuan Song, and Furong Gao. Review and big data perspectives on robust data mining approaches for industrial process modeling with outliers and missing data. *Annual Reviews in Control*, 46:107–133, 2018.

[5] Chengqi Zhang, Yongsong Qin, Xiaofeng Zhu, Jilian Zhang, and Shichao Zhang. Clustering-based missing value imputation for data preprocessing. In *2006 4th IEEE International Conference on Industrial Informatics*, pages 1081–1086. IEEE, 2006.

[6] RS Somasundaram and R Nedunchezhian. Evaluation of three simple imputation methods for enhancing preprocessing of data with missing values. *International Journal of Computer Applications*, 21(10):14–19, 2011.

[7] Jianglin Huang, Yan-Fu Li, Jacky Wai Keung, Yuen Tak Yu, and WK Chan. An empirical analysis of three-stage data-preprocessing for analogy-based software effort estimation on the isbsg data. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 442–449. IEEE, 2017.

[8] Xiaolong Xu, Weizhi Chong, Shancang Li, Abdullahi Arabo, and Jianyu Xiao. Miaec: Missing data imputation based on the evidence chain. *IEEE Access*, 6:12983–12992, 2018.

[9] F Nelwamondo and Tshilidzi Marwala. Key issues on computational intelligence techniques for missing data imputation-a review. In *Proc. of world multi conf. on systemics, cybernetics and informatics*, volume 4, pages 35–40, 2008.

[10] Little Rja and Donald B Rubin. Statistical analysis with missing data. *New York*, 1987.

[11] Donald B Rubin. Multiple imputation for survey nonresponse, 1987.

[12] Yagyanath Rimal. Multivariate imputation for missing data handling a case study on small and large data sets. *International Journal of Human Computing Studies*, 2(1):5–11, 2020.

[13] Qian Liu and Manfred Hauswirth. A provenance meta learning framework for missing data handling methods selection. In *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0349–0358. IEEE, 2020.

[14] Andrzej T. Tunkiel, Dan Sui, and Tomasz Wiktorski. Impact of data pre-processing techniques on recurrent neural network performance in context of real-time drilling logs in an automated prediction framework. *Journal of Petroleum Science and Engineering*, 208:109760, 2022.

[15] Andrzej T. Tunkiel, Dan Sui, and Tomasz Wiktorski. Reference dataset for rate of penetration benchmarking. *Journal of Petroleum Science and Engineering*, 196:108069, 2021.

[16] Andrzej T. Tunkiel, Dan Sui, and Tomasz Wiktorski. Training-while-drilling approach to inclination prediction in directional drilling utilizing recurrent neural networks. *Journal of Petroleum Science and Engineering*, 196:108128, 2021.

[17] Andrzej T. Tunkiel, Dan Sui, and Tomasz Wiktorski. Data-driven sensitivity analysis of complex machine learning models: A case study of directional drilling. *Journal of Petroleum Science and Engineering*, 195:107630, 2020.

[18] Andrzej T Tunkiel, Tomasz Wiktorski, and Dan Sui. Continuous drilling sensor data reconstruction and prediction via recurrent neural networks. In *International Conference on Offshore Mechanics and Arctic Engineering*, volume 84317, page V001T01A002. American Society of Mechanical Engineers, 2020.

[19] Andrzej T Tunkiel, Tomasz Wiktorski, and Dan Sui. Drilling dataset exploration, processing and interpretation using volve field data. In *International Conference on Offshore Mechanics and Arctic Engineering*, volume 84430, page V011T11A076. American Society of Mechanical Engineers, 2020.

[20] Richard Y Wang, Veda C Storey, and Christopher P Firth. A framework for analysis of data quality research. *IEEE transactions on knowledge and data engineering*, 7(4):623–640, 1995.

[21] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study. *Artificial intelligence review*, 22(3):177–210, 2004.

Appendices

Appendix A

Parameter Combinations

Sr. No.	P1	P2	P3
0	Measured Depth m	Hole depth (MD) m	Hole Depth (TVD) m
1			Corrected Total Hookload kkgf
2			Extrapolated Hole TVD m
3			Bit Drill Time h
4			RGX_RT unitless
5			RHX_RT unitless
6		Hole Depth (TVD) m	Corrected Total Hookload kkgf
7			Extrapolated Hole TVD m
8			Bit Drill Time h
9			RGX_RT unitless
10			RHX_RT unitless
11		Corrected Total Hookload kkgf	Extrapolated Hole TVD m
12			Bit Drill Time h
13			RGX_RT unitless
14			RHX_RT unitless
15		Extrapolated Hole TVD m	Bit Drill Time h
16			RGX_RT unitless
17			RHX_RT unitless

18		Bit Drill Time h	RGX_RT unitless
19			RHX_RT unitless
20		RGX_RT unitless	RHX_RT unitless
21	Hole depth (MD) m	Hole Depth (TVD) m	Corrected Total Hookload kkgf
22			Extrapolated Hole TVD m
23			Bit Drill Time h
24			RGX_RT unitless
25			RHX_RT unitless
26		Corrected Total Hookload kkgf	Extrapolated Hole TVD m
27			Bit Drill Time h
28			RGX_RT unitless
29			RHX_RT unitless
30		Extrapolated Hole TVD m	Bit Drill Time h
31			RGX_RT unitless
32			RHX_RT unitless
33		Bit Drill Time h	RGX_RT unitless
34			RHX_RT unitless
35		RGX_RT unitless	RHX_RT unitless
36	Hole Depth (TVD) m	Corrected Total Hookload kkgf	Extrapolated Hole TVD m
37			Bit Drill Time h
38			RGX_RT unitless
39			RHX_RT unitless
40		Extrapolated Hole TVD m	Bit Drill Time h
41			RGX_RT unitless
42			RHX_RT unitless
43		Bit Drill Time h	RGX_RT unitless
44			RHX_RT unitless

45		RGX_RT unitless	RHX_RT unitless
46	Corrected Total Hookload kkgf	Extrapolated Hole TVD m	Bit Drill Time h
47			RGX_RT unitless
48			RHX_RT unitless
49		Bit Drill Time h	RGX_RT unitless
50			RHX_RT unitless
51		RGX_RT unitless	RHX_RT unitless
52	Extrapolated Hole TVD m	Bit Drill Time h	RGX_RT unitless
53			RHX_RT unitless
54		RGX_RT unitless	RHX_RT unitless
55	Bit Drill Time h	RGX_RT unitless	RHX_RT unitless

Appendix B

Python Code

B.1 Resampling Method Optimization

```python

import fbprophet
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sys
import random
from sklearn.preprocessing import MinMaxScaler
from numpy.random import seed
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from pca_mod import shift_pca
from pca_mod import shift_notpca
from sklearn.preprocessing import StandardScaler
from statistics_module import stats
import itertools

#%matplotlib qt

df = pd.read_csv('f9ad.csv').iloc[:, 2:]
drops = []

cols = df.columns

for i in range(len(df.T)):
    if str(np.dtype(df.iloc[:, i]))=='object':
        drops.append(cols[i])

print(drops)
```

```
    df = df.drop(columns = drops)

    index = 'Measured Depth m'
    target = 'MWD Continuous Inclination dega'
    smartfill = 0.9

    s, m, per = stats(df)

```

Listing B.1: Importing data

```

    #%%
    ## Gap statistics for target
    #
    # This chart will show the percentage of dataset occupied by gaps of a certain
    # size. Gaps are normal in drilling logs and nothing to be afraid of

    x_label = per[target]['gap_sizes']
    x = np.arange(0, len(x_label),1)

    y = per[target]['percentage_cells_occupied']
    plt.figure(figsize=(15, 8))
    plt.xticks(x, x_label, rotation=90)
    plt.bar(x,y)
    plt.title(f'Gap distribution in:\n {target}')
    plt.xlabel('Gap length')
    plt.ylabel('Percentage of dataset occupied')

    x_labels = x.tolist()
    x_labels[0] = 'data'
    plt.xticks(x, x_labels)
    plt.grid()
    plt.show()

    #%%

    ## Outlier detection

    outlier_cutoff = 0.005 #arbitrarily selected

    # calculation that penalizes long, rare, continuous gaps
    out_coef = per[target]['gap_sizes'] / (per[target]['gap_counts'] * len(df))

    x = np.arange(0,len(per[target]['gap_sizes']),1)
    x_label = per[target]['gap_sizes']
    x = np.arange(0, len(x_label),1)
    plt.figure(figsize=(15, 8))
    plt.xticks(x, x_label, rotation=90)
```

```
    plt.bar(x,out_coef)
    plt.ylim(0,0.01)
    plt.plot(x,[outlier_cutoff]*len(x), color='red', label='cutoff')
    plt.legend()
    x_labels = x.tolist()
    x_labels[0] = 'data'
    plt.xticks(x, x_labels)
    plt.title(f'Gap occupancy = gap size / (relative gap quantity) \nin:{target}')
    plt.xlabel('Gap length')
    plt.ylabel('Gap occupancy')
    plt.grid()
    plt.show()

#%%
## Automatic proposal of useful area, part 1

# find the smallest outlier gap
# outlier coefficients - True when above outlier cutoff
cutoff_map = (out_coef >= outlier_cutoff)

# Using map created above, what is the smallest outlier?
lower_cutoff = np.min(np.asarray(per[target]['gap_sizes'])[cutoff_map])

# This is done to quickly mark gaps bigger than cutoff with zero and
# other with NaN. This makes a good chart.
from functools import partial

def _cutoff(x, lower_cutoff=0):
    if x >= lower_cutoff:
        return 0
    else:
        return np.nan

_cutoff_par = partial(_cutoff, lower_cutoff=lower_cutoff)
mapped_outliers = list(map(_cutoff_par, m[target]))
plt.figure(figsize=(15, 8))
plt.scatter(df[index], df[target], s=1, c='black', label='data')
plt.plot(df[index], mapped_outliers, c='red', label='Unusuable range')
#                                                  # has to be plot to avoid index
#                                                  # discontinuities

plt.grid()
plt.legend()
plt.title('Useful range analysis')
plt.xlabel(f'{index}')
```

```
    plt.ylabel(f'{target}')
    plt.show()
    #%%

    ## Automatic proposal of useful area, part 2

    # Simply finds the biggest area with acceptable gaps

    # TODO - check if the algorithm will detect a stride at the end of the dataset
    #         because I have a feeling it won't!

    strides = []

    s_start = -1
    s_stop = -1

    for i in range(len(df)):
        if mapped_outliers[i] != 0 and s_start == -1:
            s_start = i
        elif mapped_outliers[i] == 0 and s_start != -1:
            s_stop = i
            strides.append([s_start, s_stop, s_stop - s_start ])

            s_start = -1
            s_stop = -1

    strides = np.asarray(strides)
    strides = strides[strides[:,2].argsort()][::-1] # sort by length [2]
                                                    # and reverse
    print(f'''Proposed range to use is row {strides[0,0]} to row {strides[0,1]}
    for total of {strides[0,0]} rows
          ''')
    print(f'All found strides are: [start, stop, length]')
    print(strides)

    s_start = strides[0,0]
    s_stop = strides[0,1]
    #%%

    ## cut the dataframe for the selected stride, redo the stats
    #  From now on dfs is used (dataframe stride)
    margin_percent = 1  # since the edges can be a bit unpredictable, margin is
                        # removed

    s_start = s_start + int((s_stop - s_start) * 0.01*margin_percent)
    s_stop = s_stop - int((s_stop - s_start) * 0.01*margin_percent)
    dfs = df.iloc[s_start:s_stop] # dfs = DataFrameStride
```

```
    s, m, per = stats(dfs)

    #%%

    ## Removing columns that contain big gaps

    from functools import partial

    def _cutoff_inv(x, lower_cutoff=0):
        if x >= lower_cutoff:
            return 1
        else:
            return 0

    _cutoff_par = partial(_cutoff_inv, lower_cutoff=lower_cutoff)

    killed_cols = []

    for column in list(dfs):
        mapped_outliers = list(map(_cutoff_par, m[column]))
        offender_count = np.sum(mapped_outliers)
        if offender_count > 0:
            dfs = dfs.drop(columns = [column])
            killed_cols.append([column, 100*offender_count/len(dfs)])

    killed_cols = pd.DataFrame(killed_cols, columns=['Name','Percent
offending'])
    print('Removed following columns due to outlier gap (showing
under 15% only):')
    print(killed_cols.sort_values(by='Percent offending')[killed_cols
['Percent offending'] < 15])

    #%%

    ## Checking if first derivative of index is stable.

    index_dr = np.diff(dfs[index])

    index_mean = np.mean(index_dr)
    index_std = np.std(index_dr)
    index_maxgap = np.max(index_dr)
    deviation = np.abs(index_dr - index_mean)/index_std

    print(f'Maximum distance from mean is {np.max(deviation):.1f}
standard deviations')
    print(f'If this value is above 6, there may be too high sampling
frequency variation')

    #%%

    ## Counting zeros in the first derivative to see if it should be
```

```
ffilled
## or linearly interpolated

## NOTE: Actual filling will not happen here, but AFTER the data split

fill_method = {}

for attribute in list(dfs):

    dropna_diff = np.diff(dfs[attribute].dropna())
    zeros_p = np.count_nonzero(dropna_diff == 0) / len(dropna_diff)

    if zeros_p >= smartfill: # Threshold to check?
        fill_method[attribute] = 'ffill'
    else:
        fill_method[attribute] = 'linterp'


#%%

#%%

## Gap filling - but only forward filling. Linear interpolation is done later

# Resampling helps with uneven distribution of data
# Timeseries models wants data points to be evenly spaced in time domain
for attribute in list(dfs):
    if fill_method[attribute] == 'ffill':
        dfs[attribute] = dfs[attribute].ffill().bfill()#.rolling(5, center=True).mean().ffill().bfill()


```

Listing B.2: Gap Analysis

```

split = 0.6 #portion of data available
future = 0.15 #section after available, for testing

from fancyimpute import KNN, NuclearNormMinimization, SoftImpute, BiScaler
from fancyimpute import IterativeImputer

cols = dfs.columns

masked = np.nonzero(pd.isnull(dfs.values))
xx = masked[0]
yy = masked[1]
```

```
    missing_mask = np.concatenate((xx[:, None],yy[:, None]), axis=1)

    M_data = np.nan_to_num(dfs)

    masked = np.nonzero(M_data)
    xx = masked[0]
    yy = masked[1]
    observed_mask = np.concatenate((xx[:, None],yy[:, None]), axis=1)

    X_filled_knn = KNN(k=3).fit_transform(dfs)

    dfs1 = pd.DataFrame(X_filled_knn)
    dfs1.columns = cols
    dfs1.head()

   X = dfs1.drop(target, axis=1)
    y = dfs1[target].to_frame()

```

Listing B.3: KNN Imputation

```

    from tensorflow.keras import Model, Input
    from tensorflow.keras.layers import (Dense, Dropout, GRU, Flatten
,
                                         GaussianNoise, concatenate,
LSTM,
                                         Bidirectional,
TimeDistributed)
    from tensorflow.keras.layers import Conv1D
    from tensorflow.keras.layers import MaxPool1D
    from tensorflow.keras.callbacks import EarlyStopping
    from tensorflow.keras.callbacks import ModelCheckpoint
    import tensorflow as tf

    from tensorflow.keras.models import load_model
    import math

    import tensorflow as tf
    print("Num GPUs Available: ", len(tf.config.list_physical_devices
('GPU')))


```

Listing B.4: Importing Neural Networks Libraries

```

    def models(visible1, visible2, model_name="RNN"):
      if model_name == 'RNN':
        x1 = TimeDistributed(Dense(hDense4))(visible1)
        x1 = GRU(units=hGRU, kernel_initializer = 'glorot_uniform',
                 recurrent_initializer='orthogonal',
```

```
                    bias_initializer="zeros", kernel_regularizer='l2', recurrent_regularizer=None,
                    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
                    recurrent_constraint=None, bias_constraint=None, return_sequences=True,
                    return_state=False, stateful=False)(x1)

        x1 = Dense(imagination)(x1)
        x1 = Flatten()(x1)
        x1 = Dropout(hDrop1)(x1)



        x2 = TimeDistributed(Dense(hDense5))(visible2)
        dense2 = Dense(hDense1, activation="linear")(x2)
        drop2 = Dropout(hDrop2)(dense2)
        flat2 = Flatten()(drop2)
        dense2 = Dense(imagination, activation='linear')(flat2)
        drop2 = Dropout(hDrop3)(flat2)

        model = concatenate([x1, drop2])

    if model_name == 'LSTM':
        x1 = TimeDistributed(Dense(hDense4))(visible1)
        x1 = LSTM(units=hGRU, kernel_initializer = 'glorot_uniform',
                    recurrent_initializer='orthogonal',
                    bias_initializer="zeros", kernel_regularizer='l2', recurrent_regularizer=None,
                    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
                    recurrent_constraint=None, bias_constraint=None, return_sequences=True,
                    return_state=False, stateful=False)(x1)

        x1 = Dense(imagination)(x1)
        x1 = Flatten()(x1)
        x1 = Dropout(hDrop1)(x1)



        x2 = TimeDistributed(Dense(hDense5))(visible2)
        dense2 = Dense(hDense1, activation="linear")(x2)
        drop2 = Dropout(hDrop2)(dense2)
        flat2 = Flatten()(drop2)
        dense2 = Dense(imagination, activation='linear')(flat2)
        drop2 = Dropout(hDrop3)(flat2)

        model = concatenate([x1, drop2])

    if model_name=='RNN_LSTM':
```

```
        x1 = TimeDistributed(Dense(hDense4))(visible1)
        x1 = GRU(units=hGRU, kernel_initializer = 'glorot_uniform',
                 recurrent_initializer='orthogonal',
                 bias_initializer="zeros", kernel_regularizer='l2',
    recurrent_regularizer=None,
                 bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None,
                 recurrent_constraint=None, bias_constraint=None,
    return_sequences=True,
                 return_state=False, stateful=False)(x1)

        x1 = Dense(imagination)(x1)
        x1 = Flatten()(x1)
        x1 = Dropout(hDrop1)(x1)

        x3 = TimeDistributed(Dense(hDense4))(visible1)
        x3 = LSTM(units=hGRU, kernel_initializer = 'glorot_uniform',
                 recurrent_initializer='orthogonal',
                 bias_initializer="zeros", kernel_regularizer='l2',
    recurrent_regularizer=None,
                 bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None,
                 recurrent_constraint=None, bias_constraint=None,
    return_sequences=True,
                 return_state=False, stateful=False)(x3)

        x3 = Dense(imagination)(x3)
        x3 = Flatten()(x3)
        x3 = Dropout(hDrop1)(x3)



        x2 = TimeDistributed(Dense(hDense5))(visible2)
        dense2 = Dense(hDense1, activation="linear")(x2)
        drop2 = Dropout(hDrop2)(dense2)
        flat2 = Flatten()(drop2)
        dense2 = Dense(imagination, activation='linear')(flat2)
        drop2 = Dropout(hDrop3)(flat2)

        model = concatenate([x1, x3, drop2])

    return model

```

Listing B.5: Defining Machine Learning Models

```

hstep_extension = 5
resample='yes'
resample_coef = 1
#resample_weights='distance'
step_length = index_mean * hstep_extension
```

```python

  if resample != 'no':
    grid = [['radius', 'knn'], ['uniform', 'distance'], ['ball_tree', 'kd_tree'], [1, 2, 3]]
  elif resample == 'no':
    grid = [[1], [1], [1], [1]]

  AIC_vals = []
  BIC_vals = []
  parameters = []
  mae = []
  log_res = []

  for a in range(len(grid[0])):
    resample=grid[0][a]
    for b in range(len(grid[1])):
      resample_weights=grid[1][b]
      for c in range(len(grid[2])):
        algorithm_test = grid[2][c]
        for d in range(len(grid[3])):
          metric=grid[3][d]

          if split != 1:
              splitpoint = int(len(dfs)*split)
              futurepoint = int(len(dfs)*(split+future))

              X_train = X[:splitpoint]
              y_train = y[:splitpoint]
              X_test  = X[splitpoint:futurepoint]
              y_test  = y[splitpoint:futurepoint]
          else:
              X_train = X
              y_train = y
              X_test  = X
              y_test  = y


          i_train_min = np.min(X_train[index])
          i_train_max = np.max(X_train[index])
          i_test_min = np.min(X_test[index])
          i_test_max = np.max(X_test[index])

          index_train = np.arange(i_train_min, i_train_max, step_length).reshape(-1,1)
          index_test = np.arange(i_test_min, i_test_max, step_length).reshape(-1,1)

          parameters.append(np.asarray([resample, resample_weights, algorithm_test, metric]))
          print([resample, resample_weights, algorithm_test, metric])
```

```
        if resample != 'no':
            if resample == 'radius':
                from sklearn.neighbors import RadiusNeighborsRegressor
                reg = RadiusNeighborsRegressor(radius=index_maxgap*resample_coef, weights=resample_weights, algorithm=algorithm_test, p=metric)
            elif resample == 'knn':
                from sklearn.neighbors import KNeighborsRegressor
                reg = KNeighborsRegressor(weights=resample_weights, n_neighbors=resample_coef, algorithm=algorithm_test, p=metric)
            else:
                sys.exit("Error, incorrect resampling algorithms choice")

            reg.fit(X_train[index].to_numpy().reshape(-1,1), y_train[target].to_numpy())
            y_train = pd.DataFrame()
            y_train[target] = reg.predict(index_train)

            reg.fit(X_test[index].to_numpy().reshape(-1,1), y_test[target].to_numpy())
            y_test = pd.DataFrame()
            y_test[target] = reg.predict(index_test)

            X_train_resampled = pd.DataFrame()
            for attribute in list(X_train):
                reg.fit(X_train[index].to_numpy().reshape(-1,1), X_train[attribute].to_numpy())
                X_train_resampled[attribute] = reg.predict(index_train)

            X_train = X_train_resampled

            X_test_resampled = pd.DataFrame()
            for attribute in list(X_train):
                reg.fit(X_test[index].to_numpy().reshape(-1,1), X_test[attribute].to_numpy())
                X_test_resampled[attribute] = reg.predict(index_test)

            X_test = X_test_resampled

        elif resample == 'no':
            train_q = int((i_train_max - i_train_min)/step_length)
            test_q  = int((i_test_max - i_test_min)/step_length)

            #sample_train = np.sort(random.sample(range(len(X_train)), train_q))
```

```
            #sample_test = np.sort(random.sample(range(len(X_test)), test_q))

            sample_train = np.linspace(0, len(X_train)-1, train_q, dtype=int)
            sample_test = np.linspace(0, len(X_test)-1, test_q, dtype=int)

            X_train = X_train.iloc[sample_train,:]
            y_train = y_train.iloc[sample_train,:]

            X_test = X_test.iloc[sample_test,:]
            y_test = y_test.iloc[sample_test,:]

        else:
            sys.exit("Error, incorrect resampling choice")

        ## Inclination to delta inclination convertion needed here!
        convert_to_diff = []
        asel_choice = 'pca'
        hPcaScaler='mm',
        #list of parameters that are to be in local coordinate system

        for attr in convert_to_diff:
            if attr == target:
                y_train[attr] = y_train[attr].diff().bfill() # bfill to kill initial NaN
                y_test[attr] = y_test[attr].diff().bfill()
            else:
                X_train[attr] = X_train[attr].diff().bfill()
                X_test[attr] = X_test[attr].diff().bfill()
        #%%

        ## Scaling the data. Note that range is decide on the training dataset only!


        # Both conditions have same outcome, why?
        if asel_choice == 'pca' and hPcaScaler == 'ss':
            scaler_X = MinMaxScaler() # StandardScaler()
            scaler_y = MinMaxScaler()
        else:
            scaler_X = MinMaxScaler()
            scaler_y = MinMaxScaler()

        pca_allattr = X_train.columns
        X_train[X_train.columns] = scaler_X.fit_transform(X_train[X_train.columns])
        y_train[y_train.columns] = scaler_y.fit_transform(y_train
```

```
[y_train.columns])

        ## Test portion is tranformed based on the existing
scaler
        X_test[X_test.columns] = scaler_X.transform(X_test[X_test
.columns])
        y_test[y_test.columns] = scaler_y.transform(y_test[y_test
.columns])

        #%%

        ## Dataframe for use in correlation analysis, where
X_train and y_train is
        ## together
        df_train = X_train
        df_train = df_train.merge(y_train, how='outer',
left_index=True, right_index=True)

        PCA_n = -1
        hAttrCount = 3

        asel_choice = 'pca'
        #asel_choice_1 = 'AIC'

        if asel_choice == 'AIC':


            dfs_corr = df_train.corr(method='pearson')
            corr_values = dfs_corr[target].to_numpy()
            corr_index = dfs_corr[target].index.to_numpy()

            corr_m = np.column_stack((corr_values, corr_index))

            for i in range(len(corr_m)):
                if np.isnan(corr_m[i,0]):
                    corr_m[i,0] = 0
                else:
                    corr_m[i,0] = np.abs(corr_m[i,0])

            corr_m = corr_m[corr_m[:,0].argsort()]
            sns.heatmap(dfs_corr, linewidth=0.5)
            plt.show()

            corr_n = corr_m[corr_m[:, 0]>0.98][:-1, :]
            keep_columns = corr_m[-1-len(corr_n):-1,1]

            #X_train = X_train[[keep_columns[1], keep_columns[3],
 keep_columns[4]]]
            #X_test  = X_test[[keep_columns[1], keep_columns[3],
keep_columns[4]]]

```

```python
            data = itertools.combinations(keep_columns, 3)
            keep_columns_list = list(data)

        elif asel_choice == 'pearson':

            keep_columns_list = []
            dfs_corr = df_train.corr(method='pearson')
            corr_values = dfs_corr[target].to_numpy()
            corr_index = dfs_corr[target].index.to_numpy()

            corr_m = np.column_stack((corr_values, corr_index))

            for i in range(len(corr_m)):
                if np.isnan(corr_m[i,0]):
                    corr_m[i,0] = 0
                else:
                    corr_m[i,0] = np.abs(corr_m[i,0])

            corr_m = corr_m[corr_m[:,0].argsort()]

            keep_columns = corr_m[-1-hAttrCount:-1,1]

            keep_columns_list.append(keep_columns)

            #X_train = X_train[keep_columns]
            #X_test  = X_test[keep_columns]

            sns.heatmap(dfs_corr, linewidth=0.5)
            plt.show()



        elif asel_choice == 'pca':
            # PCA is not implemented here but the asel_choice was set to 'pca'. How was the code working then
            # Was there no dimensionality reduction?
            from sklearn.decomposition import PCA
            keep_columns = [] #empty for future code compatibility

            PCA_n = hAttrCount
            # applied after sensitivity analysis

        ## ppscore based

        elif asel_choice == 'ppscore':
            import ppscore as pps
            dfs_corr = pps.predictors(df_train, target, output='list')

            keep_columns_list = []
```

```
                corr_values = []
                corr_index = []
                for i in dfs_corr:
                    corr_values.append(i['ppscore'])
                    corr_index.append(i['x'])


                corr_m = np.column_stack((corr_values, corr_index))


                corr_m = corr_m[corr_m[:,0].argsort()]

                keep_columns = corr_m[-1-hAttrCount:-1,1]
                keep_columns_list.append(keep_columns)
                #X_train = X_train[keep_columns]
                #X_test  = X_test[keep_columns]
                sns.heatmap(dfs_corr, linewidth=0.5)
                plt.show()

            else:
                sys.exit("Error, incorrect attribute selection choice")

            if PCA_n != -1:

                keep_columns_list = [1]
                scaler_pca = MinMaxScaler() # new scaler here because PCA can push
                                             # variables out of (-1,1) bounds

                pca = PCA(n_components = PCA_n)

                X_train_new = pca.fit_transform(X_train)
                X_train_new = scaler_pca.fit_transform(X_train_new)

                X_test_new = pca.transform(X_test)
                X_test_new = scaler_pca.transform(X_test_new)

            print(keep_columns)


            #Model Fitting

            #AIC_vals = []
            #BIC_vals = []
            #parameters = []
            mod = 'LSTM'
            for k in range(len(keep_columns_list)):
              if PCA_n != -1:
                keep_columns = []
```

```
            else:
              keep_columns = keep_columns_list[k]
              try:
                X_train_new = X_train[keep_columns]
                X_test_new  = X_test[keep_columns]
                parameters.append(keep_columns)
              except:
                X_train_new = X_train[list(keep_columns)]
                X_test_new  = X_test[list(keep_columns)]
                parameters.append(np.asarray(list(keep_columns)))
            #%%

            ## Data shaping

            ## from now on, arrays are being morphed into shapes valid for RNN+MLP
            imagination_meters = 25
            hMemoryMeters = 25
            lcs_list = []


            ## if clean == True:
            step_length = 1
            memory = int(hMemoryMeters/step_length)
            print(memory)
            imagination = int(imagination_meters/step_length)

            X_attr = list(X_train_new)

            try:
                X_train_new = X_train_new.to_numpy()
                X_test_new = X_test_new.to_numpy()
                y_train_new = y_train.to_numpy()
                y_test_new = y_test.to_numpy()
            except:
                y_train_new = y_train.to_numpy()
                y_test_new = y_test.to_numpy()

            if split != 1:
                X_test_new = np.concatenate([X_train_new[-memory+1:,:], X_test_new], axis=0)
                y_test_new = np.concatenate([y_train_new[-memory+1:], y_test_new], axis=0)
            #%%

            # If memopry size is 10 then stacks will be as follow:
                #Value 1 to value 10
                #Value 2 to value 11
                #
                #
                #
```

```
                    #Value N to value N+10

                def prepare(data, start, stop, cut_margin = 0, lcs=
    False):
                    memory = stop-start
                    stack = []
                    for i in range(memory):
                        stack.append(np.roll(data, -i))

                    stack = np.flip(np.rot90(stack), axis=0)[start:-
    memory+1-cut_margin]

                    if lcs == True:
                        zero = stack[:,0]

                        for j in range(len(zero)):
                            stack[j] = stack[j] - zero[j]
                    return stack

                target_lcs_correction = 1

                if target in lcs_list:
                    X_train_RNN = prepare(np.squeeze(y_train_new), 0,
    memory, cut_margin = imagination, lcs=True)
                    X_test_RNN = prepare(np.squeeze(y_test_new), 0,
    memory, cut_margin = imagination, lcs=True)

                    y_train_RNN = prepare(np.squeeze(y_train_new),
    memory, memory+imagination, lcs=True)
                    y_test_RNN = prepare(np.squeeze(y_test_new), memory
    , memory+imagination, lcs=True)

                    offset_train = X_train_RNN[:,-1]
                    offset_test = X_test_RNN[:,-1]

                    for k in range(len(offset_train)):
                        y_train_RNN[k] = y_train_RNN[k] + offset_train[
    k]

                    for k in range(len(offset_test)):
                        y_test_RNN[k] = y_test_RNN[k] + offset_test[k]

                    target_lcs_correction = 1/np.max(y_train_RNN)

                    y_train_RNN = y_train_RNN * target_lcs_correction
                    X_train_RNN = X_train_RNN  * target_lcs_correction
                    y_test_RNN = y_test_RNN * target_lcs_correction
                    X_test_RNN = X_test_RNN * target_lcs_correction

                else:
                    X_train_RNN = prepare(np.squeeze(y_train_new), 0,
```

```
memory, cut_margin = imagination)
            X_test_RNN = prepare(np.squeeze(y_test_new), 0, memory, cut_margin = imagination)
            y_train_RNN = prepare(np.squeeze(y_train_new), memory, memory+imagination)
            y_test_RNN = prepare(np.squeeze(y_test_new), memory, memory+imagination)

        X_train_MLP = []
        X_test_MLP = []

        #%%
        if PCA_n == -1:
            X_lcs_correction = [1]*len(X_train_new[0])

            for i in range(len(X_train_new[0])):
                if keep_columns[i] in lcs_list:
                    X_train_MLP.append(prepare(X_train_new[:,i],memory,memory+imagination, lcs=True))
                    X_lcs_correction[i] = 1/np.max(X_train_MLP[i])
                    X_train_MLP[i] = X_train_MLP[i]*X_lcs_correction[i]
                else:
                    X_train_MLP.append(prepare(X_train_new[:,i],memory,memory+imagination))

            X_train_MLP = np.asarray(X_train_MLP)
            X_train_MLP = np.concatenate(X_train_MLP[:,:, np.newaxis], axis = 1)
            X_train_MLP = np.rot90(X_train_MLP, axes=(1,2), k=3)





            for i in range(len(X_test_new[0])):
                if keep_columns[i] in lcs_list:
                    X_test_MLP.append(prepare(X_test_new[:,i],memory,memory+imagination, lcs=True))
                    X_test_MLP[i] = X_test_MLP[i]*X_lcs_correction[i]
                else:
                    X_test_MLP.append(prepare(X_test_new[:,i],memory,memory+imagination))

            X_test_MLP = np.asarray(X_test_MLP)
            X_test_MLP = np.concatenate(X_test_MLP[:,:, np.newaxis], axis = 1)
            X_test_MLP = np.rot90(X_test_MLP, axes=(1,2), k=3)
```

```

        else:
            for i in range(len(X_train_new[0])):
                X_train_MLP.append(prepare(X_train_new[:,i], memory,memory+imagination))

            X_train_MLP = np.asarray(X_train_MLP)
            X_train_MLP = np.concatenate(X_train_MLP[:,:, np.newaxis], axis = 1)
            X_train_MLP = np.rot90(X_train_MLP, axes=(1,2), k=3)

            for i in range(len(X_test_new[0])):
                X_test_MLP.append(prepare(X_test_new[:,i], memory,memory+imagination))

            X_test_MLP = np.asarray(X_test_MLP)
            X_test_MLP = np.concatenate(X_test_MLP[:,:, np.newaxis], axis = 1)
            X_test_MLP = np.rot90(X_test_MLP, axes=(1,2), k=3)
        #%%

        X_train_RNN_m = X_train_RNN[:,:,np.newaxis]
        X_train_m = [X_train_RNN_m, X_train_MLP]


        X_test_RNN_m = X_test_RNN[:,:,np.newaxis]
        X_test_m = [X_test_RNN_m, X_test_MLP]
        #%%

        #%%

        ## Local coordinate system

        ## Just a cumsum on parameter converted to delta earlier


        #%%

        ## ML model definition

        hDense4 = 1
        hGRU = 386
        hDrop1 = 0
        hDense5 = 128
        hDrop2 = 0
        hDrop3 = 0
        hDense1 = 1
        hDense2 = 32
        hDense3 = 128
```

```
            hDense4 = 1
            hDense5 = 128
            h5prefix=''
            verbose=0
            sensitivity_analysis = False
            plot_samples = True

            #physical_devices = tf.config.list_physical_devices('
GPU')
            #tf.config.experimental.set_memory_growth(
physical_devices[0], True)

            tf.keras.backend.clear_session()

            visible1 = Input(shape=(memory,1))


            visible2 = Input(shape=((imagination),len(X_train_new
[0])))

            combined = models(visible1, visible2, mod)

            z = Dense(hDense3, activation="relu")(combined)
            z = Dense(imagination, activation="linear")(z)


            model = Model(inputs=[visible1, visible2], outputs=z)




            es = EarlyStopping(monitor='val_loss', mode='min',
verbose=0, patience=50)

            mc = ModelCheckpoint(f'{h5prefix}best_model.h5',
monitor='val_loss',
                                             mode='min', save_best_only=
True, verbose=0)


            model.compile(optimizer='adam',loss='mean_squared_error
')
            #plot_model(model, to_file='model_plot.png',
show_shapes=True, show_layer_names=True)

            ## Training
            rowcount = len(y_train_RNN)
            val_border = int(rowcount*0.85)

            X_train_m_a = []
            X_train_m_b = []
```

```
            X_train_m_a.append(X_train_m[0][:val_border])
            X_train_m_a.append(X_train_m[1][:val_border])

            X_train_m_b.append(X_train_m[0][val_border:])
            X_train_m_b.append(X_train_m[1][val_border:])



            y_train_RNN_a = y_train_RNN[:val_border]
            y_train_RNN_b = y_train_RNN[val_border:]


            history = model.fit(X_train_m_a,y_train_RNN_a, validation_data=(X_train_m_b,y_train_RNN_b),
                                          epochs=2000, verbose=verbose, batch_size=32,
                                          callbacks=[es, mc])

            model = load_model(f'{h5prefix}best_model.h5')


            result_test = model.evaluate(X_test_m, y_test_RNN, verbose=0)

            if target in convert_to_diff:
                truth = np.cumsum(y_test_RNN/target_lcs_correction/scaler_y.scale_, axis=1)
                pred = model.predict(X_test_m)
                pred = np.cumsum(pred/target_lcs_correction/scaler_y.scale_, axis=1)

            else:
                truth = y_test_RNN/target_lcs_correction/scaler_y.scale_
                pred = model.predict(X_test_m)
                pred = pred/target_lcs_correction/scaler_y.scale_

            if split == 1 or sensitivity_analysis == True: # sensitivity enable!

                y_test_RNN = y_train_RNN
                X_test_MLP = X_train_MLP
                X_test_RNN_m = X_train_RNN_m

                senstable = {}
                plt.style.use(['science','no-latex'])
                singular_sensitivity = []
                if PCA_n != -1:
                    for i in range(pca.n_features_):
                        X_test_MLP_plus = shift_pca(X_test_MLP,
```

```
                                                    scaler_pca,
                                                    pca,
                                                    channel=i,
                                                    shift=shift)

                        X_test_m_plus = [X_test_RNN_m,
   X_test_MLP_plus]
                        results_plus = scaler_y.inverse_transform(
   model.predict(X_test_m_plus))

                        X_test_MLP_minus = shift_pca(X_test_MLP,
                                                    scaler_pca,
                                                    pca,
                                                    channel=i,
                                                    shift=-shift)
                        X_test_m_minus = [X_test_RNN_m,
   X_test_MLP_minus]
                        results_minus = scaler_y.inverse_transform(
   model.predict(X_test_m_minus))

                        if target in convert_to_diff:
                            results_plus = np.cumsum(results_plus,
   axis=1)
                            results_minus = np.cumsum(results_minus
   , axis=1)

                        sens = (results_plus - results_minus)/2


                        ave = np.average(sens, axis=0)
                        perc5 = np.percentile(sens,5,axis=0)
                        perc25 = np.percentile(sens,25,axis=0)
                        perc50 = np.percentile(sens,50,axis=0)
                        perc75 = np.percentile(sens,75,axis=0)
                        perc95 = np.percentile(sens,95,axis=0)

                        plt.figure(figsize=(4,3))
                        #plt.plot(ave, linewidth=2, color='darkblue
   ')
                        plt.plot(perc5, linewidth=1, linestyle=":",
    color='black')
                        plt.plot(perc25, linewidth=1, color='black'
   )
                        plt.plot(perc50, linewidth=2, color='black'
   )
                        plt.plot(perc75, linewidth=1, color='black'
   )
                        plt.plot(perc95, linewidth=1, linestyle=":"
   , color='black')
                        plt.title(pca_allattr[i])
                        plt.grid()
```

```
                    plt.tight_layout()


                    plt.plot([],[],linewidth=1, linestyle=":", color='black',
                             label='$5^{th}$; $95^{th}$ percentile')

                    plt.plot([],[],linewidth=1, color='black',
                             label='$25^{th}$; $75^{th}$ percentile')

                    plt.plot([],[],linewidth=2, color='black',
                             label='$50^{th}$ percentile')

                    plt.legend()
                    plt.ylabel(f'Sensitivity Index\n[{pca_allattr[i]}]')
                    plt.xlabel('Output\nPrediction distance [m]')

                    myticks = np.linspace(0,len(perc5),6)
                    mylabels = np.linspace(0,imagination_meters,6).astype(int)
                    plt.xticks(myticks, mylabels)
                    plt.title(f'Average sensitivity = {np.average(sens)}')
                    senstable[pca_allattr[i]] = np.average(sens)

                    plt.savefig(f'{pca_allattr[i].replace("/","")}.pdf')
                    plt.show()
                    singular_sensitivity.append(np.average((results_plus - results_minus)/2))

                print(singular_sensitivity)
                print(pca_allattr)
            else:
                for i in range(len(keep_columns)):
                    X_test_MLP_plus = shift_notpca(X_test_MLP,
                                                   channel=i,
                                                   shift=shift)
                    X_test_m_plus = [X_test_RNN_m, X_test_MLP_plus]
                    results_plus = scaler_y.inverse_transform(model.predict(X_test_m_plus))

                    X_test_MLP_minus = shift_notpca(X_test_MLP,
                                                    channel=i,
                                                    shift=-shift)
```

```
                X_test_m_minus = [X_test_RNN_m, X_test_MLP_minus]
                results_minus = scaler_y.inverse_transform(model.predict(X_test_m_minus))

                if target in convert_to_diff:
                    results_plus = np.cumsum(results_plus, axis=1)
                    results_minus = np.cumsum(results_minus, axis=1)


                sens = (results_plus - results_minus)/2


                ave = np.average(sens, axis=0)
                perc5 = np.percentile(sens,5,axis=0)
                perc25 = np.percentile(sens,25,axis=0)
                perc50 = np.percentile(sens,50,axis=0)
                perc75 = np.percentile(sens,75,axis=0)
                perc95 = np.percentile(sens,95,axis=0)

                plt.figure(figsize=(4,3))
                #plt.plot(ave, linewidth=2, color='darkblue')
                plt.plot(perc5, linewidth=1, linestyle=":", color='black')
                plt.plot(perc25, linewidth=1, color='black')
                plt.plot(perc50, linewidth=2, color='black')
                plt.plot(perc75, linewidth=1, color='black')
                plt.plot(perc95, linewidth=1, linestyle=":", color='black')
                plt.title(pca_allattr[i])
                plt.grid()
                plt.tight_layout()


                plt.plot([],[],linewidth=1, linestyle=":", color='black',
                        label='$5^{th}$; $95^{th}$ percentile')

                plt.plot([],[],linewidth=1, color='black',
                        label='$25^{th}$; $75^{th}$ percentile')

                plt.plot([],[],linewidth=2, color='black',
                        label='$50^{th}$ percentile')
```

```

                    plt.legend()
                    plt.ylabel(f'Sensitivity Index\n[{pca_allattr[i]}]')
                    plt.xlabel('Output\nPrediction distance [m]')

                    myticks = np.linspace(0,len(perc5),6)
                    mylabels = np.linspace(0,imagination_meters,6).astype(int)
                    plt.xticks(myticks, mylabels)
                    plt.title(f'Average sensitivity = {np.average(sens)}')
                    senstable[pca_allattr[i]] = np.average(sens)
                    plt.savefig(f'{pca_allattr[i].replace("/","")}.pdf')
                    plt.show()
                    singular_sensitivity.append(np.average((results_plus - results_minus)/2))
                print(singular_sensitivity)
                print(keep_columns)




            ## Sensitivity for RNN input channel

            X_test_m_plus = [X_test_RNN_m + 0.1, X_test_MLP]
            results_plus = scaler_y.inverse_transform(model.predict(X_test_m_plus))

            X_test_m_minus = [X_test_RNN_m - 0.1, X_test_MLP]
            results_minus = scaler_y.inverse_transform(model.predict(X_test_m_minus))

            # if target in convert_to_diff:
            #     results_plus = np.cumsum(results_plus, axis=1)
            #     results_minus = np.cumsum(results_minus, axis=1)


            sens = (results_plus - results_minus)/2


            ave = np.average(sens, axis=0)
            perc5 = np.percentile(sens,5,axis=0)
            perc25 = np.percentile(sens,25,axis=0)
            perc50 = np.percentile(sens,50,axis=0)
            perc75 = np.percentile(sens,75,axis=0)
```

```
                    perc95 = np.percentile(sens,95,axis=0)

                    plt.figure(figsize=(4,3))
                    #plt.plot(ave, linewidth=2, color='darkblue')
                    plt.plot(perc5, linewidth=1, linestyle=":", color='black')
                    plt.plot(perc25, linewidth=1, color='black')
                    plt.plot(perc50, linewidth=2, color='black')
                    plt.plot(perc75, linewidth=1, color='black')
                    plt.plot(perc95, linewidth=1, linestyle=":", color='black')
                    plt.title("RNN Input sensitivity, full channel")
                    plt.grid()
                    plt.tight_layout()


                    plt.plot([],[],linewidth=1, linestyle=":", color='black',
                            label='$5^{th}$; $95^{th}$ percentile')

                    plt.plot([],[],linewidth=1, color='black',
                            label='$25^{th}$; $75^{th}$ percentile')

                    plt.plot([],[],linewidth=2, color='black',
                            label='$50^{th}$ percentile')

                    plt.legend()
                    plt.ylabel(f'Sensitivity Index\n[{target}]')
                    plt.xlabel('Output\nPrediction distance [m]')

                    myticks = np.linspace(0,len(perc5),6)
                    mylabels = np.linspace(0,hMemoryMeters,6).astype(int)
                    plt.xticks(myticks, mylabels)

                    plt.title(f'Average sensitivity = {np.average(sens)}')

                    senstable["RNN"] = np.average(sens)
                    plt.savefig(f'1.pdf')
                    plt.show()


                    singular_sens_input = []
                    for i in range(len(X_test_RNN_m[0])):
                        print(".", end="")
                        X_test_RNN_m_plus = X_test_RNN_m.copy()

                        localrange = np.abs(np.max(X_test_RNN_m_plus[:,i]) - np.min(X_test_RNN_m_plus[:,i]))

```

```
                    X_test_RNN_m_plus[:,i] = X_test_RNN_m_plus[:,i] + 0.1*localrange
                    X_test_m_plus = [X_test_RNN_m_plus, X_test_MLP]
                    results_plus = scaler_y.inverse_transform(model.predict(X_test_m_plus))

                    X_test_RNN_m_minus = X_test_RNN_m.copy()
                    X_test_RNN_m_minus[:,i] = X_test_RNN_m_minus[:,i] - 0.1*localrange
                    X_test_m_minus = [X_test_RNN_m_minus, X_test_MLP]
                    results_minus = scaler_y.inverse_transform(model.predict(X_test_m_minus))

                    sens = (results_plus - results_minus)/2

                    singular_sens_input.append(np.percentile(sens,50, axis=0))

                plt.figure(figsize=(4,3))
                vspread = np.max(np.abs(singular_sens_input))
                sns.heatmap(np.rot90(singular_sens_input), vmin = -vspread, vmax = vspread,
                            cmap="vlag",
                            cbar_kws={'label': 'Sensitivity Index'})




                len1 = len(np.rot90(singular_sens_input))
                len2 = len(np.rot90(singular_sens_input)[0])

                plt.xticks(np.linspace(0,len2,6),
                           np.linspace(-hMemoryMeters,0,6).astype(int))
                plt.yticks(np.linspace(0,len1,6),
                           np.linspace(0,imagination_meters,6).astype(int))
                plt.xlabel('RNN memory location [m]')
                plt.ylabel('Prediction distance [m]')
                plt.title(f'Average sensitivity = {np.average(sens)}')
                plt.savefig('2.pdf')


                plt.show()
                plt.figure(figsize=(4,3))
                plt.plot(np.mean(singular_sens_input, axis=0), c='black')
                plt.xticks(np.linspace(0,len1,6),
```

```
                      np.linspace(0,imagination_meters,6).astype(int))
            plt.xlabel('Prediction distance [m]')
            plt.ylabel('Sensitivity Index')
            plt.grid()
            plt.title(f'Average sensitivity = {np.average(sens)}')
            plt.savefig('3.pdf')
            plt.show()

            plt.figure(figsize=(4,3))
            plt.plot(np.mean(singular_sens_input, axis=1), c='black')
            plt.ylabel('Sensitivity Index')
            plt.xlabel('RNN memory location [m]')
            plt.xticks(np.linspace(0,len2,6),
                      np.linspace(-hMemoryMeters,0,6).astype(int))
            plt.grid()
            plt.title(f'Average sensitivity = {np.average(sens)}')
            plt.savefig('4.pdf')
            plt.show()



        # Plots
        if plot_samples==True:
            pred = scaler_y.inverse_transform(model.predict(X_train_m, verbose=0))

            if target in convert_to_diff:

                xtr = np.cumsum(scaler_y.inverse_transform(X_train_RNN), axis=1)
                off = np.rot90(np.tile(xtr[:,-1], (len(pred[0]),1) ), 3)

                pred = np.cumsum(pred, axis=1) + off
                ytr = np.cumsum(scaler_y.inverse_transform(y_train_RNN), axis=1) + off
            else:
                xtr = scaler_y.inverse_transform(X_train_RNN)
                ytr = scaler_y.inverse_transform(y_train_RNN)


            #for i in range(10):
            #    s = np.random.randint(0, len(y_train_RNN))

            #   x = np.arange(0,len(X_train_RNN[0]),1)

```

```
                    #     plt.title('Train')
                    #     plt.plot(x, xtr[s], label='RNN input')


                    #     x = np.arange(len(X_train_RNN[0]), len(X_train_RNN[0]) + len(y_train_RNN[0]),1)
                    #     plt.plot(x,ytr[s], label='RNN output, true')



                    #     plt.plot(x,pred[s], label='RNN output, predicted')
                    #     plt.legend()


                    #     plt.show()

                    pred = scaler_y.inverse_transform(model.predict(X_test_m))

                    if target in convert_to_diff:

                         xts = np.cumsum(scaler_y.inverse_transform(X_test_RNN), axis=1)
                         off = np.rot90(np.tile(xts[:,-1], (len(pred[0]),1) ), 3)

                         pred = np.cumsum(pred, axis=1) + off
                         yts = np.cumsum(scaler_y.inverse_transform(y_test_RNN), axis=1) + off
                    else:
                         xts = scaler_y.inverse_transform(X_test_RNN)
                         yts = scaler_y.inverse_transform(y_test_RNN)


                    #for i in range(5):
                    s = np.random.randint(0, len(y_test_RNN))

                    x = np.arange(0,len(X_test_RNN[0]),1)

                    plt.plot(x, xts[s], label='RNN input')

                    x = np.arange(len(X_test_RNN[0]), len(X_test_RNN[0]) + len(y_test_RNN[0]),1)
                    plt.plot(x,yts[s], label='RNN output, true')


                    plt.title('test')
                    plt.plot(x,pred[s], label='RNN output, predicted')
                    plt.legend()
                    plt.show()
```

```

            if np.isnan(result_test):
                result_test = 0
            #print(-np.log10(result_test))

            if PCA_n != -1:
                keep_columns = pca_allattr

            print(f'MAE: {np.average(np.abs(truth-pred))}')

            if split == 1 or sensitivity_analysis == True:
                print(truth, pred, keep_columns, -np.log10(result_test), senstable)

            else:
                print(truth, pred, keep_columns, -np.log10(result_test))

            difference = truth-pred

            #####
            # AIC and BIC
            ssd = np.sum(difference ** 2)
            AIC_wi = 2*model.count_params()+len(difference)*math.log(ssd/len(difference))
            BIC_wi = np.log(len(difference))*model.count_params()+len(difference)*math.log(ssd/len(difference))
            print(AIC_wi)
            print(BIC_wi)
            AIC_vals.append(AIC_wi)
            BIC_vals.append(BIC_wi)
            mae.append(np.average(np.abs(truth-pred)))
            log_res.append(-np.log10(result_test))

from google.colab import files
df_res = pd.concat((pd.DataFrame(parameters), pd.DataFrame(AIC_vals), pd.DataFrame(BIC_vals), pd.DataFrame(mae), pd.DataFrame(log_res)), axis = 1)
#if PCA_n ==-1:
#  df_res.columns = ['P1', 'P2', 'P3', 'AIC', 'BIC']
#else:
#  df_res.columns = ['AIC', 'BIC']
if resample!='no':
    df_res.columns = ['P1', 'P2', 'P3', 'P4', 'AIC', 'BIC', 'MAE', 'Log Res']
else:
    df_res.columns = ['AIC', 'BIC']
df_res.to_csv('IC_results.csv')
files.download('IC_results.csv')
```

Listing B.6: Resampling Optimization Loop

B.2 Parameter Selection

```

AIC_vals = []
BIC_vals = []
parameters = []
mod = 'LSTM'
for k in range(len(keep_columns_list)):
    if PCA_n != -1:
        keep_columns = []
    else:
        keep_columns = keep_columns_list[k]
        try:
            X_train_new = X_train[keep_columns]
            X_test_new  = X_test[keep_columns]
            parameters.append(keep_columns)
        except:
            X_train_new = X_train[list(keep_columns)]
            X_test_new  = X_test[list(keep_columns)]
            parameters.append(np.asarray(list(keep_columns)))
    #%%

    ## Data shaping

    ## from now on, arrays are being morphed into shapes valid for RNN+MLP
    imagination_meters = 25
    hMemoryMeters = 25
    lcs_list = []


    ## if clean == True:
    step_length = 1
    memory = int(hMemoryMeters/step_length)
    print(memory)
    imagination = int(imagination_meters/step_length)

    X_attr = list(X_train_new)

    try:
        X_train_new = X_train_new.to_numpy()
        X_test_new = X_test_new.to_numpy()
        y_train_new = y_train.to_numpy()
        y_test_new = y_test.to_numpy()
    except:
```

```
        y_train_new = y_train.to_numpy()
        y_test_new = y_test.to_numpy()

    if split != 1:
        X_test_new = np.concatenate([X_train_new[-memory+1:,:], X_test_new], axis=0)
        y_test_new = np.concatenate([y_train_new[-memory+1:], y_test_new], axis=0)
    #%%

    # If memopry size is 10 then stacks will be as follow:
        #Value 1 to value 10
        #Value 2 to value 11
        #
        #
        #
        #Value N to value N+10

    def prepare(data, start, stop, cut_margin = 0, lcs=False):
        memory = stop-start
        stack = []
        for i in range(memory):
            stack.append(np.roll(data, -i))

        stack = np.flip(np.rot90(stack), axis=0)[start:-memory+1-cut_margin]

        if lcs == True:
            zero = stack[:,0]

            for j in range(len(zero)):
                stack[j] = stack[j] - zero[j]
        return stack

    target_lcs_correction = 1

    if target in lcs_list:
        X_train_RNN = prepare(np.squeeze(y_train_new), 0, memory, cut_margin = imagination, lcs=True)
        X_test_RNN = prepare(np.squeeze(y_test_new), 0, memory, cut_margin = imagination, lcs=True)

        y_train_RNN = prepare(np.squeeze(y_train_new), memory, memory+imagination, lcs=True)
        y_test_RNN = prepare(np.squeeze(y_test_new), memory, memory+imagination, lcs=True)

        offset_train = X_train_RNN[:,-1]
        offset_test = X_test_RNN[:,-1]

        for k in range(len(offset_train)):
```

```
            y_train_RNN[k] = y_train_RNN[k] + offset_train[k]

        for k in range(len(offset_test)):
            y_test_RNN[k] = y_test_RNN[k] + offset_test[k]

        target_lcs_correction = 1/np.max(y_train_RNN)

        y_train_RNN = y_train_RNN * target_lcs_correction
        X_train_RNN = X_train_RNN  * target_lcs_correction
        y_test_RNN = y_test_RNN * target_lcs_correction
        X_test_RNN = X_test_RNN * target_lcs_correction

    else:
        X_train_RNN = prepare(np.squeeze(y_train_new), 0, memory, cut_margin = imagination)
        X_test_RNN = prepare(np.squeeze(y_test_new), 0, memory, cut_margin = imagination)
        y_train_RNN = prepare(np.squeeze(y_train_new), memory, memory+imagination)
        y_test_RNN = prepare(np.squeeze(y_test_new), memory, memory+imagination)

    X_train_MLP = []
    X_test_MLP = []

    #%%
    if PCA_n == -1:
        X_lcs_correction = [1]*len(X_train_new[0])

        for i in range(len(X_train_new[0])):
            if keep_columns[i] in lcs_list:
                X_train_MLP.append(prepare(X_train_new[:,i],memory, memory+imagination, lcs=True))
                X_lcs_correction[i] = 1/np.max(X_train_MLP[i])
                X_train_MLP[i] = X_train_MLP[i]*X_lcs_correction[i]
            else:
                X_train_MLP.append(prepare(X_train_new[:,i],memory, memory+imagination))

        X_train_MLP = np.asarray(X_train_MLP)
        X_train_MLP = np.concatenate(X_train_MLP[:,:, np.newaxis], axis = 1)
        X_train_MLP = np.rot90(X_train_MLP, axes=(1,2), k=3)





        for i in range(len(X_test_new[0])):
            if keep_columns[i] in lcs_list:
                X_test_MLP.append(prepare(X_test_new[:,i],memory,
```

```
memory+imagination, lcs=True))
                    X_test_MLP[i] = X_test_MLP[i]*X_lcs_correction[i]
                else:
                    X_test_MLP.append(prepare(X_test_new[:,i],memory,
memory+imagination))

            X_test_MLP = np.asarray(X_test_MLP)
            X_test_MLP = np.concatenate(X_test_MLP[:,:, np.newaxis],
axis = 1)
            X_test_MLP = np.rot90(X_test_MLP, axes=(1,2), k=3)

        else:
            for i in range(len(X_train_new[0])):
                X_train_MLP.append(prepare(X_train_new[:,i],memory,
memory+imagination))

            X_train_MLP = np.asarray(X_train_MLP)
            X_train_MLP = np.concatenate(X_train_MLP[:,:, np.newaxis],
axis = 1)
            X_train_MLP = np.rot90(X_train_MLP, axes=(1,2), k=3)

            for i in range(len(X_test_new[0])):
                X_test_MLP.append(prepare(X_test_new[:,i],memory,memory
+imagination))

            X_test_MLP = np.asarray(X_test_MLP)
            X_test_MLP = np.concatenate(X_test_MLP[:,:, np.newaxis],
axis = 1)
            X_test_MLP = np.rot90(X_test_MLP, axes=(1,2), k=3)
    #%%

    X_train_RNN_m = X_train_RNN[:,:,np.newaxis]
    X_train_m = [X_train_RNN_m, X_train_MLP]


    X_test_RNN_m = X_test_RNN[:,:,np.newaxis]
    X_test_m = [X_test_RNN_m, X_test_MLP]
    #%%

    #%%

    ## Local coordinate system

    ## Just a cumsum on parameter converted to delta earlier


    #%%

    ## ML model definition

    hDense4 = 1
```

```
    hGRU = 386
    hDrop1 = 0
    hDense5 = 128
    hDrop2 = 0
    hDrop3 = 0
    hDense1 = 1
    hDense2 = 32
    hDense3 = 128
    hDense4 = 1
    hDense5 = 128
    h5prefix=''
    verbose=0
    sensitivity_analysis = False
    plot_samples = True

    #physical_devices = tf.config.list_physical_devices('GPU')
    #tf.config.experimental.set_memory_growth(physical_devices[0], True)

    tf.keras.backend.clear_session()

    visible1 = Input(shape=(memory,1))


    visible2 = Input(shape=((imagination),len(X_train_new[0])))

    combined = models(visible1, visible2, mod)

    z = Dense(hDense3, activation="relu")(combined)
    z = Dense(imagination, activation="linear")(z)


    model = Model(inputs=[visible1, visible2], outputs=z)




    es = EarlyStopping(monitor='val_loss', mode='min', verbose=0, patience=50)

    mc = ModelCheckpoint(f'{h5prefix}best_model.h5', monitor='val_loss',
                                             mode='min', save_best_only=True, verbose=0)


    model.compile(optimizer='adam',loss='mean_squared_error')
    #plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

    ## Training
```

```
    rowcount = len(y_train_RNN)
    val_border = int(rowcount*0.85)

    X_train_m_a = []
    X_train_m_b = []

    X_train_m_a.append(X_train_m[0][:val_border])
    X_train_m_a.append(X_train_m[1][:val_border])

    X_train_m_b.append(X_train_m[0][val_border:])
    X_train_m_b.append(X_train_m[1][val_border:])



    y_train_RNN_a = y_train_RNN[:val_border]
    y_train_RNN_b = y_train_RNN[val_border:]


    history = model.fit(X_train_m_a,y_train_RNN_a,validation_data=(X_train_m_b,y_train_RNN_b),
                                          epochs=2000, verbose=verbose, batch_size=32,
                                          callbacks=[es, mc])

    model = load_model(f'{h5prefix}best_model.h5')


    result_test = model.evaluate(X_test_m, y_test_RNN, verbose=0)

    if target in convert_to_diff:
        truth = np.cumsum(y_test_RNN/target_lcs_correction/scaler_y.scale_, axis=1)
        pred = model.predict(X_test_m)
        pred = np.cumsum(pred/target_lcs_correction/scaler_y.scale_, axis=1)

    else:
        truth = y_test_RNN/target_lcs_correction/scaler_y.scale_
        pred = model.predict(X_test_m)
        pred = pred/target_lcs_correction/scaler_y.scale_

    if split == 1 or sensitivity_analysis == True: # sensitivity enable!

        y_test_RNN = y_train_RNN
        X_test_MLP = X_train_MLP
        X_test_RNN_m = X_train_RNN_m

        senstable = {}
        plt.style.use(['science','no-latex'])
        singular_sensitivity = []
```

```
        if PCA_n != -1:
            for i in range(pca.n_features_):
                X_test_MLP_plus = shift_pca(X_test_MLP,
                                            scaler_pca,
                                            pca,
                                            channel=i,
                                            shift=shift)

                X_test_m_plus = [X_test_RNN_m, X_test_MLP_plus]
                results_plus = scaler_y.inverse_transform(model.
predict(X_test_m_plus))

                X_test_MLP_minus = shift_pca(X_test_MLP,
                                             scaler_pca,
                                             pca,
                                             channel=i,
                                             shift=-shift)
                X_test_m_minus = [X_test_RNN_m, X_test_MLP_minus]
                results_minus = scaler_y.inverse_transform(model.
predict(X_test_m_minus))

                if target in convert_to_diff:
                    results_plus = np.cumsum(results_plus, axis=1)
                    results_minus = np.cumsum(results_minus, axis
=1)

                sens = (results_plus - results_minus)/2


                ave = np.average(sens, axis=0)
                perc5 = np.percentile(sens,5,axis=0)
                perc25 = np.percentile(sens,25,axis=0)
                perc50 = np.percentile(sens,50,axis=0)
                perc75 = np.percentile(sens,75,axis=0)
                perc95 = np.percentile(sens,95,axis=0)

                plt.figure(figsize=(4,3))
                #plt.plot(ave, linewidth=2, color='darkblue')
                plt.plot(perc5, linewidth=1, linestyle=":", color='
black')
                plt.plot(perc25, linewidth=1, color='black')
                plt.plot(perc50, linewidth=2, color='black')
                plt.plot(perc75, linewidth=1, color='black')
                plt.plot(perc95, linewidth=1, linestyle=":", color=
'black')
                plt.title(pca_allattr[i])
                plt.grid()
                plt.tight_layout()


                plt.plot([],[],linewidth=1, linestyle=":", color='
```

```
black',
                        label='$5^{th}$; $95^{th}$ percentile')

                plt.plot([],[],linewidth=1, color='black',
                        label='$25^{th}$; $75^{th}$ percentile')

                plt.plot([],[],linewidth=2, color='black',
                        label='$50^{th}$ percentile')

                plt.legend()
                plt.ylabel(f'Sensitivity Index\n[{pca_allattr[i]}]')
                plt.xlabel('Output\nPrediction distance [m]')

                myticks = np.linspace(0,len(perc5),6)
                mylabels = np.linspace(0,imagination_meters,6).astype(int)
                plt.xticks(myticks, mylabels)
                plt.title(f'Average sensitivity = {np.average(sens)}')
                senstable[pca_allattr[i]] = np.average(sens)

                plt.savefig(f'{pca_allattr[i].replace("/","")}.pdf')
                plt.show()
                singular_sensitivity.append(np.average((results_plus - results_minus)/2))

            print(singular_sensitivity)
            print(pca_allattr)
        else:
            for i in range(len(keep_columns)):
                X_test_MLP_plus = shift_notpca(X_test_MLP,
                                          channel=i,
                                          shift=shift)
                X_test_m_plus = [X_test_RNN_m, X_test_MLP_plus]
                results_plus = scaler_y.inverse_transform(model.predict(X_test_m_plus))

                X_test_MLP_minus = shift_notpca(X_test_MLP,
                                          channel=i,
                                          shift=-shift)
                X_test_m_minus = [X_test_RNN_m, X_test_MLP_minus]
                results_minus = scaler_y.inverse_transform(model.predict(X_test_m_minus))

                if target in convert_to_diff:
                    results_plus = np.cumsum(results_plus, axis=1)
                    results_minus = np.cumsum(results_minus, axis=1)

```

```

                sens = (results_plus - results_minus)/2


                ave = np.average(sens, axis=0)
                perc5 = np.percentile(sens,5,axis=0)
                perc25 = np.percentile(sens,25,axis=0)
                perc50 = np.percentile(sens,50,axis=0)
                perc75 = np.percentile(sens,75,axis=0)
                perc95 = np.percentile(sens,95,axis=0)

                plt.figure(figsize=(4,3))
                #plt.plot(ave, linewidth=2, color='darkblue')
                plt.plot(perc5, linewidth=1, linestyle=":", color='black')
                plt.plot(perc25, linewidth=1, color='black')
                plt.plot(perc50, linewidth=2, color='black')
                plt.plot(perc75, linewidth=1, color='black')
                plt.plot(perc95, linewidth=1, linestyle=":", color='black')
                plt.title(pca_allattr[i])
                plt.grid()
                plt.tight_layout()


                plt.plot([],[],linewidth=1, linestyle=":", color='black',
                        label='$5^{th}$; $95^{th}$ percentile')

                plt.plot([],[],linewidth=1, color='black',
                        label='$25^{th}$; $75^{th}$ percentile')

                plt.plot([],[],linewidth=2, color='black',
                        label='$50^{th}$ percentile')

                plt.legend()
                plt.ylabel(f'Sensitivity Index\n[{pca_allattr[i]}]')
                plt.xlabel('Output\nPrediction distance [m]')

                myticks = np.linspace(0,len(perc5),6)
                mylabels = np.linspace(0,imagination_meters,6).astype(int)
                plt.xticks(myticks, mylabels)
                plt.title(f'Average sensitivity = {np.average(sens)}')
                senstable[pca_allattr[i]] = np.average(sens)
                plt.savefig(f'{pca_allattr[i].replace("/","")}.pdf')
                plt.show()
                singular_sensitivity.append(np.average((
```

```
results_plus - results_minus)/2))
            print(singular_sensitivity)
            print(keep_columns)




        ## Sensitivity for RNN input channel

        X_test_m_plus = [X_test_RNN_m + 0.1, X_test_MLP]
        results_plus = scaler_y.inverse_transform(model.predict(
X_test_m_plus))

        X_test_m_minus = [X_test_RNN_m - 0.1, X_test_MLP]
        results_minus = scaler_y.inverse_transform(model.predict(
X_test_m_minus))

        # if target in convert_to_diff:
        #     results_plus = np.cumsum(results_plus, axis=1)
        #     results_minus = np.cumsum(results_minus, axis=1)


        sens = (results_plus - results_minus)/2


        ave = np.average(sens, axis=0)
        perc5 = np.percentile(sens,5,axis=0)
        perc25 = np.percentile(sens,25,axis=0)
        perc50 = np.percentile(sens,50,axis=0)
        perc75 = np.percentile(sens,75,axis=0)
        perc95 = np.percentile(sens,95,axis=0)

        plt.figure(figsize=(4,3))
        #plt.plot(ave, linewidth=2, color='darkblue')
        plt.plot(perc5, linewidth=1, linestyle=":", color='black')
        plt.plot(perc25, linewidth=1, color='black')
        plt.plot(perc50, linewidth=2, color='black')
        plt.plot(perc75, linewidth=1, color='black')
        plt.plot(perc95, linewidth=1, linestyle=":", color='black')
        plt.title("RNN Input sensitivity, full channel")
        plt.grid()
        plt.tight_layout()


        plt.plot([],[],linewidth=1, linestyle=":", color='black',
                label='$5^{th}$; $95^{th}$ percentile')

        plt.plot([],[],linewidth=1, color='black',
                label='$25^{th}$; $75^{th}$ percentile')

        plt.plot([],[],linewidth=2, color='black',
```

```
                    label='$50^{th}$ percentile')

        plt.legend()
        plt.ylabel(f'Sensitivity Index\n[{target}]')
        plt.xlabel('Output\nPrediction distance [m]')

        myticks = np.linspace(0,len(perc5),6)
        mylabels = np.linspace(0,hMemoryMeters,6).astype(int)
        plt.xticks(myticks, mylabels)

        plt.title(f'Average sensitivity = {np.average(sens)}')

        senstable["RNN"] = np.average(sens)
        plt.savefig(f'1.pdf')
        plt.show()


        singular_sens_input = []
        for i in range(len(X_test_RNN_m[0])):
            print(".", end="")
            X_test_RNN_m_plus = X_test_RNN_m.copy()

            localrange = np.abs(np.max(X_test_RNN_m_plus[:,i]) - np.min(X_test_RNN_m_plus[:,i]))

            X_test_RNN_m_plus[:,i] = X_test_RNN_m_plus[:,i] + 0.1*localrange
            X_test_m_plus = [X_test_RNN_m_plus, X_test_MLP]
            results_plus = scaler_y.inverse_transform(model.predict(X_test_m_plus))

            X_test_RNN_m_minus = X_test_RNN_m.copy()
            X_test_RNN_m_minus[:,i] = X_test_RNN_m_minus[:,i] - 0.1*localrange
            X_test_m_minus = [X_test_RNN_m_minus, X_test_MLP]
            results_minus = scaler_y.inverse_transform(model.predict(X_test_m_minus))

            sens = (results_plus - results_minus)/2

            singular_sens_input.append(np.percentile(sens,50, axis=0))

        plt.figure(figsize=(4,3))
        vspread = np.max(np.abs(singular_sens_input))
        sns.heatmap(np.rot90(singular_sens_input), vmin = -vspread, vmax = vspread,
                    cmap="vlag",
                    cbar_kws={'label': 'Sensitivity Index'})

```

```


        len1 = len(np.rot90(singular_sens_input))
        len2 = len(np.rot90(singular_sens_input)[0])

        plt.xticks(np.linspace(0,len2,6),
                np.linspace(-hMemoryMeters,0,6).astype(int))
        plt.yticks(np.linspace(0,len1,6),
                np.linspace(0,imagination_meters,6).astype(int))
        plt.xlabel('RNN memory location [m]')
        plt.ylabel('Prediction distance [m]')
        plt.title(f'Average sensitivity = {np.average(sens)}')
        plt.savefig('2.pdf')


        plt.show()
        plt.figure(figsize=(4,3))
        plt.plot(np.mean(singular_sens_input, axis=0), c='black')
        plt.xticks(np.linspace(0,len1,6),
                np.linspace(0,imagination_meters,6).astype(int))
        plt.xlabel('Prediction distance [m]')
        plt.ylabel('Sensitivity Index')
        plt.grid()
        plt.title(f'Average sensitivity = {np.average(sens)}')
        plt.savefig('3.pdf')
        plt.show()

        plt.figure(figsize=(4,3))
        plt.plot(np.mean(singular_sens_input, axis=1), c='black')
        plt.ylabel('Sensitivity Index')
        plt.xlabel('RNN memory location [m]')
        plt.xticks(np.linspace(0,len2,6),
                np.linspace(-hMemoryMeters,0,6).astype(int))
        plt.grid()
        plt.title(f'Average sensitivity = {np.average(sens)}')
        plt.savefig('4.pdf')
        plt.show()



    # Plots
    if plot_samples==True:
        pred = scaler_y.inverse_transform(model.predict(X_train_m, verbose=0))

        if target in convert_to_diff:

            xtr = np.cumsum(scaler_y.inverse_transform(X_train_RNN), axis=1)
            off = np.rot90(np.tile(xtr[:,-1], (len(pred[0]),1) ), 3)
```

```
            pred = np.cumsum(pred, axis=1) + off
            ytr = np.cumsum(scaler_y.inverse_transform(y_train_RNN)
, axis=1) + off
        else:
            xtr = scaler_y.inverse_transform(X_train_RNN)
            ytr = scaler_y.inverse_transform(y_train_RNN)


        #for i in range(10):
        #    s = np.random.randint(0, len(y_train_RNN))

        #    x = np.arange(0,len(X_train_RNN[0]),1)

        #    plt.title('Train')
        #    plt.plot(x, xtr[s], label='RNN input')


        #    x = np.arange(len(X_train_RNN[0]), len(X_train_RNN[0])
 + len(y_train_RNN[0]),1)
        #    plt.plot(x,ytr[s], label='RNN output, true')



        #    plt.plot(x,pred[s], label='RNN output, predicted')
        #    plt.legend()


        #    plt.show()

        pred = scaler_y.inverse_transform(model.predict(X_test_m))

        if target in convert_to_diff:

            xts = np.cumsum(scaler_y.inverse_transform(X_test_RNN),
 axis=1)
            off = np.rot90(np.tile(xts[:,-1], (len(pred[0]),1) ),
3)

            pred = np.cumsum(pred, axis=1) + off
            yts = np.cumsum(scaler_y.inverse_transform(y_test_RNN),
 axis=1) + off
        else:
            xts = scaler_y.inverse_transform(X_test_RNN)
            yts = scaler_y.inverse_transform(y_test_RNN)


        #for i in range(5):
        s = np.random.randint(0, len(y_test_RNN))

        x = np.arange(0,len(X_test_RNN[0]),1)
```

```python

        plt.plot(x, xts[s], label='RNN input')

        x = np.arange(len(X_test_RNN[0]), len(X_test_RNN[0]) + len(y_test_RNN[0]),1)
        plt.plot(x,yts[s], label='RNN output, true')


        plt.title('test')
        plt.plot(x,pred[s], label='RNN output, predicted')
        plt.legend()
        plt.show()


    if np.isnan(result_test):
        result_test = 0
    #print(-np.log10(result_test))

    if PCA_n != -1:
        keep_columns = pca_allattr

    print(f'MAE: {np.average(np.abs(truth-pred))}')

    if split == 1 or sensitivity_analysis == True:
        print(truth, pred, keep_columns, -np.log10(result_test), senstable)

    else:
        print(truth, pred, keep_columns, -np.log10(result_test))

    difference = truth-pred

    #####
    # AIC and BIC
    ssd = np.sum(difference ** 2)
    AIC_wi = 2*model.count_params()+len(difference)*math.log(ssd/len(difference))
    BIC_wi = np.log(len(difference))*model.count_params()+len(difference)*math.log(ssd/len(difference))
    print(AIC_wi)
    print(BIC_wi)
    AIC_vals.append(AIC_wi)
    BIC_vals.append(BIC_wi)

from google.colab import files
df_res = pd.concat((pd.DataFrame(parameters), pd.DataFrame(AIC_vals), pd.DataFrame(BIC_vals)), axis = 1)
df_res.columns = ['P1', 'P2', 'P3', 'AIC', 'BIC']
df_res.to_csv('IC_results.csv')
files.download('IC_results.csv')
```

```

```

Listing B.7: Parameter Selection Loop