UNIVERSITY OF CAPE TOWN

MASTERS MINI DISSERTATION

Automated Machine Learning for Predicting Trends in Time Series Data

---

Author:
Kouame KOUASSI

Supervisor:
A/Prof. Deshen MOODLEY

A minor dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science

in the

Centre for Artificial Intelligence Research
Department of Computer Science
University of Cape Town

October 18, 2021

Declaration of Authorship

I, Kouame KOUASSI, declare that this thesis titled, "Automated Machine Learning for Predicting Trends in Time Series Data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Signed by candidate

Date: 18 October 2021

Declaration of Publications

1. K.H. Kouassi, and D. Moodley, "An Analysis of Deep Neural Networks for Predicting Trends in Time Series Data". To appear in *Communications in Computer and Information Science* (LNCS sub-series CCIS), Volume 1342, ISBN 978-3-030-66150-2, 2020.

2. K. H. Kouassi, and D. Moodley, "Automatic deep learning for trend prediction in time series data", 2020. In Progress. arXiv: 2009.08510 [cs.LG].

"Education is the most powerful weapon which you can use to change the world."

Nelson Mandela

Abstract

Recently, a hybrid Deep Neural Network (DNN) algorithm, TreNet was proposed for predicting trends in time series data. While TreNet was shown to have superior performance to a number of alternative approaches, the validation method used did not take into account the sequential nature of time series data. It also did not deal with model update and model stability, which are important for real-world applications. Furthermore, in the TreNet paper and previous trend prediction research, the Algorithm Selection and Hyperparameter Optimisation (ASHO) is performed manually. However, manual ASHO is expensive and often results in a sub-optimal or mediocre model because it needs extensive experimentation as well as domain specific and Machine Learning (ML) expert knowledge. This dissertation replicates TreNet experiments on the same datasets using a walk-forward validation method, which includes model update. The model is tested over multiple independent runs to evaluate model stability. TreNet, which takes in both raw point data and trend line features, is compared to vanilla DNNs and traditional ML algorithms that take in raw point data features. A recent Automated Machine Learning (AutoML) namely the hybrid Bayesian optimisation and hyperband (BOHB) framework is implemented and evaluated for ASHO. The AutoML models are then compared to the manually tuned models. The results show that in general TreNet still performs better than the vanilla DNN, but not on all datasets as reported in the original TreNet paper. On non-stationary datasets, traditional ML models outperform DNN models. The AutoML experiments found optimal configurations that produced models that surpass or compare well against the average performance and stability levels of configurations found during the experiments with manual tuning for ASHO across four datasets. This work highlights the importance of using an appropriate validation method and evaluating model stability while developing and testing ML models for time series applications. It also demonstrates that AutoML techniques such as BOHB are effective to automatically finding a well-performing models for predicting trends in time series data, thus making ML model development more systematic and less error-prone.

Acknowledgements

I would like to thank God for providing me with the necessary opportunities and for supporting me throughout my education journey. I am grateful to my supervisor A/Prof. Deshen Moodley, who provided me with the guidance, advice, and help required to produce this dissertation. Professor Moodley did not hesitate go out of his way to give me the support I needed. His supervision made my Masters journey a much better experience than I imagined. I thank the Centre for Artificial Intelligence Research (CAIR) and the Department of Science and Technology for awarding me the CAIR scholarship to carry out this research. To the Youth Excellence Programme, thank you for providing me with the initial scholarship that led to this Masters dissertation. To the members of the Adaptive and Cognitive Systems Lab, to my friends and family members, thank you for your support and advice. To the Deep Learning Indaba community, thank you for the inspiration, the opportunity to learn and to teach. Further recognition goes to the anonymous reviewers of papers submitted from this work. Their comments made the overall quality of this dissertation better.

Contents

Declaration of Authorship i

Declaration of Publications ii

Abstract iv

Acknowledgements v

1 Introduction 1
- 1.1 Background and context . . . 1
- 1.2 Problem statement . . . 2
- 1.3 Aim and objectives . . . 3
- 1.4 Research methodology . . . 3
- 1.5 Contribution of the study . . . 4
- 1.6 Dissertation outline . . . 5

2 Literature review 6
- 2.1 Machine learning for time series prediction . . . 6
  - 2.1.1 Machine learning process . . . 6
  - 2.1.2 Time series data characteristics . . . 6
  - 2.1.3 Preprocessing . . . 7
  - 2.1.4 Widely used algorithms . . . 8
    - Multilayer perceptrons . . . 8
    - Long-short term memory . . . 9
    - Convolutional neural networks . . . 9
    - Support vector regression . . . 10
    - Random forests . . . 11
    - Gradient boosting machines . . . 11
  - 2.1.5 Time series prediction model evaluation . . . 11
    - Traditional time series evaluation . . . 11
    - The standard cross-validation . . . 12
    - Walk-forward evaluation . . . 12
  - 2.1.6 Performance metrics . . . 13
- 2.2 Predicting trends in time series data . . . 14
  - 2.2.1 Indirectly learning trends in time series data . . . 14
  - 2.2.2 Directly learning trends in time series data . . . 14

Trend prediction formulation . . . 14
TreNet . . . 15
2.2.3 Shortcomings of TreNet . . . 16
2.2.4 Segmentation of time series into trends . . . 16
Sliding window . . . 17
Bottom-up . . . 17
Top-down . . . 17
2.3 Algorithm selection and hyperparameter optimisation . . . 18
2.3.1 Hyperparameter optimisation . . . 18
2.3.2 Algorithm selection . . . 19
2.3.3 CASH problem formulation . . . 19
2.3.4 Approaches to the CASH problem . . . 19
Sequential model based optimisation . . . 20
HyperBand . . . 21
Bayesian optimisation and hyperband . . . 21
Summary of some AutoML tools . . . 22
2.4 CASH problem in time series prediction . . . 22
2.5 Summary . . . 22

3 Experimental design **24**
3.1 Overview of the process for predicting trends . . . 24
3.2 Datasets . . . 25
3.2.1 The voltage dataset . . . 25
3.2.2 The methane dataset . . . 26
3.2.3 NYSE dataset . . . 26
3.2.4 JSE dataset . . . 27
3.3 Preprocessing . . . 28
3.3.1 Data segmentation . . . 28
3.3.2 Input-ouput data instances . . . 28
3.4 Learning algorithm design . . . 29
3.4.1 The hybrid DNN (TreNet) . . . 29
3.4.2 MultiLayer perceptrons (MLP) . . . 30
3.4.3 Long short-term memory (LSTM) . . . 30
3.4.4 Convolutional neural network (CNN) . . . 30
3.4.5 Neural network training and regularisation . . . 30
3.4.6 Random Rorests (RF) . . . 31
3.4.7 Gradient Boosting Machines (GBM) . . . 31
3.4.8 Support Vector Regression (SVR) . . . 31
3.5 Model evaluation . . . 31
3.5.1 Walk-forward evaluation and performance metrics . . . 31
3.5.2 Model update with warm-start . . . 33
3.6 Algorithm selection and hyperparameter optimisation . . . 34

3.6.1 Manual tuning . . . 34
3.6.2 AutoML for ASHO . . . 34
3.7 AutoML framework and implementation . . . 34
3.7.1 Hyperparameter configuration space . . . 35
3.7.2 BOHB budget and configuration . . . 35
3.8 Overview of the experiments . . . 36

4 Experiments - Manual ML for predicting trends 38
4.1 Experiment 1: Predicting trends with TreNet . . . 38
4.1.1 Comparison metric . . . 38
4.1.2 Results and discussions . . . 39
4.1.3 Robustness and stability . . . 40
4.1.4 Optimal hyperparameters . . . 40
4.1.5 Analysis of the model update with *warm-start* . . . 41
4.2 Experiment 2: Predicting trends with vanilla DNNs . . . 41
4.2.1 Results and discussions . . . 43
4.3 Experiment 3: Predicting trends with traditional ML algorithms . . . 44
4.3.1 Results and discussions . . . 44
4.4 Experiment 4: Addition of trend line features . . . 45
4.4.1 Results and discussions . . . 45
4.5 Summary: Best manually tuned models . . . 46

5 Experiments - AutoML for predicting trends 48
5.1 Experiment 5: AutoML – all algorithms . . . 48
5.1.1 Optimal AutoML models . . . 48
5.1.2 Configurations evaluated . . . 49
5.2 Experiment 6: AutoML - single algorithm . . . 50
5.2.1 Results and discussions . . . 51
5.3 Summary: Best AutoML models . . . 51

6 Discussion and Conclusions 53
6.1 Manual ML for predicting trends . . . 53
6.1.1 Summary of the findings . . . 53
6.1.2 Reflection on manual ML for predicting trends . . . 54
6.2 AutoML for predicting trends . . . 54
6.2.1 Summary of the findings . . . 54
6.2.2 Reflection on AutoML for predicting trends . . . 55
6.3 Impact of the research . . . 56
6.4 Limitations and Future work . . . 57

A Set of hyperparameter values evaluated for each algorithm per dataset during the manual tuning process 58

B Speed-up obtained from using model update with warm-start 59

C The RMSE values achieved by non-hybrid algorithms after the addition of trend line features **60**

References **62**

List of Figures

2.1 Machine learning process . . . 6
2.2 Sliding window of size 20 for a 1-step ahead stock market price movement prediction. (copied from [37]) . . . 7
2.3 Multilayer perceptrons with a single hidden layer (copied from [34]) . . . 8
2.4 Repeating LSTM Cells . . . 9
2.5 CNN structure . . . 10
2.6 Traditional time series evaluation . . . 12
2.7 An example of successive training validation test sets in overlapping partition . . . 12
2.8 (a) Trend prediction on time series. (b) Associated sequence of trends (copied from [3]) . . . 15
2.9 Illustration of the hybrid architecture of TreNet (copied from [3]) . . . 15

3.1 Overview diagram of trend prediction process . . . 24
3.2 Top - The individual household voltage dataset. Bottom - Probalility distribution of the voltage dataset. . . . 25
3.3 Top - Methane concentration in air over time. Bottom - Probability distribution of the methane dataset. . . . 26
3.4 Top - The composite NYSE closing price dataset. Bottom - Probalility distribution of the NYSE dataset. . . . 27
3.5 Raw NYSE dataset. . . . 27
3.6 Top - Composite JSE closing price dataset. Bottom - Probalility distribution of the JSE dataset. . . . 28
3.7 The structure of a one layer 1D-convolution neural network. . . . 31
3.8 Warm-start fraction selection for MLP on the NYSE dataset. The red line represents the validation loss without warm-start. . . . 33

List of Tables

2.1 Comparison of Loss functions 14
2.2 Comparison of segmentation algorithms 18
2.3 List of tools for implementing algorithms for CASH 22

3.1 Summary of the characteristics of the datasets. 27
3.2 Summary of the basic statistics of the segmented datasets. 29
3.3 Summary of the input vector size per dataset and per feature type. Raw local data refers to the window size in terms of number of data points. Number of data instances refers to the number total number of data examples available for training and testing 29
3.4 Summary of the data instance partitioning 32
3.5 Summary of the hyperparameters in the configuration space 36
3.6 BOHB budget (number of epochs), mimimum budget (min), maximum budget (max) used for each dataset. All refers to the budget used when the hyperparameter configuration space contained all three vanilla deep neural networks. 37

4.1 Comparison of the slope (S), duration (D), and average (A) RMSE values achieved by TreNet and Lin et al.'s (Original) results, and their respective percentage improvements (% improv.) over the naive LVM 39
4.2 The RMSE values and the standard deviation achieved by TreNet across multiple (10) runs. 40
4.3 TreNet hyperparameters found by manual experimentation. "?" means *unknown* and $S = \{300, 600, 900, 1200\}$ 41
4.4 The reduction factor in total number of epochs (speed-up), and the average (slope and duration) RMSE using model update with/without warm-start 41
4.5 Hyperparameters optimised for the vanilla DNN algorithms and their optimal values found for each dataset 42
4.6 Comparison of the TreNet model with the vanilla DNN models 43
4.7 Hyperparameters optimised for each algorithm and their optimal values found manually for each dataset 44
4.8 Comparison of the best DNN models (Best DNN) with the traditional ML algorithms. The % improvement (% improv.) is the performance improvement of the best traditional ML model over the best DNN model 45

4.9 Performance improvement (%) in RMSE after supplementing the raw data with trend line features. 46
4.10 Average RMSE values (E) achieved by the hybrid TreNet model (Hybrid); and the best non-hybrid model (A) with raw point data features alone (Pt) and with raw point data plus trend line features (Pt + T). The % change is with respect to the TreNet algorithm. 47

5.1 The number of times each of DNN models is selected as optimal by BOHB, and the best manually tuned DNN models (Best M-DNN) 49
5.2 The number of unique configurations evaluated (No. unique) and the total number of evaluations performed (Total) manually and automatically using BOHB as well as the number of equivalent evaluations on the maximum budget (No. max. equiv.). 49
5.3 Comparison of BOHB on all algorithm (BOHB-All) and the best BOHB on the single algorithms (BOHB-Single). Model (M), Slope RMSE (S), Duration RMSE (D), Average slope and duration RMSEs (A) 51
5.4 Comparison of the best AutoML models and the best manual models. Model (M), Slope RMSE (S), Duration RMSE (D), Average slope and duration RMSEs (A) 52

A.1 Set of hyperparameter values evaluated for each algorithm per dataset during the manual tuning process 58

B.1 The reduction factor in total number of epochs (speed-up), and the average (slope and duration) RMSE using model update with/without warm-start 59

C.1 The RMSE values achieved by vanilla DNN algorithms on raw data alone and raw data and trend line features. 60
C.2 The RMSE values achieved by traditional ML algorithms on raw data alone and raw data and trend line features. 61

List of Abbreviations

AE	**A**bsolute **E**rror
AR	Auto**R**egressive
ARIMA	**A**uto**R**egressive **I**ntegrated **M**oving **A**verage
ASHO	**A**lgorithm **S**election and **H**yperparameter **O**ptimisation
Auto-WEKA	**A**utomatic **M**odel **S**election and **H**yperparameter **O**ptimization in WEKA
AutoML	**A**utomated **M**achine **L**earning
BO	**B**ayesian **O**ptimisation
BOHB	**B**ayesian **O**ptimisation and **H**yper**B**and
CASH	**A**lgorithm **S**election and **H**yperparameter **O**ptimisation
CNN	**C**onvolutional **N**eural **N**etworks
DNN	**D**eep **N**eural **N**etwork
GARCH	**G**eneralised **A**uto**R**egressive **C**onditional **H**eteroscedastic
GBM	**G**radient **B**oosting **M**achines
GP	**G**aussian **P**rocesses
HB	**H**yper**B**and
HMM	**H**idden **M**arkov **M**odels
JSE	**J**ohannesburg **S**tock **E**xchange
LVM	**L**ast **V**alue **M**odel
LSTM	**L**ong-**S**hort **T**erm **M**emory
MA	**M**oving **A**verage
MAE	**M**ean **A**bsolute **E**rror
MAPE	**M**ean **A**bsolute **P**ercentage **E**rror
ML	**M**achine **L**earning
MLP	**M**ulti**L**ayer **P**erceptrons
MTBO	**M**ulti-**T**ask **B**ayesian **O**ptimisation
NYSE	**N**ew **Y**ork **S**tock **E**xchange
pHMM	**p**attern-based **H**idden **M**arkov **M**odels
PLA	**P**iecewise **L**inear **A**pproximation
ReLU	**R**ectified **L**inear **U**nit
RF	**R**andom **F**orests
RNN	**R**ecurrent **N**eural **N**etworks
SMAC	**S**equential **M**odel-based **A**lgorithm **C**onfiguration
sMAPE	**s**ymmetric **M**ean **A**bsolute **P**ercentage **E**rror
SMBO	**S**equential **M**odel-**B**ased **O**ptimisation
SVM	**S**upport **V**ector **M**achine

SVR	**S**upport **V**ector **R**egression
SWAB	**S**liding **W**indow **A**nd **B**ottom-up
TPE	**T**ree-structured **P**arzen **E**stimation
TreNet	Hybrid LSTM and CNN **Tre**nd prediction **Net**work
UCI	**U**niversity of **C**alifornia **I**rvine
WEKA	**W**aikato **E**nvironment for **K**nowledge **A**nalysis

Dedicated to my dear mother Affoue Afeli, and my dear father Kouame Kouassi, who did not have the opportunity to earn a higher education degree …

1 Introduction

1.1 Background and context

With the advent of low cost sensors and digital transformation, time series data is being generated at an unprecedented speed and volume in a wide range of applications in almost every domain. For example, stock market fluctuations, computer cluster traces, medical and biological experimental observations, sensor networks readings are all represented in time series. Consequently, there is an enormous interest in analysing time series data, which has resulted in a large number of studies on new methodologies for indexing, classifying, clustering, summarising, and predicting time series data [1]–[5].

In certain time series prediction applications, predicting the next trend is preferred over predicting just the next value in the series [3], [4]. Segmenting the time series into a sequence of trend lines - also known as piecewise linear approximation (PLA) [1] - can provide a better representation for the underlying semantics and dynamics of the generating process of a non-stationary and dynamic time series[3], [4]. Non-stationary and dynamic time series are time series whose properties change over time. Moreover, trend lines are a more natural representation for predicting change points in the data, which may be more interesting for decision making. For example, suppose a share price in the stock market is currently rising. A trader in the stock market would ask "How long will it take and at what price will the share price peak and when will the price start dropping?" Another example application is for predicting daily household electricity consumption. Here the user may be more interested in identifying the time, scale and duration of peak or low energy consumption.

Traditional trend prediction approaches include Hidden Markov Models (HMM)s [4], [6] and multi-step ahead predictions [7]. Leveraging the success of deep neural networks (DNN) in computer vision and natural language processing [8]–[10], Lin et al. [3] proposed a hybrid Long-Short Term Memory (LSTM) and Convolutional Neural Networks (CNN) approach, called TreNet. TreNet predicts the next trend of a time series as a PLA (*trend line*) with a slope and a duration. The authors show that TreNet outperforms support vector regression (SVR), CNN, LSTM, pattern-based Hidden Markov Models (pHMM)s [4], and cascaded CNN and LSTM. However, the study had certain limitations as explained below:

Inadequacy of cross-validation: The study used standard cross-validation with random shuffling. This implies that data instances, which are generated after a given validation set, are used for training [11].

No model update: In real world applications where systems are often dynamic, models become outdated and must be updated as new data becomes available. TreNet's test error was estimated on a single hold-out set, which assumes that the system under consideration is static. TreNet's evaluation therefore does not provide a sufficiently robust performance measure for datasets that are erratic and non-stationary [11].

No evaluation of model stability: DNNs, as a result of random initialisation and possibly other random parameter settings could yield substantially different results when re-run with the same hyperparameter values on the same dataset. Thus, it is crucial that the best DNN configurations should be stable, i.e. have minimal deviation from the mean test loss across multiple runs. There is no evidence that this was done for TreNet.

The above three limitations of TreNet evaluation method are named *validation problem*.

Missing implementation details: Important implementation details in the TreNet study are not stated explicitly. For instance, the segmentation method used to transform the raw time series into trend lines is not apparent. This questions the reproducibility of the TreNet study.

Manual ASHO: In the TreNet paper and previous trend prediction research [3], [4], the algorithm selection and hyperparameter optimisation (ASHO) is performed manually. Manual ASHO is expensive and often results in a sub-optimal or mediocre model because it needs extensive experimentation as well as domain specific and machine learning (ML) expert knowledge.

1.2 Problem statement

The experimental study of TreNet showed the potential of DNNs for predicting trends in time series data. However, the study used an inadequate validation method; it did not perform model update; it did not evaluate model stability; and the paper omitted important implementation details. Besides, ensemble regression models such as Random Forests (RF) [12], and Gradient Boosting Machines (GBM) [13], which are very widely used traditional ML algorithms [12] [13] [14], [15], were not included in their baselines. Similar to other ML tasks, finding the optimal trend prediction model and its optimal hyperparameter values for a particular time series requires extensive experimentation by an ML expert. It often requires information about the characteristics of that time series. In some cases, the resulting predictive model may still be sub-optimal because the search space is very large and experts have limited time and resources to conduct the necessary experiments. The ASHO is a sub-problem of the larger field of automated machine learning (AutoML) [16], [17]. Besides evolutionary algorithms [18], some of the promising AutoML methods are: the model-based approaches such as Bayesian Optimisation (BO) [17], and the multi-armed bandit approaches such as HyperBand (HB) [19]. Recently BOHB (Bayesian Optimisation and HyperBand), a hybrid BO and HB, was proposed [20] and was shown to outperform BO and HB. BOHB has not yet been applied to time series trend prediction. Previous trend prediction studies performed the ASHO manually [3], [4].

It is important to address the highlighted limitations of TreNet, and compare TreNet to other approaches including RF and GBM to determine the optimal trend prediction model with a more appropriate validation method. Furthermore, the evaluation of BOHB for automating the ASHO for trend prediction has the potential to accelerate time series research and application.

This study therefore implements and evaluates TreNet with a walk-forward validation method which is more appropriate for time series data. A walk-forward validation with successive and overlapping partitioning (see section 2.1.5) is better suited for evaluating and comparing model performance on time series data [21]. TreNet is then compared with vanilla DNN models as well as traditional ML algorithms. Finally, BOHB is implemented and evaluated for automating ASHO for predicting trends in times series.

1.3 Aim and objectives

The overall aim of this work is to evaluate different machine learning techniques for predicting trends in time series data. The specific research objectives are to:

1. Develop and evaluate the performance and the stability of a hybrid deep neural network, i.e. TreNet for predicting trends in time series data, taking into account model update and model stability.

2. Compare the hybrid deep neural network trend prediction approach with the vanilla deep neural network approaches.

3. Compare the deep neural network trend prediction approaches with the traditional machine learning approaches.

4. Evaluate the effect of the addition of trend line features to raw point data features on the performance of non-hybrid algorithms for predicting trends in time series data.

5. Explore the potential for AutoML tools to perform algorithm selection and hyperparameter optimisation for predicting trends in time series data.

1.4 Research methodology

The study is conducted on four different time series datasets including the three datasets used in the original TreNet paper. The first dataset is a series of voltage recordings of an individual household power consumption. The second dataset is a series of methane concentration in air from a gas sensor. The third and the four datasets are stock market closing prices of the composite New York Stock Exchange (NYSE) and the composite Johannesburg Stock Exchange (JSE) respectively. TreNet was not benchmarked on the JSE dataset.

Seven ML models namely TreNet, LSTM, CNN, MultiLayer Perceptrons (MLP), RF, GBM, and SVR are implemented and compared for predicting trends in time series data.

The TreNet, LSTM, CNN, MLP are DNNs, RF and GBM are tree-based ensemble methods, and SVR is a kernel-based method. Both the ensemble methods and the kernel-based method are traditional ML algorithms. The DNNs are implemented with Pytorch [22]; RF, and SVR with sklearn [23]; and GBM with LightGBM[1]. The hybrid TreNet model takes in both raw point data and trend line features, whereas the other six non-hybrid models take in just raw point data features. The study investigates the impact of supplementing the raw point data with trend line features on the performance of the non-hybrid models.

The models are evaluated using the walk-forward evaluation method [21] instead of the standard cross-validation used in the TreNet study [3]. The walk-forward validation maintains the order of a time series sequence and deals with changes in its properties over time [21]. It also includes model update (see section 2.1.5 for more details). To evaluate the robustness of the models, each experiment is run independently 10 times with random seeds. The variance in the performance metric determines the model stability. Similar to Lin at al. [3], the Root Mean Square Error (RMSE) is used as the performance metric. In general, the performance of the models are reported as the mean and the standard deviation of the RMSE values of the slope and the duration. However, the equally weighted average slope and duration RMSE value is used as a single value metric for the comparisons.

Two ASHO approaches are explored: a manual tuning and an AutoML approach. The manual ASHO is performaned through hand-crafted experiments for each dataset. The hybrid BOHB is chosen, based on a literature survey, to implement the automated ASHO. BOHB combines the strengths of both a model-based approach and a multi-armed bandit approach. BOHB is implemented using HpBandSter[2] framework.

The focus of this dissertation is on predicting trends in time series data. It does not deal with the broader time series prediction problem, which forecasts the next data point. The study also focuses on machine learning methods. It does not explore statistical methods such as Autoregressive Integrated Moving Average (ARIMA) models, Generalised Autoregressive Conditional Heteroscedastic (GARCH) [11], [24].

1.5 Contribution of the study

This study highlights the importance of using an appropriate validation strategy, which takes into account model update and model stability, when dealing with time series data. With such an evaluation method, TreNet generally performs better than other models such as LSTM and RF, but not always so as suggested in the original TreNet paper [3]. No previous work was found which used RF and GBM for predicting trends in time series data, and this may be the first study to do this. The study also demonstrates how AutoML tools, specifically the HpBandSter framework, can be effectively used to automatically find the best DNN configuration. The hyperparameter configuration search

[1] https://lightgbm.readthedocs.io/en/latest/index.html
[2] https://github.com/automl/HpBandSter

space identifies key hyperparameter variables and ranges that most impacted model performance across all datasets during the manual experiments. The configuration file is made publicly accessible[3] to allow researchers and practitioners to replicate these results and evaluate this approach on other datasets. Thus, it decreases the barrier to entry for developing ML models. The research therefore has the potential to contribute to the expansion of the ML user community by allowing users without expert knowledge and experience in ML to rapidly build and evaluate ML models for predicting trends in time series data.

1.6 Dissertation outline

The remainder of the dissertation is structured as follows:

- Chapter 2 - **Literature review**: This chapter provides an overview of the machine learning process for time series prediction, and reviews the relevant literature on trend prediction and automated machine learning.
- Chapter 3 - **Experimental design**: This chapter describes the datasets, the design, the methods used for the experiments, and the implementation of BOHB for ASHO.
- Chapter 4 - **Manual trend prediction experiments**: This chapter describe manual analysis experiments, provides their results and discusses the findings. The limitations of TreNet are addressed, and compared to vanilla DNN models. Then DNN models are compared to traditional ML models.
- Chapter 5 - **Automated trend prediction experiments**: This chapter describes the automated trend prediction experiments, provides the results and discusses the findings.
- Chapter 6 - **Discusion and Conclusions**: This chapter starts by summarising the findings of the research, discusses to what extent they achieve the aim and the objectives of the dissertation, and reflects on the methods. It proceeds with the impact of the research, and highlights the limitations and the opportunities for future work.

[3]https://github.com/h-kouame/configuration-space-of-auto-cash
[3]https://automl.github.io/HpBandSter/build/html/best_practices.html

2 Literature review

The literature review begins with the ML process for time series prediction. It then provides a survey of the different techniques for predicting trends in time series data. It proceeds to introduce the ASHO problem in ML. Recent AutoML techniques for solving the ASHO problem are presented and analysed. The chapter ends by discussing the ASHO for predicting trends in time series data.

2.1 Machine learning for time series prediction

2.1.1 Machine learning process

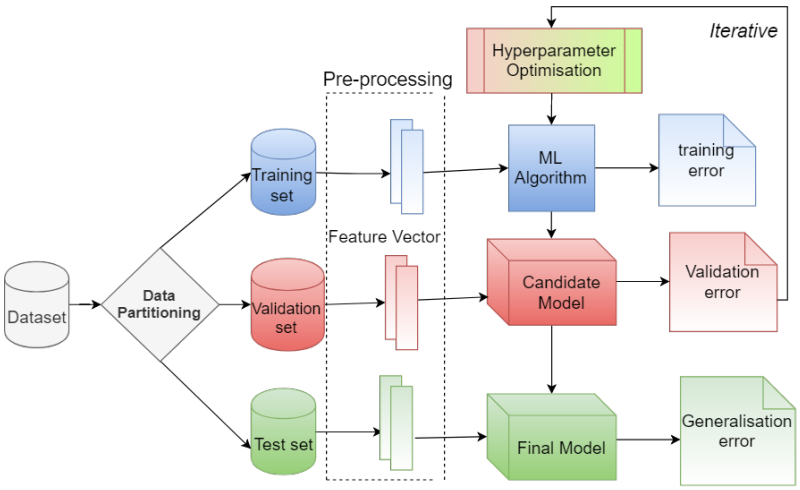

FIGURE 2.1: Machine learning process

Figure 2.1 shows an outline of a typical ML process. After acquiring the dataset, the first step consists of pre-processing and partitioning it into training, validation and test sets [16], [25]. These sets are respectively used to train, tune the hyperparameters and evaluate the model. The hyperparameter tuning or model selection, which is a very expensive process, is traditionally performed manually but may be automated by a meta-process [26].

2.1.2 Time series data characteristics

Time series present many characteristics such as trend, seasonality, periodicity, serial correlation, skewness, kurtosis, chaos, nonlinearity, and self-similarity [27], [28]. The *trend* of a time series refers to the long term change in the mean level [28]. The *seasonality* is a

pattern that repeats itself over fixed intervals of time [28]. The length of these intervals represents the *periodicity* of a seasonal time series. The *skewness* of a time series measures its degree of symmetry of its distribution around its mean value [28]. The skewness for a normal distribution is zero, and any symmetric data should have the skewness near zero. Negative values for the skewness indicate data that are skewed left, and positive values for the skewness indicate data that are skewed right. In other words, left skewness means that the left tail is heavier than the right tail. Similarly, right skewness means the right tail is heavier than the left tail. The *kurtosis* is a measure of whether the data are peaked or flat, relative to a normal distribution. A data set with high kurtosis tends to have a distinct peak near the mean, decline rather rapidly, and have heavy tails. Datasets with low kurtosis tend to have a flat top near the mean rather than a sharp peak [28].

2.1.3 Preprocessing

Similar to most ML applications, time series prediction requires the raw data to be processed into meaningful input features. This preprocessing takes on various forms depending on the application, and the properties of the time series. Thus, the raw data may first be subject to a log-transformation, deseasonalisation, detrending, moving average, and scaling [29], [30]. In addition to the raw data points, other input features include PLA [31], shapelets [5], [32] and discrete Fourier transform or discrete wavelet transform [33]. The aim of these techniques is to reduce the dimensionality and/or discover useful features with predictive capabilities. Regardless of the features used, the sliding window technique is used to create the feature vectors that are fed into the ML algorithms [11], [24], [29], [30], [34]–[36], based on the assumption that the data generating mechanism is autoregressive. Thus, given a time series $x_1, x_2, ..., x_T$, the sliding window transforms a time series into input-output vector pairs (X_t, Y_t) such that: $X_t = (x_{t-w}, x_{t-w+1}, ..., x_t)$ and $Y_t = (x_{t+1}, x_{t+2}, ..., x_{t+M})$ where w $\rightarrow$ *window size* and $M \rightarrow$ *the number of steps ahead prediction*. In effect, each feature vector pair consists of w lagged-values and M lead-values. Figure 2.2 [37] is an illustration of this preprocessing technique for 1 step ahead prediction and window size of 20. The output of the preprocessing step is fed into an ML

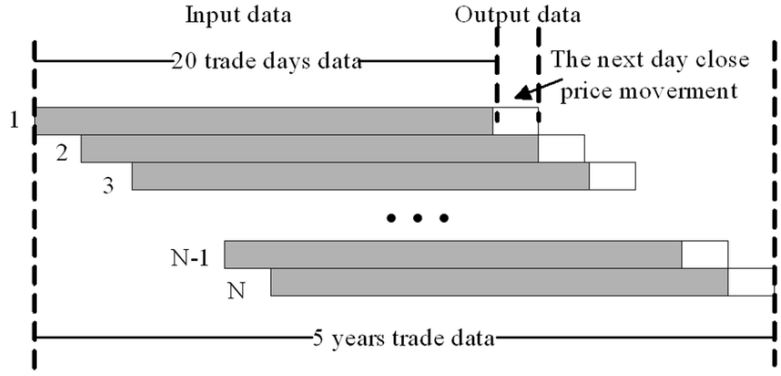

FIGURE 2.2: Sliding window of size 20 for a 1-step ahead stock market price movement prediction. (copied from [37])

algorithm for learning. The next sub-section outlines some of the widely used learning algorithms for time series data.

2.1.4 Widely used algorithms

Statistical techniques such as exponential smoothing, autoregressive (AR) models, moving average (MA), ARIMA models, GARCH are widely used in time series forecasting [11], [24]. To model the non-linearity property of real-world time series, these traditional methods require the specification of non-linear functional relationship between variables. This model-based approach is very limiting. Thus, ML based approaches are increasingly becoming more prominent [3], [35], [38], [39] in the field. There is a variety of ML algorithms for time series prediction. Some of the most common techniques include neural network based algorithms such as MLP [34], [40], LSTM-RNN [10], [39], CNN [3], and hybrid LSTM-CNN techniques such as TreNet [3]. Ensemble techniques such as RF [12] and GBM [13] are also prominent in the time series literature [14], [15]. Finally, kernel based methods such as SVR are often used as baseline [3].

Multilayer perceptrons

Multilayer perceptrons (MLP) are a class of universal approximators [41], which are able to discover and learn complex patterns from data [34]. Figure 2.3 [34] shows a typical feedforward MLP structure with a single hidden layer. MLP have been widely used for

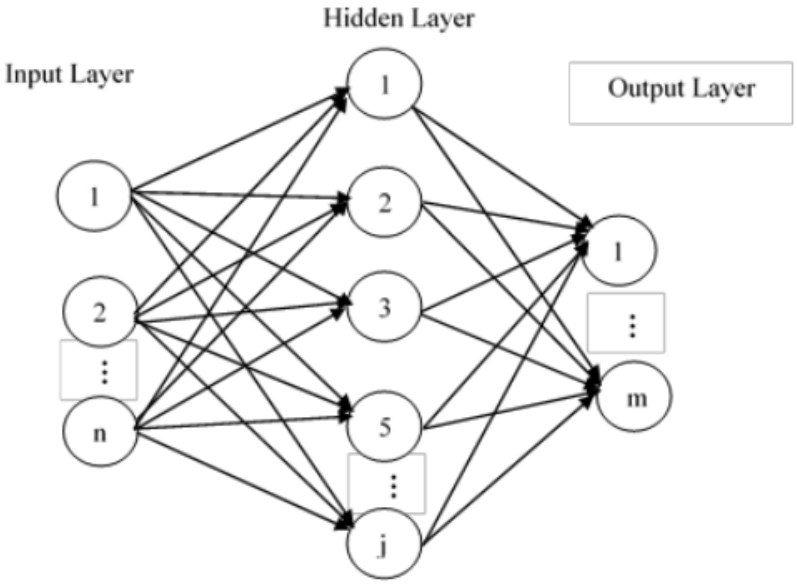

FIGURE 2.3: Multilayer perceptrons with a single hidden layer (copied from [34])

time series prediction [29], [34], [38], [42]. Ayankoya et at. [34] used MLPs for predicting futures contract prices of white maize. Given their populariy, Aras and Kocakoç [42] proposed a new model strategy for MLP model selection in time series prediction. So did Sandy and Balkin [38] focusing on univariate time series. Moreover, Ahmed et al. [29]

performed an empirical comparison of ML techniques including MLP and concluded that MLP are among the best performing techniques for time series prediction. MLPs are popular not only because of their universal approximation property - with just one hidden layer [41], but also, because they do not in theory require extensive feature engineering: they are capable of learning the features themselves. However, they are highly parametrised [29] and therefore, require the correct hyperparameters to be set for a given problem. For instance, gradient descent, one of the prominent optimisation algorithm used to train neural networks, is very sensitive to the learning rate. Thus, there is a crucial need to optimise the hyperparameters of the model for a given data to ensure an optimal generalisation ability of the model.

Long-short term memory

RNN, a special type of neural networks with recurrent properties, have been proven successful for sequence data with dependencies [3], [9], [10]. This success is partly due to long-short term memory (LSTM) RNNs. LSTMs are a special class of sophisticated gated RNNs, introduced by Hochreiter and Schmidhuber [43] in 1997 to solve the problem of long-term dependencies, the problem of vanishing, and exploding gradients present in RNNs [43]. Figure 2.4 shows how three LSTM layers can be stacked to form an ensemble

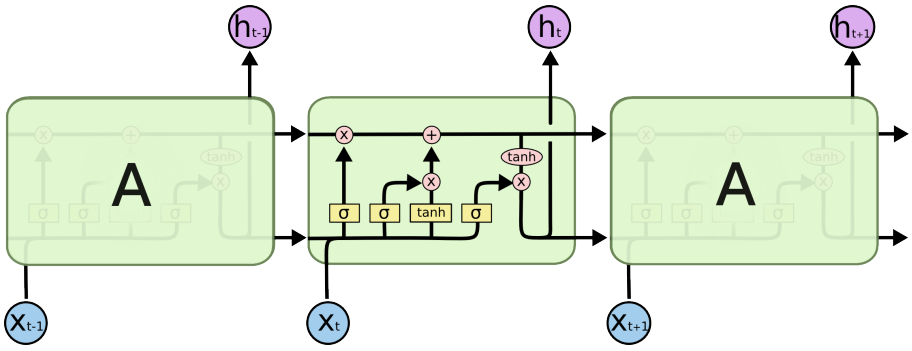

FIGURE 2.4: The repeating module in an LSTM contains four interacting layers (copied from [1]).

with the middle model showing the internal gates of a typical LSTM module. Lipton et al. [44] used LSTMs for multi-label classification of diagnoses. Malhotra et al. [45] evaluated their ability to detect anomalies in ECG time series. Guo et al. [10] used LSTMs for online time series prediction and Lin et al. [3] proposed TreNet using LSTMs for time series trend prediction. Their promising results can be explained by their ability to learn hidden long-term dependencies in sequence data [9]. However, they are generally harder to train because of the need to model long-term dependencies [46].

Convolutional neural networks

Introduced in 1989 by LeCun et al. [8], convolutional neural network (CNN) are specialised neural networks, which have shown great performance mainly on image data [8], [47]–[50]. Figure 2.5 gives an example of a CNN architecture for digit recognition.

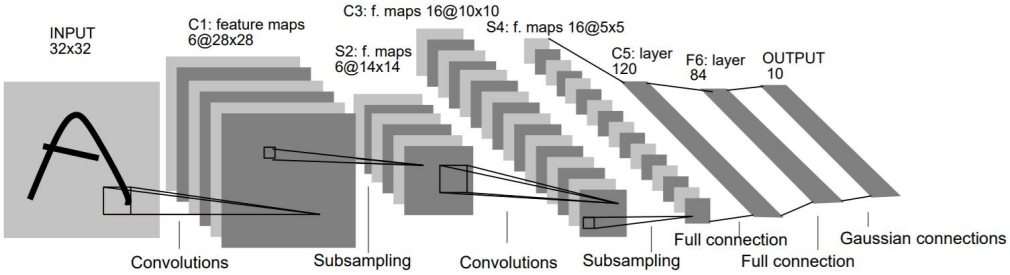

FIGURE 2.5: Architecture of LeNet-5, a convolution neural network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical. (copied from [8])

The convolutions and subsampling operations form a CNN layer. Thus, a CNN layer is composed of a convolution layer, a pooling layer, and optionally an activation non-linear function such as ReLU [51], tanh or sigmoid. For example, LeNet used sigmoid activation functions [8]. The convolution layer consists of one or more filters also known as kernels. These kernels convolve the input data to extract useful low-level features such as edges, textures for images. Each filter is characterised by a kernel size. The pooling layer is an aggregation operation used to reduce the dimension of the features extracted by the convolution layer. As such, given a pool of values, the max pooling type outputs the maximum value; the average pooling, the average of the values. Based on the network structure, the pooling layer can be optional, that is the identity pooling type. The max pooling and the average pooling type require a pooling size to be specified. Recent studies have shown that CNNs can yield comparative performance to RNN on sequential data with more efficient training [52], [53]. Thus, they have become an alternative [54] or a complement to LSTM-RNN for time series data [3].

Support vector regression

Support vector regression (SVR) is the regression version of the prominent traditional classification method called support vector machine (SVM) [55]. SVRs use kernels to transform the input variables into a high-dimensional feature space and penalises the complexity of the model by adding a penalty term to the error function [29]. Contrary to MLPs, SVRs have a strong theoretically foundation, so they are constantly used as a baseline technique for time series prediction [3]. Generally, their performance is closely related to choosing the correct kernel that suits the problem at hand. However, for direct trend prediction, the average performance of radial basis kernel was better than other kernels tested, i.e. the sigmoid and the polynomial kernels [3].

[1] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Random forests

Bagging [56] and boosting [57] methods are two-well known ensemble methods for both classification and regression problems [58]. Bagging methods build successive trees independently using a bootstrap sample of the dataset and take a simple majority vote for prediction [58]. Introduced in 2001 by Breiman [12], random forests (RF), which are a bagging ensemble method, make two changes to the standard trees. They construct each tree using a different bootstrap sample of the data, and they split each node using the best among a subset of predictors randomly chosen at that node, contrary to the standard trees, which use the best split among all variables [58]. RFs are robust to overfitting and compare very well against SVR, MLP [12]. RFs are used for time series applications such stock market movement prediction [14], [15].

Gradient boosting machines

Gradient boosting machines (GBM) are boosting methods, which are type of ensemble methods. As opposed to bagging methods, which build trees independently, boosting methods build successive trees give extra weight to points incorrectly predicted by earlier predictors. Then, a final weighted vote is taken for the prediction [58]. GBMs are used for both classification and regression tasks. The GBM regression was explicitly presented by Friedman in 2001 [13], and have since been applied to various problems including time series data.

2.1.5 Time series prediction model evaluation

The evaluation of ML models is crucial for measuring the generalisation performance and the reliability of the system. It also ensures a correct and effective model selection and hyperparameter optimisation. To effectively compare state-of-the-art methods, a common agreed upon evaluation technique on benchmark datasets is generally required. In time series prediction, there is no such consensus in the literature [11]. Some of the frequently used evaluation methods are described below.

Traditional time series evaluation

In traditional forecasting, a portion of the end of the time series is held-out for testing and the remaining sequence is used for training the model [11], [59]. The training set may be further divided into training subset and validation set if model hyperparameter tuning is necessary depending on the ML algorithm. This technique, illustrated in figure 2.6, is similar to a hold-out cross-validation without randomly splitting the data instances. Although this evaluation procedure does not suffer from the theoretical problems such as temporal evolutionary effects and dependencies within the data, it may result in sub-optimal model because it does not make use of all the available data during training [11].

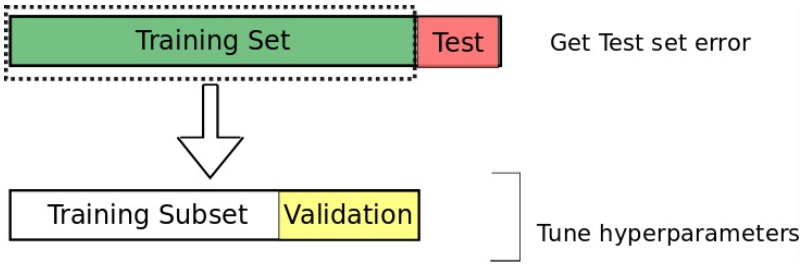

FIGURE 2.6: Traditional time series evaluation with validation set (copied from [2]).

The standard cross-validation

To make use of all the training data during model development, cross-validation is often used to evaluate ML models. The most common cross-validation method named k-fold cross-validation, randomly splits the entire dataset into k different subsets. It then trains the model k times, using k-1 subsets as training set and the remaining subset as validation set for each training iteration. Each of the k-subset serves as validation set once. Thus, every data instance is effectively used for both training and test. The final model error is an average value of all the k test errors. The random partitioning is based on the independent assumption between data instances. Theoretically, this assumption invalidates the standard k-cross-validation with random partitioning when applied to time series data, where the autoregressive property of the sequence is generally assumed. This point is even stronger for non-stationary time series data, which exhibits temporal evolutionary dynamics [11].

Walk-forward evaluation

A more suitable data partitioning of time series data is the successive training-validation-test sets in an overlapping fashion [60] as illustrated in figure 2.7. This partitioning

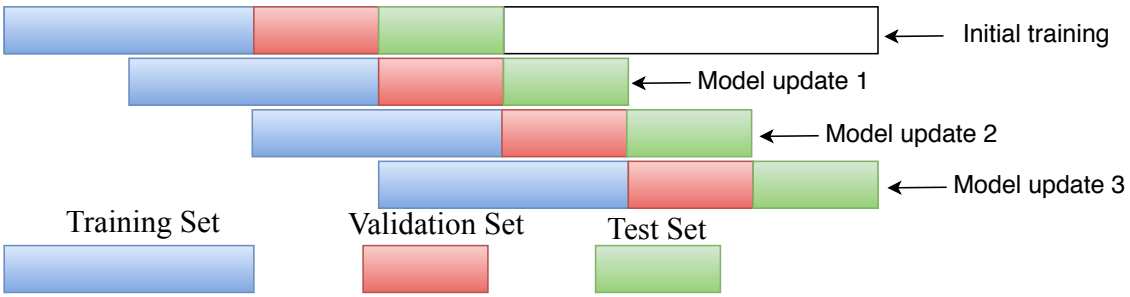

FIGURE 2.7: An example of successive training validation test sets in overlapping partition

method ensures that for a given validation and test data, not only historical but recent data is used to trained the model. It therefore deals with the potential temporal changes in the properties of the data. The multiple partitions and evaluation in a walk-forward fashion evaluates the performance of the model over time after re-training. Thus, the test size represents the model update period. The model update is the re-training of the

model with new data. This evaluation method with the successive training-validation-test partitioning is referred to as the *walk-forward evaluation* in this work. In the literature, some variants of this evaluation methods are named nested cross-validation [61], rolling-origin [59], or rolling-origin-recalibration [11]. The walk-forward evaluation combines the strengths of the traditional hold-out time series evaluation with those of the standard cross-validation with multiple subsets of data to maximise the use of training data. In effect, the walk-forward time series evaluation can be viewed as performing multiple 1-fold cross-validation with multiple time windows of the same dataset without randomising the data instances. The final model performance is the average of the performances on the test sets. In certain applications, it may be useful to concatenate all the tests sets to compute a single performance metric such as the RMSE.

2.1.6 Performance metrics

To quantify the performance of a regression model, statistical measures are required. The most intuitive measure is the mean absolute error (MAE) in equation 2.1,

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|y_t - y_t^{'}| \tag{2.1}$$

where T is the length of the sequence, y_t is the actual value, and $y_t^{'}$ is the predicted value. Although this measure is easily interpretable, it does not penalise larger error terms. The widely used root mean squared error (RMSE) in equation 2.2 for its "desirable" mathematical properties, solves this problem. However, RMSE and MAE are both scale-dependent.

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - y_t^{'})^2} \tag{2.2}$$

Another performance metric is the Mean Average Percentage Error (MAPE) in equation 2.3

$$MAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{100|y_t - y_t^{'}|}{y_t}. \tag{2.3}$$

The MAPE also has its strengths and weaknesses. It is easily interpretable, scale independent but does not penalise larger error terms, and is not defined when $y_t = 0$. To avoid this issue, the symmetric Mean Average Percentage Error (sMAPE) in equation 2.4 was proposed

$$sMAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{100|y_t - y_t^{'}|}{m_t} \tag{2.4}$$

where

$$m_t = \frac{|y_t| + |y_t^{'}|}{2} \tag{2.5}$$

Table 2.1 summarises the performance metrics, also known as loss functions in the context of optimisation, and their properties.

[2]https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9

TABLE 2.1: Comparison of Loss functions

	MAE	**MAPE**	**sMAPE**	**RMSE**
Interpretability	Intuitive	Intuitive	Not Intuitive	Not Intuitive
Scale-dependency	Scale-dependent	Scale-independent	Scale-independent	Scale-dependent
Large error-term	Not penalised	Not penalised	Not penalised	Penalised
Definition at $y_t = 0$	Defined	Undefined	Defined	Defined

2.2 Predicting trends in time series data

In certain time series prediction applications, predicting the next trend is preferred over predicting just the next value in the series [3], [4]. Segmenting the time series into a sequence of trend lines (also known as piecewise linear approximations [1]) can provide a better representation for the underlying semantics and dynamics of the generating process of a non-stationary and dynamic time series [3], [4]. A trend is an intermediate upward and downward behaviour of the successive observations [3]. It is characterised by a line segment specified by a slope, and duration, i.e. the number of data points covered by the line. Trend prediction can be performed indirectly or directly.

2.2.1 Indirectly learning trends in time series data

Traditionally trend prediction is achieved by performing multistep ahead prediction and fitting the predicted values to obtained the trend [7]. This approach suffers from accumulative prediction errors as noted by Bao et al. [62] and Taieb and Atiya [63]. It is therefore, not a trivial task [64]. An alternative approach is the use of HMMs to discover a predefined number of states in the training set, and model the transition dynamics between these states [4], [6]. The next trend is then predicted by first identifying the current state, which is used to perform the inference. However, HMMs only maintain short-term state dependences because of the memoryless Markov property. Furthermore, specifying the number of states requires task specific knowledge [3]. The multistep-ahead prediction and the p-HHM learn the trend in an indirect fashion.

2.2.2 Directly learning trends in time series data

In light of the limitations of the multistep prediction and the HMMs segmentation approaches, Lin et al. [3] proposed a new approach that focuses on directly learning trends using hybrid neural networks called TreNet. The problem of directly learning trends in time series as formulated by Lin et al. is given next.

Trend prediction formulation

Consider a univariate time series as $X = \{x_1, ..., x_T\}$, where x_t is a real-valued observation at time t. The trend sequence T for X, is denoted by $T = \{< l_1, s_1 >, ..., < s_k, l_k >\}$,

[2]https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9

and is obtained by performing a piecewise linear approximation of X [1]. l_k represents the *duration* and is given by the number of data points covered by trend k and s_k is the slope of the trend expressed as an angle between -90 and 90 degrees. Given a historical time series X and its corresponding trend sequence T, the aim is to predict the *next trend* $< s_{k+1}, l_{k+1} >$.

TreNet

Lin et al. [3] developed a hybrid neural networks architecture dubbed TreNet for predicting trends in time series. The structure of TreNet consists of a CNN layer, a RNN-LSTM layer, and a feature fusion layer. Intuitively, the CNN learns the local features of the time series from local raw data [3]. The local raw data was defined as observations preceding the subsequent trend to be predicted as shown in figure 2.8 [3]. The rationale for

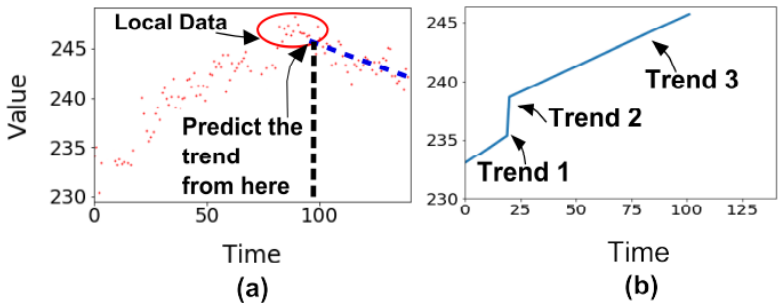

FIGURE 2.8: (a) Trend prediction on time series. (b) Associated sequence of trends (copied from [3])

the LSTM layer was to capture any potential long-range dependency in the sequence of historical trends. Thus, TreNet exploits the different strengths of the CNN and LSTM layers in a complementary fashion. Each layer is trained independently then combined by the fusion layer. This is illustrated by figure 2.9 [3]. Lin et al. reported that TreNet

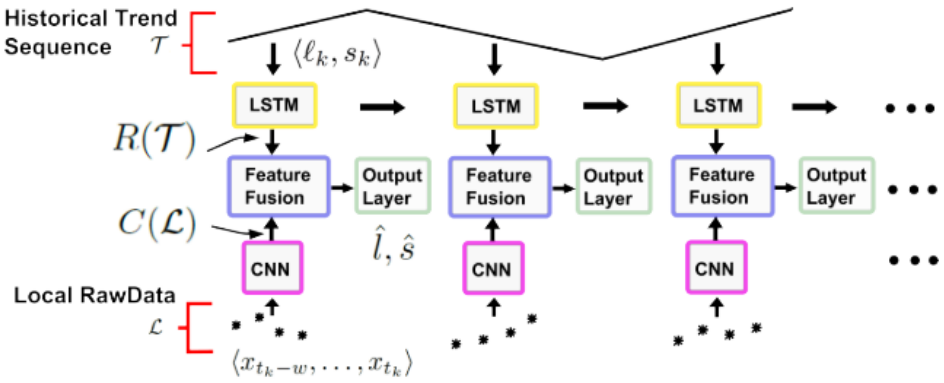

FIGURE 2.9: Illustration of the hybrid architecture of TreNet (copied from [3])

outperformed SVR, CNN, LSTM, pHHM [4], and cascaded CNN and RNN.

2.2.3 Shortcomings of TreNet

Besides exploiting the properties of CNN and LSTM, the novelties introduced by Lin et al. include the use of both the local data and the historical trends as features. Together, they are meant to provide both a local and a global view of the time series. Thus, to make inferences, the historical time series needs to be segmented into its constituant piecewise linear trends first. This may present some practical challenges especially for applications that require online inferences from a continuous stream of data. This is because the two basic PLA algorithms (bottom-up 2.2.4 and top-down 2.2.4) that produce the best approximations are offline [1], [2], [31]. Although Keogh et al. [1] proposed a competitive online hybrid PLA algorithm dubbed SWAB (Slidding Window And Bottom-up), the online segmentation may require a lot of computation resources for large time series. Thus, being able to learn trends directly using raw data as features may be advantageous mainly for common applications that require online inferences from a continuous stream of data because, this will remove the necessity of performing online segmentation during inference.

Furthermore, the TreNet paper as presented in [3] by Lin et al. has some limitations. Firstly, the pre-processing method used to segment the time series into trend lines was not specified nor was the error tolerance. This makes it difficult to replicate their results and therefore, indicates a need to establish benchmark datasets.

Moreover, some hyperparameter choices were not explicitly justified. For instance, they used 600 memory cells in LSTM but did not indicate how and why they chose this hyperparameter value. Thus, there may be a potential to improve the reported results by better exploring the hyperparameter space for each dataset.

Finally, the paper stated the following: "We then do **random shuffling** over such data instances, where 10% of the data instances is held out as the test dataset and the rest is used for cross-validation" [3]. This shows that their evaluation procedure did not consider the possible dependencies present in the data by shuffling the data and using the standard cross-validation instead of nested cross-validation as explained in 2.1.5. Besides, DNNs, as a result of random initialisation and possibly other random parameter settings could yield substantially different results when re-run with the same hyperparameter values on the same dataset. Thus, it is crucial that the best DNN configurations should be stable, i.e. have minimal deviation from the mean test loss across multiple runs. There is no evidence that this was done for TreNet. The limitations of the original TreNet paper show the need to replicate the study with an appropriate validation method which deals with model update and model stability, and re-compare TreNet's performance to other ML approaches.

2.2.4 Segmentation of time series into trends

In many time series applications, the raw data is summaried in an efficient and effective representation such as Fourier transforms [31], [65], wavelets [66], symbolic mappings [67] and PLA [31]. PLA is one of the most widely used representations [2], [4], [68]–[81].

For instance, Ge and Smyth [68] used PLA for change-point detection in semiconductor manufacturing; Hunter and McIntosh for knowledge-based event detection in complex time series; Koski et al. [70] for the recognition of ECG signals; Park et al. [76] for subsequence searches in sequence databases; Wang et al. [4] for finding semantics in time series data using patterned-based hidden Markov models. Piecewise linear approximation techniques [1] are used to segment raw time series into trends. The PLA methods approximate a window of data points with a line by joining the two end-points through interpolation or regression. Kheogh et al. [1] performed an extensive review and empirally compared different PLA techniques. They observed that PLA methods can be categorised into three main approaches: *sliding window*, *bottom-up*, *top-down*. Each approach is described below.

Sliding window

The sliding window approach considers the first data point of the time series as the left anchor of the first trend also known as line segment. Then, the segment grows to the next data point until a residual error threshold is reached. The second segment starts with the next data point not covered by the first segment. This process continues until all the data points are covered. Although this technique may be faster than others because it only considers a local view of the time series, Keogh et al. [1] mentioned the existence of various techniques to make it even faster by increasing the stride size, for example. In addition to being fast, the sliding window approach allows online segmentation. This makes it a convenient method of choice for application with online requirements such as medical data analysis [1]. However, it may result in poorer representation of the initial data point because of its lack of global view of the time series.

Bottom-up

In contrast to the sliding window, this approach considers a global view of the time series in a bottom-up fashion, thus, referred to as bottom-up PLA. The bottom-up algorithms consider the finest possible approximation as a segment: each data point is a segment. These segments are merged until a stopping criterion, such as a number of segments K or a residual error threshold E, is obtained. Compared to the sliding window, this approach may result in a better approximation [1], but, it cannot be used in an online fashion. Keogh et al. [1] proposed a hybrid online algorithm which leverages the strengths of the sliding window and the bottom-up approaches.

Top-down

Similar to the bottom-up approach, the top-down algorithms keep a global view of the time series but in a top-down fashion. The entire time series is initially considered as a single long segment. It is then partitioned until a stopping criterion is reached: a residual error E or maximum number of segments. Because of its global view, it may result in

reasonable approximations. However, similar to the bottom-up, it can only be used offline and can be very slow, although it can be easily parallelised in a divide and conquer fashion.

Table 2.2 provides a summary of the algorithms and their properties based on Table 5 in [1], where E → *Maximum error for a given segment*, ME → *Maximum error for a given segment for the entire time series*, K → *Number of segments*.

TABLE 2.2: Comparison of segmentation algorithms

Algorithm	**Parameters**	**Complexity**	**Parallelisability**	**Online**
Sliding window	E	$O(Ln)$	No	Yes
Bottom-up	E, ME, K	$O(Ln)$	Yes	No
Top-down	E, ME, K	$O(n^2K)$	Yes	No

2.3 Algorithm selection and hyperparameter optimisation

2.3.1 Hyperparameter optimisation

An ML algorithm is a computational procedure that searches a space of hypothesis functions that can best generalise on unseen data. Each hypothesis function is fully defined by a set of parameters that can be learnt using algorithms such as gradient descent. For instance, a neural network is fully specified by its weights matrix W [82]. Before learning the model parameters, some initial parameters need to be specified to determine the structure of the approximation function and the learning configurations. These parameters are referred to as hyperparameters and are set by the modeller or by a metaprocess. For example, the hyperparameters of a SVR include the kernel to use (linear, polynomial, sigmoid...), the degree of a polynomial kernel, the regularization constant C, round-off error ϵ, the tolerance parameter [82] and those of an MLP include the learning rate, the number of layers, the number of neurons per layer, etc.

Some hyperparameters may change during a training epoch [17], for example the learning rate of a neural network may dynamically vary to balance exploration and exploitation during training. Other hyperparameters are conditional on some particular hyperparameters being chosen/active: this gives rise to an hierarchical dependencies between hyperparameters [25]. As an illustration, in SVR the degree of polynomial is conditional on the polynomial kernel being active.

The performance of most ML models is highly dependent on carefully setting the hyperparameters. This raises the hyperparameter optimisation problem, formally defined by Shahriari et al. [17] and Thornton et al. [25].

2.3.2 Algorithm selection

In addition to the hyperparameter optimisation, one faces the problem of algorithm selection. The algorithm selection problem is about selecting the best algorithm from a set of candidate methods (e.g. SVR, MLP, LSTM) that will minimise a given loss function. This problem is justified by the fact that through empirical evidence, it has been established that no single algorithm will ensure the optimal performance on all datasets [16], [42]. This observation is further backed by The No Free Lunch Theorem, which states that any two algorithms are equivalent when their task performance is averaged across all possible problems [83]. A solution to the algorithm selection problem should ensure that the candidate algorithms' hyperparameters are optimised before evaluating their respective performance. Thus, the algorithm selection problem and the hyperparameter optimisation intertwine. This led to the formulation of the joint optimisation problem termed combined algorithm selection and hyperparameter optimisation (CASH) [16], [25].

2.3.3 CASH problem formulation

Thornton et al. [25] and Feurer et al. [16] defined the CASH problem as outlined in definition 2.3.1:

Definition 2.3.1. CASH:
Let $\mathcal{A} = \{A^{(1)}, ..., A^{(R)}\}$ be a set of algorithms, and let the hyperparameters of each algorithm $A^{(j)}$ have domain Λ. Further, let $D_{train} = \{(x_1, y_1), ..., (x_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{D^{(1)}_{valid}, ..., D^{(K)}_{valid}\}$ and $\{D^{(1)}_{train}, ..., D^{(K)}_{train}\}$ such that $D^{(i)}_{train} = D_{train} \backslash D^{(i)}_{valid}$ for $i = 1, ..., K$.
Finally, let $\mathcal{L}(A^{(j)}_{\lambda}, D^{(i)}_{train}, D^{(i)}_{valid})$ denote the loss that algorithm $A^{(j)}$ achieves on $D^{(i)}_{valid}$ when trained on $D^{(i)}_{train}$ with hyperparameters λ. Then, the CASH problem is to find the joint algorithm and hyperparameter setting that minimizes this loss:

$$A^*, \lambda_*, \epsilon = \underset{A^{(j)} \epsilon \mathcal{A}, \lambda \epsilon \Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}(A^{(j)}, D^{(i)}_{train}, D^{(i)}_{valid}). \tag{2.6}$$

The Definition 2.3.1 assumes the standard cross-validation. However, the standard cross-validation can be substituted by an appropriate validation method such as walk-forward validation for time series data.

2.3.4 Approaches to the CASH problem

In principle, the CASH problem can be solved using various approaches such as grid search [84], random search [85], evolutionary algorithms for deep learning algorithms [18], BO [25], and HB [19]. These methods are either sub-optimal, intractable, or they require numerically differentiable hyperparameters, or themselves have various hyperparameters to be tuned. For instance, the "widely used grid search methods for tuning hyperparameters are sample inefficient and computationally expensive" [86]. Random

search on the other hand cannot guarantee finding a good algorithm and hyperparameter configuration, especially in a low resource setting, because of its random nature. A more promising solution is the Bayesian approach. Thus, Thornton et al. [25], developed AUTO-WEKA (Automatic Model Selection and Hyperparameter Optimization in WEKA) to solve the CASH problem using tree-based Bayesian Optimisation (BO) methods with the machine learning framework WEKA [87]. In addition to the variants of Bayesian approaches, a multi-armed bandit approach such as HyperBand (HB), has also emerged in the literature [19]. BO, HB, and BOHB - a hybrid BO and HB are described below.

Sequential model based optimisation

The most promising Bayesian optimisation framework is the sequential model-based optimisation (SMBO) [25]. SMBO can be summarised in three steps. Firstly, it models the dependency between the hyperparameter configurations and their performance measure using a probabilistic model. Secondly, using the current model and an acquisition function, SMBO predicts the most promising candidate configurations of hyperparameters, in closed form. Finally, it evaluates the candidate configuration, then, updates the probabilistic model in a Bayesian fashion. The last two steps are repeated iteratively until a stopping criterion, such as a computational budget or time limit, is met. This procedure is outlined in algorithm 2.1. In a nutshell, SBMO is an approximate search framework

Algorithm 2.1 SMBO

1: initialise model $\mathcal{M}_L; \mathcal{H} \leftarrow \emptyset$
2: **while** time budget for optimization has not been exhausted **do**
3: $\lambda \leftarrow$ candidate configuration from $\mathcal{M}_L$
4: Compute $c = \mathcal{L}(A_\lambda, D^{(i)}_{train}, D^{(i)}_{valid})$
5: $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\lambda, c)\}$
6: Update $\mathcal{M}_L$ given $\mathcal{H}$
7: **end while**
8: **return** λ from $\mathcal{H}$ with minimal c

that maximises the acquisition function, which balances exploration and exploitation of the hyperparameter space. In Auto-WEKA, Thornton et al. [25] successfully used the positive expected improvement over an existing given error rate as acquisition function. There are several SMBO algorithms based on the model class used to fit the relationship between the hyperparameter configuration and the performance measure. For instance, the sequential model-based algorithm configuration (SMAC) may use Gaussian processes (GP) or random forests. In addition to SMAC, other Bayesian optimisation algorithms include tree-structured parzen estimation (TPE) [25], multi-task Bayesian optimisation (MTBO) [88].

In general, the training time of these algorithms take several hours for large datasets depending on the available computational resources. Klein et al. [26] therefore, proposed a Bayesian optimisation dubbed FABOLAS which models loss and training time as a

function of the dataset sizes. Thus, FABOLAS trades off high information gain about the global optimum against computational cost. The authors of FABOLAS recorded that it finds high-quality solutions 10 to 100 times faster than other state-of-the art Bayesian optimisation methods such as Hyperband, described next.

HyperBand

Proposed by Lin et al. [19], HyperBand (HB) is a multi-armed bandit strategy that is faster, when compared to BO. HB takes advantage of the fact that in many applications, the true function f to be learnt can be approximated by a cheap-to-evaluate approximation $\tilde{f}(., b)$, where $b \in [b_{min}, b_{max}]$ is the evaluation budget such as the number of epochs for training DNNs. HB assumes that the quality of the approximation typically increases with the budget b. Thus, $\tilde{f}(., b_{max}) = f(.)$. HB exploits this concept to evaluate multiple hyperparameter configurations on cheaper budgets to determine the most promising ones. This is done by repeatedly calling SuccessiveHalving [89] to find the best of n randomly sampled configurations as shown in Algorithm 2.2 [20]. Then, the promising

Algorithm 2.2 Hyperband

input: budgets b_{min} and b_{max}, η

1: $s_{max} = \lfloor \log_\eta \frac{b_{max}}{b_{min}} \rfloor$
2: **for** $s \in$ **do**
3: sample $n = \lceil \frac{s_{max}+1}{s+1} \cdot \eta^s \rceil$ configurations
4: run SuccessiveHalving [89] on the n configurations with $\eta^s \cdot b_{max}$ as initial budget

configurations are evaluated on higher budgets and eventually on the maximum budget to obtain the true function, i.e. the optimal model. HB is faster compared to BO however, it is not as stable as BO because the initial set of configuration is randomly selected. HB is effective at finding good configurations faster and is parallelisable. However, it is very data hungry and requires maintaining a large pool of pre-selected candidate configurations.

Bayesian optimisation and hyperband

Recently, Falkner et al. [20] proposed Bayesian optimisation and hyperband (BOHB), which combines the strengths of both HB and BO. BOHB makes use of the multi-fidelity evaluation approach of HB but selects the initial set of configurations using a model similar to BO. The authors of BOHB showed that it converges faster and guarantees better results in most cases except against Fabolas which had a similar performance. Moreover, BOHB satisfies the following properties: 1 - a strong anytime performance, 2 - a strong final performance, 3 - an effective use of parallel resources, 4 - scalability, and 5 - robustness and flexibility [20]. BOHB can therefore ensure stable and competitive results when limited compute resources are available.

Summary of some AutoML tools

Table 2.3 outlines some of the existing AutoML tools that can be used to implement automatic algorithm selection and hyperparameter optimisation.

TABLE 2.3: List of tools for implementing algorithms for CASH

Name	Tool	Description
SMAC [90]	SMAC3[3]	A hyperparameter optimiser and model selector implementing SMAC.
Fabolas [26]	RoBO[4]	A Gaussian process based implementation of Fabolas.
Hyperband [91]	RoBO / HpBandSter[5]	A multi-armed bandit hyperparameter optimiser using Successive Halving selection.
TPE & Random [92]	HyperOpt[6]	A Python framework for bayesian optimization with TPE.
MTBO [88]	RoBO	A Python implementation of multi-task Bayesian optimisation.
SMAC, warmstart & ensemble [16]	auto-sklearn[7]	An automated machine learning toolkit and a drop-in replacement for a scikit-learn estimator.
BOHB [20]	HpBandSter	A hybrid Bayesian optimisation and hyperband framework for hyperparameter optimization.

2.4 CASH problem in time series prediction

As mentioned by Thornton et al. [25] the problems of algorithm selection and hyperparameter optimisation have been widely tackled but separately. This remark holds for time series prediction as well. On one hand, to determine which machine learning algorithm best performs on time series data, Ahmed et al. [29] performed an extensive empirical study but did not tackle automatically the model selection problem. On the other hand, having chosen neural networks as algorithm, the automated hyperparameter optimisation problem was investigated by Aras and Kocakoç [42], and Balkin and Ord [38] for time series prediction. Similarly most papers on time series trend prediction select the machine learning algorithm based on expert knowledge and perform extensive or very little experiments, using techniques such as grid search [84] or random search [85], to optimise the hyperparameters [3], [4], [34]. Although frameworks such as BO [25], HB [19], and BOHB [20] have been proposed for solving the CASH problem, there is still some gaps to be filled for time series trend prediction.

2.5 Summary

This chapter reviews ML techniques for time series prediction. It highlights the importance for evaluating ML models on time series data with an appropriate validation method with model update such as the walk-forward evaluation, instead of the standard cross-validation. The chapter introduced the usefulness of predicting trends in time series data for certain applications and described existing approaches including a promising recent hybrid deep neural networks, i.e. TreNet. TreNet's paper showed that it is superior to other methods, but the study has many limitations such as the use of the

[2]https://github.com/automl/SMAC3
[3]https://github.com/automl/RoBO
[4]https://github.com/zygmuntz/hyperband
[5]https://github.com/hyperopt/hyperopt
[6]https://github.com/automl/auto-sklearn
[7]https://github.com/automl/HpBandSter

standard cross-validation to evaluate the model. Furthermore, this chapter described the algorithm selection and hyperparemeter optimisation problem faced by ML practitioners. Some recent AutoML techniques for solving the ASHO problem are described. It is observed that TreNet and previous study on predicting trends in time series performed the algorithm selection and hyperparameter optimisation manually. This chapters therefore shows the necessity to replicate TreNet's experiments with an appropriate validation method and compare its performance to other methods. It also identifies an opportunity to evaluate recent AutoML techniques for automating the ASHO for predicting trends in time series data.

3 Experimental design

This chapter describes the process for predicting trends, and the design of the experiments performed. It starts with an overview of the prediction process. It proceeds with the description of the datasets, and the data preprocessing method. The design of the learning algorithms and the optimisation of their hyperparameters are explained. The chapter continues to describe the data partitioning and the model evaluation. It concludes with an overview of the experiments performed. It is worth pointing out in advance that some design decisions are taken because of practical considerations. For instance, the number of raw local data per dataset described in section 3.3.2 was fixed instead fine-tuning to reduce the number of tuning experiments.

3.1 Overview of the process for predicting trends

Figure 3.1 provides an overview of the process followed for predicting trends in this work. The process starts with acquiring the dataset. The dataset is then segmented into

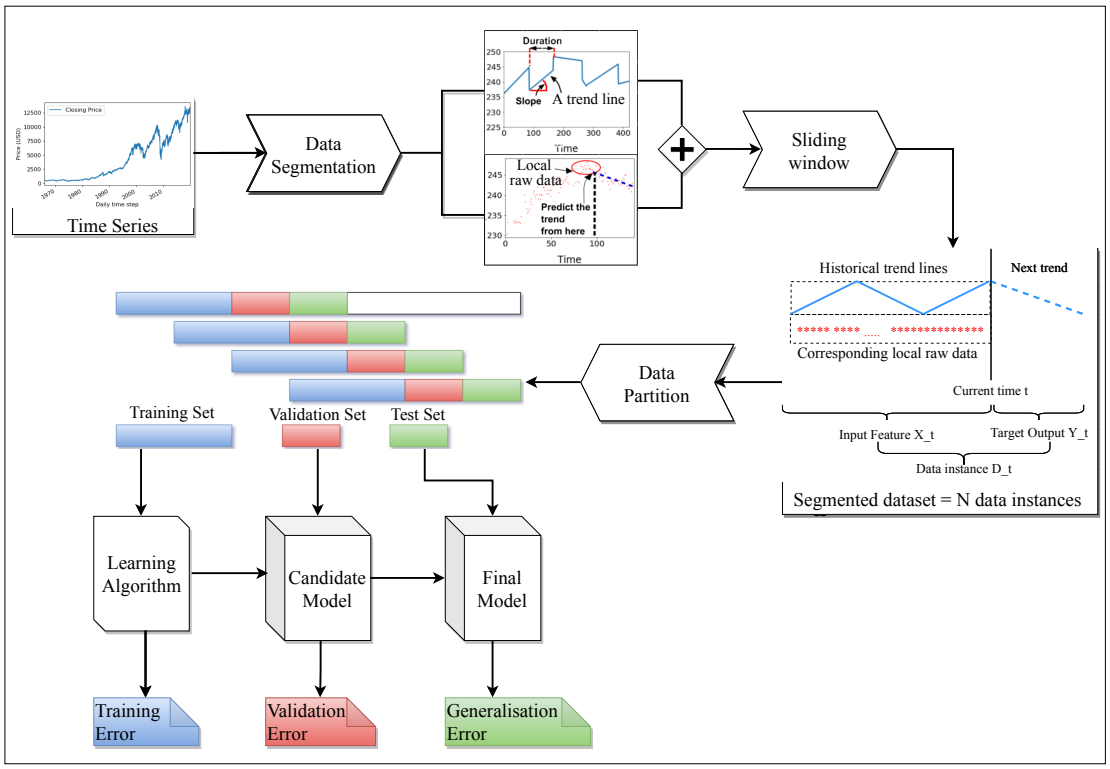

FIGURE 3.1: Overview diagram of trend prediction process

piecewise linear approximations. The output of the segmentation process is a sequence

of trend lines and their corresponding raw local data. The sliding window technique is applied to this sequence to form the data instances, i.e. the input-output pairs. The data instances are partitioned in an overlapping training-validation-test fashion. The ASHO process finds the optimal algorithm and its optimal hyperparameter configuration. More specifically, this process selects a learning algorithm, which is trained using the training sets. The output of the training process is a candidate model, which is evaluated using the validation sets. The outcome of the evaluation is used to select the best model. Finally, the generalisation ability of the final model, i.e. the best model, is estimated on the test sets.

3.2 Datasets

The experiments are conducted on four datasets namely the voltage, the methane, the NYSE, and the JSE datasets. The first three datasets are used in the original TreNet study [3]. The JSE dataset is added to extend the number of datasets.

3.2.1 The voltage dataset

The voltage dataset is obtained from the University of California Irvine (UCI) ML repository[1]. It corresponds to the power consumption dataset used by Lin et al. [3]. It is renamed to be more precise since the complete power consumption dataset contains many other time series. The raw data consists of 2075259 voltage measurements of a house in Sceaux (7km from Paris, France). These measurements were recorded over a period of 47 months - between December 2006 and November 2010 - with a one-minute sampling rate. Figure 3.2 shows the evolution of the voltage dataset over time (top sub-figure) and the probability distribution of its values (bottom sub-figure). The top sub-figure shows that

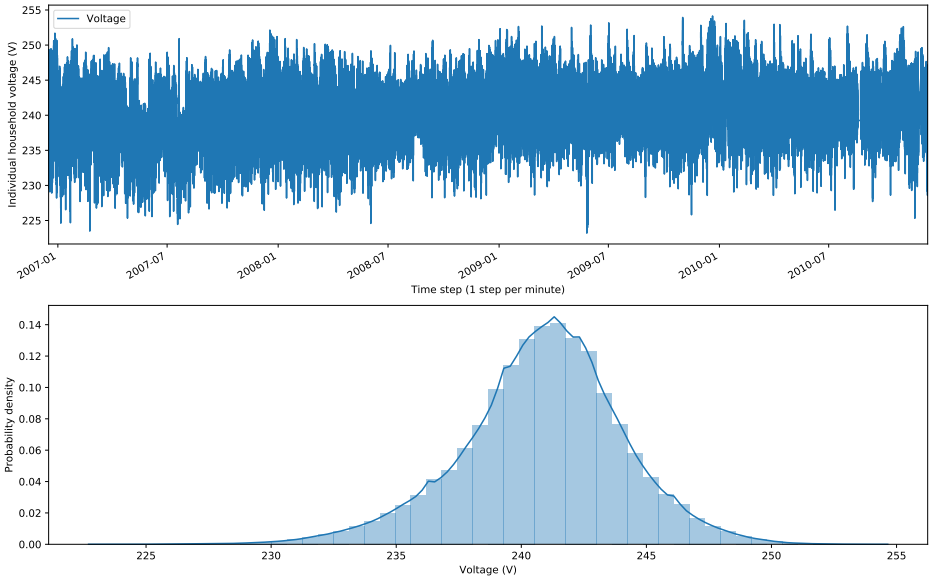

FIGURE 3.2: Top - The individual household voltage dataset. Bottom - Probalility distribution of the voltage dataset.

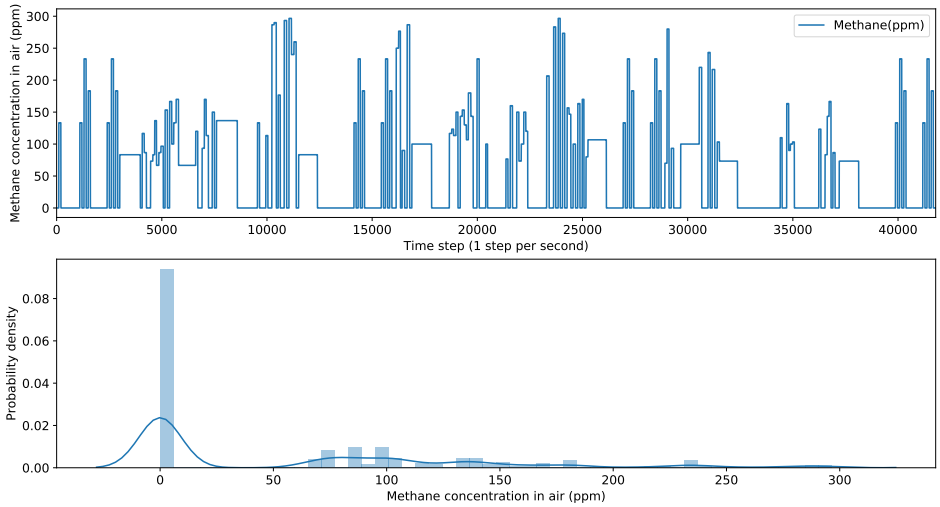

FIGURE 3.3: Top - Methane concentration in air over time. Bottom - Probability distribution of the methane dataset.

the voltage dataset follows the same pattern every year, according to the weather seasons. It also reveals that it is highly volatile, but it has an almost constant trend. A volatile time series values change rapidly in an irregular pattern. The bottom probability distribution int the bottom sub-figure shows that the voltage dataset is normally distributed.

3.2.2 The methane dataset

The methane dataset corresponds to the gas sensor dataset used by Lin et al. [3]. It is renamed for clarity because the gas sensor dataset contains many other series. It is extracted from the UCI ML repository[2]. The original dataset consist of 4178504 measurements of methane concentration in air, which were gathered at a frequency of 100Hz. A resampled set of size of 41786 at a frequency of 1Hz is used in this work. Figure 3.3 shows the evolution of the methane dataset over time (top sub-figure) and the probability distribution of its values (bottom sub-figure). The top sub-figure shows that the methane does not exhibit any clear trend or seasonality. The probability distribution in the bottom sub-figure reveals that the methane dataset is skewed to the right of its mean value. It also shows that the methane dataset has very sharp changes with medium to low volatility.

3.2.3 NYSE dataset

The NYSE dataset is the composite New York Stock Exchange closing price from 31-12-1965 to 15-11-2019. It is extracted from Yahoo finance[3] and contains 13563 raw data points. Figure 3.5 shows the evolution of the NYSE dataset over time (top sub-figure) and the probability distribution of its values (bottom sub-figure). The top sub-figure shows that the NYSE dataset has an almost linear upward trend. Its volatility is very low

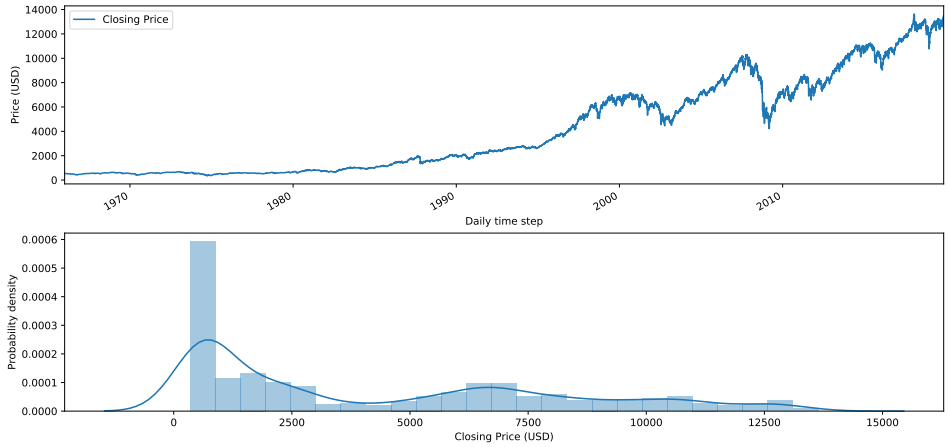

FIGURE 3.4: Top - The composite NYSE closing price dataset. Bottom - Probalility distribution of the NYSE dataset.

FIGURE 3.5: Raw NYSE dataset.

initially until before the year 2000 after which, it becomes very volatile. The probability distribution in the bottom sub-figure shows that the NYSE dataset is skewed to the right.

3.2.4 JSE dataset

The JSE dataset is the composite Johannesburg Stock Exchange closing price from 2007-09-18 to 2019-12-31. It is extracted from Yahoo finance. It is much smaller than the other three datasets, with only 3094 raw data points. Figure 3.6 shows the evolution of the JSE dataset over time (top sub-figure) and the probability distribution of its values (bottom sub-figure). The top sub-figure shows that this stock market dataset is less volatile, compared to the NYSE. The probability distribution in the bottom sub-figure shows that the JSE dataset has a more symmetrical distribution around its mean value. It is not exactly normally distributed like the voltage dataset: it has a flat top and heavy tails on both sides.

The characteristics of the four datasets are summarised in table 3.1. These characteristics are explained in section 2.1.2.

TABLE 3.1: Summary of the characteristics of the datasets.

	Trend	***Seasonality***	***Periodicity***	***Skewness***	***Volatility***
Voltage	almost constant	seasonal	yearly	symmetric	very high
Methane	no clear trend	non-seasonal	N/A	right skewness	medium then low
NYSE	upward	non-seasonal	N/A	right skewness	low then high
JSE	mostly constant	non-seasonal	N/A	almost symmetric	medium then low

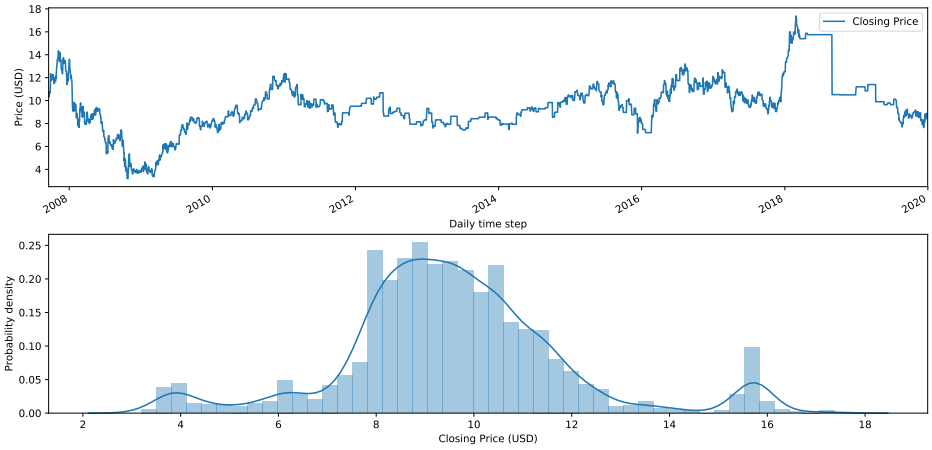

FIGURE 3.6: Top - Composite JSE closing price dataset. Bottom - Probalility distribution of the JSE dataset.

3.3 Preprocessing

The data preprocessing consists of three operations: the missing data imputation, the data segmentation, and the sliding window operation. Each missing data point is replaced with the closest preceding non-missing value. The data segmentation and the sliding window operations are explained next.

3.3.1 Data segmentation

Each dataset is segmented into piecewise linear approximations (PLA)s, referred to as trend lines or simply trends. Similar to Wang et al.[4], the segmentation is performed using the bottom-up technique (see section 2.2.4). As a reminder, a trend line consists of a slope and a duration. The slope of the trend, which is the gradient of the line, is converted to an angle between -90 and 90 degrees. This transformation eases the interpretation of the results [3]. The duration of a trend line is determined by the number of data points it covers. Table 3.2 provides a summary of the basic statistics, i.e. the number of trend lines, the mean, and the standard deviation of the segmented datasets.

3.3.2 Input-ouput data instances

The data instances, i.e. the input-output pairs are formed using the sliding window technique (see section 2.1.3 and figure 2.2). At the current sequence step t, the output is the next trend $T_{k+1} =< s_{k+1}, l_{k+1} >$. The prediction horizon is therefore 1. The input features are the local data points $L_k =< x_{t_k-w}, ..., x_{t_k} >$ for the current trend $T_k =< s_k, l_k >$. The

[1]https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption
[2]https://archive.ics.uci.edu/ml/datasets/gas+sensor+array+under+dynamic+gas+mixtures
[3]https://finance.yahoo.com/

TABLE 3.2: Summary of the basic statistics of the segmented datasets.

	Voltage	**Methane**	**NYSE**	**JSE**
Number of raw data points	2075259	41786	13563	3094
Number of trend lines	42280	4419	10015	1001
Mean ± deviation of the trend slope	-0.21 ± 10.41	0.17 ± 18.12	5.44 ± 81.27	0.21 ± 18.18
Mean ± deviation of the trend duration	50.08 ± 60.36	10.46 ± 67.03	2.35 ± 0.81	4.09 ± 5.23

prediction is based on the actual end of the previous segment not the previously predicted trend since the prediction horizon is 1. The window size w is determined by the duration of the first trend line. For TreNet the input consists of both the local data points L_k (fed into the CNN), and the current trend T_k (fed into the LSTM). Table 3.3 gives a summary of the the input vector size per feature type, and the number of data instances per dataset.

TABLE 3.3: Summary of the input vector size per dataset and per feature type. Raw local data refers to the window size in terms of number of data points. Number of data instances refers to the number total number of data examples available for training and testing

	Voltage	**Methane**	**NYSE**	**JSE**
Raw local data	19	100	4	2
Trend lines	2	2	2	2
Raw local data + Trend line	21	102	6	4
Number of data instances	42279	4418	10014	1001

3.4 Learning algorithm design

This study considered seven different algorithms namely TreNet [3], MLP, LSTM, CNN, RF, GBM, and SVR. TreNet, MLP, LSTM, and CNN are neural networks based algorithms; RF and GBM are ensemble methods; and SVR is a kernel based algorithm. The design of each of each algorithm is explained below.

3.4.1 The hybrid DNN (TreNet)

Figure 2.9 shows the generic architecture of TreNet. Its implimentation in this study is similar to the initial architecture [3] proposed by Lin et al. [3]. More specifically, the LSTM component consisted of a single LSTM layer. The CNN is composed of two stacked [3] 1D convolution layers without pooling layer. The second CNN layer is followed by a Rectified Linear Unit (ReLU) activation function to capture any potential non-linear relationship between the input and the output. The flattened output of the CNN's ReLU layer and the LSTM layer are reduced/projected to the same dimension using a fully connected layer for the fusion operation. The fusion layer consisted of a fully connected

layer that takes the element-wise addition of the projected outputs of the CNN and LSTM components as its input. It outputs the slope and duration values. A dropout layer is added to the layer before the output layer.

3.4.2 MultiLayer perceptrons (MLP)

The MLP algorithm consists of N fully connected neural network layers, where $N \in [1, 5]$. Each layer is followed by a ReLU activation function to capture non-linear patterns in the data. To prevent overfitting, a dropout layer is added after each odd number layer, except the last layer. For instance, if the number of layers $N = 5$, the layer 1 and layer 3 will be followed by a dropout layer. The weights of the network are initialised using the He initialisation technique [93] with normal distribution.

3.4.3 Long short-term memory (LSTM)

The LSTM algorithm is built with N LSTM layers, where $N \in [1, 3]$. Each LSTM is followed by a ReLU activation function to extract non-linear patterns, and a dropout layer to prevent overfitting. After the last LSTM layer, a fully connected neural network layer is added. This layer takes the feature representation extracted by the LSTM layers as its input and predicts the next trend. The fully connected layer is initialised with the He initialisation [93]. To learn long-term relationships between trends, the LSTM layers are not re-initialised at every epoch.

3.4.4 Convolutional neural network (CNN)

The architecture of the CNN algorithm consists of N 1D-convolutional layer, where $N \in [1, 3]$. Each convolutional layer, which consists of a specified number of filters of a given kernel size, is followed by a ReLU activation function, a pooling layer, and a dropout layer to prevent overfitting. The final layer of the CNN algorithm is a fully connected neural network which takes the features extracted by the convolution, activation, pooling, and dropout operations as its input and predicts the next trend. The structure of a one layer CNN is illustrated in figure 3.7. This architecture is inspired by Lin et al's implementation [3]. Both the convolutional and fully connected layers are initialised with the He initialisation technique [93].

3.4.5 Neural network training and regularisation

The Mean Square Error (MSE) is used as loss function when training the neural network based algorithms, similar to Lin et al. [3]. The slope and the duration loss are weighted equally. The Adam optimizer [94] is used for learning the optimal weights of the neural networks. It is selected because it achieves good results faster than other methods [94]. To prevent overfitting, dropout and L2 regularisation are used [3]. The actual value per algorithm per dataset is obtained through tuning. To make the neural networks robust to random initialisation, their weights are initialised using the He initialisation technique

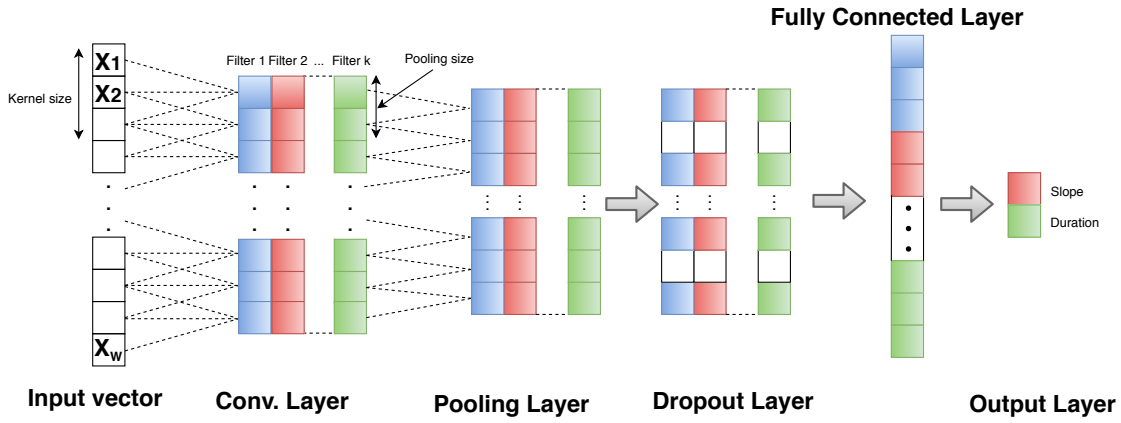

FIGURE 3.7: The structure of a one layer 1D-convolution neural network.

[93] with normal distribution. The ReLU function is selected as the non-linear function. The mode is set to *fan-in*, that is the magnitude of the variance of the weights in the forward pass is preserved.

3.4.6 Random Rorests (RF)

The RF algorithm used is sklearn's random forest regressor [23]. The MSE loss is used for training. The hyperparameters that resulted in performance gain are the *number of estimators*, the *maximum depth*, the *boostrap*, and *warm start* parameters.

3.4.7 Gradient Boosting Machines (GBM)

The LightGBM[4] framework for used to implement the GBM algorithm used in this work. The hyperparameters that improved the performance are the *boosting type*, the *number of estimators*, and the *learning rate*.

3.4.8 Support Vector Regression (SVR)

Sklearn's [23] implementation of the SVR is used in this work. Only the radial basis kernel is used because of their better results [3] and faster training time. Tuning the *gamma* and *C* hyperparameters resulted in noticeable performance gain.

3.5 Model evaluation

3.5.1 Walk-forward evaluation and performance metrics

The walk-forward evaluation procedure, with the successive and overlapping training-validation-test partition [21], is used to evaluate the performance of the models (see section 2.1.5 for more details on the walk-forward evaluation). The input-output data instances are partitioned into training, validation, and test sets in a successive and overlapping fashion [21] as shown in figure 2.7. For the methane and JSE datasets, the combined test sets make up 10% of their total data instances as per the original TreNet experiments

[3]; and 80% and 50% for the voltage and NYSE datasets respectively because of their large sizes. The partition sizes for each dataset are given in table 3.4. The number of partitions is set to 8 for the voltage, 44 for methane, 5 for NYSE and, 101 for the JSE dataset. This determines the number of model updates performed for each dataset. For example, one initial training and 7 (8-1) model updates are performed for the voltage dataset. For DNN models, the neural networks are initialised using the weights of the most recent model, during model update. This makes the training of the network faster without compromising its generalisation ability. More details about this technique which we refer to as *model update with warm-start* are given in section 3.5.2 below.

TABLE 3.4: Summary of the data instance partitioning

	Voltage	**Methane**	**NYSE**	**JSE**
Number of data instances	42279	4418	10014	1001
Chosen total test sets percentage	80%	10%	50%	10%
Chosen test set size	4227	10	1001	1
Number of splits	8	44	5	101
Validation set size	4227	10	1001	1
Training set size	4227	3967	4008	899

The equally weighted average of the RMSE values of the slope and duration is used as the evaluation metric. The RMSE formula is repeated in equation 3.1 as a reminder.

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - y_t^{'})^2} \tag{3.1}$$

where, $y_t \rightarrow$ actual next trend, $y_t^{'} \rightarrow$ predicted next trend, and $T \rightarrow$ number of data instances. In certain cases, calculating the RMSE on the test set of each split and averaging over all the splits may lead to an incorrect estimation of the model error. For example, when the test set size is 1, i.e. $T_{split} = 1$, the RMSE of each split becomes the Absolute Error (AE), as shown in equation 3.3. Averaging this RMSE over all the splits makes the overall RMSE equals to the Mean Absolute Error (MAE) as demonstrated in equation 3.3, where $T = total\ test\ size = number\ of\ splits$.

$$RMSE_{split} = \sqrt{\frac{1}{1}\sum_{t=1}^{1}(y_t - y_t^{'})^2} = \sqrt{(y_t - y_t^{'})^2} = |y_t - y_t^{'}| = AE \tag{3.2}$$

$$RMSE_{overall} = \frac{1}{S}\sum_{t=1}^{S} AE = \frac{1}{S}\sum_{t=1}^{S}|y_t - y_t^{'}| = \frac{1}{T}\sum_{t=1}^{T}|y_t - y_t^{'}| = MAE \tag{3.3}$$

To avoid such incorrect estimation of the RMSE, the predictions for all the test sets is concatenated into a single series and used to calculate the overall RMSE using equation 3.1. Besides the RMSE, the percentage improvement over the naive last value model (LVM) is used in experiment 1. The rationale for this is explained in experiment 1 in section 4.1, where the percentage improvement over the naive LVM is used.

Each experiment is run 10 times and the mean and the standard deviation across the 10 runs are reported. This provides a measure of the stability of the models.

3.5.2 Model update with warm-start

The walk-forward evaluation procedure requires as many training episodes as the number of splits: one initial training and many model updates. This can be computationally very expensive, depending on the number of splits. This is more crucial for DNNs which are more expensive to train. Thus, model update with *warm-start initialisation* is used to reduce the training time of DNNs. The *warm-start initialisation* means that the new network is initialised with the weights of the previous model during model update. In effect, the patterns learnt by the previous network are transferred to the new model, therefore, reducing the number of epochs required to learn the new best function. In practice, the walk-forward evaluation with warm-start corresponds to performing the first training with the maximum number of epochs required to converge, then using a fraction of this number for every other update. This fraction - between 0.0 and 1.0 - becomes an additional hyperparameter dubbed *warm start*. The lowest value that out-performs the model update without warm-start is used as the best value, because this technique is essentially used to speed-up the model updates. This process is illustrated in figure 3.8, for the MLP on the NYSE dataset. 0.7 is chosen as the best warm start value because it is the smallest fraction of the maximum number of epochs that resulted in a loss lower than the loss of the model without warm start - indicated by the red line.

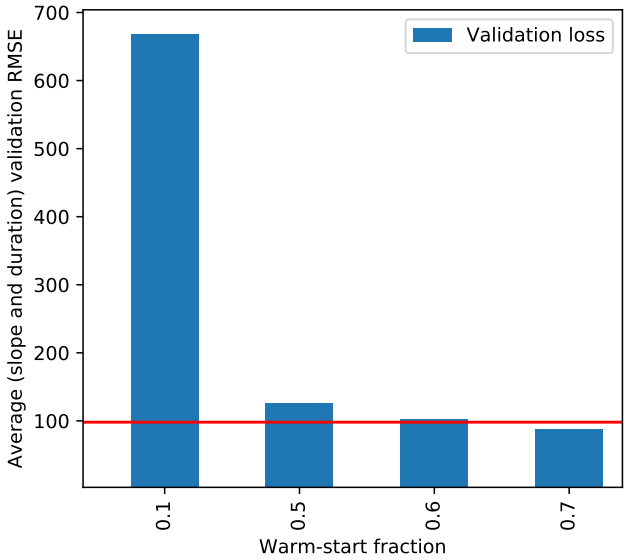

FIGURE 3.8: Warm-start fraction selection for MLP on the NYSE dataset. The red line represents the validation loss without warm-start.

The speed-up, i.e. the expected reduction factor in the total number of epochs can be computed in advance using equation 3.6. The equation 3.6 is derived from equation 3.4 and equation 3.5.

$$E' = E + E \times (S - 1) \times \omega \tag{3.4}$$

$$E' = E \times (1 + (S - 1) \times \omega) \tag{3.5}$$

$$speed\text{-}up = \frac{E}{E'} = \frac{S}{1 + (S - 1) \times \omega} \tag{3.6}$$

Where, $E' \rightarrow$ *Total epochs with warm start*, $E \rightarrow$ *Epochs per split without warm-start*, $S \rightarrow$ *Number of data partition splits*, $\omega \rightarrow$ *warm-start fraction*.

3.6 Algorithm selection and hyperparameter optimisation

This study uses two approaches for ASHO: the traditional manual tuning and an AutoML approach.

3.6.1 Manual tuning

The manual hyperparameter optimisation is done by focusing on one hyperparameter at a time, while keeping all the other hyperparameters constant. Thus, the algorithm is evaluated using the different values of the hyperparameter being optimised. For categorical hyperparameters, an exhaustive evaluation of the search space of that hyperparameter is performed, whereas for continuous hyperperparameters, the initial value, which may be chosen arbitrarily, is decreased or increased while observing the validation loss. The direction that yields a decrease in the validation loss is followed until a budget limit is reached or no more improvement is observed. For each hyperparameter, the value that results in the lowest loss is kept as its best value. Conceptually, this manual hyperparameter optimisation process can be seen as taking a slice through the loss landscape. Starting from a random point, this slice is then searched in both direction to find the optimal value for that hyperparameter. The set of hyperparameters evaluated for each algorithm per dataset is given in the appendix table A.1.

3.6.2 AutoML for ASHO

The manual optimisation is labour intensive, time consuming, and is not systematic. Thus, a recent AutoML technique namely the hybrid BOHB framework is implemented and evaluated to automate ASHO. The implementation of BOHB for predicting trends in time series data is described below.

3.7 AutoML framework and implementation

BOHB is chosen for two main reasons. Firstly, it fulfills five main desiderata: a strong anytime performance; a strong final performance; scalability; and robustness & flexibility [20]. Secondly, it is suitable for applications with low computational resources because it leverages the speed of hyperband [20], [91]. Hyperband is fast because it eliminates suboptimal hyperparameter configurations by performing many low fidelity, i.e. cheaper

[4]https://lightgbm.readthedocs.io/en/latest/index.html

model evaluations on smaller budgets, and fewer high fidelity, i.e. costlier model evaluations on near-optimal configurations.

BOHB is implemented using the HpBandSter[5] framework. HpBandSter requires the specification of a *loss metric*, the *hyperparameter configuration search space*, and a stopping criterion, or *budget*. The *loss metric* is the equally weighted average slope and duration RMSE. The implementation details of the *hyperparameter configuration space* and the *budget* choices are provided in the next sub-sections.

3.7.1 Hyperparameter configuration space

The key hyperparameter variables and ranges that are observed to have the most impact on model performance are analysed and identified, based on the experiences during the manual tuning. They are grouped into 4 types of parameters. Each type is a specification of the structure of the DNN, the training parameters, the regularisation parameters, or the algorithm type. Common hyperparameters such the learning rate, the batch size, and the dropout rate are shared to reduce the search space. The choice of algorithm is represented as a categorical hyperparameter, which splits the search spaces into sub-spaces. Each subspace contains the hyperparameters specific to that algorithm. For instance, the kernel size parameter for the CNN is only activated in the search space when CNN is selected. The entire configuration space consists of the union of sub-spaces, specific to each of the candidate algorithms, and the shared hyperparameter spaces such as the range of learning rate.

The resultant hyperparameter search space consisted of 24 different hyperparameters, 22 that are categorical or discretised, and 2 that are continuous. There is 1 hyperparameter for the algorithm, i.e. MLP, CNN, or LSTM, 6 structural hyperparameters for the MLP, 4 for the LSTM, 9 for the CNN, and 4 common training and regularisation hyperparameters that are shared accross all algorithms. Given the 2 continuous parameters, the number of possible unique configurations is infinite. A summary of the hyperparameter configuration search space is provided in Table 3.5. The full hyperparameter sets/ranges is specified using ConfigSpace [95], and can be accessed in json format via this link[6]. The implementation of BOHB is limited to vanilla DNN for simplicity.

3.7.2 BOHB budget and configuration

The *number of training epochs* of the DNN is used to estimate its fidelity with respect to the true DNN function to be learnt (see literature review on HyperBand in section 2.3.4). Thus, the lowest fidelity model is trained with on the *minimum budget*, i.e. with the minimum number of epochs, and the highest fidelity model is obtained when it is evaluated on the *maximum budget*, i.e. with the maximum number of epochs. The maximum number of epochs from the manual tuning is used to guide the number of epochs required to identify the optimal DNN configuration. For example, if the search space contains a

[5] https://github.com/automl/HpBandSter

[6] https://github.com/h-kouame/configuration-space-of-auto-cash

TABLE 3.5: Summary of the hyperparameters in the configuration space

MLP/CNN/LSTM hyperparameter	Value type	Type
Algorithm	Categorical	Algorithm
Number of hidden/CNN/LSTM layers	Discrete	Structure
Number of hidden neurons/filters/cells of layer_i	Discrete	Structure
Kernel size of CNN layer_i	Discrete	Structure
Pooling type for CNN layers	Categorical	Structure
Pooling size for CNN layers	Discrete	Structure
Batch size	Discrete	Training
Learning rate	Continuous	Training
Dropout rate	Discrete	Regularisation
Weight decay	Continuous	Regularisation

single algorithm, the maximum budget will be set to the maximum number of epochs found during the manual tuning of that algorithm (see table 3.6). For the experiments where the search space contains all three candidate DNNs, the maximum of the maximum number of epochs found during the manual tuning of the candidate DNNs is chosen (see table 3.6). An exception is made for the voltage and methane dataset, where a third of actual maximum value is used (see table 3.6). This is to constrained BOHB to find optimal models that are faster to evaluate. Otherwise, the evaluations on the maximum budget would take too long, thus making a BOHB run impracticable.

After chosing the maximum budget, the minimum budget is determined using equation 3.7, where
$\eta \rightarrow$ *a hyperparameter of the hyberband algorithm* [20], [91]; and $N \rightarrow$ *the number of medium budgets between the minimum budget and the maximum budget.*

$$minimum\ budget = \lfloor \frac{maximum\ budget}{\eta^{N+1}} \rfloor = \lfloor \frac{maximum\ budget}{3^2} \rfloor \tag{3.7}$$

Following Li. et al's recommendation [19], η is set to 3; and N to 1. The minimum and maximum budget per dataset are provided in table 3.6. The number of iterations of BOHB is set to 30. All the other BOHB parameters are kept to their default values except the *top_n_percent* and the *num_samples*, which are respectively doubled to 30, and halved to 32, following Falkner et al.'s [20] guidelines[7].

3.8 Overview of the experiments

Six experiments are performed on four different datasets for predicting trends in time series data. The first four experiments use manual tuning for ASHO, and led to the selection of the best manually tuned model on each dataset. The last two experiments used AutoML for ASHO, and led to the selection of the best automatically tuned model on each dataset.

[7] https://automl.github.io/HpBandSter/build/html/best_practices.html

TABLE 3.6: BOHB budget (number of epochs), mimimum budget (min), maximum budget (max) used for each dataset. All refers to the budget used when the hyperparameter configuration space contained all three vanilla deep neural networks.

	NYSE		JSE		Voltage		Methane	
	min	max	min	max	min	max	min	max
MLP	55	500	11	100	555	5000	1666	15000
LSTM	11	100	11	100	111	1000	1666	15000
CNN	11	100	11	100	1555	15000	111	1000
All	55	500	11	100	333	3000	333	3000

In experiment 1, a recent hybrid DNN trend prediction approach, i.e. TreNet [3] is implemented. TreNet's performance and robustness is evaluated using a walk-forward validation method. TreNet uses a hybrid DNN structure, that combines both an LSTM and a CNN, and takes in a combination of raw data points and trend lines as its input. In experiment 2, TreNet results are compared with the performance of vanilla DNN structures, i.e. MLP, CNN and LSTM, on raw point data. In experiment 3, the performance of three traditional ML techniques, i.e. SVR, RF, and GBM on raw point data are compared with the performance of the best DNN to analyse the performance difference between DNN and non-DNN approaches. In experiment 4, the raw features are supplemented with trend line features to analyse whether there is any performance improvement to the non-hybrid models - both the vanilla DNN and the traditional ML models - from Experiments 2 and 3.

In experiments 5 and 6, a recent AutoML framework, i.e BOHB [20] is used to determine to what extent ASHO can be automated and how the performance of the optimal AutoML model compares to the optimal model found in the manual experiments 1 to 4. More specifically, in experiment 5, the optimal AutoML models across multiple runs are compared with the best manually tuned models to determine whether AutoML finds similar optimal models. Experiment 5 also compares the number configurations evaluated by BOHB with the number of configurations evaluated during the manual tuning. In experiment 6, each candidate algorithm is tuned separately - as opposed to the joint tuning in experiment 5 - to evaluate the effect of an increase in computational budget on the performance of BOHB.

4 Experiments - Manual ML for predicting trends

This chapter describes the four experiments conducted with manual ASHO. It starts with experiment 1, which replicates TreNet with walk-froward validation instead of cross-validation. It proceeds with experiment 2, which compares the hybrid TreNet to vanilla DNN structures. It continues with experiment 3, which compares DNNs to traditional ML algorithms; and experiment 4, which analyses the effect of supplementing raw data point features with trend line features for non-hybrid algorithms. The chapter concludes with a summary of the best manually tuned models on each dataset based on experiments 1 to 4. The results of the experiments are provided and discussed in their corresponding sections.

4.1 Experiment 1: Predicting trends with TreNet

The TreNet approach, recently proposed by Lin et al. [3], combines an LSTM and a CNN into a hybrid neural network. While the authors of TreNet reported a marked performance improvement when compared to other approaches, the validation method used in their experiments is questionable. It suffers from the validation problem explained in section 1.1. For instance, the data was first randomly shuffled, 10% of the data was held out for testing and a cross validation approach for training with the remainder of the data. Randomly shuffling the data and using a standard cross validation approach[3] does not take into account the sequential nature of time series data and may give erroneous results. A walk-forward validation is better suited for evaluating and comparing model performance on time series data [21]. Since this brings into question the veracity of the reported results, an attempt to replicate TreNet approach is performed using a walk forward validation instead of random shuffling and cross-validation.

4.1.1 Comparison metric

In order to compare the results with the original TreNet, a similar performance measure to Lin et al. [3] is used. The percentage improvement over the naive last value model (LVM) is measured. The naive LVM simply "takes the duration and slope of the last trend as the prediction for the next one" [3]. The use of a relative metric makes comparison easier, since the RMSE is scale-dependent, and the trend lines generated in this study may differ from the original TreNet paper's [3]. The authors did not provide details of

the segmentation method they used in their paper. Furthermore, the naive LVM does not require any hyperparameter tuning, its predictions are stable and repeatable, i.e. the performance of the naive LVM does not differ when the experiment is re-run, and it is only dependent on the characteristics of the dataset.

4.1.2 Results and discussions

Table 4.1 shows the performance improvement on RMSE values over the LVM achieved by the TreNet implementation on each dataset. They are compared to the performance of the original TreNet on the three datasets they used in their experiments, i.e. the voltage, methane and NYSE datasets.

TABLE 4.1: Comparison of the slope (S), duration (D), and average (A) RMSE values achieved by TreNet and Lin et al.'s (Original) results, and their respective percentage improvements (% improv.) over the naive LVM

	Voltage			Methane			NYSE		
	S	D	A	S	D	A	S	D	A
LVM	17.09	86.51	51.80	28.54	152.86	90.70	127.16	0.33	63.75
TreNet	9.25	62.37	35.81	14.87	31.25	23.06	86.89	1.23	44.06
Our % improv.	**45.87**	27.90	30.87	**47.90**	**79.56**	**74.58**	**31.67**	−272.73	**30.89**
Original LVM	21.17	39.68	30.43	10.57	53.76	32.17	8.58	11.36	9.97
Original TreNet	12.89	25.62	19.26	9.46	51.25	30.36	6.58	8.51	7.55
Original % improv.	39.11	**35.43**	**36.71**	10.50	4.69	5.63	23.31	**25.09**	24.27

The results of this experiment differ substantially from those reported in the original TreNet paper. TreNet models' percentage improvement over the naive LVM is 13.25 (**74.58**/5.63) and 1.27 (**30.89**/24.27) times greater than Lin et al.'s [3] on the methane and NYSE datasets respectively; but 1.19 (36.71/**27.90**) times smaller on the voltage dataset. The naive LVM performs better than the TreNet model on the NYSE for the duration prediction. The 272.73% decrease in performance is due to two reasons. On the one hand, during the model training, the loss minimisation is biased towards the slope loss at the expense of the duration loss. This is because the slope loss is significantly greater compared to the duration loss, but, TreNet's loss function weights both equally. On the other hand, the durations of the trends in the NYSE dataset being very similar - with a standard deviation of 0.81 - makes the last value prediction model a favourably competitive model for the duration prediction. The first issue may be mitigated by scaling the slope and duration input values to the same value range such as $[-1, 1]$.

The greater average improvement on the methane and NYSE is attributed to the use of the walk-forward evaluation procedure. The methane and NYSE datasets undergo various changes in the generating process because of the sudden changes in methane concentrations and the economic cycles for the NYSE. Thus, the use of the walk-forward evaluation ensures that the most recent and useful training set is used for a given validation/test set. However, given that Lin et al. [3] did not drop older data from the training

dataset, the network may learn long-range relationships that are not useful for the current test set. Furthermore, they used random shuffling, which may most likely result in future data points being included in the training data.

The smaller improvement of this TreNet's implementation on the voltage dataset can be attributed to the use of a smaller window size for the *local raw data* fed into the CNN. This study used 19 compared to the best value of 700 on the voltage dataset in the original TreNet study [3]. This is one of the limitations of the replication of TreNet. For each dataset, the length of the first trend line is used as window size of the *local raw data* feature fed into the CNN, instead of tuning it to select the best value. The other limitation is the use of a sampled version of the methane dataset instead of the complete methane dataset.

4.1.3 Robustness and stability

Table 4.2 shows the RMSE achieved by TreNet along with the standard deviation across multiple (10) runs. The standard deviation provides a measure of the robustness and the stability of the model to random initialisation and to the stochastic nature of its training algorithm, i.e. the Adam optimizer [94].

TABLE 4.2: The RMSE values and the standard deviation achieved by TreNet across multiple (10) runs.

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
RMSE	9.25 ± 0.0	62.37 ± 0.01	35.81 ± 0.01	14.87 ± 0.40	31.25 ± 2.62	23.06 ± 1.51
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
RMSE	86.89 ± 0.14	1.23 ± 0.38	44.06 ± 0.26	19.65 ± 0.05	12.49 ± 0.04	16.07 ± 0.05

The standard deviation is low for all four datasets. Regardless of the stochastic nature of the training procedure, the performance of the final model of a given training episode does not deviate far away from the average performance across multiple runs. This provides a reasonable degree of confidence in the stability and the robustness of the method.

4.1.4 Optimal hyperparameters

Table 4.3 shows the optimal hyperparameters found for TreNet on each dataset. They are compared to the hyperparameters used in the original TreNet implementation by Lin et al. [3], where applicable.

Contrary to the suggestion of the original paper, the optimal dropout and L2-regularisation values show that in general no regularisation is needed for predicting trends with TreNet. Similarly, the optimal structural hyperparameters such as the LSTM cells, the number of CNN filters, and the fusion layer size differ per dataset. Using the same values for all datasets may not result in optimal performances.

TABLE 4.3: TreNet hyperparameters found by manual experimentation. "?" means *unknown* and $S = \{300, 600, 900, 1200\}$

	Dropout	**L2**	**LR**	**LSTM Cells**	**CNN Filters**	**Fusion Layer**	**Batch Size**	**Epochs**	**Warm start**
Voltage	0.0	5e-4	1e-3	[600]	[16, 16]	300	2000	100	0.2
Methane	0.0	5e-4	1e-3	[1500]	[4, 4]	1200	2000	2000	0.1
NYSE	0.0	0.0	1e-3	[600]	[128, 128]	300	5000	100	0.5
JSE	0.0	0.0	1e-3	[5]	[32, 32]	10	500	100	0.05
Lin et al. [3]	0.5	5e-4	?	[600]	[32, 32]	*from S*	?	?	*N/A*

4.1.5 Analysis of the model update with *warm-start*

Table 4.4 shows the speed-up gained with the model update with warm-start (the speed-up is calculated using formula 3.6); and the RMSE achieved by TreNet with and without warm-start.

TABLE 4.4: The reduction factor in total number of epochs (speed-up), and the average (slope and duration) RMSE using model update with/without warm-start

	Voltage	Methane	NYSE	JSE
Number of splits	8	44	5	101
Warm-start fraction	0.2	0.1	0.5	0.05
Speed-up	3.33	8.30	1.67	16.83
RMSE with warm-start	**35.79 ± 0.02**	**40.58 ± 1.43**	**44.20 ± 0.41**	**16.40 ± 0.09**
RMSE without warm-start	35.81 ± 0.01	46.49 ± 2.18	44.82 ± 0.25	17.07 ± 0.06

The warm-start initialisation reduces the training time but does not compromise the generalisation ability of the network. In general, this holds true for vanilla DNN models as shown in the appendix table B.1. The higher the number of splits, the higher the speed-up gained from the model update with warm-start. The lower the warm-start fraction, the higher the speed-up gained from the model update with warm-start. For TreNet, the highest warm-start fraction required to achieve a lower error with the warm-start than without warm-start was 0.5 on the NYSE dataset.

4.2 Experiment 2: Predicting trends with vanilla DNNs

Given the use of a different validation method, which yields different performance scores to the original TreNet results, this TreNet's implementation is compared with the vanilla DNN models to evaluate whether or not it still outperforms them. Three vanilla DNN models, i.e. MLP, LSTM, and CNN are implemented, tested using only raw data point features. Table 4.5 shows their best hyperparameters found for each dataset.

TABLE 4.5: Hyperparameters optimised for the vanilla DNN algorithms and their optimal values found for each dataset

		Voltage	Methane	NYSE	JSE
MLP	batch size	4000	250	5000	250
	warm start	0.1	0.1	0.7	0.05
	learning rate	1e-4	1e-3	1e-3	1e-3
	dropout	0.0	0.0	0.0	0.0
	weight decay	0.0	0.0	5e-4	0.0
	number of epochs	10000	15000	500	100
	layer configuration	[500, 400, 300]	[500, 400]	[500, 400, 300]	[100]
LSTM	batch size	4000	2000	5000	1000
	warm start	0.1	0.1	0.01	0.05
	learning rate	1e-2	1e-4	1e-3	1e-3
	dropout	0.0	0.0	0.5	0.5
	weight decay	0.0	0.0	5e-5	0.0
	number of epochs	1000	15000	100	100
	cell configuration	[600]	[600, 300]	[100]	[100]
CNN	batch size	2000	250	5000	1000
	warm start	0.5	0.3	0.4	0.1
	learning rate	1e-3	1e-3	1e-3	1e-3
	dropout	0.0	0.0	0.0	0.0
	weight decay	5e-5	5e-4	0.0	0.0
	number of epochs	15000	1000	12000	100
	filter configuration	[16]	[32, 32]	[32]	[32, 32]
	kernel configuration	[2]	[2, 4]	[1]	[1, 1]
	Pooling type	Max	Max	Identity	Identity
	Pooling size	2	5	N/A	N/A

4.2.1 Results and discussions

Table 4.6 shows the average RMSE values achieved by the MLP, LSTM, CNN and TreNet on each dataset across 10 independent runs. The deviation across the 10 runs is also shown to provide an indication of the stability of the models across the runs. The average slope and duration RMSE values are used as an overall comparison metric. The % improvement is the improvement of the best vanilla DNN model over TreNet. The best model is chosen based on the overall comparison metric.

TABLE 4.6: Comparison of the TreNet model with the vanilla DNN models

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
MLP	**9.04 ± 0.06**	62.82 ± 0.04	35.93 ± 0.05	14.57 ± 0.10	49.79 ± 4.85	32.18 ± 2.48
LSTM	10.30 ± 0.0	62.87 ± 0.0	36.59 ± 0.0	**14.21 ± 0.19**	56.37 ± 1.77	35.29 ± 0.49
CNN	9.24 ± 0.10	62.40 ± 0.13	35.82 ± 0.12	15.07 ± 0.35	54.79 ± 4.55	34.93 ± 2.45
TreNet	9.25 ± 0.0	**62.37 ± 0.01**	**35.81 ± 0.01**	14.87 ± 0.40	**31.25 ± 2.62**	**23.06 ± 1.51**
% Improvement	-0.11	-0.05	-0.03	**2.02**	-59.33	-39.55
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
MLP	90.76 ± 4.43	33.08 ± 42.08	61.92 ± 23.26	19.87 ± 0.01	12.51 ± 0.09	16.19 ± 0.05
LSTM	**86.56 ± 0.01**	**0.41 ± 0.08**	**43.49 ± 0.05**	19.83 ± 0.01	12.68 ± 0.01	16.25 ± 0.01
CNN	89.31 ± 1.38	12.21 ± 12.17	50.76 ± 6.78	19.90 ± 0.06	**12.48 ± 0.21**	16.19 ± 0.14
TreNet	86.89 ± 0.14	1.23 ± 0.38	44.06 ± 0.26	**19.65 ± 0.05**	12.49 ± 0.04	**16.07 ± 0.05**
% Improvement	**0.38**	**66.67**	**1.29**	-1.12	-0.16	-0.75

In general TreNet still performs better than the vanilla DNN models, but does not outperform the vanilla models on all the datasets. The most noticeable case is on the NYSE, where the LSTM model outperforms the TreNet model on both the slope and duration prediction. This contradicts Lin et al. [3]'s findings, where TreNet clearly outperforms all other models including LSTM. Lin et al. [3]'s results show that the TreNet model outperforms the LSTM model by 22.48%; whereas here, the TreNet model underperforms the LSTM model by 1.31%. However, Lin et al. [3]'s LSTM model appears to be trained using trend line input features only and not raw point data. This study's LSTM model uses *local raw data* input features. It must also be noted that the validation method used here is substantially different from the one used by Lin et al. [3].

The large performance difference between the TreNet model and the vanilla models on the methane dataset is because for this dataset the raw local data features do not provide the global information about the time series since it is non-stationary. This is confirmed by the increase in the performance of the MLP (23.83%), LSTM (11.02%) and CNN (24.05%) after supplementing the raw data features with trend line features (see experiment 4 in section 4.4).

4.3 Experiment 3: Predicting trends with traditional ML algorithms

Given the new validation method, the performance of DNN models are compared with the performance of traditional ML models. Three traditional ML models, i.e. radial-based SVR, RF, and GBM are implemented and tested. Table 4.7 shows their best hyperparameters found on each dataset. No previous work was found which uses RF and GBM for

TABLE 4.7: Hyperparameters optimised for each algorithm and their optimal values found manually for each dataset

		Voltage	Methane	NYSE	JSE
RF	number of estimators	50	50	200	100
	maximum depth	2	10	1	1
	bootstrap	False	False	True	False
	warm start	False	False	True	True
GBM	boosting type	gbdt	gbdt	gbdt	gbdt
	number of estimators	1	10000	1	4
	learning rate	2000	1.5	0.2	0.1
SVR	gamma	0.1	1e-4	1e-1	1e-4
	C	4	10000	100	500

predicting trends in time series data. This study may be the first to do so. However, Lin et al. [3] compared their approach against multiple SVR kernels that took in both local raw data and trend line features. In this experiment, the models take in *local raw data* features alone.

4.3.1 Results and discussions

Table 4.8 shows the RMSE values achieved by the traditional ML algorithms and the best DNN models on each dataset. The best DNN model is TreNet on all datasets except on the NYSE, on which LSTM is the best model. The percentage improvement (% improv.) is the performance improvement of the best traditional ML model over the best DNN model, where, the best model is selected based on the equally weighted average slope and duration RMSE (referred to as *Average* in table 4.8).

The best traditional ML algorithm underperformed the best DNN algorithm by 0.47% and 1.74% respectively on the (almost) normally distributed datasets, i.e. the voltage and the JSE datasets. However, the RF model outperformed the best DNN model, i.e. TreNet by 33.04% on the methane dataset; while the SVR model matched the performance of the best DNN model, i.e. LSTM on the NYSE dataset. TreNet learns long-range dependencies from trend line features with its LSTM component. Although this is useful for stationary and less evolving time series such as the voltage and JSE datasets, it appears that it can be detrimental in the case of dynamic and non-stationary time series such as the methane dataset. This may explain why the traditional ML models, which do not keep long-term memory, performed better on this dataset.

TABLE 4.8: Comparison of the best DNN models (Best DNN) with the traditional ML algorithms. The % improvement (% improv.) is the performance improvement of the best traditional ML model over the best DNN model

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
RF	9.53 ± 0.0	63.11 ± 0.20	36.32 ± 0.10	$\mathbf{10.09 \pm 0.01}$	$\mathbf{20.79 \pm 0.01}$	$\mathbf{15.44 \pm 0.01}$
GBM	10.0 ± 0.0	62.67 ± 0.0	36.34 ± 0.0	13.05 ± 0.0	75.10 ± 0.0	44.08 ± 0.0
SVR	9.32 ± 0.0	62.58 ± 0.0	35.95 ± 0.0	14.98 ± 0.0	34.39 ± 0.0	24.69 ± 0.0
Best DNN	$\mathbf{9.25 \pm 0.0}$	$\mathbf{62.37 \pm 0.01}$	$\mathbf{35.81 \pm 0.01}$	14.87 ± 0.40	31.25 ± 2.62	23.06 ± 1.51
% improv.	-0.76	-0.34	-0.47	**32.15**	**33.47**	**33.04**
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
RF	88.75 ± 0.17	$\mathbf{0.29 \pm 0.0}$	44.52 ± 0.09	20.21 ± 0.0	12.67 ± 0.0	16.44 ± 0.0
GBM	86.62 ± 0.0	0.42 ± 0.0	43.52 ± 0.0	20.08 ± 0.0	12.62 ± 0.0	16.35 ± 0.0
SVR	$\mathbf{86.55 \pm 0.0}$	0.42 ± 0.0	43.49 ± 0.0	20.01 ± 0.0	12.85 ± 0.0	16.43 ± 0.0
Best DNN	86.56 ± 0.01	$\mathbf{0.41 \pm 0.08}$	43.49 ± 0.05	$\mathbf{19.65 \pm 0.05}$	$\mathbf{12.49 \pm 0.04}$	$\mathbf{16.07 \pm 0.05}$
% improv.	**0.01**	2.44	0.0	-2.19	-1.04	-1.74

The fact that the radial-based SVR performed better than TreNet on the NYSE dataset contradicts Lin et al. [3]'s results. This is attributed the use of *local raw data* features alone, instead of *local raw data* plus *trend line* features used by Lin et al. [3].

4.4 Experiment 4: Addition of trend line features

In this experiment, the raw data are supplemented with trend line features to analyse whether this yields any performance improvement to the non-hybrid models: both the DNN and the traditional ML models from Experiments 2 and 3. The hyperparameter values found using the *raw data* features alone are retained for this experiment.

4.4.1 Results and discussions

Table 4.9 shows the average performance improvement (%) after supplementing the raw data with trend line features. The negative sign indicates a drop in performance, and *Average* is the mean and the standard error of the improvements over the algorithm or the dataset. The actual RMSE values are shown in table C.1 and table C.2 in the appendix.

The addition of trend line features improved the performance of both DNN and traditional ML models 10 times out of 24 cases. In general, it improves the performance of dynamic and non-stationary time series such as the methane and NYSE datasets. This is because local raw data features do not capture the global information about the time series for non-stationary time series. Thus, addition of trend line features brings new information to the models. In 12 out of 24 cases, the addition of trend line features reduced the performance of both the DNN and the traditional ML models except the GBM

TABLE 4.9: Performance improvement (%) in RMSE after supplementing the raw data with trend line features.

	MLP	LSTM	CNN	RF	GBM	SVR	Average
Voltage	**0.03**	0.0	-73.14	-0.13	**0.06**	-0.36	-12.26 ± 12.18
Methane	**23.83**	**11.02**	**24.05**	-4.47	**42.88**	-6.28	$\mathbf{15.17 \pm 7.71}$
NYSE	**6.49**	0.0	-1.00	**2.36**	**0.23**	-0.02	$\mathbf{1.34 \pm 1.12}$
JSE	-4.14	-1.17	-5.37	-7.60	**0.37**	-10.96	-4.81 ± 1.70
Average	$\mathbf{6.55 \pm 6.16}$	$\mathbf{4.93 \pm 2.87}$	-13.87 ± 20.79	-2.46 ± 2.22	$\mathbf{10.89 \pm 10.67}$	-4.41 ± 2.62	

models. For these cases, the additional trend line features introduce noisy or duplicate information, which the models did not deal with successfully. This may be because the best hyperparameters for the raw data features alone may not be optimal for the raw data and the trend line features combined. For instance, DNN models are generally able to extract the true signal from noisy or duplicate input features, however, they are sensitive to the hyperparameter values.

The above results show that the addition of trend line features has the potential to improve the performance of both DNN and traditional ML models on non-stationary time series. This comes at the cost of additional complexities and restrictions. The first complexity is related to the model complexity because the bigger the input feature size, the more complex the model becomes. Secondly, the trend line features require the segmentation of the time series into trends, which brings new challenges and restrictions during inference. For instance, trend prediction applications that require online inference need an online segmentation method such as SWAB [1]. It is therefore necessary to evaluate whether the performance gain over raw data features alone justifies these complexities and restrictions.

It must be noted that this experiment is exploratory. Retaining the same hyperparameters used for raw local data during this experiment does not make the findings conclusive. This is a limitation of the work.

4.5 Summary: Best manually tuned models

Table 4.10 provides a summary of the best manually tuned models, and their average performance, based on experiments 1 to 4. TreNet models outperform the non-hybrid models on the voltage and the JSE datasets, but the performance difference is marginal $< 1\%$. Interestingly, the traditional ML models outperformed TreNet and the vanilla DNN algorithms on the methane and NYSE datasets.

The addition of trend lines to the point data (experiment 4) did not yield any substantial change in the results. It must be noted though that this was an exploratory experiment and that no hyperparameter optimisation was done to cater for the introduction of a new input feature. It may well be the case that better models could be found of a new hyperparameter optimisation process was undertaken.

TABLE 4.10: Average RMSE values (E) achieved by the hybrid TreNet model (Hybrid); and the best non-hybrid model (A) with raw point data features alone (Pt) and with raw point data plus trend line features (Pt + T). The % change is with respect to the TreNet algorithm.

		% Change	*Pt*	*Hybrid*	*Pt + T*	*% Change*
Voltage	*A*	-	CNN	**TreNet**	MLP	-
	E	-0.03	35.82 ± 0.12	$\mathbf{35.81 \pm 0.01}$	35.92 ± 0.05	-0.31
Methane	*A*	-	**RF**	TreNet	RF	-
	E	**33.04**	$\mathbf{15.44 \pm 0.01}$	23.06 ± 1.51	16.13 ± 0.01	30.05
NYSE	*A*	-	SVR	TreNet	**GBM**	-
	E	**1.29**	43.49 ± 0.0	44.06 ± 0.26	$\mathbf{43.42 \pm 0.0}$	1.45
JSE	*A*	-	MLP	**TreNet**	GBM	-
	E	-0.75	16.19 ± 0.05	$\mathbf{16.07 \pm 0.05}$	16.29 ± 0.0	-1.37

It is clear from these results that TreNet generally performs well on most datasets. However, it is not the clear winner, and there are some datasets where traditional models can substantially outperform TreNet. It is also clear that models built with point data alone can generally reach the performance levels of TreNet.

Manually finding the best model for a particular time series required extensive and often ad-hoc experimentation to find the best hyperparameter configuration for each algorithm on each dataset. In the next chapter, a recent AutoML, i.e. BOHB is implemented and evaluated to perform automated ASHO for predicting trends in time series data.

5 Experiments - AutoML for predicting trends

This chapter describes the two experiments performed to explore AutoML for ASHO. More specifically, the experiments evaluate BOHB for automating the ASHO for predicting trends in time series data (See section 3.7 for the implementation details). The chapter starts with experiment 5, which compares the best AutoML model found by BOHB with the best model found by manual tuning on each dataset. Experiment 5 also compares the number of configurations evaluated by BOHB and by the manual tuning process. It proceeds with experiment 6, which evaluates the effect of increasing the AutoML computational budget on its performance. The chapter concludes with a summary of the best AutoML model and its performance on each dataset, based on experiment 5 and 6. The performance of best AutoML models are compared with the performance of the best manually tuned models.

5.1 Experiment 5: AutoML – all algorithms

The aim of this experiment is two-fold. The experiment first compares the best model found by BOHB with the best model found by manual tuning. Then, it compares the number of configurations evaluated by BOHB with the number of configurations evaluated by manual tuning process during the manual ML experiments. In this experiment, the full AutoML search space is searched for different model configurations across all three vanilla DNN algorithms. Given the stochastic nature of BOHB, 10 independent runs of the experiment are conducted on each dataset.

5.1.1 Optimal AutoML models

Table 5.1 shows the number of times each candidate DNN is found by BOHB as the best algorithm across these 10 runs as well as the best manually tuned DNN.

The best model - found 9 times out of 10 runs - by BOHB is the same as the best manually tuned model for the NYSE and the JSE datasets. It is also interesting to note that BOHB favoured the simpler/faster MLP model over the CNN for the JSE dataset. On the voltage dataset, the best manually tuned model is found 5 times by BOHB as the optimal model. The other 5 times, it found a different model namely MLP instead of CNN. However, the average performance of the 5 MLP models is better than the average performance of the 5 CNN models (RMSE of 36.21 and RMSE of 36.46). Again the MLP

TABLE 5.1: The number of times each of DNN models is selected as optimal by BOHB, and the best manually tuned DNN models (Best M-DNN)

	Voltage	Methane	NYSE	JSE
MLP	**5**	3	0	**9**
LSTM	0	0	**9**	0
CNN	**5**	**7**	1	1
Total runs	10	10	10	10
Best M-DNN	**CNN**	MLP	**LSTM**	**MLP/CNN**

is favoured over the CNN, which shows a bias towards simpler/faster models. The interesting result is the methane dataset, where the CNN is favoured over the MLP. BOHB found the CNN to be the optimal model 7 times out of 10 instead of the optimal manually tuned model, i.e. MLP, which BOHB found 3 times as the optimal model. It is important to that although BOHB found a different model than the manual tuning process, the average RMSE performance of the optimal AutoML models across the 10 runs is better than that of the optimal manual DNN model, i.e. $\mathbf{30.05 \pm 2.76}$ vs. 32.18 ± 2.48 (see table 4.6 and table 5.3)

Regarding the optimal hyperparameters, each run of BOHB finds a different optimal configuration given the same optimal algorithm. However, the variance in the performances of these configurations is low. This is shown by the small standard deviation of the RMSE values achieved by the different configurations (BOHB-All) in table 5.3. This confirms the *robustness* of BOHB as suggested by Falkner et al [20].

5.1.2 Configurations evaluated

During the manual ML experiments in chapter 4, each configuration is evaluated 10 times. For the BOHB experiments, an initial pool of configurations is evaluated first on the minimum budget (number of epochs). Then, the budget is increased to the medium budget, i.e. $3 \times (\textit{min budget})$ (see section 3.7.2) and the best half of these configurations are evaluated. Finally, the best half of the medium budget configurations are evaluated with the maximum budget (highest fidelity). In effect, promising configurations are first identified using the minimum budget and further explored using a higher budget.

Table 5.2 shows the number of unique configurations evaluated during the manual ML and the AutoML approach for predicting trends. BOHB evaluated more unique con-

TABLE 5.2: The number of unique configurations evaluated (No. unique) and the total number of evaluations performed (Total) manually and automatically using BOHB as well as the number of equivalent evaluations on the maximum budget (No. max. equiv.).

	Manual				BOHB			
	Voltage	Methane	NYSE	JSE	Voltage	Methane	NYSE	JSE
No. unique	55	48	46	46	150	150	150	150
Total	550	480	460	460	200	200	200	200
No. max. equiv.	-	-	-	-	80	80	80	80

figurations than the manual tuning process. For each AutoML run, BOHB evaluated 150 unique hyperparameter configurations. However, the manual tuning evaluated a maximum of 55 unique configurations on the voltage dataset for the three candidate algorithms, i.e. MLP, LSTM, and CNN.

Since each configuration is evaluated 10 times during the manual tuning, the total number of evaluations performed for each dataset by the manual ASHO process is 10 times the number unique configurations evaluated. Thus, the minimum total number of evaluations performed by the manual ASHO is 460 on the JSE and NYSE datasets for 46 unique configurations. The automated ASHO performed 200 total evaluations for more unique configurations (150). The difference of 50 (200 - 50) is due to the re-evaluation of promising candiate configurations on the medium, and the maximum budget by BOHB. The 200 total BOHB evaluations are equivalent to 80 evaluations on the maximum (full) budget, i.e. if 80 evaluations are performed at the maximum number of epochs it would use roughly the same computational power as the 200 BOHB evaluations. In this way, BOHB is able to maximise the specified budget to explore more candidate configurations in the search space. BOHB therefore performs a more explorative algorithm and hyperparameter configuration search with less total computational budget. However, the manual tuning process as explained in section 3.6.1 performs a more focused search informed by expert judgement.

5.2 Experiment 6: AutoML - single algorithm

The aim of this experiment is to evaluate the effect of increasing the computational budget on the performance of BOHB. This is accomplished by comparing the performance of the best model when the search space is constrained to a single algorithm with the performance of the best model when the search space contains all three candidate algorithms. The second scenario corresponds to the setting in experiment 5 in section 5.1, where BOHB is run once with the full search space.

In the first scenario, BOHB is run three different times for each of the three candidate algorithms. Thus, 150 unique configurations are evaluated for a single algorithm instead of 150 configurations across 3 algorithms. This in some sense provides an increase in budget. Practically, searching each algorithm at a time allowed setting the BOHB's maximum budget for each algorithm to the best number of epochs found during the manual experiment for that algorithm. This is a more accurate estimation of the true function to be learnt given that algorithm. However, it resulted in an increase or a decrease of the minimum/maximum budget for certain algorithms on certain datasets, compared to the combined search (see the change in budgets from *All* to *single algorithm* in table 3.6). For instance, the maximum budget for the LSTM on the NYSE decreased to 100 from 500, chosen for the combined search based on the manually found optimal number of epochs.

5.2.1 Results and discussions

Table 5.3 provides a comparison of the performance of the optimal model found when all algorithms (BOHB-All) are used and when only a single algorithm (BOHB-Single) is used, i.e when the budget - number of configurations evaluated per algorithm - is increased. The mean and deviation of the optimal model from each experiment trained and evaluated over 10 independent runs is shown.

TABLE 5.3: Comparison of BOHB on all algorithm (BOHB-All) and the best BOHB on the single algorithms (BOHB-Single). Model (M), Slope RMSE (S), Duration RMSE (D), Average slope and duration RMSEs (A)

		BOHB-All	*BOHB-Single*	*% Delta*
re based on the actual end of the segment	*M*	MLP	CNN	-
Voltage	*S*	9.70 ± 0.44	$\mathbf{9.08 \pm 0.04}$	**6.39**
	D	62.97 ± 0.14	$\mathbf{62.35 \pm 0.02}$	**0.98**
	A	36.34 ± 0.29	$\mathbf{35.72 \pm 0.03}$	**1.71**
	M	CNN	MLP	-
Methane	*S*	15.29 ± 1.00	$\mathbf{14.01 \pm 0.21}$	**8.37**
	D	44.80 ± 4.52	$\mathbf{40.09 \pm 6.95}$	**10.51**
	A	30.05 ± 2.76	$\mathbf{27.05 \pm 3.58}$	**9.98**
	M	LSTM	LSTM	-
NYSE	*S*	$\mathbf{86.61 \pm 0.03}$	86.62 ± 0.03	-0.01
	D	$\mathbf{0.55 \pm 0.15}$	0.72 ± 0.30	-30.91
	A	$\mathbf{43.58 \pm 0.09}$	43.67 ± 0.17	-0.21
	M	MLP	CNN	-
JSE	*S*	20.00 ± 0.13	$\mathbf{19.96 \pm 0.17}$	**0.20**
	D	$\mathbf{12.46 \pm 0.18}$	$\mathbf{12.46 \pm 0.12}$	0.0
	A	16.23 ± 0.16	$\mathbf{16.21 \pm 0.15}$	**0.12**

By evaluating more configurations per algorithm a much better model is found for the methane dataset, but only marginally better model for the voltage and JSE datasets. There is a slight drop in performance for the NYSE. This could be because when the search is constrained to a single algorithm, the best configuration found by BOHB, i.e. LSTM is evaluated with a smaller number of epochs compared to the LSTM found by BOHB when all the algorithms are searched together (100 vs. 500 - see table 3.6). When BOHB focused on each algorithm, the best model found for each dataset matches the performance of the best manual models.

5.3 Summary: Best AutoML models

The risk/reward of using AutoML for predicting trends are now evaluated with respect to all the manually tuned models (G-) and with respect to the manually tuned candidate AutoML models alone (C-). Table 5.4 compares the best AutoML model and its performance with the best manually tuned model and its performance.

For the voltage dataset, the average performance of the best AutoML model, i.e. CNN is 35.72. An increase of 0.25% on the best manually tuned model, i.e. TreNet. The reader is reminded that the manually tuned CNN model yielded a decrease of 0.03% on the manually tuned TreNet. This shows that AutoML is able to find a less complex model, which the manual tuning process missed. For the three other datasets, the best AutoML model did not match the best manually tuned models. The most noticeable is on the methane dataset, where there is a decrease of 75.19%. This is mainly because the best manual model is the RF model, but, RF is not part of the candidates models used during

TABLE 5.4: Comparison of the best AutoML models and the best manual models. Model (M), Slope RMSE (S), Duration RMSE (D), Average slope and duration RMSEs (A)

		G-Reward (%)	*G-Manual*	*AutoML*	*C-Manual*	*C-Reward (%)*
Voltage	*M*	-	TreNet	CNN	CNN	-
	S	**1.84**	9.25 ± 0.0	$\mathbf{9.08 \pm 0.04}$	9.24 ± 0.10	**1.73**
	D	**0.03**	62.37 ± 0.01	$\mathbf{62.35 \pm 0.02}$	62.40 ± 0.13	**0.08**
	A	**0.25**	35.81 ± 0.01	$\mathbf{35.72 \pm 0.03}$	35.82 ± 0.12	**0.28**
Methane	*M*	-	RF	MLP	MLP	-
	S	-38.85	$\mathbf{10.09 \pm 0.01}$	14.01 ± 0.21	14.57 ± 0.10	**3.84**
	D	-92.83	$\mathbf{20.79 \pm 0.01}$	40.09 ± 6.95	49.79 ± 4.85	**19.48**
	A	-75.19	$\mathbf{15.44 \pm 0.01}$	27.05 ± 3.58	32.18 ± 2.48	**15.94**
NYSE	*M*	-	SVR/LSTM	LSTM	LSTM	-
	S	-0.07	$\mathbf{86.55 \pm 0.0}/86.56 \pm 0.01$	86.61 ± 0.03	86.56 ± 0.01	-0.06
	D	-30.95	$0.42 \pm 0.0/\mathbf{0.41 \pm 0.08}$	0.55 ± 0.15	$\mathbf{0.41 \pm 0.08}$	-34.15
	A	-0.21	$\mathbf{43.49 \pm 0.0}/\mathbf{43.49 \pm 0.05}$	43.58 ± 0.09	$\mathbf{43.49 \pm 0.05}$	-0.21
JSE	*M*	-	TreNet	CNN	MLP/CNN	-
	S	-1.58	$\mathbf{19.65 \pm 0.05}$	19.96 ± 0.17	$19.87 \pm 0.01/19.90 \pm 0.06$	-0.45
	D	**0.24**	12.49 ± 0.04	$\mathbf{12.46 \pm 0.12}$	$12.51 \pm 0.09/12.48 \pm 0.21$	**0.40**
	A	-0.87	$\mathbf{16.07 \pm 0.05}$	16.21 ± 0.15	$16.19 \pm 0.05/16.19 \pm 0.14$	-0.12

the automated ASHO. When compared to the best manually tuned candidate model, i.e. the manually tuned MLP model (see table 4.6), the AutoML model shows an increase of 9.98% on this dataset. This shows that the performance of AutoML heavily depends on a correct selection of the candidate algorithms.

6 Discussion and Conclusions

The overall aim of this work is to evaluate different ML techniques for predicting trends in time series data, taking into account model update and model stability. A manual approach and an AutoML approach are explored to determine the best algorithm and its best hyperparameter configuration, given a dataset. This chapter first summarises the findings and reflects on both these approaches. It proceeds by highlighting the impacts of the study, and ends with the limitations of the study and the avenues for future work.

6.1 Manual ML for predicting trends

While the hybrid DNN approach, i.e. TreNet, was no doubt an important development in terms of the application of DNNs for predicting trends, the validation method used in the experiments suffers from the validation problem explained in section 1.1. It did not take into account the sequential and dynamic nature of most real world time series data sets. It also did not deal with model update and stability. This study addressed these limitations by predicting trends with walk-forward validation, which includes model update. The study also measures the stability of the models across multiple independent runs. Given the new evaluation method, six experiments are performed with manual tuning for ASHO to compare the performance of TreNet, three vanilla DNNs, and three traditional ML algorithms for predicting trends. The hybrid TreNet model takes in both raw point data and trend line features, but, the non-hybrid models take in raw point data features. The effect of supplementing raw point data features with trend line features on the performance of the non-hybrid models is evaluated. The aim of these experiments is to use manual tuning to determine the best model for predicting trends in each of the four time series.

6.1.1 Summary of the findings

The results of the experiments substantially differs from those reported in the original TreNet paper. In general, the performance improvement over the naive last value model (LVM) achieved in this work is greater than that of the original TreNet especially for non-stationary datasets, where model update is necessary.

Compared to the vanilla DNN models, the hybrid TreNet model, which feeds raw point data into the CNNs and trend lines into the LSTM, performs better on the (almost) normally distributed datasets. It must be noted that TreNet does not greatly outperform the best vanilla DNN networks - with less than 1% performance difference.

On the non-stationary datasets, the traditional ML models outperform the DNN models. For instance, the RF model, trained with raw point data features, comes out as a clear winner on the methane dataset with more than 30% improvement over the best DNN model. The GBM with raw point data and trend line features is superior on the JSE dataset but not greatly so with 1.37% improvement over TreNet, the best DNN model.

In general, the best manually tuned models exhibit a robust and stable performance with an average RMSE deviation between 0.0 and 0.05 across multiple (10) runs.

6.1.2 Reflection on manual ML for predicting trends

The manual ASHO is an effective method for finding the best ML model for predicting trends in time series data. It revealed that no single algorithm and a single hyperparameter configuration, nor a single feature type is optimal for all datasets, i.e. all trend prediction applications. ML expertise and domain knowledge are both necessary for selecting the initial pool of candidate algorithms, the sensible hyperparameters, and the feature types for a given trend prediction problem. Besides, extensive and often expensive experimentation need to be performed to identify the best model. Simply put, manually identifying the best model for predicting trends is a time consuming and expensive exercise, which stakeholders such as researchers and industry practitioners cannot often afford. This hinders the fast advancement of the trend prediction research and the democratisation of its application to real-world problems because of the time, costs, and the expertise associated with developing state-of-the-art models. Although the manual tuning procedure used in this study is sensible, it remains an ad-hoc and hand-crafted engineering endeavour. It is not systematic and it may result in the best model being sub-optimal and inefficient due to a lack of time and resources as observed by Guyon et al. [96]. This is a reflection of the the current state of the broader ML research and practice. The second approach is an attempt to mitigate some of these issues by leveraging AutoML for predicting trends.

6.2 AutoML for predicting trends

In the TreNet paper and previous research on predicting trends, the ASHO is performed manually. However, manual ASHO is expensive and often results in a sub-optimal or mediocre model because it needs extensive experimentation as well as domain specific and ML expert knowledge. This study leveraged recent developments in AutoML for automating the ASHO for predicting trends in time series data.

6.2.1 Summary of the findings

A survey of recent AutoML techniques led to the selection of the hybrid BOHB framework for automatic ASHO for predicting trends in time series data. BOHB is selected because it is robust and flexible, it ensures a strong anytime performance, and it is suitable for low resource settings. In general, BOHB finds the same best algorithm as the manual

process, conducted by a human expert. When BOHB finds a different best algorithm, the average performance of the best AutoML models is better than the performance of the best manually tuned model. This shows that AutoML can at least be used to narrow the list of candidate algorithms for a given problem.

BOHB generally finds different best hyperparameter configurations across multiple runs with the same settings. However, the variance in the performance of the different best configurations is generally low. This confirms the robustness of BOHB. It also shows that in general the algorithm and hyperparameter configuration space of trend prediction problems contains multiple local maxima.

In general AutoML surpasses the performance of the best manually tuned models - from the candidate algorithms involved in the AutoML experiments. Its performance increases when BOHB is run N different times for each of the N candidate algorithms. In other words, the performance of BOHB generally increases as the allocated computation budget increases.

In certain cases, AutoML models underperforms the manually tuned models but not greatly so with less than 1% performance difference. When compared to all the manually tuned models (not just the manually tuned counterparts of the candidate AutoML models), the best AutoML model significantly underformed on the methane dataset. The best manually tuned RF model beat the best AutoML model, i.e. the Auto-MLP model by 75.19%. This is a warning that AutoML can result in a performance loss compared to a best hand-crafted model. This risk should be accounted for when considering to use AutoML for predicting trends in time series data. Besides this risk, BOHB has shown to be an effective AutoML framework for finding a well performing model for predicting trends in time series data.

It is important to note that the loss signal used by BOHB to guide its search is potentially noisy as opposed to the manual tuning process. This is because the validation loss used for a given hyperparameter configuration during the AutoML experiments is for a single training and validation episode, in order too reduce the running time of BOHB. During the manual tuning, the validation loss used is the average of 10 different training and validation episodes, in order to get a robust validation loss signal for a given hyperparameter configuration. This shows that the performance of BOHB may increase if the loss metric used is the average across 10 different independent runs (instead of a single run), provided that more budget in computation resource and/or time is made available.

6.2.2 Reflection on AutoML for predicting trends

This study shows that recent AutoML frameworks and tools can assist both researchers and ML practitioners in developing models for predicting trends in time series data. This can take the form of automating intensive and expensive hyperparameter optimisation experiments with smarter and robust AutoML algorithms such as BOHB instead of random search, or grid search. The use of AutoML frameworks such as HpBandSter[1] to find well-performing models for predicting trends is one-step towards finding a systematic method for ASHO.

The performance of BOHB is closely dependent on a correct specification of the problem. For example, it requires the specification of the hyperparameter configuration search space. The search space is of a high importance because it determines the pool of possible candidate hyperparameter configurations (therefore models) that can be found by BOHB. A too small search space may exclude the best configuration, whereas, a too large search space makes finding the best configuration harder and more expensive. In addition to the definition of the search space, the choice of a correct budget measure with reasonable minimum and maximum values, as well as the selection of a sensible and a robust loss signal/metric highly impact the performance of BOHB. For a new problem, an effective problem specification often requires a combination of the following: a good knowledge of the problem space, and multiple iterations. This increases the cost of an already expensive exercise and makes its adoption more difficult.

Although AutoML frameworks such as BOHB provide a means to automate or semi-automate model development, their performance is still dependent on correctly setting their own hyperparameters. At best, this creates an additional overhead for its users since it needs to be fine tuned. At worst, it results in a mediocre model for an inexperienced user. This study used some fine tuning iterations and recommended hyperparameters in the literature to achieve satisfactory results. However, it is important to develop AutoML algorithms that have minimal or are less dependent on their hyperparameter settings.

In summary, recent AutoML frameworks such as BOHB is useful for well-known and defined trend prediction problem spaces with sensible time and computational resources as well as expert AutoML knowledge. However, they currently move some tasks such as the hyperparameter optimisation problem from the ML algorithm level to the meta/AutoML level since they also need to be tuned. They still require knowledge of the problem domain for optimal performance.

6.3 Impact of the research

This study highlights the importance of using an appropriate validation strategy, which takes into account model stability and model update, when dealing with time series data. It also demonstrated how AutoML tools, specifically the HpBandSter framework, can be effectively used to automatically find the best DNN configuration.

Even though trend prediction has broad real world applications, there are relatively few studies that apply DNNs to this area when compared to applications for computer vision, natural language processing and speech recognition. Furthermore, there is limited research on the use of AutoML for finding well-performing ML models for predicting trends. Despite the availability of recent AutoML frameworks such as BOHB, most papers on trend prediction select the ML based on expert knowledge and perform extensive or very little experiments to optimise the hyperparameters [3], [4]. There has been

[1] https://github.com/automl/HpBandSter

some attempt to use AutoML for other time series problems [38], [42], [97], [98]. However, these studies did not focus on predicting trends in time series data nor did they incorporate recent AutoML frameworks such as BOHB.

The results of this study can be used to accelerate time series trend prediction research and practical evaluation of these algorithms for real world applications. The hyperparameter configuration search space identifies key hyperparameter variables and ranges that most impacted model performance across all datasets during the manual tuning experiments. The configuration file is made publicly accessible[2] to allow researchers and practitioners to replicate these results and evaluate this approach on other datasets. This could be a first step towards formalising and sharing ML knowledge.

It is important to note the AutoML method will not necessarily find the best model. The proposed method can be used to automatically find a near optimal model that performs close to or exceeds a typical manual tuning experiment. If this suffices then it can serve as a fully automated solution. However, if the performance of the resultant model is not sufficient then the model can provide a configuration, which can be manually tuned to further improve its performance or as a baseline model for comparison and exploration of other configurations. In both scenarios the approach may save time and effort. Furthermore, since it decreases the barrier to entry for developing ML models, the proposed method is likely to expand the ML user community. It allows people without expert knowledge and experience in ML to rapidly build and evaluate ML models.

6.4 Limitations and Future work

There are many avenues to probe the results of this work further. Firstly, only four datasets were used. While these included all three datasets used in the original TreNet paper [3], testing on more data sets is required to probe the generalisation of these findings. The results could be improved by investigating the effect of varying the window size. In this study, the window size was fixed to the length of the first trend line. Besides, instead of using a sampled version of the methane dataset, the complete dataset could be used.

Furthermore, during experiment 4, the hyperparameter used for the raw local data were retained for all the algorithms. Instead, the experiment could be run with a single algorithm with tuned hyperparameters to obtained conclusive findings.

Moreover, when exploring the automated ASHO, the hyperparameter search space is restricted to the three vanilla DNNs. It could be expanded to include other algorithms and more hyperparameters, but this will of course affect the budget and the running time. Finally, the raw point data is used as input features during the AutoML. Exploring the addition of trend lines features and adding this to the configuration space would be sensible to automate the feature selection process.

[2] https://github.com/h-kouame/configuration-space-of-auto-cash

A Set of hyperparameter values evaluated for each algorithm per dataset during the manual tuning process

TABLE A.1: Set of hyperparameter values evaluated for each algorithm per dataset during the manual tuning process

Algorithm	*Hyperparameter*	*Voltage*	*Methane*	*NYSE*	*JSE*
MLP	*Batch size*	{1000, 2000, 4000, 5000}	{250, 500, 1000, 2000, 4000}	{1000, 2000, 5000}	{100, 250, 500, 1000}
	Warm start	{0.1, 0.5, 1.0}	{0.1, 1.0}	{0.1, 0.5, 0.6, 0.7, 1.0}	{0.05, 0.1, 1.0}
	Learning rate	{1e-5, 1e-4, 1e-3, 1e-2}	{1e-4, 1e-3, 1e-2}	{1e-4, 1e-3, 1e-2}	{1e-4, 1e-3, 1e-2}
	Dropout	{0.0, 0.1, 0.5}	{0.0, 0.5}	{0.0, 0.5}	{0.0, 0.1, 0.5}
	Weight decay	{0.0}	{0.0, 5e-4}	{0.0, 5e-5, 5e-4, 5e-3}	{0.0, 5e-4}
	Number of epochs	{500, 1000, 3000, 5000, 10000}	{1000, 5000, 15000}	{50, 100, 400, 500, 600, 1000}	{50, 100, 200}
	Layer configuration	{[500], [500, 400], [500, 400, 300], [500, 400, 300, 200], [500, 400, 300, 200, 100]}	{[500], [500, 400], [500, 400, 300]}	{[500], [500, 400], [500, 400, 300], [500, 400, 300, 200]}	{[10], [20], [100], [500], [500, 400], [500, 400, 300]}
LSTM	*Batch size*	{1000, 2000, 4000, 5000}	{1000, 2000, 4000}	{1000, 2000, 5000}	{250, 500, 1000}
	Warm start	{0.1, 0.5, 1.0}	{0.1, 1.0}	{0.01, 0.05, 0.1, 0.2, 1.0}	{0.05, 0.1, 1.0}
	Learning rate	{1e-4, 1e-3, 1e-2, 1e-1}	{1e-4, 1e-3, 1e-2}	{1e-4, 1e-3, 1e-2}	{1e-4, 1e-3, 1e-2}
	Dropout	{0.0, 0.5}	{0.0, 0.5}	{0.0, 0.5}	{0.0, 0.5}
	Weight decay	{0.0, 5e-4}	{0.0, 5e-4}	{0.0, 5e-6, 5e-5, 5e-4, 5e-3, 5e-2}	{0.0, 5e-4}
	Number of epochs	{1000, 3000, 5000}	{1000, 5000, 10000, 15000}	{10, 50, 100, 200, 500}	{25, 50, 100, 200}
	Cell configuration	{[600], [600, 300], [600, 300, 100]}	{[600], [600, 300], [600, 300, 100], [600, 300, 100]}	{[600], [300], [100], [600, 300] [600, 300, 100]}	{[600], [300], [100], [600, 300]
CNN	*Batch size*	{500, 1000, 2000, 5000}	{250, 500, 1000, 2000, 4000}	{1000, 2000, 5000}	{250, 500, 1000}
	Warm start	{0.1, 0.2, 0.3, 0.4, 0.5, 1.0}	{0.1, 0.2, 0.3, 1.0}	{0.1, 0.2, 0.3, 0.4, 0.5, 1.0}	{0.05, 0.1, 1.0}
	Learning rate	{1e-4, 1e-3, 1e-2}	{1e-4, 1e-3, 1e-2}	{1e-4, 1e-3, 1e-2}	{1e-4, 1e-3, 1e-2}
	Dropout	{0.0, 0.1, 0.5}	{0.0, 0.5}	{0.0, 0.5}	{0.0, 0.5}
	Weight decay	{0.0, 5e-5, 5e-4, 5e-3}	{0.0, 5e-5, 5e-4, 5e-3}	{0.0, 5e-6, 5e-5, 5e-4, 5e-3}	{0.0, 5e-4}
	Number of epochs	{50, 100, 200, 500, 1000, 2000, 5000, 10000, 15000}	{50, 100, 200, 1000}	{50, 100, 200, 1000, 10000, 12000, 15000}	{50, 100, 200} {50, 100, 200}
	Filter configuration	[16]	[32, 32]	[32]	[32, 32]
	Kernel configuration	{1, 2, 4, 8}	{1, 2, 4, 8}	{1, 2, 4, 8}	{1, 2, 4, 8}
	Pooling type	Max, Average, Identity	Max, Average, Identity	Max, Average, Identity	Max, Average, Identity
	Pooling size	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}
RF	*Number of estimators*	{5, 10, 50, 100, 500}	{5, 10, 50, 100, 200}	{10, 50, 100, 200, 400, 500}	{50, 100, 200}
	Maximum depth	{1, 2, 3, 4, 5, 10, None}	{5, 10, 15, None}	{1, 2, 5, 10, 15, None}	{1, 2, 10, None}
	Bootstrap	{False, True}	{False, True }	{False, True}	{False, True}
	Warm start	{False, True}	{False, True }	{False, True}	{False, True}
GBM	*Boosting type*	{gbdt, dart, goss}	{gbdt, dart, goss}	{gbdt, dart, goss}	{gbdt, dart, goss}
	Number of estimators	{1, 2, 5, 10, 25, 50, 100, 200, 500, 1000}	{50, 100, 500, 1000, 5000, 10000}	{1, 5, 10, 50, 100, 500}	{1, 2, 3, 4, 10, 20, 50, 100, 500}
	Learning rate	{1e-3, 1e-2, 5e-2, 1e-1, 1.5e-2, 2e-1, 5e-1}	{1e-2, 1e-1, 2e-1, 5e-1, 9e-1, 1.0, 1.5, 2.0}	{1e-2, 1e-1, 2e-1, 5e-1, 1.0}	{1e-2, 9e-2, 1e-1, 2e-1, 5e-1}
SVR	*Gamma*	{auto, scale, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0}	{auto, scale, 1e-5, 1e-4, 1e-3}	{auto, scale, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0}	{auto, scale, 1e-2, 1e-1, 1.0}
	C	{1, 2, 3, 4, 5, 6, 8, 10, 100, 1000}	{1, 10, 1000, 10000, 100000}	{1, 10, 50, 100, 1000, 10000}	{1, 10, 100, 200, 500, 1000, 2000, 5000, 10000}

B Speed-up obtained from using model update with warm-start

TABLE B.1: The reduction factor in total number of epochs (speed-up), and the average (slope and duration) RMSE using model update with/without warm-start

	MLP		LSTM		CNN	
	Voltage	Methane	Voltage	Methane	Voltage	Methane
Number of splits	8	44	8	44	8	44
Warm-start fraction	0.1	0.1	0.1	0.1	0.5	0.3
Speed-up	4.71	8.30	4.71	8.30	1.78	3.17
RMSE with warm-start	**36.37 ± 0.04**	**49.79 ± 0.19**	**36.41 ± 0.0**	**45.98 ± 0.16**	**36.56 ± 0.16**	**55.11 ± 3.18**
RMSE without warm-start	36.59 ± 0.72	50.80 ± 0.20	36.47 ± 0.02	51.19 ± 0.13	36.59 ± 0.15	56.62 ± 0.64

(A) Voltage and methane dataset

	MLP		LSTM		CNN	
	NYSE	JSE	NYSE	JSE	NYSE	JSE
Number of splits	5	101	5	101	5	101
Warm-start fraction	0.7	0.05	0.01	0.05	0.4	0.1
Speed-up	1.32	16.83	4.81	16.83	1.92	9.18
RMSE with warm-start	**91.50 ± 19.17**	**16.20 ± 0.13**	43.57 ± 0.02	**16.27 ± 0.01**	**97.77 ± 25.10**	**16.22 ± 0.05**
RMSE without warm-start	101.52 ± 16.82	16.21 ± 0.12	**43.49 ± 0.01**	**16.27 ± 0.01**	120.51 ± 19.07	16.33 ± 0.23

(B) NYSE and JSE dataset

C The RMSE values achieved by non-hybrid algorithms after the addition of trend line features

TABLE C.1: The RMSE values achieved by vanilla DNN algorithms on raw data alone and raw data and trend line features.

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
Raw data	9.04 ± 0.06	62.82 ± 0.04	35.93 ± 0.05	14.57 ± 0.10	49.79 ± 4.85	32.18 ± 2.47
Raw data + Trend lines	$\mathbf{9.03 \pm 0.06}$	62.81 ± 0.04	$\mathbf{35.92 \pm 0.05}$	$\mathbf{14.56 \pm 0.19}$	$\mathbf{34.46 \pm 2.79}$	$\mathbf{24.51 \pm 1.49}$
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
Raw data	90.76 ± 4.43	33.08 ± 42.08	61.92 ± 23.26	$\mathbf{19.87 \pm 0.01}$	$\mathbf{12.51 \pm 0.09}$	$\mathbf{16.19 \pm 0.05}$
Raw data + Trend lines	90.45 ± 2.55	25.34 ± 24.09	57.90 ± 13.32	21.13 ± 0.30	12.59 ± 0.14	16.86 ± 0.22

(A) MLP

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
Raw data	10.30 ± 0.0	62.87 ± 0.0	36.59 ± 0.0	$\mathbf{14.21 \pm 0.19}$	56.37 ± 1.77	35.29 ± 0.68
Raw data + Trend lines	10.30 ± 0.0	62.87 ± 0.0	36.59 ± 0.0	14.77 ± 0.51	$\mathbf{48.03 \pm 5.74}$	$\mathbf{31.40 \pm 3.13}$
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
Raw data	86.56 ± 0.01	$\mathbf{0.41 \pm 0.08}$	$\mathbf{43.49 \pm 0.05}$	$\mathbf{19.83 \pm 0.01}$	$\mathbf{12.68 \pm 0.01}$	$\mathbf{16.26 \pm 0.01}$
Raw data + Trend lines	$\mathbf{86.50 \pm 0.01}$	0.47 ± 0.03	$\mathbf{43.49 \pm 0.02}$	20.16 ± 0.03	12.74 ± 0.02	16.45 ± 0.03

(B) LSTM

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
Raw data	$\mathbf{9.24 \pm 0.10}$	$\mathbf{62.40 \pm 0.13}$	$\mathbf{35.82 \pm 0.12}$	15.07 ± 0.35	54.79 ± 4.55	34.93 ± 2.45
Raw data + Trend lines	33.26 ± 19.41	90.78 ± 53.17	62.02 ± 36.29	15.14 ± 0.28	$\mathbf{37.92 \pm 4.11}$	$\mathbf{26.53 \pm 2.20}$
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
Raw data	89.31 ± 1.38	12.21 ± 12.17	50.76 ± 6.78	$\mathbf{19.90 \pm 0.06}$	$\mathbf{12.48 \pm 0.21}$	$\mathbf{16.19 \pm 0.14}$
Raw data + Trend lines	90.44 ± 1.74	14.05 ± 9.52	52.25 ± 5.63	21.41 ± 0.33	12.71 ± 0.15	17.06 ± 0.24

(C) CNN

TABLE C.2: The RMSE values achieved by traditional ML algorithms on raw data alone and raw data and trend line features.

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
Local raw data	9.53 ± 0.0	$\mathbf{63.11 \pm 0.20}$	36.32 ± 0.10	$\mathbf{10.09 \pm 0.01}$	20.79 ± 0.01	$\mathbf{15.44 \pm 0.01}$
Local raw data + Trend lines	$\mathbf{9.35 \pm 0.0}$	63.19 ± 0.29	$\mathbf{36.27 \pm 0.15}$	11.53 ± 0.0	$\mathbf{20.73 \pm 0.01}$	16.13 ± 0.01
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
Local raw data	88.75 ± 0.17	$\mathbf{0.29 \pm 0.0}$	44.52 ± 0.09	$\mathbf{20.21 \pm 0.0}$	$\mathbf{12.67 \pm 0.0}$	$\mathbf{16.44 \pm 0.0}$
Local raw data + Trend lines	86.53 ± 0.01	0.41 ± 0.0	$\mathbf{43.47 \pm 0.01}$	22.68 ± 0.0	12.69 ± 0.0	17.69 ± 0.0

(A) RF

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
Local raw data	$\mathbf{10.0 \pm 0.0}$	62.67 ± 0.0	36.34 ± 0.0	13.05 ± 0.0	75.10 ± 0.0	44.08 ± 0.0
Local raw data + Trend lines	10.01 ± 0.0	$\mathbf{62.63 \pm 0.0}$	$\mathbf{36.32 \pm 0.0}$	$\mathbf{12.02 \pm 0.0}$	$\mathbf{38.34 \pm 0.0}$	$\mathbf{25.18 \pm 0.0}$
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
Local raw data	86.62 ± 0.0	0.42 ± 0.0	43.52 ± 0.0	20.08 ± 0.0	$\mathbf{12.62 \pm 0.0}$	16.35 ± 0.0
Local raw data + Trend lines	$\mathbf{86.42 \pm 0.0}$	$\mathbf{0.41 \pm 0.0}$	$\mathbf{43.42 \pm 0.0}$	$\mathbf{19.93 \pm 0.0}$	12.65 ± 0.0	$\mathbf{16.29 \pm 0.0}$

(B) GBM

	Voltage			Methane		
	Slope	Duration	Average	Slope	Duration	Average
Raw data	$\mathbf{9.32 \pm 0.0}$	$\mathbf{62.58 \pm 0.0}$	$\mathbf{35.95 \pm 0.0}$	14.98 ± 0.0	$\mathbf{34.39 \pm 0.0}$	$\mathbf{24.69 \pm 0.0}$
Raw data + Trend lines	9.54 ± 0.0	62.62 ± 0.0	36.08 ± 0.0	17.95 ± 0.0	34.52 ± 0.0	26.24 ± 0.0
	NYSE			JSE		
	Slope	Duration	Average	Slope	Duration	Average
Raw data	86.55 ± 0.0	0.42 ± 0.0	$\mathbf{43.49 \pm 0.0}$	$\mathbf{20.01 \pm 0.0}$	$\mathbf{12.85 \pm 0.0}$	$\mathbf{16.43 \pm 0.0}$
Raw data + Trend lines	$\mathbf{86.54 \pm 0.0}$	0.45 ± 0.0	43.50 ± 0.0	23.27 ± 0.0	13.19 ± 0.0	18.23 ± 0.0

(C) SVR

References

[1] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," *Proceedings 2001 IEEE International Conference on Data Mining*, pp. 289–296, 2001, ISSN: 15504786. DOI: 10.1109/ICDM.2001.989531. [Online]. Available: http://ieeexplore.ieee.org/document/989531/.

[2] E. Keogh and M. Pazzani, "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback," *Kdd*, vol. 98, pp. 239–243, 1998, ISSN: <null>. DOI: 10.1.1.42.1358. [Online]. Available: http://www.aaai.org/Papers/KDD/1998/KDD98-041.pdf.

[3] T. Lin, T. Guo, and K. Aberer, "Hybrid Neural Networks for Learning the Trend in Time Series," *IJCAI - Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pp. 2273–2279, 2017, ISSN: 10450823. DOI: 10.24963/ijcai.2017/316. [Online]. Available: https://www.ijcai.org/proceedings/2017/316.

[4] P. Wang, H. Wang, and W. Wang, "Finding semantics in time series," *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*, p. 385, 2011, ISSN: 07308078. DOI: 10.1145/1989323.1989364. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1989323.1989364.

[5] L. Ye and E. Keogh, "Time Series Shapelets: A New Primative for Data Mining," *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, p. 947, 2009. DOI: 10.1145/1557019.1557122. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1557019.1557122.

[6] Y. Matsubara, Y. Sakurai, and C. Faloutsos, "Autoplait: Automatic mining of co-evolving time sequences," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14, Snowbird, Utah, USA: Association for Computing Machinery, 2014, 193–204, ISBN: 9781450323765. DOI: 10.1145/2588555.2588556. [Online]. Available: https://doi.org/10.1145/2588555.2588556.

[7] L. Chang, P. Chen, and F. Chang, "Reinforced two-step-ahead weight adjustment technique for online training of recurrent neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 8, pp. 1269–1278, 2012.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *CoRR*, vol. abs/1412.3555, 2014.

[10] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer, and K. Funaya, "Robust Online Time Series Prediction with Recurrent Neural Networks," *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 816–825, 2016. DOI: 10.1109/DSAA.2016.92. [Online]. Available: http://ieeexplore.ieee.org/document/7796970/.

[11] C. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," vol. 191, 2012, ISSN: 0020-0255. DOI: 10.1016/j.ins.2011.12.028. [Online]. Available: https://doi.org/10.1016/j.ins.2011.12.028.

[12] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, 5–32, Oct. 2001, ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. [Online]. Available: https://doi.org/10.1023/A:1010933404324.

[13] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001. DOI: 10.1214/aos/1013203451. [Online]. Available: https://doi.org/10.1214/aos/1013203451.

[14] I. Kumar, K. Dogra, C. Utreja, and P. Yadav, "A comparative study of supervised machine learning algorithms for stock market trend prediction," in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, pp. 1003–1007.

[15] N. Sharma and A. Juneja, "Combining of random forest estimates using lsboost for stock market index prediction," in *2017 2nd International Conference for Convergence in Technology (I2CT)*, 2017, pp. 1199–1202.

[16] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and Robust Automated Machine Learning," *Advances in Neural Information Processing Systems 28*, pp. 2944–2952, 2015, ISSN: 10495258. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf.

[17] B Shahriari, K Swersky, Z Wang, R. P. Adams, and N de Freitas, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016, ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2494218.

[18] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15, Austin, Texas: ACM, 2015, 4:1–4:5, ISBN: 978-1-4503-4006-9. DOI: 10.1145/2834892.2834896. [Online]. Available: http://doi.acm.org/10.1145/2834892.2834896.

[19] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, and U. Berkeley, "HYPERBAND: BANDIT-BASED CONFIGURATION EVAL-UATION FOR HYPER-PARAMETER OPTIMIZATION," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://github.com/automl/RoBO..

[20] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1437–1446. [Online]. Available: http://proceedings.mlr.press/v80/falkner18a.html.

[21] L. Luo and X. Chen, "Integrating piecewise linear representation and weighted support vector machine for stock trading signal prediction," *Applied Soft Computing*, vol. 13, no. 2, pp. 806 –816, 2013, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2012.10.026. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494612004796.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[24] J. G. De Gooijer and R. J. Hyndman, "25 Years of Time Series Forecasting," *International Journal of Forecasting*, vol. 22, no. 3, pp. 443–473, 2006, ISSN: 01692070. DOI: 10.1016/j.ijforecast.2006.01.001. arXiv: Rodgers,J.L.,{\&}Nicewander,W.A.(2008).ThirteenWaystoLookattheCorrelationCoefficient,42(1),59âĂŞ66..

[25] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA," *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, p. 847, 2013, ISSN: 16130073. DOI: 10.1145/2487575.2487629. arXiv: 1208.3719. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487629{\%}0Ahttp://dl.acm.org/citation.cfm?doid=2487575.2487629.

[26] A. Klein, S. Falkner, and S. Bartels, "Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets arXiv : 1605 . 07079v2 [cs . LG] 7 Mar 2017," vol. 54, 2017. arXiv: arXiv:1605.07079v2.

[27] X. Wang, K. Smith, and R. Hyndman, "Characteristic-based clustering for time series data," *Data Min. Knowl. Discov.*, vol. 13, no. 3, 335–364, Nov. 2006, ISSN: 1384-5810. DOI: 10.1007/s10618-005-0039-x. [Online]. Available: https://doi.org/10.1007/s10618-005-0039-x.

[28] X. Wang, K. Smith-Miles, and R. Hyndman, "Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series," *Neurocomputing*, vol. 72, no. 10, pp. 2581 –2594, 2009, Lattice Computing and Natural Computing (JCIS 2007) / Neural Networks in Intelligent Systems Designn (ISDA 2007), ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2008.10.017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231208005134.

[29] N. K. Ahmed, A. F. Atiya, N. El Gayar, and H. El-Shishiny, "An empirical comparison of machine learning models for time series forecasting," *Econometric Reviews*, vol. 29, no. 5, pp. 594–621, 2010, ISSN: 07474938. DOI: 10.1080/07474938.2010.481556.

[30] M. Qi and G. P. Zhang, "Trend time-series modeling and forecasting with neural networks," *IEEE Transactions on Neural Networks*, vol. 19, no. 5, pp. 808–816, 2008, ISSN: 10459227. DOI: 10.1109/TNN.2007.912308.

[31] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases," *Knowledge and Information Systems*, vol. 3, no. 3, pp. 263–286, 2001, ISSN: 0219-1377. DOI: 10.1007/PL00011669. [Online]. Available: http://link.springer.com/10.1007/PL00011669.

[32] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning Time-series Shapelets," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14, New York, NY, USA: ACM, 2014, pp. 392–401, ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623613. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623613.

[33] T. Ferenti, "Biomedical applications of time series analysis," *2017 IEEE 30th Neumann Colloquium (NC)*, pp. 000 083–000 084, 2017. DOI: 10.1109/NC.2017.8263256. [Online]. Available: http://ieeexplore.ieee.org/document/8263256/.

[34] K. Ayankoya, A. P. Calitz, and J. H. Greyling, "Using Neural Networks for Predicting Futures Contract Prices of White Maize in South Africa," *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists on - SAICSIT '16*, pp. 1–10, 2016. DOI: 10.1145/2987491.2987508. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2987491.2987508.

[35] A. Khamis, S. Nabilah, and S. Binti, "Forecasting wheat price using Backpropagation and NARX Neural Network," *The International Journal of Engineering and Science*, vol. 3, no. 11, pp. 19–26, 2014.

[36] M. B. Shrestha and G. R. Bhatta, "Selecting appropriate methodological framework for time series data analysis," 2017. DOI: 10.1016/j.jfds.2017.11.001.

[37] Z. Guo, H. Wang, J. Yang, and D. J. Miller, "A stock market forecasting model combining two-directional two-dimensional principal component analysis and radial basis function neural network," *PLoS ONE*, vol. 10, no. 4, 2015, ISSN: 19326203. DOI: 10.1371/journal.pone.0122385.

[38] S Balkin and J Ord, "Automatic Neural Network Modeling for Univariate Times Series," *International Journal of Forecasting*, vol. 16, pp. 509–515, 2000.

[39] J. Connor, R. Martin, and L. Atlas, "Recurrent Neural Networks and Robust Time Series Prediction," *Neural Networks, IEEE ...*, vol. 5, no. 2, pp. 240–254, 1994, ISSN: 1045-9227. DOI: 10.1109/72.279188. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=279188-.

[40] G. Batres-Estrada, "Deep Learning for Multivariate Financial Time Series," p. 94, 2015.

[41] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[42] S. Aras and T. D. Kocakoç, "A new model selection strategy in time series forecasting with artificial neural networks: IHTS," *Neurocomputing*, vol. 174, pp. 974–987, 2016, ISSN: 18728286. DOI: 10.1016/j.neucom.2015.10.036.

[43] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[44] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzel, "Learning to Diagnose with LSTM Recurrent Neural Networks," 2015. arXiv: 1511.03677. [Online]. Available: http://arxiv.org/abs/1511.03677.

[45] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," 2015.

[46] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27*, Z Ghahramani, M Welling, C Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 3104–3112. [Online]. Available: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, 1097–1105.

[48] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. arXiv: 1409.1556 [cs.CV].

[49] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[51] A. F. Agarap, *Deep learning using rectified linear units (relu)*, 2018. arXiv: 1803.08375 [cs.NE].

[52] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. DOI: 10.3115/v1/D14-1181. [Online]. Available: https://www.aclweb.org/anthology/D14-1181.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[54] M. Wen, P. Li, L. Zhang, and Y. Chen, "Stock Market Trend Prediction Using High-Order Information of Time Series," *IEEE Access*, vol. 7, pp. 28 299–28 308, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2901842. [Online]. Available: https://ieeexplore.ieee.org/document/8653278/.

[55] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001, ISBN: 0262194759.

[56] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, 123–140, Aug. 1996, ISSN: 0885-6125. DOI: 10.1023/A:1018054314350. [Online]. Available: https://doi.org/10.1023/A:1018054314350.

[57] R. Schapire, Y. Freund, P. Bartlett, and W. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," English (US), *Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686, Oct. 1998, ISSN: 0090-5364. DOI: 10.1214/aos/1024691352.

[58] A. Liaw and M. Wiener, "Classification and regression by randomforest," 2007.

[59] L. J. Tashman, "Out-of-sample tests of forecasting accuracy: An analysis and review," *International Journal of Forecasting*, vol. 16, no. 4, pp. 437–450, 2000, ISSN: 01692070. DOI: 10.1016/S0169-2070(00)00065-0.

[60] L. Luo, S. You, Y. Xu, and H. Peng, "Improving the integration of piece wise linear representation and weighted support vector machine for stock trading signal prediction," *Applied Soft Computing Journal*, 2017, ISSN: 15684946. DOI: 10.1016/j.asoc.2017.03.007.

[61] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC Bioinformatics*, vol. 7, pp. 1–8, 2006, ISSN: 14712105. DOI: 10.1186/1471-2105-7-91.

[62] Y. Bao, T. Xiong, and Z. Hu, "Multi-step-ahead time series prediction using multiple-output support vector regression," *Neurocomputing*, vol. 129, pp. 482 –493, 2014, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2013.09.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121300917X.

[63] S. Ben Taieb and A. Atiya, "A bias and variance analysis for multistep-ahead time series forecasting," English, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 1, pp. 62–76, Jan. 2016, ISSN: 2162-237X. DOI: 10.1109/TNNLS.2015.2411629.

[64] A. Venkatraman, M. Hebert, and J. A. Bagnell, "Improving multi-step prediction of learned time series models," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15, Austin, Texas: AAAI Press, 2015, 3024–3030, ISBN: 0262511290.

[65] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases," in *Foundations of Data Organization and Algorithms*, D. B. Lomet, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 69–84, ISBN: 978-3-540-48047-1.

[66] Kin-Pong Chan and Ada Wai-Chee Fu, "Efficient time series matching by wavelets," in *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, 1999, pp. 126–133. DOI: 10.1109/ICDE.1999.754915.

[67] R. A. K.-l. Lin and H. S. S. K. Shim, "Fast similarity search in the presence of noise, scaling, and translation in time-series databases," in *Proceeding of the 21th International Conference on Very Large Data Bases*, Citeseer, 1995, pp. 490–501.

[68] X. Ge and P. Smyth, "Segmental semi-markov models for endpoint detection in plasma etching," *IEEE Transactions on Semiconductor Engineering*, 2001.

[69] J. Hunter and N. McIntosh, "Knowledge-based event detection in complex time series data," in *Proceedings of the Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making*, ser. AIMDM '99, Berlin, Heidelberg: Springer-Verlag, 1999, 271–280, ISBN: 354066162X.

[70] A. Koski, M. Juhola, and M. Meriste, "Syntactic recognition of ecg signals by attributed finite automata," *Pattern Recognition*, vol. 28, no. 12, pp. 1927 –1940, 1995, ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(95)00052-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0031320395000526.

[71] E. J. Keogh and M. J. Pazzani, "Relevance feedback retrieval of time series data," in *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '99, Berkeley, California, USA: Association for Computing Machinery, 1999, 183–190, ISBN: 1581130961. DOI: 10.1145/312624.312676. [Online]. Available: https://doi.org/10.1145/312624.312676.

[72] E. Keogh and P. Smyth, "A probabilistic approach to fast pattern matching in time series databases," in *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, ser. KDD'97, Newport Beach, CA: AAAI Press, 1997, 24–30.

[73] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan, "Mining of concurrent text and time series," in *KDD-2000 Workshop on Text Mining*, Citeseer, vol. 2000, 2000, pp. 37–44.

[74] C.-S. Li, P. S. Yu, and V. Castelli, "Malm: A framework for mining sequence database at multiple abstraction levels," in *Proceedings of the Seventh International Conference on Information and Knowledge Management*, ser. CIKM '98, Bethesda, Maryland, USA: Association for Computing Machinery, 1998, 267–272, ISBN: 1581130619. DOI: 10.1145/288627.288666. [Online]. Available: https://doi.org/10.1145/288627.288666.

[75] R. Osaki, M. Shimada, and K. Uehara, "Extraction of primitive motion for human motion recognition," in *Proceedings of the Second International Conference on Discovery Science*, ser. DS '99, Berlin, Heidelberg: Springer-Verlag, 1999, 351–352, ISBN: 354066713X.

[76] S. Park, S.-W. Kim, and W. W. Chu, "Segment-based approach for subsequence searches in sequence databases," in *Proceedings of the 2001 ACM Symposium on Applied Computing*, ser. SAC '01, Las Vegas, Nevada, USA: Association for Computing Machinery, 2001, 248–252, ISBN: 1581132875. DOI: 10.1145/372202.372334. [Online]. Available: https://doi.org/10.1145/372202.372334.

[77] S. Park, D. Lee, and W. W. Chu, "Fast retrieval of similar subsequences in long sequence databases," in *Proceedings 1999 Workshop on Knowledge and Data Engineering Exchange (KDEX'99) (Cat. No.PR00453)*, 1999, pp. 60–67. DOI: 10.1109/KDEX.1999.836610.

[78] Y. Qu, C. Wang, and X. S. Wang, "Supporting fast search in time series for movement patterns in multiple scales," in *Proceedings of the Seventh International Conference on Information and Knowledge Management*, ser. CIKM '98, Bethesda, Maryland, USA: Association for Computing Machinery, 1998, 251–258, ISBN: 1581130619. DOI: 10.1145/288627.288664. [Online]. Available: https://doi.org/10.1145/288627.288664.

[79] H. Shatkay and S. B. Zdonik, "Approximate queries and representations for large data sequences," in *Proceedings of the Twelfth International Conference on Data Engineering*, 1996, pp. 536–545. DOI: 10.1109/ICDE.1996.492204.

[80] H. J. L. M. Vullings, M. H. G. Verhaegen, and H. B. Verbruggen, "Automated ecg segmentation with dynamic time warping," in *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Vol.20 Biomedical Engineering Towards the Year 2000 and Beyond (Cat. No.98CH36286)*, 1998, 163–166 vol.1. DOI: 10.1109/IEMBS.1998.745863.

[81] Changzhou Wang and X. Sean Wang, "Supporting content-based searches on time series via approximation," in *Proceedings. 12th International Conference on Scientific and Statistica Database Management*, 2000, pp. 69–81. DOI: 10.1109/SSDM.2000.869779.

[82] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, no. 1, pp. 1–16, 2016, ISSN: 21926670. DOI: 10.1007/s13721-016-0125-6.

[83] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997, ISSN: 1089778X. DOI: 10.1109/4235.585893.

[84] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, 1889–1897.

[85] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012, ISSN: 1532-4435. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188385.2188395.

[86] S. Paul, V. Kurin, and S. Whiteson, "Fast efficient hyperparameter tuning for policy gradient methods," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019, pp. 4616–4626. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/743c41a921516b04afde48bb48e28ce6-Paper.pdf.

[87] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009, ISSN: 1931-0145. DOI: 10.1145/1656274.1656278. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278.

[88] K. Swersky, J. Snoek, and R. P. Adams, "Multi-Task Bayesian Optimization," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L Bottou, M Welling, Z Ghahramani, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013,

pp. 2004–2012. [Online]. Available: http://papers.nips.cc/paper/5086-multi-task-bayesian-optimization.pdf.

[89] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*, 2016, pp. 240–248.

[90] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, C. A. C. Coello, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523, ISBN: 978-3-642-25566-3.

[91] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, 6765–6816, Jan. 2017, ISSN: 1532-4435.

[92] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: A python library for model selection and hyperparameter optimization," *Computational Science & Discovery*, vol. 8, no. 1, p. 014 008, 2015. DOI: 10.1088/1749-4699/8/1/014008. [Online]. Available: https://doi.org/10.1088%2F1749-4699%2F8%2F1%2F014008.

[93] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015. arXiv: 1502.01852 [cs.CV].

[94] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. arXiv: 1412.6980 [cs.LG].

[95] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter, "Boah: A tool suite for multi-fidelity bayesian optimization & analysis of hyperparameters," *arXiv:1908.06756 [cs.LG]*, 2019.

[96] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W.-W. Tu, and E. Viegas, "Analysis of the automl challenge series 2015–2018," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Cham: Springer International Publishing, 2019, pp. 177–219, ISBN: 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5_10. [Online]. Available: https://doi.org/10.1007/978-3-030-05318-5_10.

[97] K. Kawabata, Y. Matsubara, and Y. Sakurai, "Automatic sequential pattern mining in data streams," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, ser. CIKM '19, Beijing, China: Association for Computing Machinery, 2019, 1733–1742, ISBN: 9781450369763. DOI: 10.1145/3357384.3358002. [Online]. Available: https://doi.org/10.1145/3357384.3358002.

[98] T. Honda, Y. Matsubara, R. Neyama, M. Abe, and Y. Sakurai, "Multi-aspect mining of complex sensor sequences," in *2019 IEEE International Conference on Data Mining (ICDM)*, 2019, pp. 299–308.