THÈSE DE DOCTORAT DE

L'UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE N° 551
Mathématiques, Informatique, Physique Théorique et Ingénierie des Systèmes

par

ZUOKUN OUYANG

Time Series Forecasting: From Econometrics to Deep Learning

Thèse présentée et soutenue à Orléans, le 11/07/2023
Spécialité de doctorat : Sciences et Technologies Industrielles, Traitement du Signal
Unité de recherche : Université d'Orléans, INSA CVL, Laboratoire PRISME (EA 4229)
Convention CIFRE avec ANRT N° 2019/0551

Rapporteur.trice.s avant soutenance :

Karine ZEITOUNI	Professeure	UVSQ, Université Paris-Saclay, France
Romain TAVENARD	Professeur	Université de Rennes 2, France

Composition du Jury :

Examinateur.trice.s :	Paul HONEINE	Professeur - Université de Rouen Normandie
	Karine ZEITOUNI	Professeure - UVSQ, Université Paris-Saclay
	Romain TAVENARD	Professeur - Université de Rennes 2
	Sylvie TREUILLET	Maître de Conférences (HDR) - Université d'Orléans
Directeur de thèse :	Philippe RAVIER	Professeur - Université d'Orléans
Co-encadrante :	Meryem JABLOUN	Maître de Conférences - Université d'Orléans

Invité.e.s :

Benoit LAHAYE	Président-Directeur Général, ATTILA
Natacha OLIVIER	Manager Business Unit Innovation, La Technopole d'Orléans

To my parents Jia GAO and Weiqing OUYANG,
for their unconditional love and support.

Acknowledgments

Similar to many other journeys like Harry Potter's Hogwarts and Alice's wonderland, my Ph.D. journey at the PRISME Laboratory, University of Orléans, INSA-CVL was full of adventures. There were bad times, especially during the COVID-19 pandemic period, but I have still enjoyed being a doctoral student at PRISME. This thesis cannot be completed without the help of many people. Some of them have helped me directly with this thesis, while others have given their help in other forms or impacted my worldview. I am grateful to have had a chance to meet, work with, and learn from them. In the following, I would like to express my sincere gratitude to:

To my supervisors, Prof. Philippe RAVIER and Assoc. Prof. Meryem JABLOUN at the University of Orléans, INSA-CVL, PRISME. I want to thank Philippe for his patient guidance, thoughtful advice, and tremendous support. I also want to thank Meryem for her dynamic discussion, rigorous revision, and humorous exchanges. As an old Chinese proverb says, "A teacher is one who imparts knowledge, teaches skills, and resolves doubts.（师者，所以传道受业解惑也。——唐·韩愈《师说》）". As supervisors, Philippe and Meryem are competent and strict. As friends, they are sincere and caring. I highly appreciate Philippe and Meryem for all they have done for me on this journey.

To the CEO of ATTILA, Mr. Benoit LAHAYE, for his trust, support, and encouragement. He gave us a chance to complete this industrial thesis and has provided every possible company resource. I also want to thank other company members, in particular: Mr. Pascal SIDOT, Mrs. Mylène MARTIN, Mrs. Marion FRICOT, Ms. Manon VIOSSAT, Mrs. Leslie FALLEAU, Mrs. Laure LAHAYE, and Mrs. Mirella TECHER.

To Prof. Frédéric ROS, Dr. Natacha OLIVIER, and Dr. Gilles MARY from Orléans Val de Loire Technopole, for their great help with my doctor's application.

To Prof. Karim ABED-MERAIM, Assoc. Prof. Sylvie TREUILLET, and Mrs. Denise PELIZZARI CARMES. I want to thank Karim for his philosophical thought and encouragement. I also want to thank Sylvie and Denise for their enormous help these years since I first came to France. My gratefulness also goes to my other colleagues at the PRISME Laboratory, especially: Assoc. Prof. Rodolphe WEBER, Assoc. Prof. Raphaël CANALS, Assoc. Prof. Aladine CHETOUANI, Dr. Antonio DAVALOS, Dr. Dian BAH, Kamel LADROUZE, and Mohamed Amine KERKOURI.

To four of my dear friends, especially Dr. Guanglie OUYANG, Dr. LE Trung Thanh, Zihao LYU, and Min WANG, for their friendship, support, and encouragement. Guanglie is one of the first people I met in France. He is one of my closest friends. We are like real brothers as we share the same family name (even though we are from different parts of China) and have many common hobbies. Trung Thanh (from his first day in the lab, we have been calling him Mr. LE) is a great friend and colleague. He has helped me a lot in many different ways, in terms of both work and life. He has also provided me

with so many invaluable advice for my research, which is highly appreciated. Zihao and Min are both my very good friends, like my family in France. I also want to devote my thankfulness to my other friends, especially: Dr. Marc MONCHI, Mrs. Huiying MONCHI, Dr. Jingwei ZUO, Dr. Yichang WANG, Assoc. Prof. Yong LIU, Assoc. Prof. Shiyong LI, Qi ZHAO, Xiaocheng HONG, Qin HU, Peng PAN, Guanhua SHU, Lin GAN, Yi HONG, and Jun ZHU. I am grateful to have met them.

To the friends I met online who have had some important impact on me: Mingbai, Tian JI, Stormzhang, Caoz, Fenng, MacTalk, and Mr. Hao CHEN, for their great help and inspiration in both my life and career.

To Mr. Jay CHOU and Mr. Eason CHAN for their great music and songs, which have given me a lot of comfort, inspiration, and motivation.

To my parents, Jia GAO and Weiqing OUYANG, for their enormous and unconditional love and support. During the three-year pandemic, I have not even had a chance to return to China to reunite with them. I cannot imagine how much they have missed me, as I have missed them so much. They are always my good listeners, my best friends, and my biggest supporters, though the distance is long and life is short. “I am so grateful to be your son. I love you all.”

To the memory of my uncle Haizhou GAO 高海洲 who passed away at age 64 after a long fight with liver cancer when I was drafting this manuscript, for his love, care, comfort, support, and all the effort he has made for the whole family through all these years. May he rest in peace. I will always miss him.

Thank you!

Orléans, 26 May 2023 Z. O.

Contents

List of Figures

List of Tables

Acronyms

ACF	AutoCorrelation Function
ADMM	Alternating Direction Method of Multipliers
AIC	Akaike's Information Criterion
AR	AutoRegressive
ARCH	AutoRegressive Conditional Heteroskedasticity
ARMA	AutoRegressive Moving Average
ARIMA	AutoRegressive Integrated Moving Average
BIC	Bayesian Information Criterion
BPTT	Back-Propagation Through Time
CART	Classification And Regression Tree
CNN(s)	Convolutional Neural Network(s)
CRM	Client Relationship Management
DFM	Dynamic Factor Model
DL	Deep Learning
DNN(s)	Deep Neural Network(s)
EMSSM	External Memory Augmented State Space Model
EPS	Earnings Per Share
ERP	Enterprise Resource Planning
ETS	ExponenTial Smoothing
FFT	Fast Fourier Transform
FPE	Final Prediction Error
GAN	Generative Adversarial Network
GARCH	Generalized AutoRegressive Conditional Heteroskedasticity
GP	Gaussian Process
GRU	Gated Recurrent Unit
HQ	Hannan-Quinn
HTTP	HyperText Transfer Protocol
IC	Information Criteria
IID	Independent and Identically Distributed
k**NN**	k Nearest Neighbors
LAD	Least Absolute Deviation

LSTM	Long Short-Term Memory
MA	Moving Average
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MAAPE	Mean Arctangent Absolute Percentage Error
MSE	Mean Squared Error
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLP(s)	Multiple Layer Perceptron(s)
MTS	Multivariate Time Series
MTSF	Multivariate Time Series Forecasting
MVC	Model-View-Controller
NID	Normally and Independent Distributed
OLE	Optimized Linear Estimation
OLS	Ordinary Least Square
PACF	Partial AutoCorrelation Function
RF	Random Forest
RNN(s)	Recurrent Neural Network(s)
ROI	Return On Investment
SARIMA	Seasonal AutoRegressive Integrated Moving Average
SC	Schwarz Criterion
SSMs	State Space Models
SVR	Support Vector Machine
TSF	Time Series Forecasting
UMBRAE	Unscaled Mean Bounded Relative Absolute Error
UTS	Univariate Time Series
VAR	Vector AutoRegression
VECM	Vector Error Correction Model
VSMC	Variational Sequential Monte Carlo
WLS	Weighted Least Square

Chapter 1

Introduction

Contents

1.1 Background

The contemporary epoch, often called *the era of data*, has witnessed an unprecedented proliferation of structured and unstructured data. Every day, quintillions of bytes of data are generated, spanning various sectors such as finance, healthcare, social media, and environmental sciences. This surge in data has been paralleled by the evolution of techniques and methodologies designed to analyze, interpret, and harness this data for predictive purposes. One of the most prominent and widely used techniques for forecasting in this data-intensive environment is based on *time series* (TS) data, i.e., *Time Series Forecasting* (TSF).

TS is a series of data points collected over time, such as the stock price, weather, and traffic volume, to name a few. TSF is a fundamental technique in econometrics, statistical learning, and data analysis involving finding historical patterns in TS data to extrapolate them into the future. These patterns may encompass a variety of characteristics, such as trends, seasonality, and cyclic behavior, which form the crux of time series data. Understanding and accurately modeling these components is crucial for effective TSF.

Given its capacity to extrapolate historical data trends and predict future occurrences, TSF techniques have been extensively adopted across various disciplines. For instance, TSF techniques are employed in the financial sector to forecast stock prices, exchange rates, and market trends, empowering investors and policy-makers with actionable insights for informed decision-making [1]–[3]. In public health, these methods are used to predict future disease incidence or patient admissions, facilitating efficient resource planning and timely interventions. We have seen many applications during these years of the COVID-19 pandemic [4]–[6]. In environmental science, TSF aids in predicting weather patterns, climatic changes, and environmental degradation, providing crucial input for policy formulation and planning [7]. Moreover, in supply chain and operations, TSF techniques are also pivotal in demand forecasting, inventory management, and capacity planning [8]–[11].

1.1.1 Time Series

A time series is a sequence of data points that are recorded over time, typically at regular intervals. These observations can be discrete or continuous, depending on the nature of the data. For instance, the daily temperature of a city is a continuous time series, while the number of daily visitors to a website is a discrete time series.

In a time series, each observation is time-dependent, meaning its value is influenced by the time at which it was recorded and by the values of its previous observations. This temporal relationship can be further complicated by the presence of trends, which represent a long-term increase or decrease in the values over time. These trends can be linear or nonlinear and are typically driven by a range of factors, including population growth, technological changes, or economic cycles.

Additionally, a time series may exhibit seasonality, characterized by a periodic pattern that repeats over a specific interval. These seasonal patterns could be daily, weekly, monthly, or yearly and are often influenced by natural or social phenomena, such as weather patterns, holidays, or business cycles.

The concept of autocorrelation is also crucial in time series analysis, indicating that the value of an observation at a particular time might be related to the value of a previous observation. Autocorrelation can result from a myriad of factors, including memory effects, feedback loops, or self-reinforcing dynamics.

It is also essential to note that in many real-world contexts, time series data can be multivariate, i.e., *Multivariate Time Series* (MTS), meaning it consists of multiple variables that may interact and exhibit interdependencies. Thus, the analysis of such data requires techniques that can effectively capture and model these intricate relationships.

Analyzing TS involves using statistical and mathematical models to analyze and forecast TS data. These models may consider various characteristics of TS data, which can be used to make predictions of the series' future values.

1.1.2 Challenges for Time Series Forecasting

While TSF has a broad array of applications across diverse disciplines and has significantly evolved, it is a field that grapples with numerous challenges. These challenges, which encompass both technical and conceptual difficulties, have profound implications for the

accuracy, efficiency, and practical applicability of forecasting models.

One fundamental technical challenge in TSF pertains to handling missing or irregular data. Data collection processes are often imperfect in the real world due to equipment failure, human error, or external circumstances. Consequently, time series data may be plagued with missing values, irregular intervals, or errors, which can significantly undermine the performance of forecasting models. Current methods for handling these issues, such as imputation or interpolation, often involve assumptions that may not hold in many cases and can thus introduce additional bias or error into the forecasts.

Another major technical issue is the challenge of non-stationarity. Traditional TSF models assume stationarity, i.e., a constant mean and variance over time. However, many real-world time series data exhibit non-stationarity, with patterns that change over time due to various factors. The presence of trends, seasonality, or sudden changes in the underlying process can violate the assumption of stationarity, leading to poor model performance. While there are methods to transform non-stationary data into stationary forms, these transformations can sometimes oversimplify or distort the underlying dynamics of the data, thereby reducing the accuracy of the forecasts.

A more complex problem in TSF is the management of multivariate time series. In reality, many systems involve multiple interrelated time series, and the ability to forecast these series jointly can provide more accurate and holistic insights. However, multivariate TSF is a highly complex task, as it requires modeling not only the temporal dynamics of each series but also the interdependencies between them. Traditional univariate forecasting models are ill-suited to this task, and while there are multivariate models available, they often involve high computational complexity and can be difficult to interpret.

In addition to these technical issues, TSF also faces the challenge of achieving reliable long-term predictions. While short-term forecasts can often be made with reasonable accuracy, the uncertainty tends to increase with the forecast horizon. This is due to various factors, including the potential for unforeseen events or changes, the accumulation of prediction errors over time, and the limitations of the models themselves. The difficulty of long-term forecasting is a significant impediment in many applications where strategic decisions and planning require reliable forecasts over extended horizons.

The advent of Big Data has added a new dimension to these challenges. With the increasing volume, velocity, and variety of data, traditional econometric methods often struggle with scalability and complexity. The high dimensionality and heterogeneity of Big Data pose additional challenges for TSF, including computational efficiency, overfitting, and the curse of dimensionality. Furthermore, Big Data often involves noisy, unstructured, or semi-structured data, which require more advanced preprocessing and feature extraction techniques.

1.2 General Objectives

As described in the previous section, TSF demands a wide range of amelioration. In this thesis, we contribute to TSF in the following three aspects.

- **Time series decomposition.** Time series decomposition is an invaluable tool in forecasting and data analysis, facilitating the isolation of specific components such as trends, seasonality, and random noise within a series. This technique is

often times used as a preprocessing step to improve the accuracy of forecasting models. However, choosing and applying appropriate decomposition techniques for different scenarios and forecasting models remain problematic. The improvement in forecasting accuracy is also not guaranteed, and in some cases, decomposition can even harm the performance of forecasting models. We want to investigate the impact of decomposition on different forecasting models and provide insights into the best practices for applying decomposition techniques.

- **Multi-step forecasting.** In reality, generating reliable and accurate multi-step forecasts poses significant challenges. Traditional methods often suffer from error accumulation, where inaccuracies in early forecasts compound in later ones. Furthermore, the issue of maintaining model performance in the face of non-stationary and high-dimensional data exacerbates the difficulty of multi-step forecasting. In this thesis, we will explore and address these problems, aiming to improve the accuracy and reliability of multi-step forecasts, particularly with deep learning models.

- **Integrations of econometric methods.** In addition to these technical issues, another area warrants attention: integrating machine learning (ML) and deep learning (DL) techniques with traditional econometric methods. ML and DL methods have shown great promise in handling high-dimensional and complex data and have achieved remarkable successes in many areas of data science. However, their application in TSF is still a relatively new and rapidly evolving field, and there are many unresolved issues and potential areas for improvement. For instance, how can we best incorporate econometric knowledge into ML models? How can we design deep architectures to detect correlation? Is it possible to integrate decomposition techniques with deep models so we can improve their performance? These questions are crucial for the development of effective and practical forecasting models. We try to answer them by proposing new models and methodologies in this thesis.

In conclusion, while TSF has made significant strides and has a wide range of applications, many challenges and open questions still need to be addressed. Solving these issues requires not only technical innovations but also a deep understanding of the underlying principles and assumptions of TSF models. Careful consideration of different applications' practical needs and constraints is also vital. Hopefully, this thesis will contribute to this ongoing effort by providing new insights and methodologies for TSF.

1.3 Thesis Description

1.3.1 Thesis Outline

The rest of this thesis is organized into the following eight chapters. See Fig. 1.1 for an overview.

The first two chapters are dedicated to state of the art for econometric TSF methods and DL models for TFS, respectively.

- Chapter 2 introduces the mathematical concepts and background behind econometric methods for TSF. It covers the basics of time series analysis and discusses

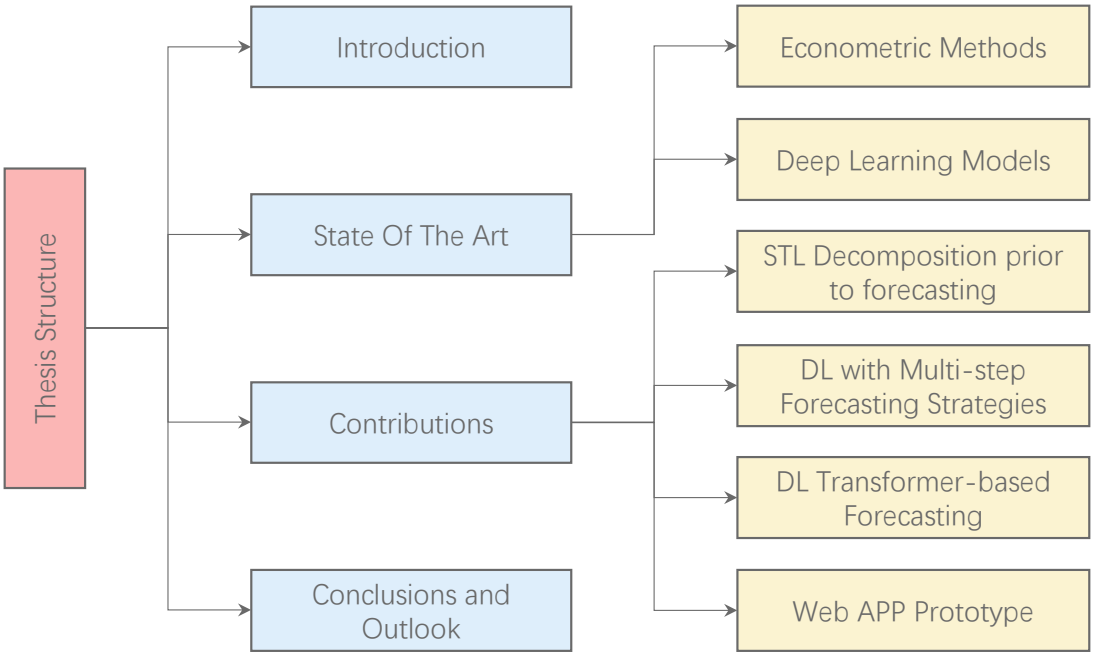


Figure 1.1: Thesis structure.

the main econometric methods for TSF, including AutoRegressing Integrated Moving Average (ARIMA), ExponenTial Smoothing (ETS), and Vector AutoRegressing (VAR) model. This chapter also presents time series decompositions prior to forecasting as well as the decomposition-based Theta method and its generalizations/variants.

- Chapter 3 introduces the state of the art of traditional DL models for TSF. We discuss the bottleneck of econometric and ML models and present the main DL models for TSF and their main challenges. We also discuss several hybrid models combining econometric and DL models. The later part of this chapter revisits the Transformer model and provides a comprehensive overview of the Transformer model and its variants for the TSF problem, including 14 different Transformer-based models. We discuss their improvements over the vanilla Transformer as well as their precursors.

Our main contributions are presented in the following four chapters.

- Chapter 4 investigates the effect of putting STL decomposition as a preprocessing step for econometric and ML models. We compare three econometric and five ML methods. Extensive experiments are conducted on the M3-Competition datasets. The results show that when applied to the monthly industrial M3-Competition dataset as a preprocessing step, STL decomposition can benefit forecasting using econometric methods but harms ML ones. We also discuss the reasons behind these results.

- Chapter 5 analyzes three DL models for the multivariate TSF problem under five forecasting strategies for multi-step forecasting. We compare the performance of

these models under different strategies on various datasets. Our results reveal that these models constantly suffer from accumulated errors under the Recursive strategy and thus cannot carry out actual multi-step forecasting tasks. However, carefully combining them with the MIMO/MISMO strategy can tackle this problem and thus enables one-step-ahead deep learning models for multi-step forecasting. Furthermore, we discuss the effect on models' capacity to capture temporal patterns when combined with different forecasting strategies facing different series seasonalities.

- Chapter 6 proposes a novel Transformer-based model, namely Rankformer, leveraging the rank correlation function and decomposition architecture for long-term TSF tasks. We also present STLformer, an updated version of Rankformer that utilizes the STL decomposition architecture to improve forecasting performance. The combination of econometric methods and the advanced DL models enables Rankformer and STLformer to outperform four state-of-the-art Transformers and two RNN models across multiple datasets and forecasting horizons.

- Chapter 7 outlines a web application prototype's design, implementation, and deployment. It starts with the functional requirements, then details the design aspects, including user interface, technology stack, architecture, core functionalities, and embedded forecasting algorithms. Python, Flask, Bootstrap, and Plotly were used for development, following an MVC design pattern for easy maintenance and extension. The application is dockerized for seamless execution on any compatible machine. The chapter concludes by discussing the application's deployment and maintenance processes.

Chapter 8 concludes the thesis with our main results and provides an outlook on future work. Particularly, we present this thesis's main contributions to econometric and DL domains, respectively, and discuss our work's limitations. We also present several research challenges and open problems that can be considered for future TSF problems research. They are data, model, and task requirements. Some possible solutions are also discussed.

1.3.2 List of Publications

Most of the above results have been published/submitted in the following papers:

[1] **Z. Ouyang**, P. Ravier, and M. Jabloun, "STL Decomposition of Time Series Can Benefit Forecasting Done by Statistical Methods but Not by Machine Learning Ones," *Eng. Proc.*, vol. 5, no. 1, p. 42, 2021.

[2] **Z. Ouyang**, P. Ravier, and M. Jabloun, "Are Deep Learning Models Practically Good as Promised? A Strategic Comparison of Deep Learning Models for Time Series Forecasting," in *Proc. EUSIPCO*, 2022.

[3] **Z. Ouyang**, P. Ravier, and M. Jabloun, "Une comparaison des modèles d'apprentissage profond combinés avec des différentes stratégies pour la prédiction multi-étape des séries temporelles," in *Proc. GRETSI*, 2022.

[4] G. Ouyang, K. Abed-Meraim, and **Z. Ouyang**, "Magnetic-Field-Based Indoor Positioning Using Temporal Convolutional Networks," *Sensors*, vol. 23, no. 3, p. 1514, 2023.

[5] **Z. Ouyang**, M. Jabloun, and P. Ravier, "Rankformer: Leveraging Rank Correlation for Transformer-based Time Series Forecasting," in *Proc. IEEE SSP*, 2023.

[6] **Z. Ouyang**, M. Jabloun, and P. Ravier, "STLformer: Exploit STL decomposition and Rank Correlation for Time Series Forecasting," in *Proc. EUSIPCO*, 2023.

Appendix: Company Introduction

This thesis is funded by ATTILA Gestion, under the CIFRE[1] convention $N°2019/0551$ contracted with the French National Association for Research and Technology (ANRT). This section provides a brief introduction to the company.

Research Context in the Company

ATTILA is a well-established franchise network in the French roofing market, specializing in repairing, maintaining, and preserving all types of roofs. With over 100 agencies and 750 collaborators throughout France, the company has grown rapidly over the years, catering to an expanding customer base and fulfilling various requests. As the company continues to expand, ATTILA recognizes the need to provide better support to its collaborators and customers. In order to address this need, ATTILA is looking to develop an assistance system that can:

1. Predict the future performance of each agency;
2. Detect and notify potential problems;
3. Provide tailored advice on best practices and methodologies.

By leveraging advanced technologies and data-driven insights, ATTILA aims to enhance its overall efficiency and quality of service, ensuring continued success and growth in the competitive roofing market.

Therefore, ATTILA proposed this CIFRE thesis to address these challenges. The aim of the thesis is to analyze the evolution of the agencies' data and predict their performances. Thus, the thesis will compare classical time series analysis methods and machine learning techniques. Additionally, the thesis will focus on developing algorithms to be integrated into the assistance system that can detect and notify the occurrence of problems and provide appropriate advice to help improve the methodologies used by ATTILA's collaborators. Overall, this program will help ATTILA better support its collaborators and improve its services to its customers.

This CIFRE thesis aims to develop a model that can predict the performance of ATTILA agencies in terms of sales and profitability. The model will be used to help the company make better decisions regarding the expansion of its network. The model will

[1] Conventions Industrielles de Formation par la REcherche

be based on the data collected by the company, which will be used to train a machine-learning (ML) algorithm. The model will be used to predict an agency's performance, allowing the company to manage its resources better and provide better service to its clients.

The company's data are collected from our internal system ATLAS, where each agency can report its indicators, such as Client Relationship Management (CRM), Enterprise Resource Planning (ERP), and accounting systems, to name a few. The data are then processed and stored in a data warehouse maintained by MB CLOUD, which we will use to train the ML algorithm. The data are also used to generate reports and dashboards that are used by the company's management to make decisions.

In this thesis, data are treated in the form of time series, as the data are collected on a daily/monthly basis. We will focus on TSF techniques and applying the developed models to the company's data. By comparing the forecasted values with the actual values, we will be able to evaluate the agency's performance and help the local manager/management department determine what to do for the agency's needs.

The History of ATTILA

In 2001, while working as a construction manager for a roofing company, Mr. Benoît LAHAYE had several reflections regarding the roofing industry, which led to the founding of ATTILA. These reflections were:

- Construction companies tend to prioritize big repairs and neglect small and medium-sized roof repairs;
- Roof maintenance is often overlooked;
- Roofs are not well-formed in terms of industrial sealing or safety;
- Roofing companies lack proper structuring.

In 2002, Mr. LAHAYE collaborated with two robotics technicians to design a prototype roof-cleaning robot, which they named ATTILA. See Fig. 2. They tested the robot's capabilities on a church roof, and the test was a resounding success.

Mr. LAHAYE's realization and prototype success paved the way for his innovative concept of a specialized sign solely focused on roof maintenance and repair, emphasizing the preservation of "Capital-toit". In 2003, he founded his own company, called *ATTILA System*[2], paying tribute to his robot prototype. In 2006, Mr. LAHAYE won the gold medal in the Lépine contest for his roof cleaning system.[3][4] As demand grew, he decided to expand his business into franchise mode that same year.

Nowadays, ATTILA has established itself as a prominent national network that specializes in the repair and maintenance of diverse types of roofs and has expanded its operations with over 100 branches located across France. The company has established a new profession with its unique market offering, providing expertise to both professionals

[2]The company was later renamed to *ATTILA* on January 1st, 2018.

[3]`https://www.concours-lepine.com/wp-content/uploads/2012/04/attila124.pdf`

[4]`https://www.concours-lepine.com/palmares/2006-2/concours-lepine-regional-strasbourg-2006/`

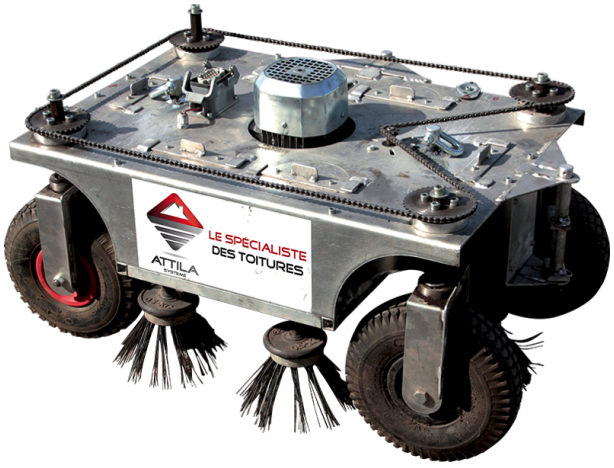


Figure 2: Prototype of the robot: ATTILA.

and private individuals to protect their roofs as a valuable asset. Thanks to its experience and the expertise of its employees, ATTILA achieved a turnover of 83 million euros in 2021. As of 2022, the network has more than 750 employees and over 100 agencies across the country, with plans to expand to 200 agencies by 2025. The company's growth has resulted in approximately ten new agencies opening each year.

Business Activities

Missions

ATTILA provides comprehensive, innovative, and personalized solutions for repairing, renovating, and maintaining roofs of all types, including tiles, slates, chimneys, gutters, downspouts, skylights, cladding, and asbestos zones. ATTILA serves a diverse range of clients, including industry professionals, property managers, private and public organizations, and homeowners with new or old roofs, whether they are tenants or owners of their premises.

ATTILA is committed to preserving the roof from the end of its construction to the day before its refurbishment. Regular maintenance and minor repairs are essential to prolonging the lifespan of the roof, and ATTILA's services are designed to achieve just that. On average, maintaining a roof for 20 years costs five times less than completely refurbishing it. ATTILA has created a new profession that ensures the sustainability of roofs in an ecological context.

At ATTILA, four types of services are provided to clients, regardless of their type of roof:

- The first service involves conducting a diagnostic assessment of the roof and its surroundings, which includes identifying and addressing leaks and performing a comprehensive audit of the roof and its associated elements.

- The second service pertains to repairing and maintaining roofs, whether industrial, terrace, or traditional. ATTILA performs the necessary repairs to make the roof safer and more durable and takes measures to prevent external elements from affecting the roof.

- The third service focuses on improving the additional elements of roofs, such as those that provide natural light, manage rainwater and prevent pest infestation. ATTILA installs and maintains these elements to prolong the roof's life.

- Finally, the fourth and last service involves emergency management. When a delay in intervention can cause additional damage or endanger lives, ATTILA provides rapid water removal to limit water damage, temporary waterproofing, or prevention of the fall of roof elements.

All of these services have a common goal: to effectively protect the clients' "Capital-toit" while adhering to safety regulations.

Company Structure

ATTILA has several departments that collaborate to ensure the efficient operation of the franchise network. The company's headquarters is located in the city of Montargis, France, and the main activities of ATTILA are divided into the following departments:

1. ATTILA has a *Management Department* called ATTILA Gestion whose objective is to drive all services and make strategic and operational decisions. This department ensures the proper functioning of the franchise network, aiming to meet the needs of clients and employees.

2. The *Marketing Department* sets up various communication tools to take care of the ATTILA brand. They work on branding elements such as building and vehicle dressing, signage on construction sites, brochures, flyers, business cards, and stands. Additionally, the department manages internal communication through a weekly newsletter and a quarterly journal.

3. ATTILA also has a *Purchasing Department* whose responsibility is to control the quality/price ratio offered to customers by all agencies. The department provides technical and commercial support to ensure that the products provided by the agencies meet quality standards. This department also communicates with the suppliers to address any concerns the agencies may have about the products.

4. ATTILA has an *IT Department* called MB CLOUD, which manages the company's IT tools, including the computer network, databases, websites, and intranet. This department ensures that all IT tools are optimally and well-maintained to guarantee the company's proper functioning.

5. A *Training Department* helps every network member learn best practices, regardless of their position. Indeed, the training center offers numerous safety, technique, management, office automation, and professional skills courses. The training department calls on many internal or external speakers to carry out its missions successfully. Finally, this department also serves as a point of contact between members of the different franchises. These courses allow employees from each franchise to meet people from different cities in France who also work at ATTILA, with whom they can exchange their experiences and thus create bonds.

6. In order to fulfill its role as a building company, ATTILA has a *Technical Department*. The objective of this department is to provide support for building sites and to keep an eye on the managers' tools, such as correcting estimates, providing safety information, and technical assistance. Moreover, this department creates technical training programs for the employees, creating a solid link with the Training department.

7. Once the employees have received training, the *Animation Department* takes over. This department acts as the intermediary between the head-end and the network. The animators verify that ATTILA's proprietary techniques are being implemented correctly by the network members. They then identify and promote good practices proposed and experienced by the collaborators directly in the field.

8. In order to efficiently manage multi-site clients, ATTILA has established the *GrandsComptes Department*. Its purpose is to centralize the administrative, commercial, and technical needs and requests of multi-site clients such as Conforama, E. Leclerc, and Point.P.

9. ATTILA also relies on its *Wealth Management Department* for everything related to its premises. Indeed, the objective of this department is to technically, financially, and regulatory manage the buildings used by ATTILA Gestion. It also has a mission to ensure legal monitoring of premises regulation and workplace safety. The Wealth Management department is available for network members to answer questions regarding their premises and arrangement.

10. ATTILA has an *Administrative and Financial Department* responsible for registering all the invoices and managing the company's social side (such as pay slips with an external accounting firm). This department also organizes tracking charts and monitors budget management for the headquarter. Additionally, the department manages the general accounting for the headquarter.

11. In order to experiment, test, and prove the efficiency of the ATTILA franchise model, a *Pilot Unit* has been established. This service aims to manage integrated agencies and verify that the orders given by the Service Direction have a positive effect before expanding the model modifications to the entire network. Currently, the pilot unit has four internal agencies: Orange, Montargis, Orléans, and Villefranche. This department aims to identify and address potential issues before they affect the whole network, thus ensuring the success of the franchise model.

12. ATTILA also has a *Franchisee Commission* whose mission is to collect feedback from agencies, propose modifications, which will be implemented after feasibility verification by the relevant department and validation from the franchisor via a voting system, and organize brainstorming meetings if necessary. The commission links franchisees and franchisors, ensuring their voices are heard and considered when making decisions.

In conclusion, ATTILA is a franchise network, and each franchisee operates independently while the headquarter provides support and guidance to ensure the proper functioning of the network. As such, each agency has its organizational structure, which will not be described here.

Chapter 2

Econometric Time Series Forecasting

Contents

This chapter presents a detailed overview of the state-of-the-art econometric methods for time series analysis and forecasting. It is organized as follows: Sec. 2.1 introduces the basic definitions and notations of time series, including the autocovariance function, stationarity, the AutoCorrelation Function (ACF), and the Partial AutoCorrelation Function (PACF). In Sec. 2.2, we present in detail the AutoRegressive Integrated Moving Average (ARIMA), ExponenTial Smoothing (ETS), and Vector AutoRegressive (VAR) families for TSF. Sec. 2.3 presents time series components and two decomposition techniques prior to forecasting. The Theta method based on the concept of decomposition is also explained. We conclude this chapter in Sec. 2.4.

2.1 Time Series Definition and Basics

In this section, some basics of time series are given as preliminary knowledge. We will discuss time series' basic representation, stationarity, and the autocorrelation function.

A time series is a sequence that consists of the historical measurements y_t of an observable variable y at typically equal time intervals. Usually, the observations come from

many different fields as disparate as economy, social sciences, energy, environment, climate, biology, and engineering.

Time series that contains records for only one single variable is called univariate series. In contrast, a time series that includes more than one variable is multivariate. Time series can be either continuous or discrete. Intuitively, the temperature or the flow of a river can be recorded as a continuous-time series, while the number of customers of a company or the population of a city may be counted as a discrete-time series. Usually, a discrete-time series is recorded consecutively at equal time-space, like hourly, daily, monthly, or yearly.

Fig. 2.1 gives an example of a univariate time series with the Sunspots data collected by the National Geographic Data Center.[1]

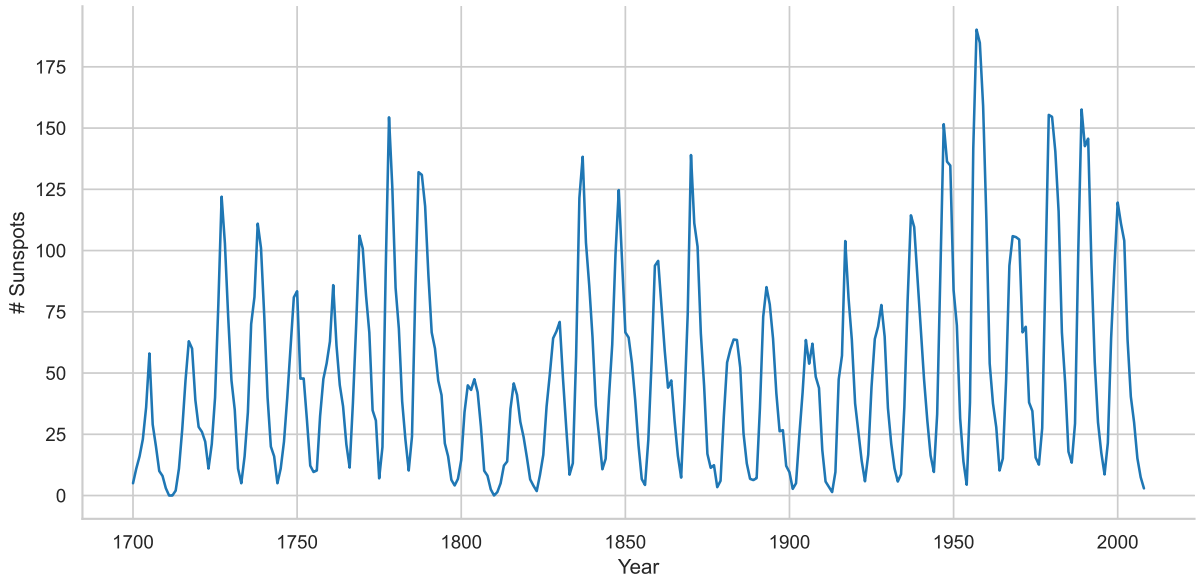


Figure 2.1: Sunspots data from the National Geographic Data Center.

Time series is often represented as a set of observations indexed by typically equal time order:

$$(y_1, y_2, \cdots, y_T).$$

Normally, we can have previous and more recent data. Thus this observation sample can be seen as a finite fragment from an infinitely long sequence $\{y_t\}_{t=-\infty}^{\infty}$:

$$\{y_t\}_{t=-\infty}^{\infty} = (\cdots, y_{-1}, y_0, y_1, y_2, \cdots, y_T, y_{T+1}, y_{T+2}, \cdots).$$

2.1.1 Autocovariance Function and Stationarity

The covariance between two random variables in time series $\{y_t\}$ is called the *autocovariance*, noted as $\text{Cov}(y_s, y_t) = E[(y_s - \mu)(y_t - \mu)]$ where μ is the mean of $\{y_t\}$. If $\text{Cov}(y_s, y_t) = \gamma_{|t-s|}$ depends only on $|t - s|$, we call:

$$\gamma_k = \text{Cov}(y_{t-k}, y_t), \quad k = 0, 1, 2, ...$$

[1]Dataset provided by `statsmodels`: `https://www.statsmodels.org/stable/datasets/generated/sunspots.html`.

the autocovariance function of the time series $\{y_t\}$. It is easy to see $\text{Cov}(y_s, y_t) = \text{Cov}(y_t, y_s)$, so $\gamma_k = \gamma_{-k}$. $\gamma_0 = \text{Var}(y_t)$. Based on the Cauchy-Schwartz inequality, we have:

$$|\gamma_k| = |E[(y_{t-k} - \mu)(y_t - \mu)]| \leq [E(y_{t-k} - \mu)^2 E(y_t - \mu)^2]^{1/2} = \gamma_0$$

where $\mu = E(y_t)$ and $\gamma_0 = E[(y_t - \mu)^2] = \text{Var}(y_t)$.

One crucial property of a time series is its stationarity. A time series is stationary if its econometric properties are invariant with the time at which it is observed. Precisely, the *self-covariance stationary* or *weak stationary* is a more practical condition. We call a time series $\{y_t\}$ weak stationary if it has a limited second-order moment and satisfies:

1. $E(y_t) = \mu$, μ is independent with t.
2. $\text{Var}(y_t) = \gamma_0$, γ_0 is independent with t.
3. $\gamma_k = \text{Cov}(y_{t-k}, y_t), k = 1, 2, ..., \gamma_k$ is independent with t.

2.1.2 Autocorrelation and Partial Autocorrelation Function

Autocorrelation Function

We define the *correlation coefficient* between two random variables X and Y as:

$$\rho(X, Y) = \rho_{x,y} = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} = \frac{E[(X - \mu_x)(Y - \mu_y)]}{\sqrt{E(X - \mu_x)^2 E(Y - \mu_y)^2}}.$$

Then the *autocorrelation coefficients* can be defined as:

$$\rho(y_{t-k}, y_t) = \frac{\text{Cov}(y_{t-k}, y_t)}{\sqrt{\text{Var}(y_{t-k})\text{Var}(y_t)}} = \frac{\gamma_k}{\sqrt{\gamma_0 \gamma_0}} = \frac{\gamma_k}{\gamma_0}, \quad k = 0, 1, 2, ..., \forall t.$$

Denote $\rho_k = \gamma_k / \gamma_0$. It is the correlation coefficient between y_{t-k} and y_t and irrelevant with t. We call ρ_k the *AutoCorrelation Function (ACF)* of y_t, with $\rho_0 = 1$.

Partial Autocorrelation Function

Let $X_1, ..., X_n, Y$, and Z be random variables. Consider the estimation problem:

$$L(Y|X_1, \ldots, X_n) \triangleq \underset{\hat{Y} = a_0 + a_1 X_1 + \cdots + a_n X_n}{\arg\min} E(Y - \hat{Y})^2. \tag{2.1}$$

Equation (2.1) is Y's Optimized Linear Estimations (OLE) on $X_1, ..., X_n$. $E(Y - \hat{Y})^2 = \|Y - \hat{Y}\|^2$ is the mean square error of the estimation. The correlation coefficient between $Y - L(Y|X_1, \ldots, X_n)$ and $Z - L(Z|X_1, \ldots, X_n)$ is called the *partial correlation coefficient* between Y and Z deducting the linear impact of $X_1, ..., X_n$.

Given a stationary time series $\{y_t\}$, we have:

$$L(y_t|y_{t-1}, \ldots, y_{t-n}) = \phi_{n0} + \phi_{n1} y_{t-1} + \cdots + \phi_{nn} y_{t-n}, \quad n = 1, 2, ...$$

where $\phi_{nj}, j = 0, 1, ..., n$ is irrelevant with t. We call ϕ_{nn} the *partial autocorrelation coefficients* of the time series $\{y_t\}$. Series $\{\phi_{nn}\}$ is defined as the *Partial AutoCorrelation Function (PACF)* of $\{y_t\}$. If a limited number, e.g. T, of samples are used to estimate ϕ_{nn}, we have the estimated sample PACF $\hat{\phi}_{nn}, n = 1, 2, ..., T$.

Both ACF and PACF are valuable tools for analyzing time series. The upper half of Fig. 2.2 shows the ACF of the Sunspots time series, and the lower half displays its PACF.

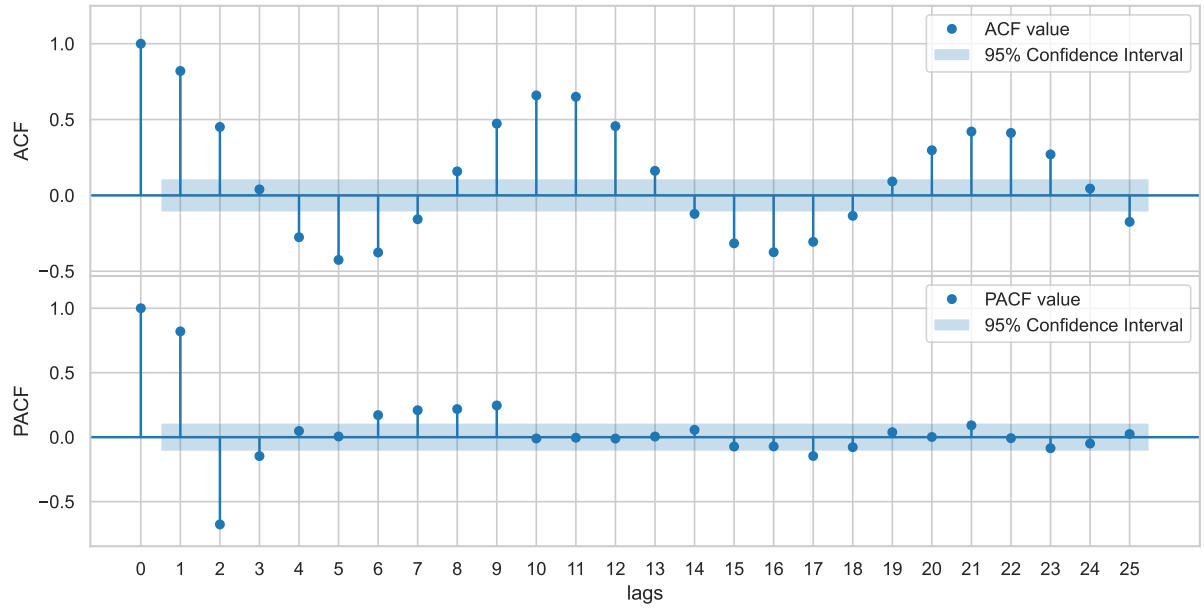


Figure 2.2: ACF and PACF of the Sunspots time series.

2.2 Econometric Time Series Forecasting

2.2.1 AutoRegressive Integrated Moving Average

This subsection discusses the AutoRegressive Integrated Moving Average (ARIMA) family. The author starts with the multivariate linear regression defined as follows.

Given a dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_m, y_m)\}$, with $\boldsymbol{x}_i = (x_{i1}; x_{i2}; ...; x_{id}), y_i \in \mathbb{R}$, a linear regression tries to learn:

$$f(\boldsymbol{x}_i) = \boldsymbol{w}^\top \boldsymbol{x}_i + b \tag{2.2}$$

to make $f(\boldsymbol{x}_i) \simeq y_i$. The parameters $\boldsymbol{w}$ and b can be estimated by the Ordinary Least Square (OLS) method. Fig. 2.3 gives an example of linear regression on a synthetic dataset.

AutoRegression

In multivariate linear regression, we search the relationships among the independent variables x_i and the dependent variable y_i. We can unfold its matrix format in (2.2) into the following representation:

$$y = w_1 x_1 + w_2 x_2 + \cdots + w_N x_N + b.$$

We now substitute the independent variables in the multivariate linear regression model with the past values of y_t:

$$y_t = \phi_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t. \tag{2.3}$$

In many applications, a time series can be represented by the sum of historical measurements and a noise term, as shown in (2.3). AutoRegression (AR) is a model that uses this form to describe the time series.

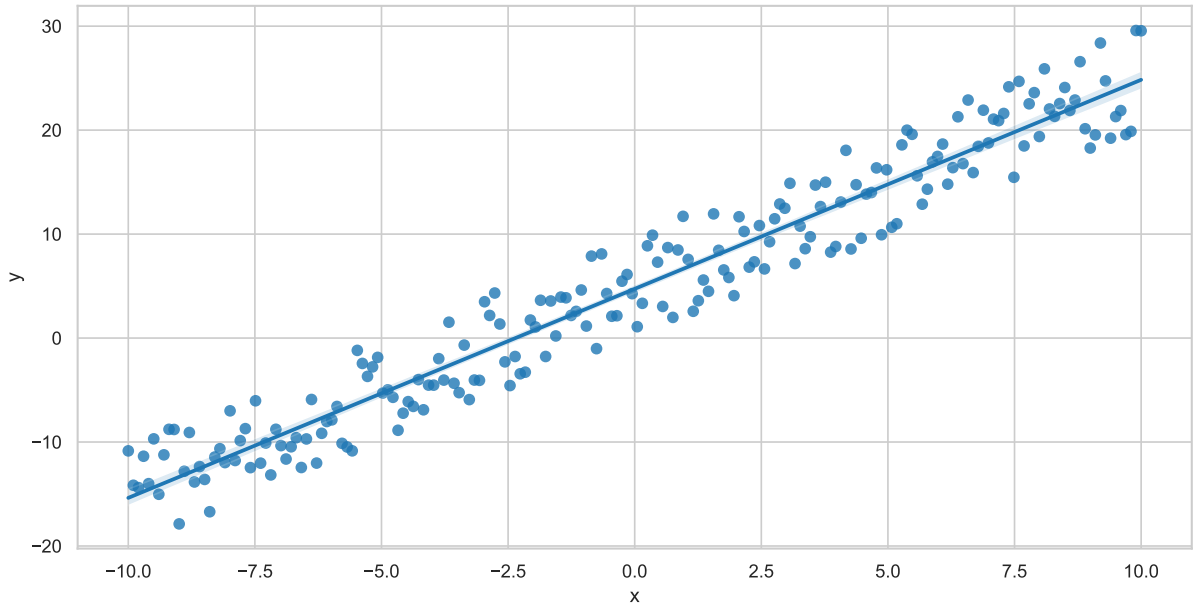


Figure 2.3: An example of linear regression on a synthetic dataset.

The simplest AR model is AR(1):

$$y_t = \phi_0 + \phi_1 y_{t-1} + \varepsilon_t, \tag{2.4}$$

where $\varepsilon_t \sim \text{NID}(0, \sigma^2)$, ϕs are model parameters and $|\phi_1| < 1$.

A generalization of AR(1) is the AR(p) model, which has the exact same representation as (2.3), with a stationary condition that ϕs satisfies that all complex roots z_* of the following equation fulfill $|z_*| > 1$:

$$1 - \phi_1 z - \cdots - \phi_p z^p = 0. \tag{2.5}$$

Equation (2.5) is called the characteristic equation of (2.3). The characteristic equation's complex roots (eigenvectors) must be outside the unit circle.

Often the differencing operator is called *Backshift Operator* and noted $\mathcal{B} y_t = y_{t-1}$. So an AR($p$) in (2.3) can be represented as:

$$\Phi(\mathcal{B}) y_t = \phi_0 + \varepsilon_t, \tag{2.6}$$

where ϕs are represented as $\Phi(\mathcal{B}) = 1 - \sum_{i=1}^{p} \phi_i \mathcal{B}^i$.

Order Determination of AR Model In the real-world application of the AR(p) model, order p is unknown. Two methods are commonly used to determine the proper order: 1) The cut-off of PACF; 2) The minimization of the Information Criterion.

1. The cut-off of PACF. If $\{y_t\}$ obeys the following AR(p) model:

$$y_t = \phi_0 + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t, \quad \phi_p \neq 0,$$

it means when using the linear combination of $y_{t-1}, y_{t-2}, \ldots$ to estimate y_t, only $y_{t-1}, \ldots, y_{t-p}$ are required. Adding $y_{t-p-1}, y_{t-p-2}, \ldots$ will not improve the estimation, which means for $k > p$, $\phi_{kk} = 0$. This phenomenon is called the *cut-off* of an AR(p) model's PACF.

The PACF calculated on T samples $\hat{\phi}_{kk}$ has the following properties:

- When $T \to \infty$, $\hat{\phi}_{pp} \to \phi_p \neq 0$.
- For $k > p$, $\hat{\phi}_{kk} \to 0$ $(T \to \infty)$.
- For $k > p$, the asymptotic variance is $\frac{1}{T}$.

By putting the upper and lower bound of $\pm\frac{2}{\sqrt{T}}$ on the PACF plot, we can exploit it to locate the cut-off position to determine the proper AR order. The lower half of Fig. 2.2 shows the PACF of the sunspots time series. A cut-off can be found at the second position. So choosing an AR(2) model for modeling the sunspots data is favorable.

2. The minimization of the Information Criterion. Information Criteria (IC) are commonly used tools for model comparison. The basic idea behind the information criterion is a compromise between the fitting quality and the model complexity.

Akaike's Information Criterion (AIC) [12] is a frequently used IC. The AIC of an AR(p) model with IID white noise $\varepsilon_t \sim \text{NID}(0, \sigma^2)$ is defined as follows:

$$\text{AIC} = \ln \tilde{\sigma}_k^2 + \frac{2k}{T}, \tag{2.7}$$

k is the order, T is the sample size, and $\tilde{\sigma}_k^2$ is the Maximum Likelihood Estimation (MLE) of ε_t's variance under this order. The first term $\ln \tilde{\sigma}_k^2$ represents the fitting accuracy. A higher value dictates a worse fitting. The second term $\frac{2k}{T}$ is a regularization of the model complexity. A higher value indicates a more complex, thus less stable model with a worse generalization ability for future values. Minimizing the AIC(k) by choosing k in a certain range compromises the fitting quality and the model complexity.

Another commonly used IC is the Bayesian Information Criterion (BIC) [13]:

$$\text{BIC} = \ln \tilde{\sigma}_k^2 + \frac{k \ln T}{T}. \tag{2.8}$$

BIC tends to choose a lower order than AIC.

Moving Average

Consider a special AR(∞) model:

$$y_t = \phi_0 - \sum_{j=1}^{\infty} (-\theta_1)^j y_{t-j} + \varepsilon_t, \tag{2.9}$$

where $0 < |\theta_1| < 1$. Reform (2.9) as:

$$y_t + \sum_{j=1}^{\infty} (-\theta_1)^j y_{t-j} = \phi_0 + \varepsilon_t. \tag{2.10}$$

Replace it with $t-1$ and multiply $-\theta_1$ at both sides:

$$-\theta_1 y_{t-1} + (-\theta_1) \sum_{j=1}^{\infty} (-\theta_1)^j y_{t-1-j} = -\phi_0 \theta_1 - \theta_1 \varepsilon_{t-1}.$$

We then have:

$$\sum_{j=1}^{\infty}(-\theta_1)^j y_{t-j} = -\phi_0\theta_1 - \theta_1\varepsilon_t. \tag{2.11}$$

Put (2.11) into (2.10), we have:

$$y_t = \phi_0(1+\theta_1) + \varepsilon_t + \theta_1\varepsilon_{t-1}.$$

We call a model with such form a 1-order Moving Average (MA) or MA(1) model.

It can be easily derived that an MA model uses the moving average value over white noise to model a time series. MA(1) is the simplest MA model:

$$y_t = \theta_0 + \varepsilon_t + \theta_1\varepsilon_{t-1}, \tag{2.12}$$

where $\varepsilon_t \sim \text{NID}(0, \sigma^2)$, and $|\theta_1| < 1$. It is easy to find that $\{y_t\}$ is a stationary time series.

The general MA(q) model has the following form:

$$y_t = \theta_0 + \varepsilon_t + \theta_1\varepsilon_{t-1} + \cdots + \theta_q\varepsilon_{t-q}. \tag{2.13}$$

Equation (2.13) has a characteristic equation:

$$1 + \theta_1 z + \cdots + \theta_q z^q = 0.$$

If all the eigenvalues are outside the unit circle, the MA model is *invertible*, which will be discussed later in this subsection.

An MA(q) model in (2.13) can also be represented in the form of the backshift operator $\mathcal{B}$:

$$y_t = \theta_0 + \Theta(\mathcal{B})\varepsilon_t, \tag{2.14}$$

where θs are represented as $\Theta(\mathcal{B}) = 1 - \sum_{j=1}^{q}\theta_j\mathcal{B}^j$.

Properties of MA Models Consider the MA(1) model in (2.12). It is easy to derive its expectation, variance, and autocovariance function:

$$E(y_t) = \theta_0, \forall t,\ \text{Var}(y_t) = \sigma^2(1+\theta_1^2),$$

$$\text{Cov}(y_{t-k}, y_t) = E[(y_{t-k}-\theta_0)(y_t-\theta_0)] = \begin{cases} \sigma^2(1+\theta_1^2), & k=0, \\ \sigma^2\theta_1, & k=1, \\ 0, & k>1. \end{cases}$$

Then its ACF should be:

$$\rho_k = \begin{cases} 1, & k=0, \\ \frac{\theta_1}{1+\theta_1^2}, & k=1, \\ 0, & k>1. \end{cases} \tag{2.15}$$

Equation (2.15) verifies that an MA(1) series is weak stationary and presents a cut-off of the ACF at the first lag. It is easy to prove that for an MA(q) model, its ACF also represents this cut-off at the k-th lag, i.e., $\rho_k = 0, \forall k > q$.

For an MA(1) model, when $|\theta_1| < 1$, based on (2.10), we have:

$$\varepsilon_t = -\phi_0 + y_t + \sum_{j=1}^{\infty}(-\theta_1)^j y_{t-j}. \tag{2.16}$$

ε_t represents the incremental information at time t. We call it *innovation*.

Equation (2.16) means that the innovation can be represented with a linear combination of the current observation y_t and historical observation $y_{t-j}, j = 1, 2, ...$ The further the historical value situates, the less impact it has on the innovation. In other words, $\{y_t, y_{t-1}, ...\}$ and $\{\varepsilon_t, \varepsilon_{t-1}, ...\}$ can linearly represent each other, $\forall t \in \mathbb{Z}$. This is called the *invertibility* of the MA model.

Order Determination of MA Model Similar to the AR(p) model, there are also two ways of determining the order of an MA(q) model.

1. The cut-off of the ACF. As discussed in the previous section, the ACF of an MA(q) model cuts off at the q-th position, which can help us to determine its order.

2. The minimization of the IC. On can also use AIC or BIC to determine q.

AutoRegressing Moving Average

The AutoRegressive Moving Average (ARMA) model combines AR and MA while having similar goodness of fit with a simpler structure. An ARMA(p, q) model has the following form:

$$y_t = \phi_0 + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \varepsilon_{t-q}. \tag{2.17}$$

Both the ACF and the PACF of an ARMA model present no cut-off feature. Thus, it is difficult to determine the orders accordingly, but favorable to use the IC.

AutoRegression Integrated Moving Average

The aforementioned AR, MA, and ARMA models are mainly for stationary time series. In many cases, time series can dissatisfy the stationary condition. The *random walk* is a classic one:

$$p_t = p_{t-1} + \varepsilon_t,$$

where $\varepsilon_t \sim \text{NID}(0, \sigma^2)$. A random walk has a very similar formula as the AR(1) in (2.4), but as $\text{Var}(p_t) = \sigma^2 t$, it does not satisfy the stationary condition.

Consider another random walk in the following form:

$$p_t = \mu + p_{t-1} + \varepsilon_t. \tag{2.18}$$

It is easy to learn that $E(p_t|p_0) = p_0 + \mu t$ and $\text{Var}(p_t) = \sigma^2 t$. This means the model has a trend of $y = p_0 + \mu t$. Thus it is called the random walk with a *drift*.

If we substitute ε_t in (2.18) with a stationary ARMA(p, q) series $\{x_t\}$ with $E(x_t) = 0$ and $\text{Var}(x_t) = \sigma_x^2$, we get:

$$y_t = y_{t-1} + \mu + x_t. \tag{2.19}$$

Since $E(y_t|y_0) = y_0 + \mu t, \text{Var}(y_t) = \sigma_x^2 t$, $\{y_t\}$ is non-stationary. But a 1-order differencing operation can make $\{y_t\}$ stationary: $\Delta y_t = y_t - y_{t-1} = \mu + x_t$. We then call

models with the form of (2.19) the AutoRegression Integrated Moving Average model, noted ARIMA$(p, 1, q)$. Here, *integration* is the reverse of differencing operation. An ARIMA$(p, 1, q)$ model can be rewritten with the backshift operator $\mathcal{B}$:

$$y_t - y_{t-1} = (1 - \mathcal{B})y_t = \mu + x_t. \tag{2.20}$$

Sometimes multiple times of differencing can be necessary to make a time series stationary. In this situation, one can construct an ARIMA(p, d, q) model, with p, d, and q corresponding to the orders of AR, differencing, and MA terms.

Tab. 2.1 are some special cases of ARIMA models.

Table 2.1: Special cases of ARIMA models [14].

Model	ARIMA Combination
White noise	ARIMA$(0, 0, 0)$ with no constant
Random walk	ARIMA$(0, 1, 0)$ with no constant
Random walk with drift	ARIMA$(0, 1, 0)$ with a constant
AutoRegression	ARIMA$(p, 0, 0)$
Moving Average	ARIMA$(0, 0, q)$

Parameters Estimation Many estimation methods exist for AR, MA, ARMA, and ARIMA models, such as OLS in linear regression, MLE based on the normal distribution, and Yule-Walker equation-based method. The discussions of these methods are far beyond the content of this thesis and thus omitted.

The implementations of the model estimation methods are widely available on many computational tools, like the `statsmodels` [15] and `pmdarima` [16] libraries in Python and the `forecast` package [17] in R, e.g., the `auto.arima()` function in R library `forecast` and the `arima.auto_arima()` function in Python package `pmdarima` employ a grid search over p, d, and q, estimate the parameters, and by default select the combination of (p, d, q) with the lowest AIC.[2]

Seasonal ARIMA

So far, we have talked about the non-seasonal data and the non-seasonal ARIMA models. Nevertheless, in many cases, especially for economic and financial data, time series can present noticeable periodic changes, e.g., a monthly time series can exhibit similar changes every three months. We call these periodic changes *seasonality*. Time series which contain seasonality are *seasonal time series*. Fig. 2.4 shows the quarterly Earnings Per Share (EPS) of Coca-cola from 1983 Q1 to 2009 Q3.[3] Fig. 2.5 demonstrates the monthly totals

[2]When selecting the best differencing order, the ICs are not always well practical, for the reason that differencing changes the calculation of the likelihood function. It is favorable to find d with other methods and select p and q with AIC_c afterward, which will be discussed later in the section.

[3]`https://gist.github.com/f-loguercio/30b5b55c139f602efea645cc6f4f302b`

of international airline passengers from 1949 to 1960.[4] They exhibit an evident change every four quarters and every 12 months.

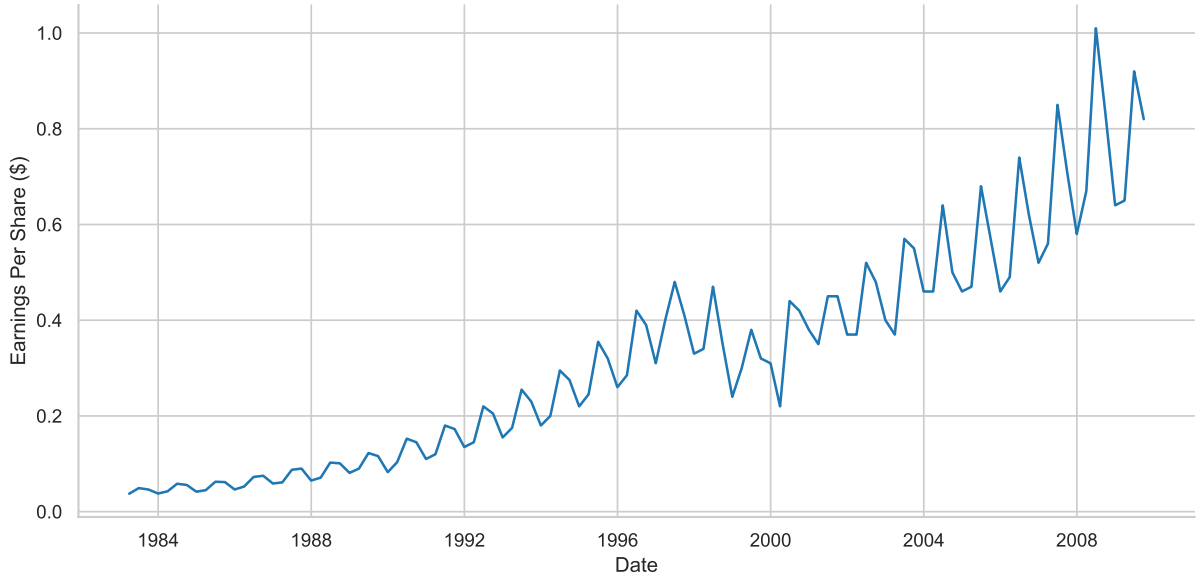


Figure 2.4: Coca-Cola quarterly EPS from 1983 Q1 to 2009 Q3.

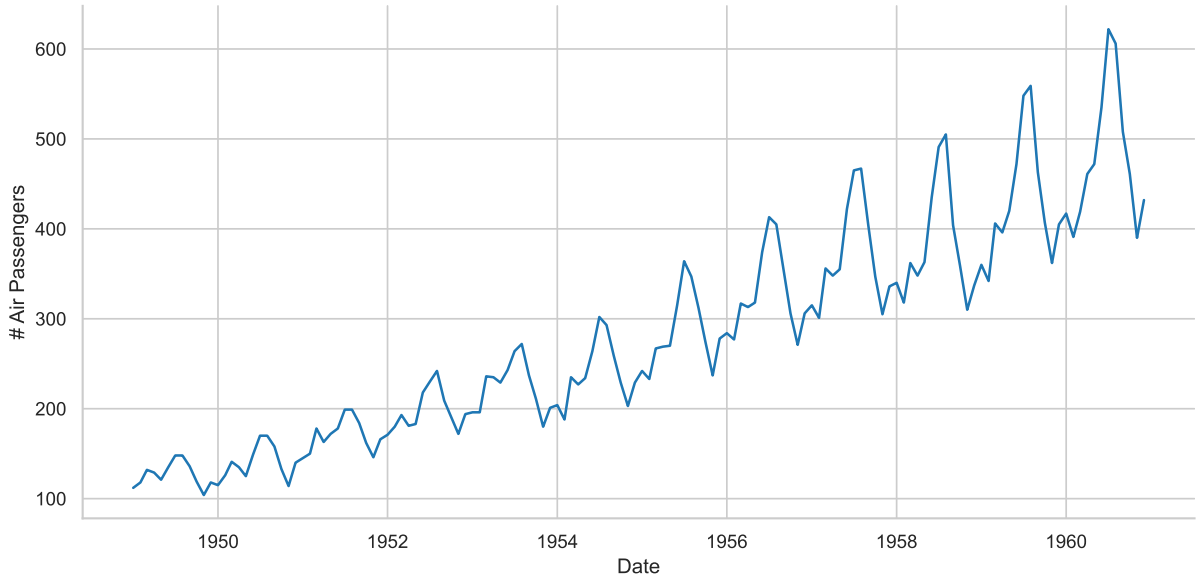


Figure 2.5: Monthly totals of international airline passengers from 1949 to 1960.

ARIMA models can also deal with seasonal series as long as we include additional seasonal terms.

A Seasonal ARIMA model adjusts the time series by differencing it with the seasonal order, e.g., in Coca-Kola quarterly earnings data, we difference the earnings value of the first quarter in the $k+1$-th year with that of the same quarter in the k-th year

[4] `https://rdrr.io/r/datasets/AirPassengers.html`

and perform that for all four quarters. This operation is called *seasonal differencing*: $(1-\mathcal{B}^4)y_t = y_t - y_{t-4}$.

A Seasonal ARIMA model is often written as ARIMA$(p,d,q)(P,D,Q)_s$, where (p,d,q) is the non-seasonal part of the model and $(P,D,Q)_s$ is the seasonal part. s is the series period or seasonality (e.g., 12 for monthly data and 4 for quarterly data). It has the following form:

$$\Phi(\mathcal{B})\Phi_s(\mathcal{B}^s)(1-\mathcal{B})^d(1-\mathcal{B}^s)^D y_t = c + \Theta(\mathcal{B})\Theta_s(\mathcal{B}^s)\varepsilon_t,$$

where $\Phi(\mathcal{B})$ and $\Theta(\mathcal{B})$ are the same parameter representation in (2.6) and (2.14). $\Phi_s(\mathcal{B}^s) = 1 - \sum_{i=1}^{P}\Phi_i\mathcal{B}^{si}$ and $\Theta_s(\mathcal{B}^s) = 1 - \sum_{j=1}^{Q}\Theta_j\mathcal{B}^{sj}$ are the parameter representation for the seasonal part of ARIMA$(p,d,q)(P,D,Q)_s$. A more commonly used form is:

$$(1-\mathcal{B})^d(1-\mathcal{B}^s)^D y_t = c + \frac{\Theta(\mathcal{B})\Theta_s(\mathcal{B}^s)}{\Phi(\mathcal{B})\Phi_s(\mathcal{B}^s)}\varepsilon_t.$$

The Seasonal ARIMA models the seasonal time series with the same non-seasonal ARIMA, and the seasonal terms are then integrated simply by multiplication.

The aforementioned `auto.arima()`/`auto_arima()` functions in R and Python can also perform order selection and parameter estimation for Seasonal ARIMA models.

Forecasting with ARIMA Models

Performing point forecast of ARIMA models is straightforward [14]:

1. Put y_t on the left side of the model equation and all other terms on the right side.
2. Replace t with $T+h$, where T and h are the observation length and forecast horizon.
3. Replace the current errors with 0 and the past errors with residuals.

Consider an ARIMA$(2,1,2)$ model:

$$(1-\phi_1\mathcal{B}-\phi_2\mathcal{B}^2)(1-\mathcal{B})y_t = (1+\theta_1\mathcal{B}+\theta_2\mathcal{B}^2)\varepsilon_t. \tag{2.21}$$

For the first step, reform (2.21) into:

$$y_t = (1+\phi_1)y_{t-1} - (\phi_1-\phi_2)y_{t-2} - \phi_2 y_{t-3} + \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2}. \tag{2.22}$$

Substitute t with $T+1$ (for one-step prediction) in (2.22):

$$y_{T+1} = (1+\phi_1)y_T - (\phi_1-\phi_2)y_{T-1} - \phi_2 y_{T-2} + \varepsilon_{T+1} + \theta_1\varepsilon_T + \theta_2\varepsilon_{T-1}.$$

Then for the third step, we have the forecasting model:

$$\hat{y}_{T+1|T} = (1+\phi_1)y_T - (\phi_1-\phi_2)y_{T-1} - \phi_2 y_{T-2} + \theta_1 e_T + \theta_2 e_{T-1}.$$

For multi-step forecasting, replace future observations with their forecasts and future errors with 0:

$$\begin{aligned}
\hat{y}_{T+2|T} &= (1+\phi_1)\hat{y}_{T+1|T} - (\phi_1-\phi_2)y_T - \phi_2 y_{T-1} + \theta_2 e_T,\\
\hat{y}_{T+3|T} &= (1+\phi_1)\hat{y}_{T+2|T} - (\phi_1-\phi_2)\hat{y}_{T+1|T} - \phi_2 y_T,\\
&\vdots
\end{aligned}$$

This way, all future values can be obtained recursively.

The prediction intervals of ARIMA models are largely beyond the scope of this thesis and thus omitted.

2.2.2 Exponential Smoothing

Simple Exponential Smoothing

Exponential smoothing is initially an intuitive forecasting model. For a time series $\{y_t\}$, exponential smoothing uses the linear combination of historical observations to predict the value in the next time step, with the weights decaying exponentially:

$$wy_{t-1} + w^2 y_{t-2} + \cdots = \sum_{j=1}^{\infty} w^j y_{t-j},$$

where $0 < w < 1$. Since the combination should be a weighted average, by taking $\sum_{j=1}^{\infty} w^j = \frac{w}{1-w}$, we can write the exponential smoothing model as:

$$y_t = (1-w)(y_{t-1} + wy_{t-2} + w^2 y_{t-3} + \cdots) = (1-w)\sum_{j=1}^{\infty} w^{j-1} y_{t-j}. \tag{2.23}$$

Equation (2.23) is called the Simple Exponential Smoothing (SES) model. It has a more commonly used formula with $0 < \alpha < 1$ as the smoothing parameter:

$$y_t = \alpha \sum_{j=0}^{\infty} (1-\alpha)^j y_{t-j}.$$

SES also has a component form [14] with a forecast equation and a level equation:

$$\text{Forecast equation} \quad \hat{y}_{t+h|t} = \ell_t, \tag{2.24}$$

$$\text{Level equation} \quad \ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}. \tag{2.25}$$

ℓ_t is the level of the series at time t. The forecast equation dictates that the forecast value at $t+1$ is the estimated level at time t. The level equation exhibits the form of exponential smoothing and gives the estimation value of the level at each t.

Holt's Trend and Holt-Winters' Trend-Seasonal Method

In 1956, Charles Holt extended the SES to capture the trend [18]:

$$\text{Forecast equation} \quad \hat{y}_{t+h|t} = \ell_t + hb_t, \tag{2.26}$$

$$\text{Level equation} \quad \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}), \tag{2.27}$$

$$\text{Trend equation} \quad b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}. \tag{2.28}$$

$0 < \alpha < 1$ and $0 < \beta^* < 1$ are the smoothing parameters for level and trend respectively. Holt's trend method has a level ℓ_t component consisting of a weighted average of the observation y_t and the combination of historical level and trend $(\ell_{t-1} + b_{t-1})$ at time $t-1$. Holt also leverages the weighted average of the estimated trend $(\ell_t - \ell_{t-1})$ at time t and the previously calculated trend b_{t-1} to model b_t. The forecast is no longer flat as SES but trending.

Holt's trend method has another version where the estimated trend is damped rather than a linear one [19], by adding a damping parameter ϕ into (2.26)-(2.28):

$$\begin{aligned}
\text{Forecast equation} \quad & \hat{y}_{t+h|t} = \ell_t + (\phi + \phi^2 + \cdots + \phi^h) b_t, \\
\text{Level equation} \quad & \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1}), \\
\text{Trend equation} \quad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}.
\end{aligned}$$

$0 < \phi < 1$ is the damping parameter. With the effect of ϕ, the trend decays to a constant at some point in the future. When $h \to \infty$, $\hat{y}_{t+h|t} \to \ell_T + \phi b_T/(1-\phi)$.

In 1957 and 1960, Holt[18] and Winters [20] extended Holt's trend method so that their model can model the seasonality:

$$\begin{aligned}
\text{Forecast equation} \quad & \hat{y}_{t+h|t} = \ell_t + h b_t + s_{t+h-m(k+1)}, && (2.29) \\
\text{Level equation} \quad & \ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}), && (2.30) \\
\text{Trend equation} \quad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*) b_{t-1}, && (2.31) \\
\text{Seasonal equation} \quad & s_t = \gamma^*(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma^*) s_{t-m}. && (2.32)
\end{aligned}$$

m is the seasonal period, and $k = \lfloor (h-1)/m \rfloor$ means the seasonal estimation is from the last period of the seasonal term. The level equation indicates a weighted average between the seasonally adjusted value $(y_t - s_{t-m})$ and the non-seasonal value $(\ell_{t-1} + b_{t-1})$. The seasonal equation shows a weighted average between the current and the same seasons in the last period. α is the smoothing parameter for the level, β^* is the smoothing parameter for the trend, and γ is the smoothing parameter for the season. $0 < \alpha, \beta^* < 1$, $0 < \gamma < (1-\alpha)$ [14].

The equations (2.29)-(2.32) show the additive form of Holt-Winters' seasonal model. Multiplicative seasonality can sometimes exist; thus Holt-Winters' model has a multiplicative form:

$$\begin{aligned}
\text{Forecast equation} \quad & \hat{y}_{t+h|t} = (\ell_t + h b_t) s_{t+h-m(k+1)}, \\
\text{Level equation} \quad & \ell_t = \alpha \frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + b_{t-1}), \\
\text{Trend equation} \quad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*) b_{t-1}, \\
\text{Seasonal equation} \quad & s_t = \gamma^* \frac{y_t}{\ell_{t-1} - b_{t-1}} + (1-\gamma^*) s_{t-m}.
\end{aligned}$$

Like Holt's trend method, Holt-Winters' seasonal method also has a trend-damped version. For the multiplicative seasonal method, its damped version is:

$$\begin{aligned}
\text{Forecast equation} \quad & \hat{y}_{t+h|t} = [\ell_t + (\phi + \phi^2 + \cdots + \phi^h) b_t] s_{t+h-m(k+1)}, \\
\text{Level equation} \quad & \ell_t = \alpha \frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + \phi b_{t-1}), \\
\text{Trend equation} \quad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}, \\
\text{Seasonal equation} \quad & s_t = \gamma^* \frac{y_t}{\ell_{t-1} - \phi b_{t-1}} + (1-\gamma^*) s_{t-m}.
\end{aligned}$$

State Space Models Representation

If we take (2.24) and (2.25) and rearrange the level equation, we have:

$$\begin{aligned}\ell_t &= \alpha y_t + (1-\alpha)\ell_{t-1} \\ &= \ell_{t-1} + \alpha(y_t - \ell_{t-1}) \\ &= \ell_{t-1} + \alpha(y_t - \hat{y}_{t|t-1}) \\ &= \ell_{t-1} + \alpha e_t,\end{aligned}$$

where $e_t = y_t - \ell_{t-1} = y_t - \hat{y}_{t|t-1}$ is the residual of the estimation from time t. This means we must adjust the level for the estimation process through $t = 1, ..., T$. We can also write $y_t = \ell_{t-1} + e_t$ so that the observation of time t can be represented by the previous level plus an error term [14].

If we specify a distribution for e_t, e.g., we choose the normal distribution for the additive error: $\varepsilon_t = e_t \sim \text{NID}(0, \sigma^2)$, we then have a state space model:

$$\text{Measurement equation} \quad y_t = \ell_{t-1} + \varepsilon_t, \tag{2.33}$$

$$\text{State equation} \quad \ell_t = \ell_{t-1} + \alpha\varepsilon_t. \tag{2.34}$$

We call (2.33) and (2.34) an *innovations state space model*, for ε_t represents the incremental information, i.e., the innovation, at time t, as explained in (2.16). The measurement equation shows that the observation y_t has two parts: the predictable part level ℓ_{t-1} and the unpredictable part random error ε_t. The state equation shows how the level changes through time.

Similarly, we can derive the state space model representation for Holt's trend method (with $\beta = \alpha\beta^*, 0 < \beta < \alpha$ for a simpler representation):

$$\text{Measurement equation} \quad y_t = \ell_{t-1} + b_{t-1} + \varepsilon_t, \tag{2.35}$$

$$\text{State equation 1} \quad \ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t, \tag{2.36}$$

$$\text{State equation 2} \quad b_t = b_{t-1} + \beta\varepsilon_t. \tag{2.37}$$

Holt-Winters' seasonal method also has its state space model's representation (with $\gamma = (1 - \alpha\gamma^*), 0 < \gamma < 1 - \alpha$ for simplicity):

$$\begin{aligned}\text{Measurement equation} \quad & y_t = \ell_{t-1} + b_{t-1}\varepsilon_t, \\ \text{State equation 1} \quad & \ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t, \\ \text{State equation 2} \quad & b_t = b_{t-1} + \beta\varepsilon_t, \\ \text{State equation 3} \quad & s_t = s_{t-m} + \gamma\varepsilon_t.\end{aligned}$$

We use a more general notation for state space models: **ETS**$(\cdot, \cdot, \cdot)$. Each $\cdot$ term stands for **E**rror, **T**rend, and **S**easonality, respectively. It can also be interpreted as **E**xponen**T**ial **S**moothing. A taxonomy for ETS models with every component having its own possible values is as follows: Error = {A, M}, Trend = {N, A, A_d}, and Seasonality = {N, A, M}, where "A" refers to "Additive", "A_d" means "Additive damped", "M" means "Multiplicative", and "N" means "None" [14].

The SES model is equivalent to an ETS(A,N,N) model. Holt's trend method equals an ETS(A,A,N) model and ETS(A,A_d,N) for the damped one. The Holt-Winters' additional seasonal method is an ETS(A,A,A), while its multiplicative one corresponds to ETS(A,A,M).

Parameter Estimation and Model Selection

For ETS models, the parameters to estimate are as follows:

- The smoothing parameters: α, β, γ, and ϕ.
- The initial states: ℓ_0, b_0, s_0, $s_{-1}, ..., s_{-m+1}$.

Commonly, we use OLS or MLE method to estimate the parameters. When the error term is additive, OLS and MLE give the same result, while for multiplicative error, MLE will give out different estimations for the parameters [21].

For model selection, apart from the aforementioned AIC and BIC, a corrected version of the AIC, i.e., AIC_c is also used for small sample bias:

$$\text{AIC}_\text{c} = \ln \tilde{\sigma}_k^2 + \frac{2k}{T-k-1}.$$

While multiplicative models are often used for strictly positive time series, they can be numerically unstable if they contain zeros or negative values. Thus, only additive models will be considered in these situations. Functions like `AutoETS()` in `sktime` [22] and `ets()` in `forecast` [17] both provide automatic interfaces for modeling ETS models.

Forecasting with ETS Models

Point forecasting with ETS models is straightforward: Iterating the model with $t = T, T+1, ..., T+h$ and setting $\varepsilon_t = 0, \forall t > T$, we can easily get the point forecast from the ETS model.

For an ETS(A,A,N), retake (2.35) and (2.37):

$$\begin{aligned} y_t &= \ell_{t-1} + b_{t-1} + \varepsilon_t, \\ \ell_t &= \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t, \\ b_t &= b_{t-1} + \beta\varepsilon_t. \end{aligned}$$

Replace t with $T+1$, we have: $y_{T+1} = \ell_T + b_T + \varepsilon_{T+1}$, so $\hat{y}_{T+1|T} = \ell_T + b_T$.

Similarly, $y_{T+2} = \ell_{T+1} + b_{T+1} + \varepsilon_{T+2} = \ell_T + b_T + \alpha\varepsilon_{T+2} + b_T + \beta\varepsilon_{T+1} = \ell_T + 2b_T + (\alpha+\beta)\varepsilon_T$, so $\hat{y}_{T+2|T} = \ell_T + 2b_T$. This is equivalent to the original representation of Holt's trend method in (2.26), (2.27), and (2.28).

ETS models can also generate prediction intervals. For most of the ETS models, the prediction intervals (PI) can be written as:

$$\text{PI} = \hat{y}_{T+h|T} \pm c\hat{\sigma}_h,$$

where $\hat{\sigma}_h$ is the standard deviation of the forecasting, and c is the multiplier to be selected for the coverage probability. For the two commonly used coverage probabilities, i.e., 80% and 95%, c is set to 1.28 and 1.96, respectively. More details are explained in [21].

ARIMA v.s. ETS

ARIMA models and ETS models can be equivalent in some cases. Tab. 2.2 presents an equivalence relationship between some ARIMA and ETS models.

While some ARIMA and ETS models are identical with specific parameter settings, they are generally not interchangeable. Although the information criteria can be beneficial when selecting the orders, they cannot be used among different categories of models, i.e., one cannot use the AIC to compare an ARIMA and an ETS model. Thus, it is usually favorable to perform time series cross-validation to compare ARIMA and ETS models rather than using the information criteria.

Table 2.2: Equivalence relationships between some ARIMA and ETS models [14].

ARIMA model	ETS model	Parameters Setting
ARIMA$(0,1,1)$	ETS(A,N,N)	$\theta_1 = \alpha - 1$
ARIMA$(0,2,2)$	ETS(A,A,N)	$\theta_1 = \alpha + \beta - 2$ $\theta_2 = 1 - \alpha$
ARIMA$(1,1,2)$	ETS(A,A_d,N)	$\phi_1 = \phi$ $\theta_1 = \alpha + \phi\beta - 1 - \phi$ $\theta_2 = (1-\alpha)\phi$
ARIMA$(0,1,m)(0,1,0)_m$	ETS(A,N,A)	-
ARIMA$(0,1,m+1)(0,1,0)_m$	ETS(A,A,A)	-
ARIMA$(1,0,m+1)(0,1,0)_m$	ETS(A,A_d,A)	-

2.2.3 Multivariate Time Series and Vector AutoRegression

So far, all the models we considered are for univariate time series, i.e., only one series is being analyzed. Sometimes, several time-aligned series can exhibit interdependencies among them. We call the combination of multiple time series a *multivariate time series*. One example can be the stock market. The change in one company's stock price can diffuse to other companies promptly and vice versa. The investors are also willing to know their Return On Investment (ROI) relationships among multiple assets. Fig. 2.6 shows the quarterly revenues of Apple Inc. by different product types, which turns out to be a multivariate time series.[5]

Vector AutoRegression Model

The Vector AutoRegressive (VAR) model allows modeling this kind of interdependencies or inter-influence relationships among series (or variables). In a VAR model, all variables are *endogenous* as they influence other variables equally.

[5]Data retrieved from **Apple Investor Relations**: `https://investor.apple.com/investor-relations/default.aspx`.

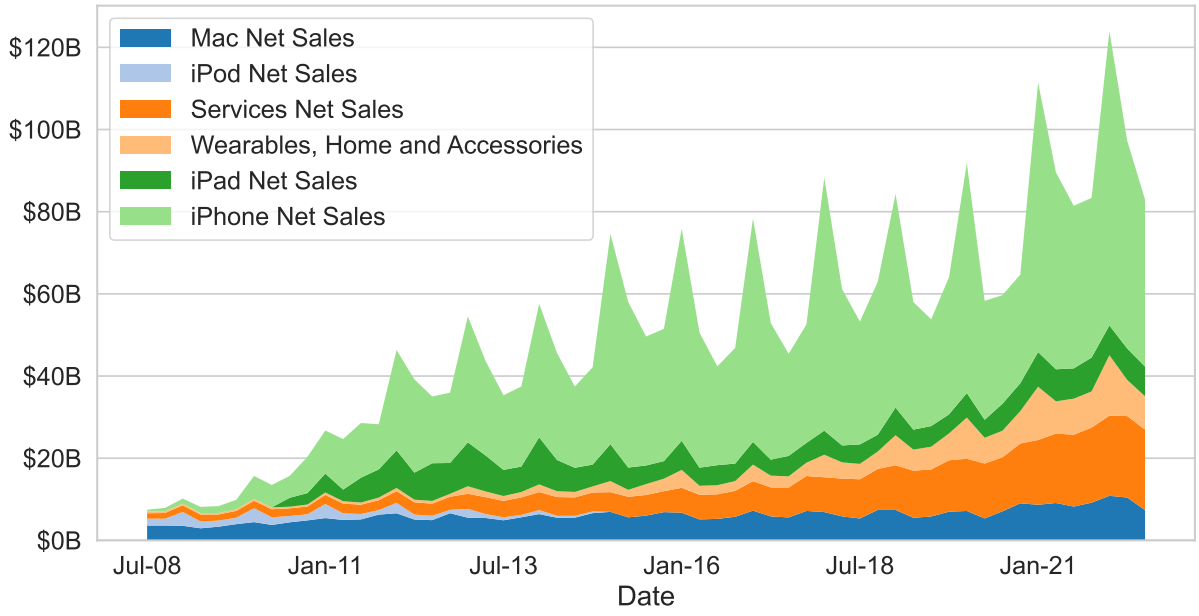


Figure 2.6: Apple Inc. quarter revenue by product type.

This part will only discuss the simplest VAR model: VAR(1). Other VAR models with higher orders are much beyond the scope of the thesis. The content about these models can be found in [23]. A 2-dimensional VAR(1) model has the following form:

$$y_{1,t} = \phi_{1,0} + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-1} + \varepsilon_{1,t},$$
$$y_{2,t} = \phi_{2,0} + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-1} + \varepsilon_{2,t}.$$

$\phi_{i,0}$ is the constant term and $\phi_{ij,\ell}$ means the influence level that y_j has at time ℓ on variable y_i. $\varepsilon_{i,t}$ are white noise processes.

Parameter Estimation and Model Selection

If the multivariate series are stationary, we can directly fit the VAR model with the historical data, and the model is called *VAR in levels*. When the series are non-stationary, we need to use the differences of the historical observations to fit the model, which is called *VAR in differences*. For each case, the OLS method is performed on $\varepsilon_{i,t}$ to estimate the parameters.

For higher-order VAR(p) models, information criteria are also applicable for determining the order p. Apart from the AIC, BIC, and AIC$_c$, Lütkepohl [23] suggested two other information criteria: The Final Prediction Error (FPE) criterion and the Hannan-Quinn (HQ) criterion. Here are the vectorized AIC, BIC, FPE, and HQ definitions.

The vectorized AIC is defined as:

$$\mathrm{AIC}(p) = \ln \det\left(\tilde{\Sigma}_u(p)\right) + \frac{2}{T} p K^2.$$

The vectorized BIC (often called the Schwarz Criterion (SC)) is defined as:

$$\mathrm{SC}(p) = \ln \det\left(\tilde{\Sigma}_u(p)\right) + \frac{\ln T}{T} p K^2.$$

The FPE criterion is defined as:

$$\mathrm{FPE}(p) = \left[\frac{T + Kp + 1}{T - Kp - 1}\right]^K \det\left(\tilde{\Sigma}_\mu(p)\right).$$

Moreover, the HQ criterion is defined as follows:

$$\mathrm{HQ}(p) = \ln\left|\tilde{\Sigma}_\mu(p)\right| + \frac{2\ln\left(\ln T\right)}{T} pK^2,$$

where T is the sample size, K is the dimension of the time series, and $\tilde{\Sigma}_\mu(p)$ is an estimate of the covariance matrix $\Sigma_\mu(p)$ of the white noise given by the approximation of the one-step-ahead forecast MSE.

The functions `VAR()` in R package `vars` [24] and in Python library `statsmodels` [15] both use the four ICs mentioned above to determine VAR's order.

Forecasting with VAR Models

VAR models can provide predictions for every variable included. The forecasting is generated iteratively by following the same process for ARIMA models:

1. Put y_ts on the left side of the model equation and all other terms on the right side.
2. Replace t with $T+h$, where T and h are the observation length and forecast horizon.
3. Replace the current errors with 0.

Every VAR(p) model for K-dimension time series contains $K \times (1 + pK)$ parameters. For example, a VAR(3) model for a 5-dimension time series has 80 parameters. Due to the high computation complexity which grows with the order, it is often favorable to choose SC, for it reports lower orders than AIC when choosing the proper order for VAR models for prediction.

2.3 Time Series Decomposition Prior to Forecasting

In this section, we present a helpful tool to analyze time series data, i.e., time series decomposition that segregates a time series into several components, as well as another decomposition-based econometric method, i.e., the Theta method.

2.3.1 Time Series Components

A time series can have many underlying patterns, and decomposition is one method that can reveal them by splitting a time series into several different components. A time series that consists of several components can be decomposed additively into the following form:

$$y_t = S_t + T_t + R_t, \tag{2.38}$$

where S_t, T_t, and R_t represent the seasonal, trend-cycle, and residual terms. We call this a *decomposition* of time series. Furthermore, (2.38) represents an *additive decomposition*.

A time series can be decomposed in a multiplicative way:

$$y_t = S_t \cdot T_t \cdot R_t. \tag{2.39}$$

Equation (2.39) is called the *multiplicative decomposition*.

An additive decomposition is more suitable if the variation of the seasonal term or the trend-cycle does not vary with the time series level. If one of them is proportional to the time series level, a multiplicative decomposition is more appropriate [14]. In financial time series, the multiplicative one is more often used.

The multiplicative decomposition can be cast into the additive one by performing the log transformation:

$$y_t = S_t \cdot T_t \cdot R_t \quad \Leftrightarrow \quad \log y_t = \log S_t + \log T_t + \log R_t.$$

2.3.2 Two Decomposition Methods of Time Series

There are many decomposition methods for time series. In this section, we will introduce the two most commonly used ones, which are also the most intuitive: the canonical decomposition and the STL decomposition.

Moving Average Smoothing

The Moving Average (MA) being discussed in this section is not the same as the Moving Average model, which refers to a moving average of error terms, as introduced in the previous section. The trend-cycle estimated by an s-order MA, often written as s-MA for simplicity, can be expressed as:

$$\hat{T}_t = \frac{1}{s} \sum_{j=-k}^{k} y_{t+j},$$

where $s = 2k + 1$. The trend-cycle estimation at time t is calculated by averaging values in a window around t of size s, and the window slides through all time steps. Thus, we also call it a *sliding window*. The s-MA reduces the randomness in the data and gives a smoother trend-cycle term. It is a powerful and frequently used tool in financial analysis. Fig. 2.7 gives an example, showing the daily close price of Apple Inc. in recent months and its MA-Smoothed lines.[6]

Centered Moving Average We can perform an MA to another MA to make an even-order MA symmetric. For example, a 2-MA can be applied to a 4-MA, i.e., 2×4-MA:

$$\begin{aligned}\hat{T}_t &= \frac{1}{2}\left(\frac{1}{4}(y_{t-2} + y_{t-1} + y_t + y_{t+1}) + \frac{1}{4}(y_{t-1} + y_t + y_{t+1} + y_{t+2})\right) \\ &= \frac{1}{8}y_{t-2} + \frac{1}{4}y_{t-1} + \frac{1}{4}y_t + \frac{1}{4}y_{t+1} + \frac{1}{8}y_{t+2}.\end{aligned}$$

It is called a *centered moving average of order 4*. It is pretty helpful in eliminating the impact of seasonality in the data, as the first and last terms which belong to the same

[6]Data retrieved from **Yahoo! Finance**: `https://finance.yahoo.com/quote/AAPL?p=AAPL`.

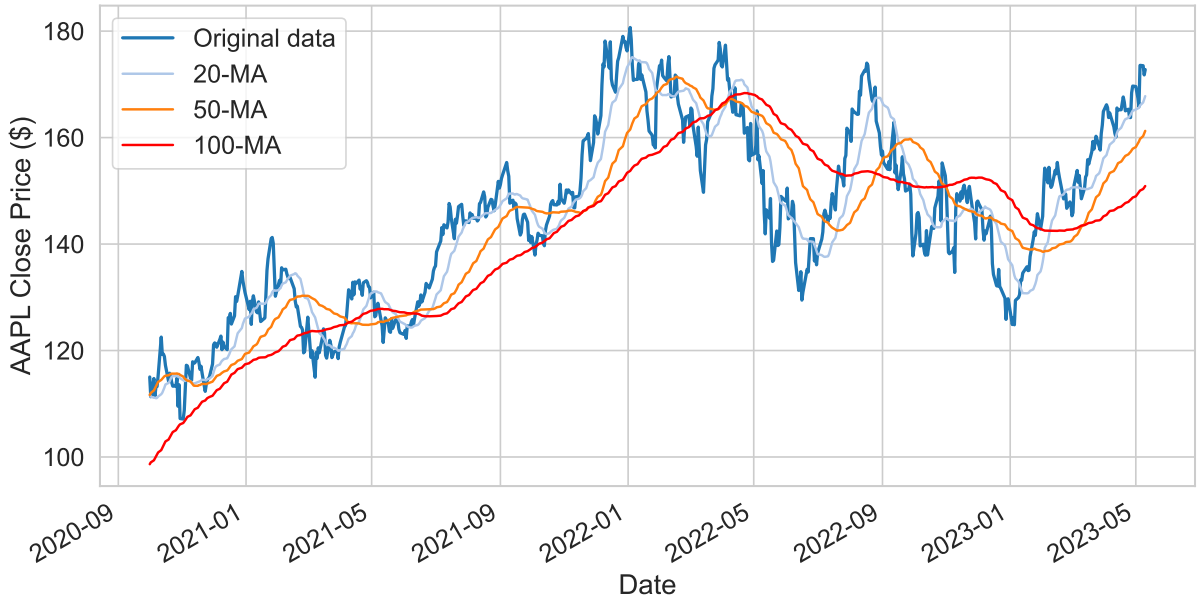


Figure 2.7: Apple Inc. (AAPL) daily stock price and its MA-Smoothed lines.

season are averaged. For example, for a 2×4-MA applied to the Coca-Cola quarterly earnings data mentioned in Sec. 2.2.1, the seasonal term can be removed entirely, as demonstrated in Fig. 2.8. Similarly, 2×12-MA and 7-MA can be exploited to remove the monthly and daily seasonalities respectively. In general, to extract the trend-cycle of a time series with a seasonal period of s, we apply a $2 \times s$-MA for an even s and an s-MA for odd ones.

To make a centered moving average symmetric, an even-order MA should be followed by an even-order MA and an odd-order MA should be followed by an odd-order MA [14].

Canonical Decomposition

Generally, a decomposition method will segregate a time series into trend-cycle, seasonal, and residual components. Thanks to the centered MA, we can easily extract the trend-cycle $\hat{T}_t$ from the time series. Here, we will describe the most used decomposition method, i.e., the classical or canonical decomposition of time series.

As described previously, depending on the data's nature, one may prefer additive or multiplicative decomposition. The canonical additive decomposition has the following procedures:

1. Extract the trend-cycle component $\hat{T}_t$ by the centered MA method described above, concerning different s.
2. Detrend the time series. $y_t - \hat{T}_t$.
3. Calculate the seasonal component $\hat{S}_t$. It has three steps:
    - Average the values in each seasonal position, e.g., for monthly data, averaging all the detrended January values to have the seasonal value of January.
    - Arrange the seasonal values for each seasonal period.

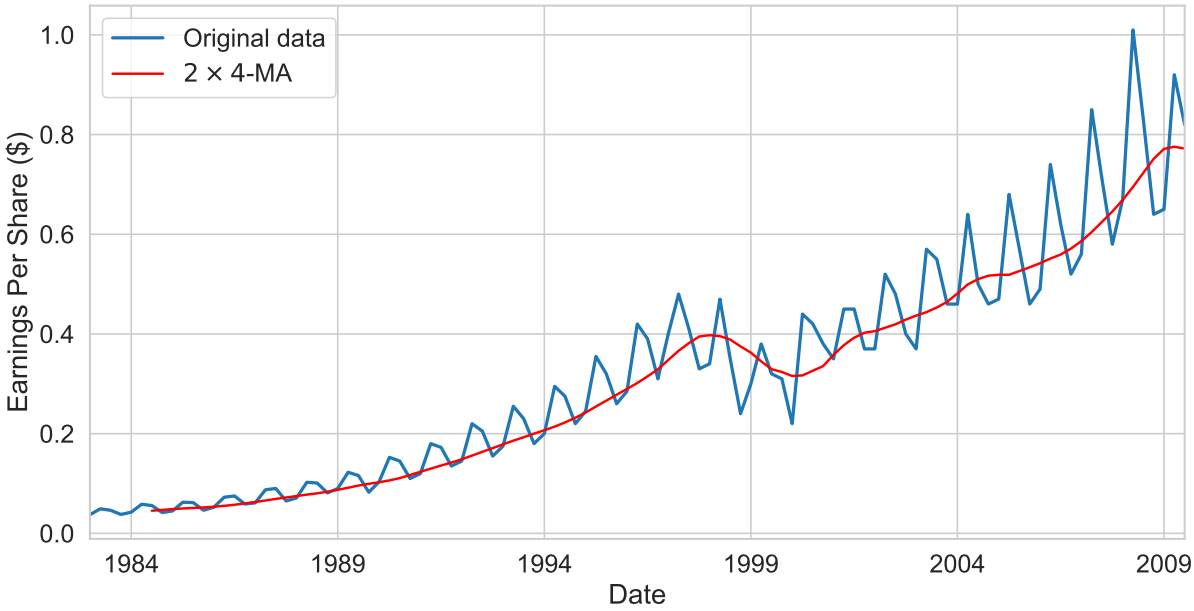


Figure 2.8: Coca-Cola quarterly EPS and its 2×4-MA line.

- Tile the ordered seasonal values several times for the final seasonal component.

4. Calculate the residuals by subtracting the trend-cycle and the seasonal components from the time series. $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$.

The multiplicative decomposition has similar procedures to the additive one, except the multiplicative one replaces the subtraction with a division for the detrending step and residual calculation. Fig. 2.9 demonstrates both the additive and the multiplicative decomposition for the Coca-Cola quarterly earnings.

Although the canonical time series decomposition method is still widely used, it has several following shortcomings:

- Canonical decomposition cannot generate the first and last s trend-cycle values due to the s-MA applied and also leaves NaN values for the residual component at the same positions.
- Canonical decomposition supposes the seasonal term in one period repeats every season, which is reasonable in many cases. However, seasonal terms may vary with time for long series, making this assumption irrational.
- Canonical decomposition is not robust to outliers.

In 1990, Cleveland *et al.* [25] proposed a decomposition method called *STL decomposition* that can address the aforementioned problems of the canonical decomposition method. STL stands for *Seasonal-Trend decomposition using LOESS*. To demonstrate how STL decomposition works, LOESS must be detailed.

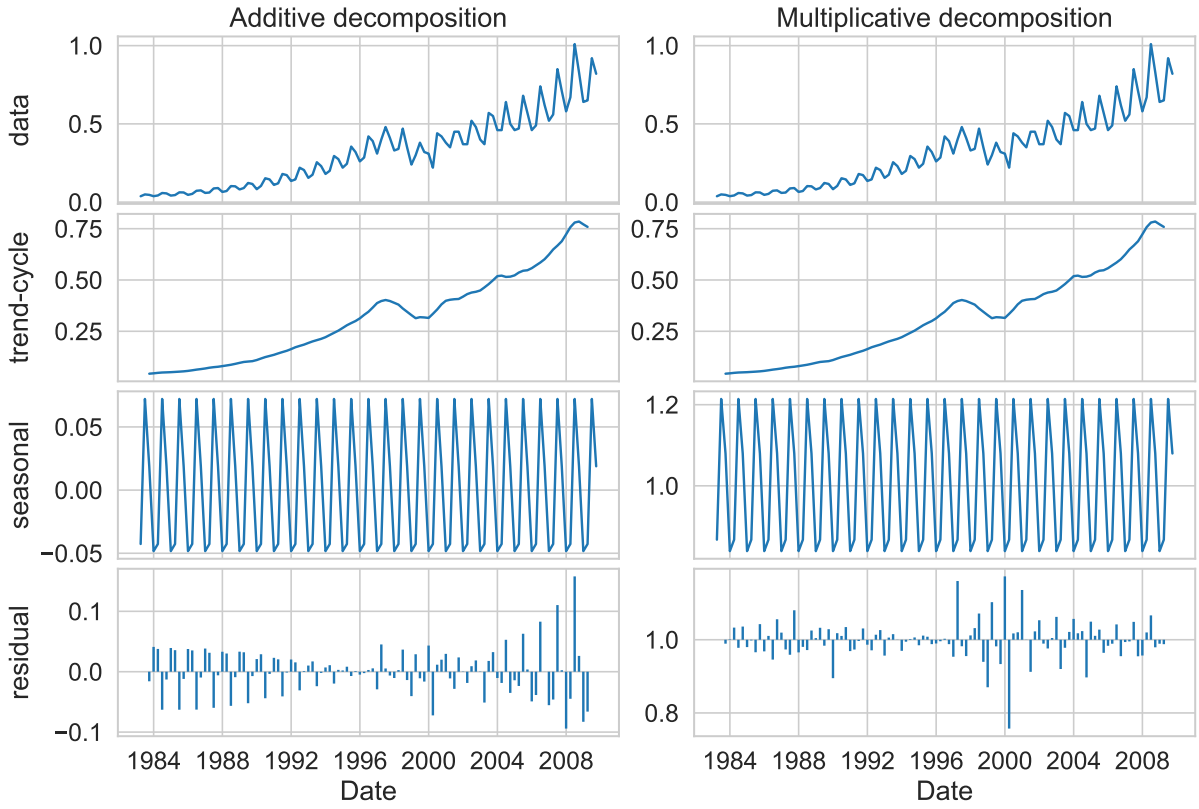


Figure 2.9: Additive and multiplicative decompositions of Coca-Cola quarterly earnings.

LOESS Regression

Generally, the relationship between two variables can be intricate, and it is thus insufficient to describe this relationship with straight lines or parametric curves. In this case, the nonparametric regression can be beneficial. The difference between parametric and nonparametric methods lies in whether we assume a certain kind of relationship for the data, e.g., a linear relationship can be modeled by a linear equation where we only need to calculate the model parameters. In contrast, an analysis based only on the data without assuming any model type is a nonparametric method. Without any limitations on the model type, the nonparametric methods usually fit a curve that better describes the relationship of the variables than the parametric ones.

LOESS stands for *LOcal regrESSion* or *LOcally Estimated Scatterplot Smoothing*. It is a nonparametric robust locally weighted regression method for smoothing a scatterplot, $(x_i, y_i), i = 1, ..., n$, in which the fitted value at x_k is the value of a polynomial fit to the data using weighted least squares, where the weight for (x_i, y_i) is large if x_i is close to x_k and small if it is not [26].

LOESS splits the data into several small subsections, performs weighted linear regressions on different subsections, and connects the center of these curves to form the complete regression curve. Specifically, LOESS is defined by the following sequence of operations:

1. For one data point, often called the *focal point*, select k nearest points around it to form a local window. Every focal point has a corresponding local window.

2. Calculate the weights of every point in the window through a weight function W.
3. Fit a weighted linear regression in the window. For n focal points, we have n weighted linear regressions.
4. Connect the center points of the n weighted regressions to form the final fitted curve.

Based on the above description, there are several tunable parameters for a LOESS regression:

- Local window length n.
- Weight function $W(x)$.
- Number of iterations i. It might be favorable to iterate the local regression several times.
- Regression interval δ. It might not be necessary to perform regression for every point. We can fit δ-interval regressions and perform interpolations for those non-fitted data points.

By default, the selection of local window length n is 0.49 times the length of data but can vary with different problems. In [26], the author discussed using d-degree polynomial regression for the local window and concluded that $d = 1$, i.e., the linear regression, strikes a good balance between the computational ease and the fitting goodness in most cases. Here, we will discuss the weight function and the regression iteration.

Weight Functions The weight function $W(x)$ is not fixed but should have the following properties [26]:

- $W(x) > 0, \forall |x| < 1$.
- $W(x) = 0, \forall |x| \geq 1$.
- $W(x) = W(-x)$.
- $W(x)$ is a nonincreasing function for $x \geq 0$.

Here the idea of choosing a weight function is that it is positive and symmetric, applies on $[-1, 1]$, and has greater values in the middle (around 0) and smaller values on two sides (-1 and 1) to distribute the weight accordingly. The author proposed to use two weight functions as follows:

$$\text{Bisquare function} \quad B(x) = \begin{cases} (1 - |x|^2)^2, & \text{for } |x| < 1, \\ 0, & \text{for } |x| \geq 1. \end{cases}$$

$$\text{Tricube function} \quad T(x) = \begin{cases} (1 - |x|^3)^3, & \text{for } |x| < 1, \\ 0, & \text{for } |x| \geq 1. \end{cases}$$

The difference between the bisquare and the tricube functions is that the tricube one decreases the weights faster for the neighbors, resulting in better smoothing but increasing the residual variance.

For the weighted linear regression, by default, LOESS calculates the neighborhood weights using the tricube weight function:

$$\text{Neighborhood weights} \quad \eta_i = T\left(\frac{|x_i - x|}{\lambda_q(x)}\right),$$

where x is the focal point's abscissa, and x_i is the i-th data's abscissa in the local window. For $q \leq n$, $\lambda_q(x)$ is the distance of the q-th farthest x_i from x in the window of length n. For $q > n$, $\lambda_q(x) = \lambda_n(x)\frac{n}{q}$.

A robust version of LOESS called *Robust LOESS* also exploits the bisquare function to calculate its robustness weights:

$$\text{Robustness weight} \quad \rho_i = B\left(\frac{|e_i|}{6h}\right),$$

where $e_i = y_i - \hat{y}_i$ is the residual of the current fitted value and $h = \text{median}(|e_i|)$ is the median of $|e_i|$. y_i and $\hat{y}_i$ are the i-th data's ordinate and its estimation. The final weights the robust LOESS use are the product of η_i and ρ_i.

Iteration Every iteration for the weighted regression in the local window in LOESS is conducted in the following sequences:

1. Calculate the neighborhood weight η_i with $T(x)$ for every i in the local window.
2. Perform the weighted regression with η_i and calculate $\hat{y}_i$.
3. Calculate the residual e_i.
4. Calculate the robustness weights ρ_i with $B(x)$.
5. Replace η_i with $\eta_i \times \rho_i$ and repeat steps 2, 3, and 4.

This procedure does not have a certain converge criterion, but practically it converges within two iterations.

Interpolations The computational time will grow drastically during the weighted regression if every point in the local window is calculated. In contrast, if we regress on only several data points and use interpolation in other places, the whole algorithm will compute faster without losing too much effectiveness. Linear, quadratic, and cubic interpolations can be used.

The distance δ within which to use interpolation instead of weighted regression is suggested to be $\delta = 0.01 \times T$ for $T > 5000$ where T is the length of the data. In practice, for each x_i, regressions are skipped for points closer than δ. The subsequent regression is fitted for the farthest point within δ of x_i, and all points in between are estimated by linearly interpolating between the two regression fits [15]. Fig. 2.10 gives a simple illustration of a LOESS regression.

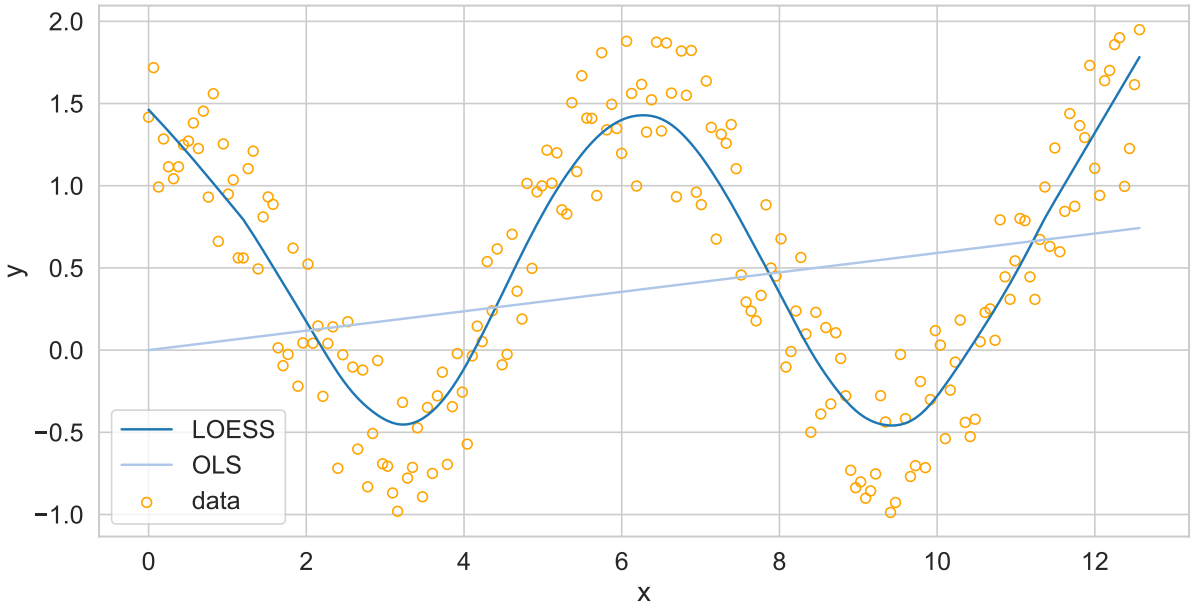


Figure 2.10: A LOESS example.

STL Decomposition

As revealed by its name, *A Seasonal-Trend decomposition procedure based on LOESS*, Cleveland *et al.* [25] leverage the LOESS regression for this decomposition algorithm. For simplicity, the LOESS used in STL decomposition is formalized as $\hat{y}_t \leftarrow \text{LOESS}(y_t, n)$, where n is the local window length.

STL decomposition segregates a time series into the same form as the additive canonical decomposition: $y_t = S_t + T_t + R_t$. It consists of an inner loop and an outer loop where the former fits the trend and calculates the seasonal component while the latter adds robustness to the algorithm.

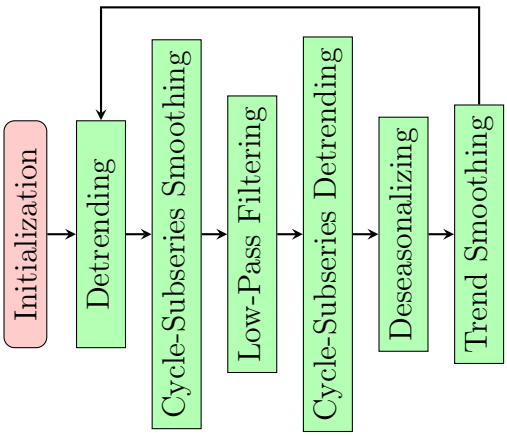


Figure 2.11: The inner loop of STL decomposition.

The Inner Loop Every inner loop performs a seasonal smoothing that updates the seasonal component $S_t^{(k)}$ in the current loop k and a trend smoothing that updates the

trend component $T_t^{(k)}$. The inner loop has six steps, as demonstrated in Fig. 2.11.

The Outer Loop The outer loop of STL decomposition is designed for robustness against anomalies. After the inner loop is executed once, the residual term $R_t = y_t - T_t - S_t$ is tested for outliers. In cases where outliers are detected in R_t, the LOESS regression used in *Detrending* and *Trend Smoothing* inside the inner loop will be replaced by the Robust LOESS, which uses a bisquare weight function and the loop repeats. In this case, the inner loop repeats only once.

Usually, for the inner loop, very few (often two) iterations suffice, and the outer loop iterates often 15 times to ensure a certain convergence.

Algorithm 1 gives a detailed illustration of the whole procedure of the STL decomposition. One can easily exploit STL decomposition by calling `STL()` provided by `statsmodels` [15] in Python or `stl()` in the R package `forecast` [17].

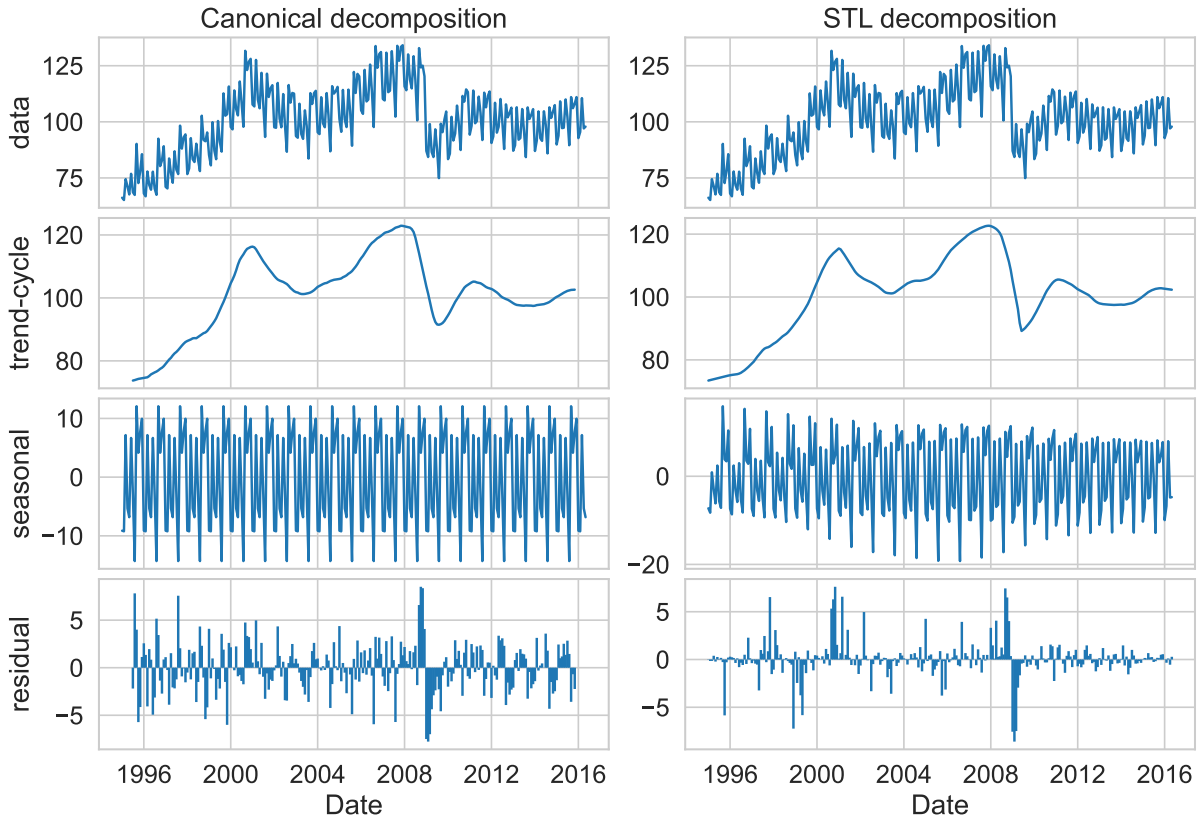


Figure 2.12: Comparison of canonical and STL decomposition of the production of electrical equipment in the EU.

Fig. 2.12 gives a comparison of canonical and STL decompositions of the production of electrical equipment in the EU.[7] We can observe the seasonal term of STL decomposition varies with time and the financial crisis in the year of 2008 is more clearly described by STL than by the canonical one.

[7]Data retrieved from **Eurostat**: `https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Archive:Manufacture_of_electrical_equipment_statistics_-_NACE_Rev._2`

Algorithm 1 STL Decomposition

Require: Time series y_t, frequency $n_{(p)}$, LOESS parameters $n_{(s)}, n_{(l)}, n_{(t)}$, #inner/outer loops $n_{(i)}, n_{(o)}$

function DETRENDING($y_t, T_t^{(k)}$)
  return $\tilde{y}_t^{(k)} = y_t - T_t^{(k)}$
end function

function CYCLE-SUBSERIESSMOOTHING($\tilde{y}_t^{(k)}, n_{(p)}, n_{(s)}$)
  $\tilde{C}_t^{(k)} \leftarrow$ `reshape`$(\tilde{y}_t^{(k)}, n_{(p)})$ ▷ Cycle-Subseries: Value of each position in the seasonal cycle of $\tilde{y}_t^{(k)}$
  if $k == 0$ **then** ▷ Smooth seasonal subseries
    $\hat{C}_t^{(k+1)} \leftarrow$ LOESS$(\tilde{C}_t^{(k)}, n_{(s)})$
  else
    $\hat{C}_t^{(k+1)} \leftarrow$ ROBUSTLOESS$(\tilde{C}_t^{(k)}, n_{(s)})$
  end if
  $\hat{C}_t^{(k+1)} \leftarrow$ `expand`$(\hat{C}_t^{(k+1)})$ ▷ Expand one point at each side
  $C_t^{(k+1)} \leftarrow$ `concatenate`$(\hat{C}_t^{(k+1)})$ ▷ Concatenat in time order
  return $C_t^{(k+1)}, t = -n_{(p)} + 1, -n_{(p)} + 2, ..., T + n_{(p)}$
end function

function LOW-PASSFILTERING($C_t^{(k+1)}, n_{(p)}, n_{(l)}$)
  $C_t^{(k+1)} \leftarrow 3 \times n_{(p)} \times n_{(p)}$-MA$(C_t^{(k+1)})$ ▷ Smooth $C_t^{(k+1)}$ with three low-pass filters
  $L_t^{(k+1)} \leftarrow$ LOESS$(C_t^{(k+1)}, n_{(l)})$ ▷ Smooth $C_t^{(k+1)}$ with LOESS filter
  return $L_t^{(k+1)}$, $t = 1, 2, ..., T$.
end function

function CYCLE-SUBSERIESDETRENDING($C_t^{(k+1)}, L_t^{(k+1)}$)
  return $S_t^{(k+1)} = C_t^{(k+1)} - L_t^{(k+1)}$ ▷ Detrend the cycle-subseries
end function

function DESEASONALIZING($y_t, S_t^{(k+1)}$)
  return $\tilde{R}_t = y_t - S_t^{(k+1)}$ ▷ Deseasonalize the time series with $S_t^{(k+1)}$
end function

function TRENDSMOOTHING($\tilde{R}_t, n_{(t)}$)
  if $k == 0$ **then** ▷ Smooth the deseasonalized time series with LOESS
    return $T_t^{(k+1)} \leftarrow$ LOESS$(\tilde{R}_t, n_{(t)})$
  else
    return $T_t^{(k+1)} \leftarrow$ ROBUSTLOESS$(\tilde{R}_t, n_{(t)})$
  end if
end function

procedure INNEROROUTERLOOP($y_t, n_{(i)}, n_{(o)}, n_{(p)}, n_{(s)}, n_{(l)}, n_{(t)}$)
  $k = 0, T_t^{(0)} \equiv 0$ ▷ Initialize the trend component at pass 0
  repeat
    $\tilde{y}_t^{(k)} \leftarrow$ DETRENDING$(y_t, T_t^{(k)})$
    $C_t^{(k+1)} \leftarrow$ CYCLE-SUBSERIESSMOOTHING$(\tilde{y}_t^{(k)}, n_{(p)}, n_{(s)})$
    $L_t^{(k+1)} \leftarrow$ LOW-PASSFILTERING$(C_t^{(k+1)}, n_{(p)}, n_{(l)})$
    $S_t^{(k+1)} \leftarrow$ CYCLE-SUBSERIESDETRENDING$(C_t^{(k+1)}, L_t^{(k+1)})$
    $\tilde{R}_t \leftarrow$ DESEASONALIZING$(y_t, S_t^{(k+1)})$
    $T_t^{(k+1)} \leftarrow$ TRENDSMOOTHING$(\tilde{R}_t, n_{(t)})$
    $k++$
  until $\frac{\max_t\left(|U_t^{(k)} - U_t^{(k+1)}|\right)}{\max_t\left(U_t^{(k)}\right) - \min_t\left(U_t^{(k)}\right)} < 0.01$ or $k == n_{(i)}$ or $n_{(o)}$ ▷ $U_t^{(k)}$ can be either $T_t^{(k)}$ or $S_t^{(k)}$
  return $S_t = S_t^{(k+1)}, T_t = T_t^{(k+1)}, R_t = y_t - S_t - T_t$
end procedure

Advantages and drawbacks of STL Compared with canonical decomposition, STL has several advantages:

- The seasonal component extracted by STL can vary over time.
- The smoothness of the trend-cycle component can be controlled by manipulating the LOESS regression.
- STL can provide a far more robust decomposition than the canonical one.

On the other hand, STL has some shortcomings:

- It provides only additional decomposition.
- It can only handle simple seasonality, i.e., it accepts only one frequency.
- It is robust but still less robust when facing abrupt changes in trend-cycle and residual, especially when used for anomaly detection.

The first problem can be easily resolved by employing log transformation or Box-Cox transformation described in [27]:

$$B(x,\lambda) = \begin{cases} \frac{x^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0, \\ \log(x), & \text{if } \lambda = 0. \end{cases}$$

The second disadvantage is addressed by the MSTL decomposition proposed by Bandara, Hyndman, and Bergmeir [28] where they use STL multiple times adapting to different seasonalities and thus construct a decomposition with multiple seasonal patterns:

$$y_t = \hat{S}_t^1 + \hat{S}_t^2 + \cdots + \hat{S}_t^n + \hat{T}_t + \hat{R}_t.$$

The third issue can be mitigated by a recent algorithm called *RobustSTL*, and its successor *Fast RobustSTL* both proposed by Wen *et al.* [29], [30] in 2019 and 2020. The former seeks to extract the trend component robustly by solving a regression problem using the LAD loss and bilateral filtering and then extracts the seasonal term with non-local seasonal filtering. The latter extends RobustSTL to handle multiple seasonality by extending the non-local seasonal filter in a weighted form and speeds up the decomposition with a special generalized ADMM algorithm.

Decomposition Features

Once a time series is decomposed, one can propose some natural questions such as *"How strong are the contributions of those components?"* or *"Among multiple different time series, which has the strongest/weakest trend or seasonality?"*

Wang, Smith, and Hyndman [31] proposed a measure of the trend and seasonality strengths. Consider a decomposition $y_t = T_t + S_t + R_t$, where T_t, S_t, and R_t are the trend-cycle, seasonal, and residual components.

For a time series that has a strong trend, the seasonally adjusted (deseasonalized) series $T_t + R_t$ should have a stronger variance than the residual itself, contributing a small

value of $\frac{\text{Var}(R_t)}{\text{Var}(T_t+R_t)}$. In contrast, for a weak-trended series, the variances of the two terms should be close. Therefore, the trend strength can be defined as:

$$F_T = \max\left(0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(T_t + R_t)}\right).$$

Similarly, we can define the seasonality strength:

$$F_S = \max\left(0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(S_t + R_t)}\right).$$

A small value of F_T or F_S represents a weak trend or seasonality and vice versa. This measure is handy when finding the time series with the weakest/strongest trend and seasonality features in multiple series.

2.3.3 Theta Method

Basic Theta Model

The Theta model is yet another econometric model initially proposed by Assimakopoulos and Nikolopoulos [32] in 2000. It constructs a decomposition of time series of several *Theta lines* through a coefficient θ by leveraging the second-order differences of the data.

A Theta line $z(\theta)$ should have the following representation:

$$z_t''(\theta) = \theta \cdot y_t'', \tag{2.40}$$

where $y_t'' = y_t - 2y_{t-1} + y_{t-2}$ is the second-order differencing of the series y and refers to the local curvatures of the time series.

For $t = 1, 2$, $z_t(\theta)$ can be obtained by minimizing $\sum_{t=1}^{T}[y_t - z_t(\theta)]^2$. Then by solving (2.40), the Theta model can be written as [33]:

$$\begin{aligned} z_t(\theta) &= \theta y_t + (1-\theta) z_t(0) \\ &= \theta y_t + (1-\theta)(\hat{a} + \hat{b}t), \quad \text{t=1,2,...} \end{aligned}$$

$\hat{a}, \hat{b}$ are the intercept and the slope, respectively, of the linear regression over $y_1, ..., y_T$ against $1, ..., T$, i.e., $z_t(0)$, given by:

$$\begin{aligned} \hat{b} &= \frac{6}{T^2 - 1}\left(\frac{2}{T}\sum_{t=1}^{T} t y_t - \frac{T+1}{T}\sum_{t=1}^{T} y_t\right), \\ \hat{a} &= \frac{1}{T}\sum_{t=1}^{T} y_t - \frac{T+1}{2}\hat{b}. \end{aligned}$$

Depending on θ's value, Theta lines can exhibit deflations or dilations of the original time series. A coefficient of $0 < \theta < 1$ can lead to a flatter Theta line closer to $z_t(0)$ and be utilized as a measure for the long-term pattern, e.g., trend. In contrast, a $\theta > 1$ will dilate the Theta line further from $z_t(0)$; thus, the coarser Theta line can capture the

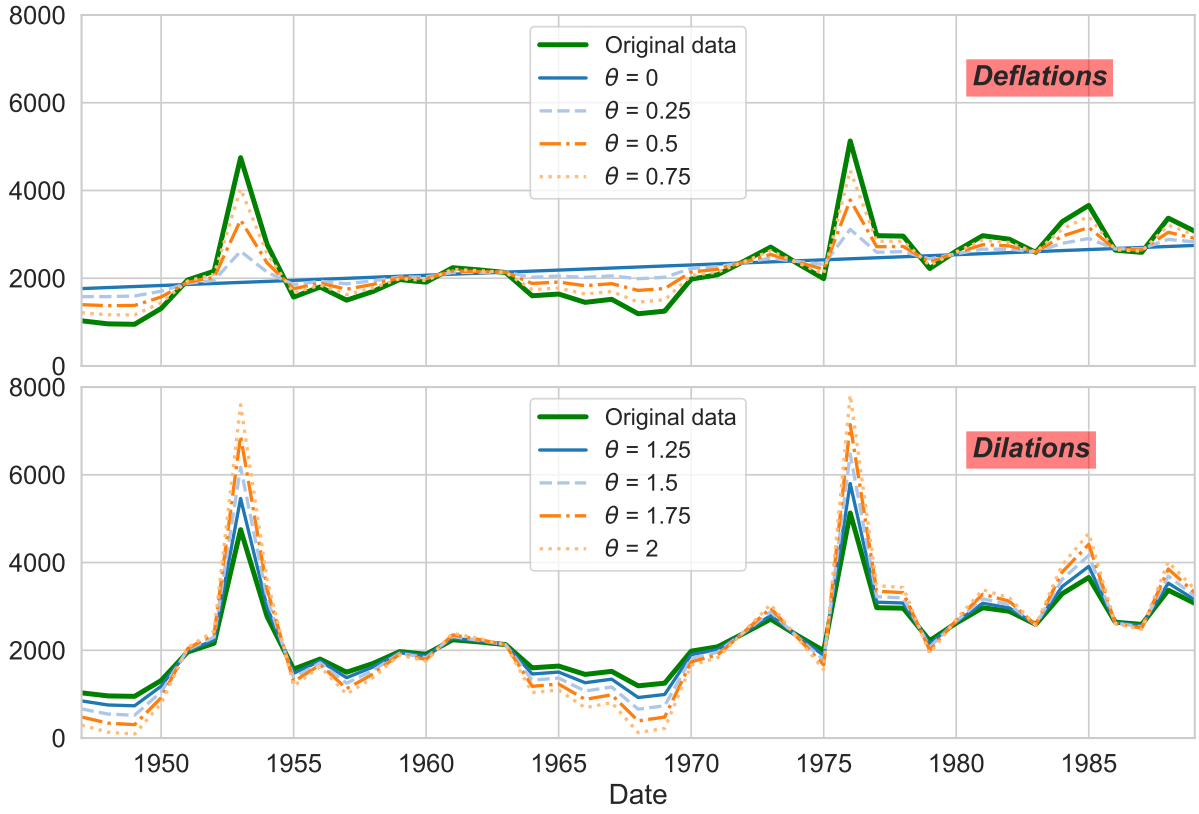


Figure 2.13: Theta lines deflations and dilations of the Series N200 in M3-Competition.

short-term fluctuations in the series. Fig. 2.13 gives two examples of the series N200 in the M3-Competition dataset [34], showing deflations and dilations with different θ values.[8]

The original Theta model consists of only two Theta lines, i.e., $z_t(0)$ and $z_t(2)$. Nonetheless, there should be no limitation on the number and type of Theta lines. Exploiting a double-lined Theta model with $\theta = 0$ and $\theta \neq 2$ can adjust the slope of the forecasts [35] while a triple-lined model can leverage more patterns of the series to benefit the forecasting [35]–[37].

However, combining its simplicity and efficiency, the double-lined Theta model is most often used [38]:

$$y_t = \omega z_t(\theta_1) + (1-\omega) z_t(\theta_2),$$

where ω and $(1-\omega)$ are the weights corresponding to the two Theta lines with $\omega = \frac{\theta_2 - 1}{\theta_2 - \theta_1}$, $\theta_1 < 1$, and $\theta_2 \geq 1$.

Typically, we use a more straightforward model with $\theta = 0$ for better maintenance of the long-term trend pattern:

$$\begin{aligned} y_t &= \frac{\theta - 1}{\theta} z_t(0) + \frac{1}{\theta} z_t(\theta) \\ &= \frac{\theta - 1}{\theta}(\hat{a} + \hat{b}t) + \frac{1}{\theta} z_t(\theta), \quad \theta \geq 1. \end{aligned} \tag{2.41}$$

[8]Data retrieved from **International Institute of Forecasters**: `https://forecasters.org/resources/time-series-data/m3-competition/`

Forecasting with Theta Models

Forecasting with the classic Theta model is carried out as follows [35]:

1. **Deseasonalization.** A seasonality significance test is performed on the original series. If it is seasonal, seasonal adjustment with multiplicative decomposition is conducted.

2. **Theta decomposition.** The seasonally adjusted series is decomposed into two Theta lines, i.e., $z_t(0)$ and $z_t(2)$.

3. **Extrapolation**. $z_t(0)$ is linearly extrapolated as it is a linear regression, while $z_t(2)$ is extended using SES in (2.24).

4. **Combination.** The predictions of $z_t(0)$ and $z_t(2)$ are averaged with equal weights.

5. **Reseasonalization.** The last period of the seasonal component is integrated into the averaged forecasted values in the previous step if the series is tested seasonal.

Hyndman and Billah claimed in their paper [39] that the classic Theta model with $\theta = 0$ and $\theta = 2$ is equivalent to an *SES with drift* since the level of the final prediction is a combination of SES-extrapolated $z_t(2)$ with half of the slope of $z_t(0)$. Nikolopoulos and Thomakos argued that although an SES with drift might resemble a Theta model, its smoothing parameter is identical to the one obtained in the Theta model only by chance. "In the case of the Theta model, the smoothing parameter a is calculated via a mean squared error (MSE) minimization procedure on a different time series: Theta line with $\theta = 2$; not on the original data." More discussions can be found in Nikolopoulos' book *Forecasting with the Theta Method: Theory and Applications* [37].

Generalized Theta Model

Despite its efficiency and simplicity, the classic Theta model has several obvious limitations:

- As the drift part of Theta is a linear regression, the model can perform poorly on the time series with a non-linear trend.

- Theta exploits the additional average of the trend and level components captured by $z_t(0)$ and $z_t(\theta)$, respectively. It leverages the multiplicative decomposition for the seasonal part that is later integrated multiplicatively. However, not all time series components are organized this way. In the ETS model, for example, different components can have their own possible values.

Some modifications are proposed to overcome these limitations in terms of the non-linear trend and multiplicative expression of the Theta model. We explore these modifications in what follows.

Trend Adjustment For non-linear trends, especially in exponential cases, substituting the linear trend component $z_t(0)$ with an exponential one can be beneficial [35]:

$$\begin{aligned} \text{Linear trend} \qquad & z_t(0) = \hat{a} + \hat{b}t, \\ \text{Exponential trend} \qquad & x_t(0) = \hat{p}e^{\hat{q}t}, \text{ or} \\ & \log(x_t(0)) = \log(\hat{p}) + \hat{q}t. \end{aligned} \tag{2.42}$$

$\hat{p}, \hat{q}$ are the intercept and slope of the similar linear regression over $\log y_1, ..., \log y_T$ against $1, ..., T$.

Additive and Multiplicative Expression In the aforementioned ETS models, components can be assembled in both additive and multiplicative ways, giving the model the most flexibility in dealing with different time series accordingly. But as Theta averages the trend and level for its forecast, assuming an additive expression of the connection between the trend and level components while always integrating a multiplicative seasonal component. This can potentially limit the Theta model's flexibility for various time series.

This problem can be addressed by introducing a different multiplicative expression of the Theta lines:

$$\begin{aligned} \text{Additive Theta line} \qquad & a_t(\theta) = \theta y_t + (1-\theta)u_t(0), \\ \text{Multiplicative Theta line} \qquad & m_t(\theta) = y_t^{\theta} \times u_t^{(1-\theta)}(0), \end{aligned}$$

where $u_t(0)$ denotes the linear trend or the exponential trend in (2.42). Then the Theta model in (2.41) can be represented in both additive and multiplicative expressions:

$$\begin{aligned} \text{Additive expression} \qquad & y_t = \frac{\theta - 1}{\theta} u_t(0) + \frac{1}{\theta} a_t(\theta), \\ \text{Multiplicative expression} \qquad & y_t = u_t^{\frac{\theta-1}{\theta}}(0) \times m_t^{\frac{1}{\theta}}(\theta). \end{aligned}$$

Another possible modification to the original Theta model is replacing the multiplicative decomposition with an additive one for seasonal time series.

Similar to the ETS model, we now have a taxonomy for Theta models: Theta$(\cdot, \cdot, \cdot)$. The first term represents the type of the Theta line (A, M), where "A" refers to an "**A**dditive" linear trend and "M" refers to a "**M**ultiplicative" exponential trend. The second term exhibits the type of the expression (A, M), where "A" refers to the "**A**dditive" expression and "M" means a "**M**ultiplicative" one. The final term is the type of seasonal decomposition, which can be "**N**one", "**A**dditive" and "**M**ultiplicative" [35].

Parameter Estimation and Model Selection

Theta model generates forecasts which are drifted according to coefficient $\hat{b}$. The drift also depends on the value of θ, being $\frac{\theta-1}{\theta}$ times that of $\hat{b}$ [35]. θ should be selected precisely to prevent a prediction too pessimistic or too optimistic than the reality.

A common way to estimate the θ value is to minimize the in-sample Mean Absolute Error (MAE) of the model:

$$\text{MAE} = \frac{1}{T} \sum_{t=1}^{T} |y_t - \hat{y}_t|,$$

where $\hat{y}_t$ is the one-step-ahead prediction given by the model. Spiliotis, Assimakopoulos, and Makridakis [35] detailed that the optimization implemented with the Brent method, which combines the golden-section search and the successive parabolic interpolation, ensures a rapid convergence into a reliable solution.

For the model selection, given different combinations of terms in Theta$(\cdot, \cdot, \cdot)$, their complexities in terms of parameters are the same. Thus the information criteria described in the previous section, e.g., AIC and BIC that penalize the model over its complexity, do not apply to the Theta models. A primitive comparison over the MAE of different models is thus performed, and the final model is selected accordingly.

2.4 Conclusions

This chapter comprehensively overviewed several state-of-the-art econometric models for TSF tasks. It began with an introduction to basic time series concepts like stationarity and ACF/PACF, followed by a detailed exploration of AR, MA, and ARMA models, including their formulations, properties, and model selection methods. It also covered non-stationary and seasonal ARIMA models, explaining their practical applications through real-world examples.

ETS models, from the basic SES to the more complex Holt-Winters' Trend Seasonal model, were discussed, including their corresponding state space models and equivalence to some ARIMA models. Furthermore, the chapter explored multivariate time series and VAR models, along with their parameter estimation and model selection methods.

The chapter also delved into two decomposition methods, i.e., canonical and STL decompositions, as well as a decomposition-based econometric model, the Theta method. These methods were explained using real-world examples, highlighting their procedures, advantages, and limitations. Detailed comparisons between different decomposition methods and solutions to specific problems were provided.

This chapter also underscored the practical performance of these traditional methods, particularly in linear time series, and their accessibility through various scientific libraries. These methods are lauded for their fast convergence, robust mathematical proofs, and human interpretability. This foundational knowledge sets the stage for the subsequent discussion on deep learning solutions for the TSF problem.

Chapter 3

Deep Learning for Time Series Forecasting

Contents

Deep Learning (DL) is a subfield of Machine Learning (ML) that aims to mimic the workings of the human brain in learning and understanding patterns in vast amounts of data. Although it is part of AI research, DL itself has many unique techniques and methods.

The name "Deep Learning" comes from the Artificial Neural Networks (ANNs) structure it uses. These networks can have many layers. Hence they are referred to as "deep" networks. Each layer processes part of the information from the input data and passes the processed information to the next layer. This layered processing allows DL models to recognize and understand very complex patterns.

One significant application of DL models is in image and speech recognition, where many of the state-of-the-art systems are based on DL models. For example, DL models are used for object recognition in autonomous vehicles and for speech recognition and understanding in voice assistants like Amazon's Alexa or Apple's Siri. Additionally, DL models are used in natural language processing, machine translation, bioinformatics, and many other fields [40].

The critical advantage of DL models is their ability to handle multiple inputs and outputs, making them well-suited for TSF tasks. They are particularly useful in scenarios with complex, nonlinear dependencies between the input and output variables or multivariate inputs and multi-step forecasting problems. DL architectures such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) offer

additional capabilities that further enhance their performance in TSF tasks. In recent years, Transformer-based models have also been proposed to address the limitations of traditional DL models. These models have achieved state-of-the-art results in various TSF tasks, demonstrating their effectiveness and potential.

This chapter presents an overview of the state-of-the-art DL-based models for TSF. In Sec. 3.1, we discuss the traditional DL models, i.e., models prior to Transformer models, for the TSF problem and their advantages and disadvantages. Sec. 3.2 introduces the state-of-the-art Transformer model and its variants dedicated to the TSF problem. We conclude this chapter in Sec. 3.3.

3.1 Traditional Deep Learning Models

This section presents the traditional DL models for the TSF problem, including the econometric and ML models' bottlenecks and the traditional DL models prior to Transformer (i.e., MLPs, CNNs, RNNs, Attention Mechanism, and hybrid models) and their advantages and disadvantages.

3.1.1 Econometric and ML Models' Bottlenecks

Forecasting time series data presents a unique set of challenges compared to more straightforward problems like classification or regression, as time series data incorporates the added complexity of order or temporal dependence between observations. Consequently, handling this data requires specialized techniques for fitting and evaluating models. However, the temporal structure of time series data can also be beneficial for modeling as it provides additional information, such as trends and seasonality, which can improve model accuracy. Traditionally, econometric methods like ARIMA have been widely applied to TSF tasks due to their effectiveness and well-established theoretical foundations. However, these classical methods have limitations, such as:

- **Data integrity.** Incomplete, missing, or corrupted data is usually unsuitable for analysis.
- **Assumption of linearity.** Econometric models may not be appropriate for modeling complex nonlinear relationships.
- **Fixed temporal dependence.** Econometric models require the relationship between observations at different times to be diagnosed and specified, i.e., the model must be retrained if the relationship changes.
- **Limited forecasting horizon.** Econometric models are limited to forecasting a fixed number of steps ahead.
- **Univariate modeling.** Most econometric models are limited to univariate modeling.

ML techniques can effectively handle complex TSF problems with multiple input variables, complex nonlinear relationships, and missing data. These techniques, however,

often necessitate the use of hand-engineered features that are typically prepared by domain experts or practitioners with a background in signal processing. These features are essential in representing and extracting the underlying patterns and relationships in the time series data. Hence, their careful selection and engineering are crucial for the model's performance.

However, the process of feature engineering is time-consuming and requires domain expertise. It is also prone to human error and bias. Furthermore, the features are often specific to the problem at hand and cannot be reused for other tasks. These limitations have motivated the development of DL models that can automatically learn the features from the data, thereby eliminating the need for manual feature engineering.

3.1.2 MLPs, CNNs, RNNs, Attention Mechanism, and Hybrid Models

Multiple Layer Perceptrons (MLPs)

MLPs are straightforward neural networks capable of approximating a mapping function from input to output variables. In the context of MLPs, as demonstrated by Fig. 3.1, each neuron in a layer is connected to all neurons in the previous layer. These connections are associated with weights learned during the training process.

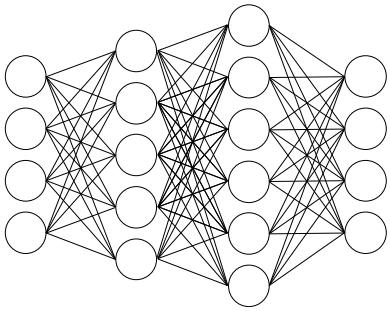


Figure 3.1: A four-layer MLP.

MLPs offer several key advantages that make them suitable for complex prediction tasks. MLPs excel in capturing nonlinear relationships between input and output variables, an essential capability for problems with intricate and nonlinear dependencies. Their resilience against noise in input data and ability to handle missing values also make them ideal for real-world applications where data may be incomplete or corrupted. Furthermore, MLPs can manage multiple input variables, making them apt for forecasting problems with various predictors. They also support an arbitrary number of output values, thereby facilitating multi-step forecasting.

Despite their strengths, MLPs face certain limitations when dealing with time series data, particularly those with a large number of input variables. MLPs are not structured to handle high-dimensional data nor to learn intricate nonlinear relationships between different variables. They require a fixed number of log input variables, akin to traditional econometric TSF methods. Also, like all neural networks, MLPs require careful hyperparameter tuning and are prone to overfitting if not appropriately trained. They cannot model temporal dependencies, a key characteristic of time series data, due to the absence of built-in mechanisms for handling sequential data and capturing temporal relationships.

Due to the limitations of MLP itself and the superiority of other DL models, MLPs are rarely used directly in TSF tasks. However, MLPs can be used as a baseline model for comparison with other DL models.

Convolutional Neural Networks (CNNs)

CNNs are a class of neural networks that use convolutional operations to extract features from the input data. The convolution operations involve passing a filter, also known as a kernel, over the input data, such as an image, to produce a transformed version called a feature map. This filter then detects specific features or patterns in the input, like edges, corners, or textures. The output feature map is a compressed representation of the original input, highlighting the detected features. Fig. 3.2 illustrates this process where the input image $\mathbf{I}$ is convolved with a kernel $\mathbf{K}$ to produce the output feature map $\mathbf{I} * \mathbf{K}$.[1]

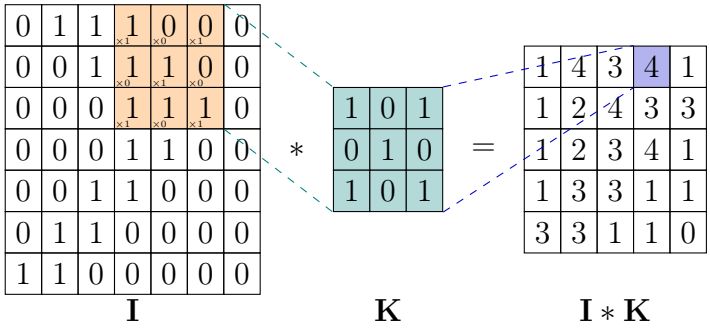


Figure 3.2: A 2D convolution operation.

CNNs were originally developed to handle grid-like data, such as images, with high efficiency. They can automatically and adaptively extract spatial hierarchies of features using convolution and pooling operations. CNNs have demonstrated remarkable performance on a variety of computer vision tasks, achieving state-of-the-art results in image classification and playing a crucial role in hybrid models for emerging problems such as object localization and image captioning. Fig. 3.3 shows the architecture of a typical CNN for classification tasks.

The distinguishing characteristic of CNNs is that they operate directly on raw data, such as raw pixel values, rather than relying on domain-specific or handcrafted features derived from the raw data. By doing so, the model can learn to extract features that are directly useful for the task automatically from the raw data, without the need for pre-processing or feature engineering, which can improve the model's performance and reduce the amount of human labor involved in the modeling process. This is known as representation learning, and CNNs can extract features in a way that is invariant to transformations or distortions in the data, which is ensured by three architectural ideas, i.e., local receptive fields, shared weights, and spatial or temporal subsampling.

The capability of CNNs to automatically learn and extract features from raw input data can also be leveraged for TSF tasks. In this context, a time series can be viewed as a one-dimensional image that can be processed by a CNN model, which can then distill the most informative elements from the sequence of observations. This approach has the

[1] TikZ figure obtained from `https://tikz.net/conv2d/`.

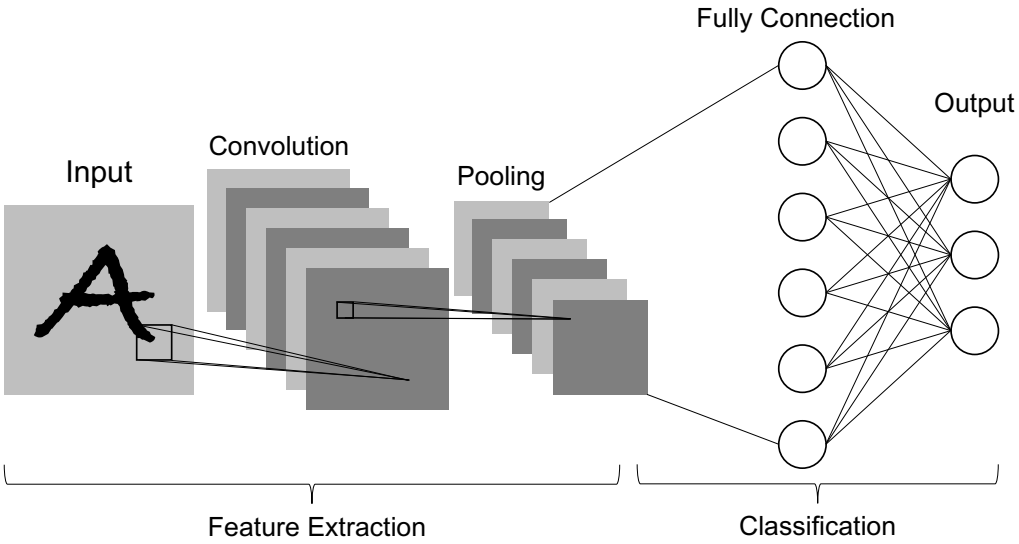


Figure 3.3: The architecture of a typical CNN.

potential to uncover complex patterns in time series data, particularly in cases where econometric methods struggle to capture nonlinear relationships and high-dimensional structures. The CNN architecture's ability to exploit the local structure in the data through the use of convolutional filters can help identify meaningful patterns that could be crucial for accurate forecasting.

Like MLP, CNNs offer several advantages for TSF, including handling multivariate input and output data and learning complex, nonlinear relationships, but do not necessitate direct input from lag observations. They can perform automatic feature extraction, providing a more efficient way of learning representations for TSF tasks. This capability of CNNs has shown to be effective in various time series classification tasks [41]–[44].

Recurrent Neural Networks (RNNs)

RNNs are neural networks designed to recognize patterns in data sequences, such as text, genomes, and time series data. Unlike MLPs or CNNs, RNNs retain a form of memory as they process inputs sequentially, allowing past information to influence the current output, making them particularly well-suited for tasks where ordinal information is important. This is achieved by introducing a feedback loop that allows information to persist, shown in Fig. 3.4. This feedback loop enables RNNs to capture temporal dependencies in the data, making them suitable for TSF tasks.

RNNs such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks are particularly suitable for TSF as they can explicitly handle the temporal order of observations when learning a mapping function for the inputs over time to outputs. This feature is not offered by other neural network architectures such as MLPs or CNNs. They have proven highly effective in complex natural language processing tasks, such as neural machine translation, where the model must learn the intricate interrelationships between words within and across languages. This powerful capability can also be applied to TSF. RNNs like LSTMs can automatically learn the temporal dependencies from the data, making them suitable for problems with nonlinear relationships and complex temporal

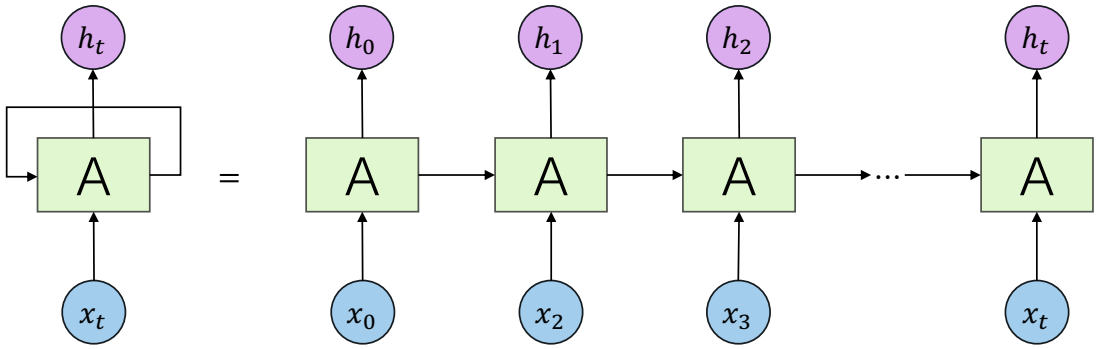


Figure 3.4: The fold and unfold architecture of a typical RNN.

dynamics.

Specifically, an RNN model is trained by being presented with one observation at a time. It learns to determine which previous observations are relevant for the forecasting task. As such, the model learns the mapping from inputs to outputs and which contextual information from the input sequence is important for making accurate predictions. The model can dynamically adjust this context as it processes each new observation in the sequence.

Besides, LSTMs and GRUs can solve the problem of gradient vanishing and exploding, which is a common problem in RNNs. This is achieved by introducing a mechanism called *gates* that control the flow of information through the network. The gates are designed to learn when information should be passed on and when it should be forgotten. This allows the model to learn long-term dependencies in the data, which is crucial for accurate forecasting. In addition, LSTMs and GRUs can be stacked to form a DL architecture, which can further improve the model's performance.

LSTMs and GRUs have been widely exploited either as the base architecture or as an effective module in time series problems from different domains as reported in various works of literature in recent years [2], [45]–[52]

Attention Mechanism

At the International Conference on Learning Representations (ICLR) in 2015, Bahdanau, Cho, and Bengio [53] creatively leveraged the joint learning to align and translate and thus proposed a new architecture called *Attention Mechanism* for neural translation tasks and pushed the machine translation community a huge step forward. Fig. 3.5 demonstrates this attention mechanism. It computes a weighted sum of input features based on their relevance to the current context, effectively allowing the model to "pay attention" to important parts and ignore less relevant ones.

The attention mechanism can also be a powerful tool used in DL to enhance the performance of TSF models. It aims to provide context to the model by enabling it to focus on relevant parts of the input sequence when making predictions. The technical details of attention mechanism implementation can vary depending on the application and network architecture. However, the core idea is to use learned attention weights to dynamically adjust the importance of different input features for a given prediction. This can allow for more effective complex time series modeling with nonlinear relationships and dependencies.

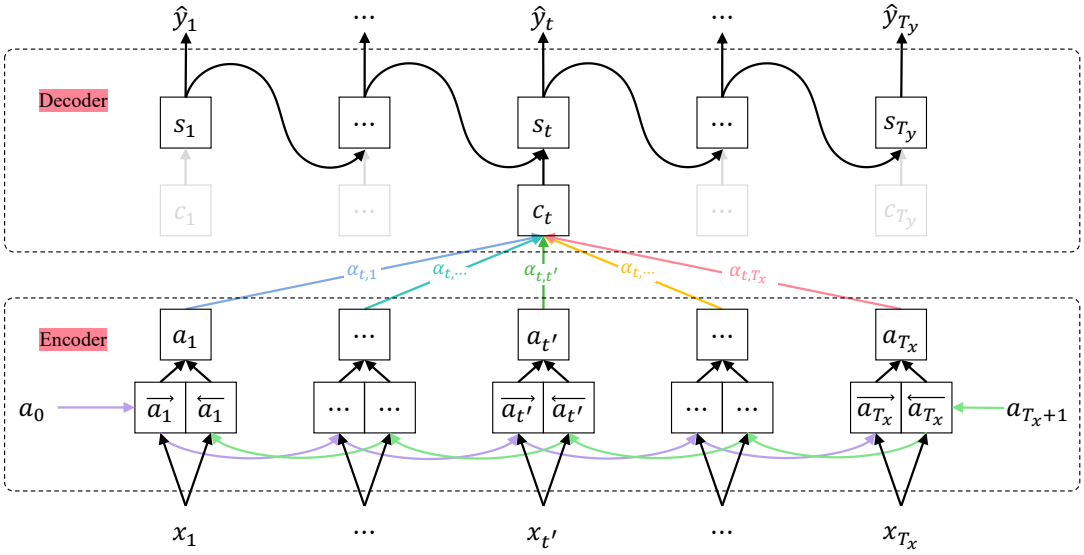


Figure 3.5: The architecture of the attention network.

The attention mechanism significantly enhances the performance of forecasting models, primarily through improved accuracy, interpretability, flexibility, and an ability to handle missing values. It enables models to concentrate on the most pertinent parts of the input sequence, thereby increasing precision—especially for long time series, where essential features can be identified for accurate predictions. This mechanism also offers better interpretability by allowing us to visualize the relevance of different input components via attention weights, offering insights into the model's decision-making. Furthermore, attention provides flexibility as it can be incorporated into various deep learning architectures like CNNs and RNNs and can be applied to both univariate and multivariate time series data. Lastly, it can handle missing data by assigning lower attention weights, ensuring precise predictions despite data gaps.

Despite its advantages, the attention mechanism still presents several challenges, such as computational cost, risk of overfitting, complex implementation, and interpretation difficulties. Calculating attention weights for each input element in long time series can be computationally demanding, increasing training time and memory requirements. Additionally, incorrect implementation can lead to overfitting, where the model becomes too specialized on specific parts of the input sequence, negatively impacting its predictive accuracy on new data. The complexity of implementing the attention mechanism also requires extra coding and modifications to the model architecture, necessitating more expertise. Lastly, despite enhancing model interpretability, the attention mechanism can sometimes make it difficult to discern the relevance of certain features, especially those assigned with low attention weights.

In conclusion, the attention mechanism is a powerful tool and has been proven to significantly improve the performance of TSF models [54]–[56].

Hybrid Models

In recent years, hybrid models of econometric and DL methods have been proposed to improve the performance of TSF models. These hybrid models combine different techniques,

such as exponential smoothing, ML, and state-space models, to produce accurate predictions. As hybrid models rarely have the same architecture, we summarize several articles that propose different hybrid methods and discuss their advantages and disadvantages.

- The Deep Factors for Forecasting method proposed by Wang *et al.* [57] is a hybrid model that combines the benefits of classical time series models and deep neural networks to produce probabilistic forecasts for large collections of similar and/or dependent time series. The method uses a global-local framework that systematically combines deep neural networks and probabilistic models while also developing an efficient and scalable inference algorithm for non-Gaussian likelihoods that generally applies to any normally distributed probabilistic models. This allows for accurate forecasting while handling uncertainty through a local classical model.

- De O. Santos Júnior, De Oliveira, and De Mattos Neto [58] proposed a hybrid system that combines the strengths of ARIMA and ANNs to improve forecasting accuracy. The authors use an intelligent model that combines linear and nonlinear techniques based on their respective forecasts. This hybrid system aims to find the most suitable combination function for describing the relationship between the forecasts of linear and nonlinear models. The authors present experimental results for well-known time series datasets, demonstrating the effectiveness of their proposed approach.

- Smyl [59] proposed ES-RNN, the winning solution of the M4 forecasting competition hosted by the International Institute of Forecasters in 2018. The innovation of this method lies in its use of a dynamic computational graph neural network system, which allows for the integration of exponential smoothing and LSTM networks into a common framework. The method also employs a hierarchical structure that combines a global part learned across many time series with a time series-specific part and ensembles at multiple levels. ES-RNN has shown promising results in competitions and real-world applications.

- Dudek, Pełka, and Smyl [60] proposed a hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting: This hybrid DL model combines exponential smoothing, LSTM, and ensembling for midterm load forecasting. The model dynamically extracts the main components of each individual time series and learns their representation. The multilayer LSTM is equipped with dilated recurrent skip connections and a spatial shortcut path from lower layers to better capture long-term seasonal relationships.

- One common assumption when training neural networks on time series data is that the errors at different time steps are uncorrelated. However, due to the temporal nature of the data, errors are often autocorrelated, which can lead to inaccurate maximum likelihood estimation. Sun, Lang, and Boning [61] proposed jointly learning the autocorrelation coefficient with the model parameters to adjust for autocorrelated errors. By doing so, it aims to improve the accuracy of these models and provide a more robust approach to modeling time series data with neural networks.

- The External Memory Augmented State Space Model (EMSSM) was invented by Sun *et al.* [62]. It is a novel approach for TSF that addresses the limitations of

conventional State Space Models (SSMs). This method provides multi-step ahead probabilistic forecasts with non-Markovian state transitions without accumulating prediction errors like autoregressive models. The external memory system efficiently utilizes weighted particles sampled by Variational Sequential Monte Carlo (VSMC) and can handle long-term nonlinear temporal dependencies. The dynamic function used in the EMSSM also contributes to lower Kullback–Leibler divergence than other SSMs.

Hybrid methods for TSF offer improved accuracy and the ability to handle complex patterns and dependencies. However, they may also be more complex and computationally intensive than single methods and require strong structural assumptions.

Summary

To summarize, traditional DL models for TSF tasks are usually based on RNNs and CNNs. At the same time, Attention Mechanism is a powerful tool proven to significantly improve the performance of TSF models. Hybrid models of econometric and DL methods have also been proposed to improve the performance of TSF models. These hybrid models combine different techniques, such as exponential smoothing, ML, and state-space models, to produce accurate predictions. Tab. 3.1 summarizes the advantages and disadvantages of the above methods.

3.2 Transformers for Time Series Forecasting

This section presents the Transformer model and its variants dedicated to the TSF problem.

3.2.1 Transformer Basic

Since its inception in 2017, Transformer models have gained increasing popularity and have been successfully applied in various fields, including machine translation, computer vision, and text generation [63]–[66].

The Transformer was first proposed by Vaswani *et al.* [63]. It is based solely on the attention mechanism. Transformer is a sequence-to-sequence model that uses attention pooling to compute the representation of the input sequence. As a generalization of the encoder-decoder architecture, it comprises an encoder and a decoder. The whole architecture of the Transformer is shown in Fig. 3.6.

Transformer's encoder and decoder are both composed of multiple layers of self-attention and feed-forward neural networks. The encoder takes an input sequence and maps it to a sequence of hidden representations, while the decoder takes the encoder output and generates an output sequence. The self-attention mechanism is used to learn the dependencies between the elements of the input and output sequences.

In traditional RNNs, the network is unrolled across time, and the state of the network at each time step is fed into the next time step. This approach makes it difficult to parallelize the training process and capture long-range dependencies between sequence elements. The self-attention mechanism solves this problem by allowing the model to attend to different input sequence elements with different weights.

Table 3.1: Advantages and Disadvantages of DL Methods for TSF.

Model	**Advantages**	**Disadvantages**
MLP	• Multivariate data support • Nonlinear relationships support • Fast training	• Careful feature engineering required • Limited capability to handle temporal dependencies • Very high-dimensional data unsupported
CNN	• Multivariate data support • Nonlinear relationships support • Automatic feature extraction	• Limited ability in long-term forecasting tasks • Careful tuning of hyperparameters required • Translation invariant
RNN	• Native sequential and time-dependent data support • Long-term dependencies support • LSTMs/GRUs can partially help the vanishing gradient problem	• Vanishing gradient problem for long series • Sensitive to the initial values of the hidden states • Slow training speed
Attention	• Improved accuracy • Better interpretability • Flexible combination with CNNs and RNNs • Robust to missing values	• Computationally expensive • Suffers from overfitting • Complex implementation • Interpretation challenges
Hybrid	• Combine multiple models to improve forecasting accuracy • Support both linear and nonlinear patterns • Balance between accuracy and interpretability	• Complex optimization and parameter tuning required • Computationally expensive • Difficult to interpret the contribution of each component

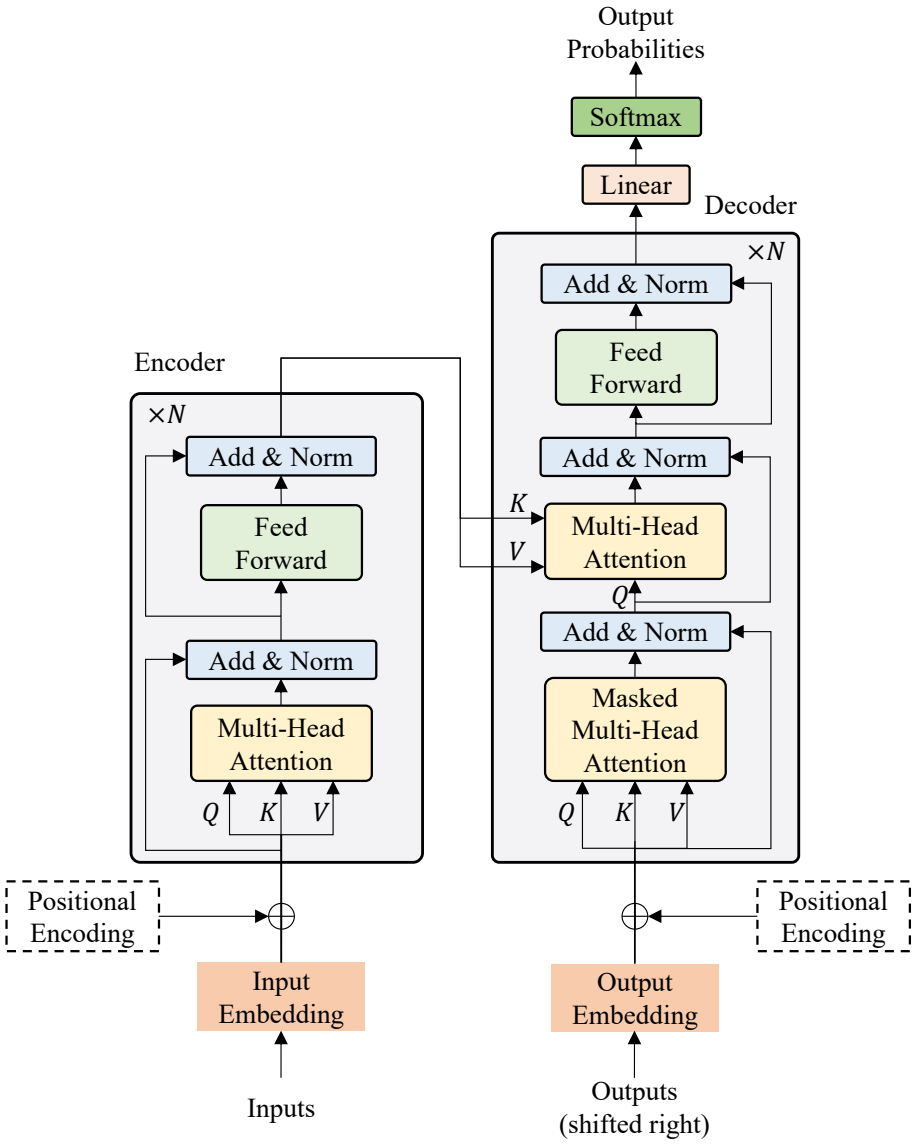
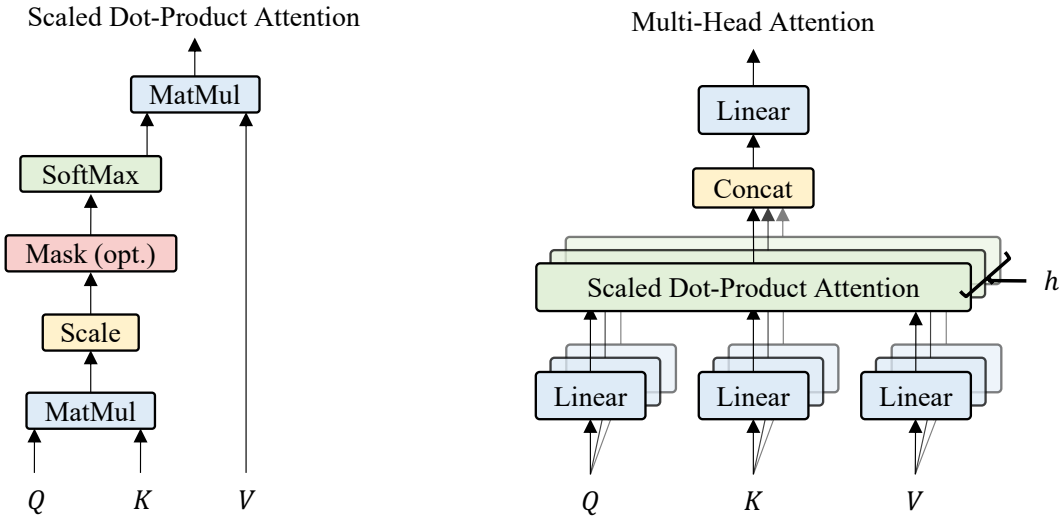


Figure 3.6: The architecture of Transformer.

The self-attention mechanism is based on attending to different parts of the input sequence. Each element in the input sequence is associated with a `query`, a `key`, and a `value` vector. The `query` vector is used to score the relevance of each `key` vector with respect to the `query`, and the `values` are weighted by the scores and summed up to form the output. This process is repeated for every element in the input sequence, resulting in a weighted sum of all the `values`, which captures the relevant information for the `query`. This procedure is illustrated in Fig. 3.7a.

Multi-head attention is another key element for Transformer. It is a variant of the self-attention mechanism that allows the model to attend to multiple parts of the input sequence simultaneously. In multi-head attention, the `query`, `key`, and `value` vectors are projected into multiple subspaces, and the self-attention mechanism is applied independently in each subspace. The results are then concatenated and linearly transformed, resulting in the final output, as shown in Fig. 3.7b. This allows the model to capture

different aspects of the input sequence and attend to multiple parts simultaneously.



(a) Self-Attention. (b) Multi-head attention.

Figure 3.7: Attention mechanism in Transformer.

Another critical component of the Transformer model is positional encoding. Unlike recurrent neural networks, the Transformer does not explicitly encode the position of each element in the sequence. Instead, positional encoding is added to the input embeddings to provide the model with positional information. In the primitive version of Transformer, the positional encoding is based on a set of sinusoidal functions of different frequencies and is added to the input embeddings before they are passed through the self-attention and feed-forward layers.

To conclude, Transformer leverages the attention mechanism solely, replacing the recurrent layers in the encoder-decoder architecture with multi-head attention. It is a milestone in the field of sequence modeling and has been successfully applied in various fields. In the later subsection, we will introduce Transformer models for TSF.

3.2.2 A gentle survey of Transformers for TSF tasks

The modeling ability of Transformers makes them naturally suitable for time series, which are also a type of sequence data structure. However, time series have many characteristics that are different from traditional sequence data. For example, time series often exhibit autocorrelation or periodicity, and TSF tasks frequently involve predicting sequences with very long periods. These characteristics present new challenges for the application of Transformers in TSF and have led to the development of a new generation of Transformers specifically designed for time series tasks. In this subsection, we will introduce the application of Transformers in TSF, with a focus on recent representative works up to 2023.

Transformer-based Model

At the International Conference on Machine Learning (ICML) in 2020, Wu *et al.* [67] proposed to apply a self-attention mechanism for the TSF problem. This is likely **the first formal publication to use Transformer structures in TSF**. Its overall structure directly adopts the encoder-decoder framework of Transformer, without much special upgrading, and applies it to a specific scenario. However, this opens the door to the application of Transformers in TSF.

Temporal Fusion Transformers

In 2020, Lim *et al.* [68]proposed a novel Transformer-based model for TSF called Temporal Fusion Transformers (TFT). This work uses **a combination of LSTM and Transformer** for TSF. The time series data is first input into the LSTM, which can extract contextual information similar to CNN and serve as a Position Encoding, leveraging the LSTM's sequential modeling ability to replace the Position Embedding trained with the Transformer model. For the input features, TFT also provides a detailed design, where a feature selection module (an attention mechanism) is used to compute the importance of each feature for the current time step. This importance is also used for visualizing and analyzing the TSF results to see the importance of each feature at each time step for the prediction.

Log-Sparse Transformer

Li *et al.* [69] published the Log-Sparse Transformer (LogTrans) at the Conference on Neural Information Processing Systems (NeurIPS) 2020. This work is based on the Transformer architecture and uses **a sparse attention mechanism to reduce the computational complexity** of the model. The attention operation of the Transformer is point-to-point, but contextual information is also important for time series data. Sometimes, the time series values of the two points are the same, but they are not useful as a reference because there is a large difference in the shape of their surrounding sequence. On the contrary, when the overall shape is very similar, even if the values of the two regions are very different, the two regions have a greater reference value and should have increased attention weights. This example illustrates that in time series problems, we cannot rely solely on the attention mechanism between two points, as in NLP, but should also consider the surrounding context information. In their work, the authors use a combination of convolution and Transformer. The time series data is first input into a one-dimensional convolutional layer, which extracts the context information of each time step. Then, the multi-head attention mechanism learns the relationship between time steps. This approach allows attention to consider the value of each point and the surrounding context information, which can establish connections between regions with similar shapes.

Adversarial Sparse Transformer

In 2020, Wu *et al.* [70] introduced the Adversarial Sparse Transformer (AST) by **applying the concept of Generative Adversarial Networks (GANs) to the Sparse Transformer architecture**. While many point prediction models are limited in capturing the randomness of time series data, AST is designed to address this issue by improving

the model's ability to represent time series and predict multiple future steps at a higher confidence level. This is achieved through adversarial training and an encoder-decoder structure, which allows the model to better capture the sparsity of dependencies between time series steps. Using a discriminator further enhances the sequence-level prediction performance of the model.

State Space Decomposition Neural Network

In 2021, Lin, Koprinska, and Rana [71] introduced the State Space Decomposition Neural Network (SSDNet), which **combines the Transformer with state space models (SSMs) to achieve a balance between model performance and interpretability**. SSDNet utilizes the Transformer architecture to learn temporal patterns and directly estimate the parameters of an SSM. This allows the model to identify which parts of the history are most important for prediction by utilizing a fixed-form SSM to provide trend and seasonal components and an attention mechanism of the Transformer. As a result, SSDNet can effectively capture the spatiotemporal characteristics of time series data while providing transparent explanations of the model's behavior, which is a promising approach for TSF.

Informer

Informer is a transformer-based approach for long sequence TSF, optimized for efficiency, proposed by Zhou *et al.* [72]. The paper was awarded the best paper at the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI) in 2021. Transformer's time complexity increases exponentially with the length of the input sequence, which is problematic for long sequences. To address this, Informer introduces **ProbSparse Attention**, which reduces computation by forming sparse attention only between key and critical query pairs. By calculating the Kullback-Leibler divergence between the distribution of attention scores and a uniform distribution, Informer determines the important queries, ignoring the unimportant ones, which significantly increases computation efficiency and reduces the time complexity to $\mathcal{O}(N \log N)$. In addition, Informer introduces **self-attention distilling** by adding a convolution layer between every two attention blocks to reduce sequence length and training cost. Informer leverages a generative inference on both the training and inferencing phase thanks to its **generative style decoder**. It can thus predict multiple time steps at once, alleviating the issue of error accumulation compared to traditional autoregressive methods.

Autoformer

Autoformer is another transformer model specifically designed for TSF introduced by Wu *et al.* [73]. Rather than using the point-to-point correlation as the attention information, Autoformer introduces **a novel self-attention mechanism that incorporates the autocorrelation information** of the time series. It discovers the period-based dependencies by calculating the series autocorrelation and aggregates similar sub-series by time delay aggregation and thus can replace the self-attention seamlessly. Specifically, the AutoCorrelation Mechanism calculates the period-based dependencies by calculating the series autocorrelation and leverages FFT and Wiener-Khinchin theorem for fast

$\mathcal{O}(N \log N)$ computation. Then it selects the top k time series with the highest autocorrelation and aggregates them by time delay aggregation. The aggregated time series is then fed into the self-attention mechanism. By integrating the autocorrelation information into the attention mechanism, the model is better equipped to capture the temporal dependencies in the data. Autoformer also covers a Deep Decomposition Architecture, which involves decomposing the input time series into trend and seasonal components. This decomposition allows the model to separately capture the patterns and dependencies present in each component, leading to a better understanding of the underlying time series structure, thus solving the problem of intricate temporal pattern discovery.

Pyraformer

Liu *et al.* [74] proposed Pyraformer, a tree-structured Transformer for long-term TSF in 2022. **Pyraformer can effectively describe both short and long temporal dependencies with low time and space complexity.** It uses a Coarser Scale Construction Module (CSCM) to construct a multi-resolution C-ary tree at a coarser scale. A pyramid attention module is designed to propagate messages across and within scales. As the sequence length N increases, it can achieve theoretical $\mathcal{O}(N)$ complexity and a maximum signal traversal path length of $\mathcal{O}(1)$. Pyraformer has practical applications in decision-making and risk management based on accurate predictions of future events using time series data.

Frequency Enhanced Decomposed Transformer

Zhou *et al.* [75] proposed FEDformer, a frequency-enhanced decomposed Transformer for TSF in 2021. When using the regular Transformer for TSF, there is often a large gap between the predicted and true data distribution, as Transformer predicts each time point independently using self-attention, which may ignore the overall properties of the time series. FEDformer addresses this issue by exploiting **Fourier transform**. The Fourier transform module in FEDformer transforms the input time series to the frequency domain and replaces the `query`, `key`, and `value` with the Fourier-transformed frequency domain information to perform the attention operation. FEDformer also achieves a **linear time and space cost through low-rank approximation** by selecting a subset of Fourier components, which are small enough to avoid overfitting and big enough to preserve most of the history information. FEDformer also composes a decomposition architecture to capture the trend and seasonality of the time series as per Autoformer but with a more complex moving average approach.

Non-stationary Transformer

The non-stationarity of time series is a common and challenging problem in the real world. The main approach to addressing this issue is to perform some normalization techniques, e.g., min-max normalization and standard normalization, to make the sample statistics consistent across different time steps. However, this solution can harm Transformer models. Although the normalized sequence statistics are consistent, the process also causes the loss of specific information in the data, resulting in a convergence of attention matrices

for different sequences, which is called over-stationarization. Liu *et al.* [76] proposed Non-stationary Transformers at NeurIPS 2022 to address this issue. The proposed framework involves two interdependent modules: **Series Stationarization and De-stationary Attention**. Series Stationarization adopts a simple normalization strategy to unify the key statistics of each series without extra parameters, while De-stationary Attention approximates the attention of unstationarized data and compensates for the inherent non-stationarity of raw series. The proposed Non-stationary Transformers framework increases the predictability while re-incorporating non-stationary information of raw series.

PatchTST

Nie *et al.* [77] presented a new TS forecasting and representation learning method based on Transformer called PatchTST, **transforming time series data into a patch form similar to the Vision Transformer** proposed by Dosovitskiy *et al.* [66], achieving remarkable results. The core idea of the proposed PatchTST is twofold. Firstly, TSF models usually input each time step into the model, which is inefficient when dealing with long historical sequences and contains limited information per time step. PatchTST adopts the **Patching** approach inspired by Vision Transformer and models time series in patch form. The patching design retains local semantic information in the embedding, reduces computation and memory usage of the attention maps quadratically given the same look-back window, and allows the model to attend longer history. Secondly, **Channel-independence** is utilized, where each variable in the MTS is mapped to a separate embedding instead of combining multiple variables into one embedding. This design allows each patch to have its own set of learnable parameters independent of other patches. It reduces the number of parameters and computation complexity while still capturing local semantic information and benefiting from longer look-back windows.

CrossFormer

Previous works on using the Transformer model for time series problems focused on how to model the relationships between different time steps more effectively using the self-attention mechanism. However, in the context of MTS, the relationships between variables are also crucial, as the meaning of each variable in the time series is different, and each variable may align at different time steps. Zhang and Yan [78] proposes CrossFormer, which converts the time series into **patches**, adds attention between variables, and incorporates a **"router" mechanism to improve efficiency**. The model includes three main modules: Dimension-Segment-Wise Embedding, which converts the time series into patch embeddings; Two-Stage Attention Layer, which applies attention across both the time and variable dimensions; and Hierarchical Encoder-Decoder, which forms a hierarchical encoding and decoding structure using different patch sizes and partitioning schemes. CrossFormer achieves state-of-the-art performance on various real-world datasets.

ETSformer

Woo *et al.* [79] proposed a novel time-series Transformer architecture called ETSformer. The approach is inspired by **the classical ETS method**, which makes the architecture more effective and efficient for long-term forecasting. ETSformer incorporates inductive

biases of time-series structures by performing a layer-wise level, growth, and seasonal decomposition. By leveraging the high capacities of deep architectures and an effective residual learning scheme, ETSformer can extract a series of latent growth and seasonal patterns and model their complex dependencies. It also introduces a novel Exponential Smoothing Attention (ESA) and Frequency Attention (FA) to replace vanilla self-attention.

Scaleformer

Shabani *et al.* [80] proposed Scaleformer, a novel multi-scale framework for TSF problems. Scaleformer introduces **an iterative scale-refinement paradigm** that can be adapted to a variety of transformer-based architectures. To minimize distribution shifts between scales and windows, Scaleformer also incorporates **a cross-scale normalization on the outputs** of the Transformer. Scaleformer can use many state-of-the-art architectures as backbones, resulting in decent performance improvement on several real-world datasets.

3.3 Conclusions

This chapter contains two main parts: a comprehensive introduction to the traditional DL models and a gentle overview of the existing Transformer models for the TSF problem.

In the first part, we discussed the bottlenecks of the econometric and ML models for the TSF problem. Then, we introduced several DL models dedicated to the TSF problem, including the basic MLPs, CNNs, RNNs, and Attention Mechanism. We summarized their advantages and disadvantages and discussed their applications to the TSF problem. We also discussed several hybrid models combining econometric methods with the DL models to improve the performance of TSF.

In the second part, we concisely revised the Transformer architecture. We then reviewed the recent advances in TSF using Transformer models, including 14 different Transformer-based models. We discussed their improvements over the vanilla Transformer as well as their precursors. This chapter provides a short comprehensive overview of the Transformer model and its variants for TSF.

To sum up, this chapter not only gives a foundation of the DL but also provides a comprehensive overview of the state-of-the-art DL models for the TSF problem. We hope this chapter can help the readers to understand the DL models for the TSF problem and inspire them to develop new DL models for the TSF problem.

In the following chapters, we present our contributions as four consecutive studies. In Chapter4, we conduct a comprehensive comparison of the STL decomposition prior to forecasting methods (decomposition & preprocessing). In Chapter 5, deep learning multi-step forecasting strategies are introduced (forecasting strategies & deep models). We propose deep learning Transformer-based forecasting with rank correlation function and STL decomposition in Chapter 6 (Transformer & rank correlation & STL decomposition). Finally, Chapter 7 presents a prototype application for time series forecasting (web application).

Chapter 4

STL decomposition prior to econometric and ML models

Contents

In this chapter, we investigate the different aforementioned forecasting methods, combine them with STL decomposition, and compare their performance. The forecasting methods we consider are as follows:

- Three econometric methods: ARIMA, ETS, and Theta.
- Five frequently used machine learning methods: kNN, SVR, CART, RF, and GP.

We conduct our forecasting test on six horizons: 1, 6, 12, 18, and 24. Our results show that, when applied to the monthly industrial M3-Competition dataset as a preprocessing

step, STL decomposition can benefit forecasting using econometric methods but harms the machine learning ones. Moreover, the STL-Theta combination method displays the best forecasting results on four of the five forecasting horizons.

The rest of this chapter is organized as follows. In Sec. 4.1, we give a brief introduction to the forecasting competition and this comparison study. Then, we present a concise description of all the involved models and the decomposition methods in Sec. 4.2. Sec. 4.3 presents how we organized and conducted the experiments. In Sec. 4.4, we present the comparison results and discussions based on these results. Sec. 4.5 gives the conclusion of this comparative study. This chapter can also be found in the corresponding journal version of the article [81].

4.1 Introduction

Since some forecasting methods can be more appropriate than others to certain scenarios, comparing their performance becomes a natural way when selecting forecasting strategies.

Although forecasting comparisons can be found among ancient Greeks, forecasting competitions have a relatively short history, with the first one traced back to 1974. Nowadays, the famous forecasting competition is the *M-Competitions*. The first one, i.e., the M1-Competition, was held by Spyros Makridakis and Michèle Hibon back in 1982, and four subsequent M-Competitions, namely M2–M4-Competitions, have been organized since then. There are many interesting discussions and debates about these competitions and their results during these years. Readers can find a brief yet thought-provoking history of forecasting competitions in [34].

Over the last decade, Artificial Intelligence (AI) has gained significant prominence, especially in Computer Vision [82], natural language processing (NLP) [83], and autonomous driving [84]. Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision [85]. Recurrent Neural Networks (RNNs) and Transformer models revolutionized NLP fields, such as machine translation and speech recognition [63], [86], [87]. In the field of time series, many machine learning methods such as support vector regression (SVR), neural networks (NN), classification and regression tree (CART), and k-nearest neighbor (kNN) were proven able to model and forecast time series as well [81], [88], [89].

There are some discussions on comparing the performance of different forecasting approaches. Ahmed *et al.* [90] performed an empirical comparison of eight machine learning models over the 1045 monthly series involved in the M3-Competition, but only one-step-ahead forecasting was considered. Makridakis, Spiliotis, and Assimakopoulos [91] did some similar works, comparing econometric and machine learning methods, but without any decomposition method being introduced as a preprocessing step. Using the M1-Competition dataset, Theodosiou [92] compared a new STL-based method with some common benchmarks but without combining STL with them, and only up to 18-month forecasting was considered.

As the preprocessing step often plays an integral part in prediction tasks and substantially impacts the results, we propose to conduct a new comparison work to identify its benefit: (1) by exploring STL decomposition when using it as a preprocessing step for all methods; and (2) by considering multiple forecasting horizons.

4.2 Methods

Although there are many different variations of each model, we considered only the primitive versions of each model in our experiments. As this study is inspired heavily by the M3 and M4-Competitions, we kept the six benchmark methods used in these two competitions by the organizer [93].

4.2.1 Benchmark Methods

Below is a list of descriptions of the benchmarks utilized in the M4-Competition.

- **Naïve 1.** Naïve 1 assumes future values are identical to the last observation.
- **Naïve S.** Naïve S assumes future values are identical to the values from the last known period, which, in our case, is 12 months.
- **Naïve 2.** Naïve 2 is similar to Naïve 1, except the data are seasonally adjusted by a conventional multiplicative decomposition if tested seasonal. We performed a 90% autocorrelation test at lag 12 for each series.
- **Simple Exponential Smoothing (SES).** SES forecasts future values as exponentially decayed weighted averages of past observations.
- **Holt.** Holt's linear trend method extends SES for data with a trend.
- **Damped.** The damped model dampens the trend in Holt's method.

4.2.2 Decomposition Methods

Here, we redescribe briefly the two commonly used decomposition methods.

Canonical Decomposition

The canonical multiplicative decomposition algorithm for a series with a seasonal period of s has four steps:

1. Compute the trend-cycle component $\hat{T}_t$ using a simple MA smoothing.
2. Detrend the series: $y_t/\hat{T}_t$.
3. Compute the seasonal component $\hat{S}_t$ by averaging the corresponding season's detrended values.
4. Compute the remainder component $\hat{R}_t$: $\hat{R}_t = y_t/(\hat{T}_t\hat{S}_t)$.

STL Decomposition

STL decomposition consists of two recursive procedures: an inner loop and an outer loop. The inner loop fits the trend and calculates the seasonal component. Every inner loop consists of six steps in total:

1. Detrending. Calculate a detrended series $y_t - T_t^{(k)}$. For the first pass, $T_t^{(0)} = 0$.
2. Cycle-Subseries Smoothing. Use LOESS to smooth the subseries of values at each position of the seasonal cycle. The result is marked as $C_t^{(k+1)}$.
3. Low-Pass Filtering of Smoothed Cycle-Subseries. This procedure consists of two MA filters and a LOESS smoother. The result is marked as $L_t^{(k+1)}$.
4. Detrending of Smoothed Cycle-Subseries. $S_t^{(k+1)} = C_t^{(k+1)} - L_t^{(k+1)}$.
5. Deseasonalizing. $y_t - S_t^{(k+1)}$.
6. Trend Smoothing. Use LOESS to smooth the deseasonalized series to get the trend component of this iteration $T_t^{(k+1)}$.

If any anomaly is detected, an outer loop will be applied, and replace the LOESSs at the second and sixth steps of the inner loop with the robust LOESS.

4.2.3 Econometric Methods

We select ARIMA, ETS, and Theta models as representatives of econometric methods.

- **ARIMA.** An ARIMA model assumes future values to be linear combinations of past values and random errors, contributing to the AR and MA terms, respectively. SARIMA (Seasonal ARIMA) is an extension of ARIMA that explicitly supports time series data with a seasonal component. Once STL decomposition is applied, SARIMA models degenerate into regular ARIMA models as STL handles the seasonal part.
- **ETS.** The ETS models are a family of time series models with an underlying state space model consisting of a level component, a trend component (T), a seasonal component (S), and an error term (E). Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older. After concatenating STL on the ETS model, the full ETS model degenerates into Holt's method as the seasonal equation is handled by STL.
- **Theta Method.** The Theta method performed exceptionally well in the M3-Competition and was used as a benchmark in the M4-Competition. The Theta method is based on the concept of modifying the local curvature of the time series through a coefficient θ, which is applied directly to the second difference of the data.

4.2.4 Machine Learning Methods

It is interesting to closely examine how machine learning methods perform in TSF tasks. Using the embedding strategy to transform this task into a supervised learning problem [94], we can apply machine learning techniques to TSF tasks. The following briefly introduces the machine learning methods used in this experiment.

- ***k*-NN.** k-NN is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors based on the Euclidean distances.
- **SVR.** SVM is a successful method that tries to find a separation hyperplane to maximize the margin between two classes, while SVR seeks a hyperplane to minimize the margin between the support vectors and the hyperplane.
- **CART.** CART is one of the most generally used machine learning methods and can be used for classification and regression. CART dichotomizes each feature recursively and divides the input space into several cells. CART computes the probability distributions of the corresponding prediction in it.
- **RF.** RF is an ensemble learning algorithm based on the Decision Tree [95]. Similar to CART, RF can be used for both classification and regression. It operates by constructing many decision trees at training time and calculating the average predictions from the individual trees.
- **GP.** A GP is a generalization of the Gaussian probability distribution [96]. It uses a measure of homogeneity between points as a kernel function to predict an unknown point's value from the input training data. Its prediction results contain the value of the point and the uncertainty information, i.e., its one-dimensional Gaussian distribution [91].

4.3 Experiments

This section presents how we organized and performed our experiment.

4.3.1 Dataset

We selected 332 monthly series from the industry category, which contains the highest number of points per series from the M3-Competition dataset.[1] We set 84 as the length of the historical data and tested five different forecast horizons, i.e., 1, 6, 12, 18, and 24 months. Thus, the total length required for an appropriate series is 108. The two series N2011 (78 points) and N2118 (104 points), were thus removed from the original 334-series dataset.

[1]Data retrieved from **International Institute of Forecasters**: `https://forecasters.org/resources/time-series-data/m3-competition/`

4.3.2 Pipeline for Machine Learning Methods

Data Preprocessing

In our experiment, three preprocessing techniques were conducted on all the series:

1. **Deseasonalizing**: A 90% autocorrelation test at lag 12 is performed to decide whether the series is seasonal. We perform a conventional multiplicative decomposition or an STL decomposition if the series is seasonal and extract the seasonal part.
2. **Detrending**: A one-order differencing is performed to eliminate the trend.
3. **Scaling**: A standardization step is applied to remove the mean and scale the features to unit variance.

Supervised Learning Setting

A time series prediction problem can be transformed into a supervised learning task that machine learning methods can do. A commonly used approach is to formulate a training set by lagging and stacking the historical series several times.

Typically, for a one-step-ahead prediction problem, we can construct a training set $\{X, Y\}$ as follow:

$$X = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ y_2 & y_3 & \cdots & y_{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{N-n} & y_{N-n+1} & \cdots & y_{N-1} \end{bmatrix}, Y = \begin{bmatrix} y_{n+1} \\ y_{n+2} \\ \vdots \\ y_N \end{bmatrix},$$

where N is the total length of the series, and n is the number of times we lag the series, often referred to as the window length. Each row in X represents a training example, while its label corresponds to Y's element with the same index.

In an h-step-ahead case, the training set evolves into:

$$X = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ y_2 & y_3 & \cdots & y_{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{N-n-h+1} & y_{N-n-h+2} & \cdots & y_{N-h} \end{bmatrix}, Y = \begin{bmatrix} y_{n+1} & y_{n+2} & \cdots & y_{n+h} \\ y_{n+2} & y_{n+3} & \cdots & y_{n+h+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{N-h+1} & y_{N-h+2} & \cdots & y_N \end{bmatrix}.$$

This transformation is often referred to as the embedding technique in the R implementation [97]

Results Post-Processing

The post-processing part comprises the inverted operations of the three preprocessing steps:

1. **Rescaling**: A rescaling step is performed by inverting the standardization operation.

2. **Retrending**: A cumulated summing is conducted to bring back the trend eliminated by the first-order differencing.

3. **Reseasonalizing**: A reseasonalization step is executed to integrate the seasonal component into the prediction. The first h points in the last season in the seasonal component are selected where h is the forecast horizon, and when $h > 12$, data from the last season are concatenated to provide a proper length.

Fig. 4.1 shows a scheme of the machine learning pipeline.

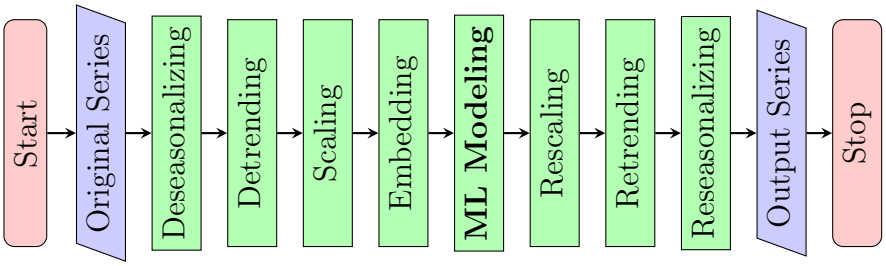


Figure 4.1: Flowchart of the machine learning pipeline.

4.3.3 Pipeline for Econometric Methods

econometric methods require no preprocessing or post-processing as the Machine Learning and Deep Learning methods demand. However, the same deseasonalization and reseasonalization steps are necessary for the STL-based methods.

In our experiment, we built some hybrid STL decomposition-based econometric methods. We performed an STL decomposition and constructed the ARIMA, ETS, and Theta models upon the seasonally adjusted series to compute the point forecasts. It comprises four procedures:

1. **STL decomposition.** Perform the STL decomposition to extract the underlying trend, seasonal and residual components. Seasonality test is performed as per ML/DL method in advance.

2. **Deseasonalization.** Compute the deseasonalized series by subtracting the seasonal component. This step is also called seasonal adjustment.

3. **Point forecasting.** Construct the ARIMA, ETS, and Theta models on the seasonally adjusted data and calculate the forecasting values.

4. **Reseasonalization.** Add the seasonal component back to the forecasting to calculate the final forecasting results.

One effect of applying the STL decomposition on econometric methods is that it cancels these econometric methods' intrinsic seasonality handlers.

Fig. 4.2 is the scheme of the econometric methods pipeline.

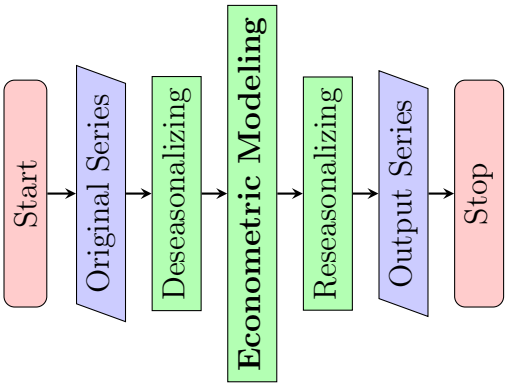


Figure 4.2: Flowchart of the econometric methods pipeline.

4.3.4 Implementation and Parameters Tuning

Econometric Methods

All of the econometric methods, as well as their STL-based versions, were conducted using the `forecast-8.13` package [17] in R 4.0.2.

Machine Learning Methods

The machine learning methods and their STL-based versions were tested by exploiting Python `3.8.5` with the help of `sktime-0.4.2` [22], `scikit-learn-0.23.2` [98], and `statsmodels-0.12.1` [15] libraries.

4.3.5 Evaluation Metrics

Three evaluation metrics were used in this experiment.

We used the symmetric Mean Absolute Percentage Error (sMAPE) [99]. It has the following formula:

$$\text{sMAPE} = \frac{2}{h}\sum_{t=1}^{h}\frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|} \times 100\%, \tag{4.1}$$

where h is the forecasting horizon, y_t is the actual values at time t, and $\hat{y}_t$ is the forecast produced by the model.

We also used the Mean Absolute Scaled Error (MASE) introduced by Rob Hyndman [100]:

$$\text{MASE} = \frac{1}{h}\frac{\sum_{t=1}^{h}|y_t - \hat{y}_t|}{\frac{1}{n-s}\sum_{t=s+1}^{n}|y_t - y_{t-s}|}, \tag{4.2}$$

where n is the number of observations, and s is the number of periods per season.

The Overall Weighted Average (OWA) w.r.t. Naïve 2 was also adopted [93]:

$$\text{OWA} = \frac{1}{2}\left(\frac{\text{sMAPE}_{\text{Model X}}}{\text{sMAPE}_{\text{Naïve 2}}} + \frac{\text{MASE}_{\text{Model X}}}{\text{MASE}_{\text{Naïve 2}}}\right). \tag{4.3}$$

4.4 Results and Discussions

4.4.1 Results

The results of our experimentation are presented in Tab. 4.1, Fig. 4.3–4.5, and the following contents.

Tab. 4.1 represents the forecast results of different methods on different forecast horizons. Note that Naïve 2 was chosen as the reference method for the OWA indicator, meaning that OWA equals 1, whatever the horizon value h. At first glance, in Tab. 4.1, most of the econometric methods give better forecasting results w.r.t. naïve methods (OWA < 1) than the machine learning methods (OWA > 1). This result confirms the conclusion from the M3-Competition that sophisticated machine learning methods do not assure a more accurate prediction than simple econometric methods.

This result becomes obvious in Fig. 4.3, showing OWA ≤ 0.910 performance results for the three advanced econometric methods (ARIMA, ETS, and Theta) by comparison with Fig. 4.4, showing OWA ≥ 0.914 performance results for the five machine learning methods. Above all, Fig. 4.3 and Fig. 4.4 show the impact of STL decomposition as a preprocessing step of econometric and ML methods on the forecasting performance results.

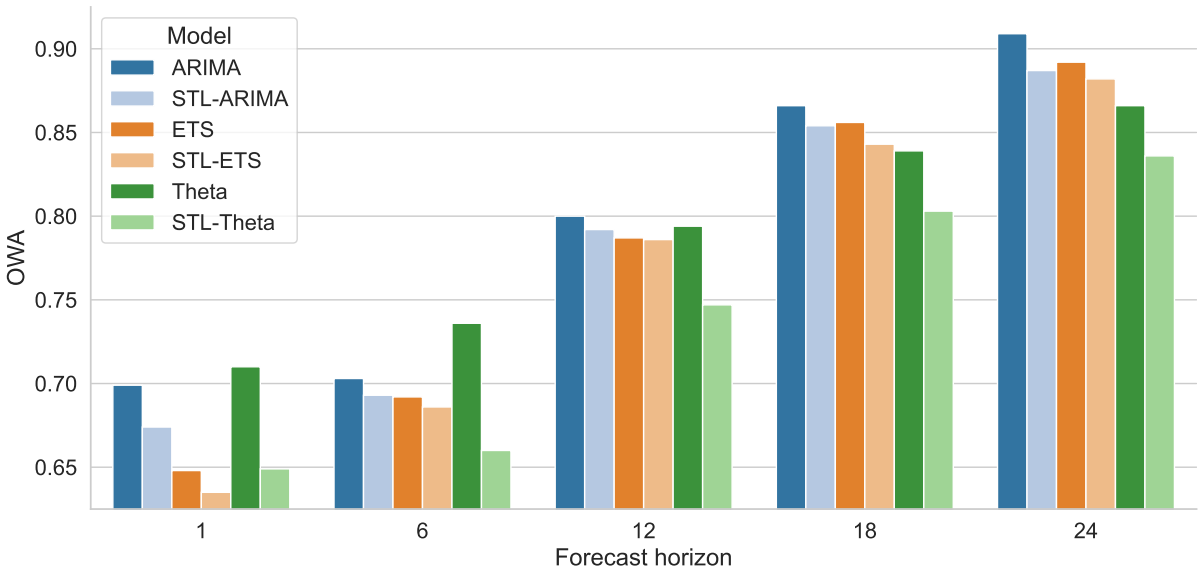


Figure 4.3: OWAs for STL decomposition on econometric models.

Significant improvement from STL decomposition was found for econometric methods. Among all the tested STL-based methods, the STL-Theta method outperforms the other methods on almost all forecast horizons. The STL-Theta method can even give a lower OWA on a 24-month forecast horizon than the other methods on the 18-month one.

In Fig. 4.4, we can find that the SVR model gives the best result. No significant improvement from STL preprocessing was detected.

Fig. 4.5 shows the mean and standard deviation of the gain brought by STL decomposition calculated as $\frac{OWA - OWA_{STL}}{OWA}$. On average, STL improves the OWA of ARIMA by 1.5%, ETS by 0.9%, and Theta by 5%, but it conducts a loss of OWA for machine learning methods. It harms SVR by 2.3%, RF by 3.3%, GP by 2.3%, kNN by 2.2%, and

Table 4.1: Forecasting results of different methods on different forecast horizons.

	h = 1			**h = 6**			**h = 12**			**h = 18**			**h = 24**		
Econometric	**sMAPE**	**MASE**	**OWA**	**sMAPE**	**MASE**	**OWA**	**sMAPE**	**MASE**	**OWA**	**sMAPE**	**MASE**	**OWA**	**sMAPE**	**MASE**	**OWA**
Naive	12.536	1.006	1.071	16.011	1.280	1.177	16.238	1.312	1.152	17.480	1.395	1.153	18.044	1.456	1.143
sNaive	12.464	0.882	1.002	12.001	0.874	0.842	12.726	0.925	0.857	14.088	1.033	0.891	14.689	1.094	0.894
Naive2	11.704	0.939	1.000	13.813	1.071	1.000	14.374	1.118	1.000	15.431	1.189	1.000	16.053	1.254	1.000
SES	9.277	0.723	0.781	11.386	0.844	0.806	12.376	0.931	0.847	13.640	1.017	0.870	14.397	1.092	0.884
Holt	9.734	0.741	0.810	11.669	0.865	0.826	13.522	1.004	0.920	15.710	1.161	0.997	17.197	1.293	1.051
Damped	9.288	0.720	0.780	11.388	0.844	0.806	12.572	0.942	0.859	13.985	1.036	0.889	14.740	1.110	0.902
ARIMA	8.643	0.623	0.701	10.037	0.730	0.704	11.824	0.873	0.802	13.581	1.015	0.867	14.794	1.127	0.910
ETS	7.805	0.591	0.648	9.875	0.716	0.692	11.718	0.849	0.787	13.608	0.987	0.856	14.751	1.085	0.892
Theta	8.645	0.640	0.710	10.668	0.749	0.736	11.862	0.854	0.794	13.403	0.962	0.839	14.399	1.047	0.866
STL-ARIMA	8.245	0.604	0.674	9.915	0.717	0.693	11.755	0.856	0.792	13.457	0.993	0.854	14.481	1.093	0.887
STL-ETS	7.760	0.569	**0.635**	9.882	0.704	0.686	11.728	0.845	0.786	13.433	0.969	0.843	14.552	1.074	0.882
STL-Theta	7.963	0.580	0.649	9.502	0.678	**0.660**	11.177	0.801	**0.747**	12.817	0.921	**0.803**	13.891	1.011	**0.836**
	h = 1			**h = 6**			**h = 12**			**h = 18**			**h = 24**		
ML & DL	**sMAPE**	**MASE**	**OWA**	**sMAPE**	**MASE**	**OWA**	**sMAPE**	**MASE**	**OWA**	**sMAPE**	**MASE**	**OWA**	**sMAPE**	**MASE**	**OWA**
*k*NN	13.636	0.965	1.096	16.070	1.166	1.126	17.781	1.326	1.212	20.421	1.497	1.291	21.741	1.610	1.319
SVR	11.253	0.855	**0.936**	12.732	0.971	**0.914**	14.712	1.116	**1.011**	17.485	1.284	**1.107**	19.526	1.429	1.178
CART	14.080	1.025	1.147	19.081	1.377	1.334	25.490	1.930	1.750	30.934	2.314	1.975	35.956	2.596	2.155
RF	11.756	0.898	0.980	13.668	1.027	0.974	15.432	1.186	1.067	17.831	1.369	1.153	19.692	1.496	1.210
GP	12.540	0.972	1.053	14.268	1.093	1.027	15.528	1.195	1.075	17.395	1.313	1.116	18.720	1.418	**1.148**
STL-*k*NN	15.318	1.077	1.228	18.359	1.390	1.313	17.980	1.306	1.210	22.513	1.698	1.444	22.154	1.610	1.332
STL-SVR	12.978	1.006	1.090	15.285	1.225	1.125	14.919	1.109	1.015	19.338	1.484	1.251	19.589	1.410	1.172
STL-CART	15.820	1.191	1.310	21.715	1.660	1.561	25.157	1.862	1.708	32.285	2.446	2.075	35.715	2.537	2.124
STL-RF	13.667	1.054	1.145	16.401	1.289	1.195	15.880	1.177	1.079	20.237	1.581	1.321	19.947	1.465	1.205
STL-GP	14.163	1.120	1.201	16.950	1.351	1.244	15.782	1.187	1.080	19.624	1.526	1.278	18.974	1.408	1.152

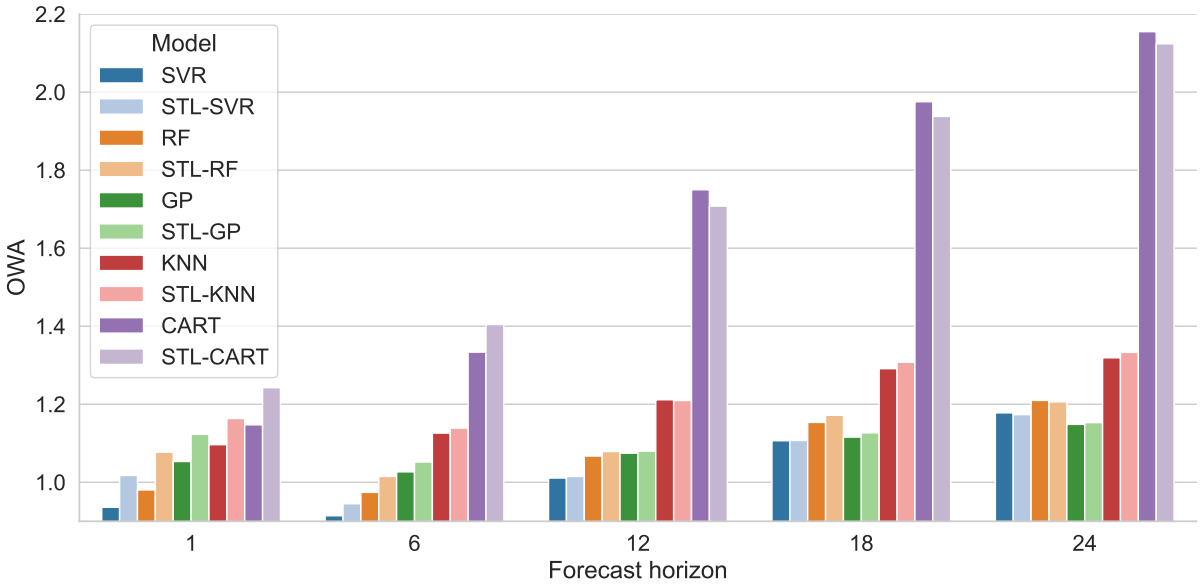


Figure 4.4: OWAs for STL decomposition on machine learning models.

CART by 1.1%.

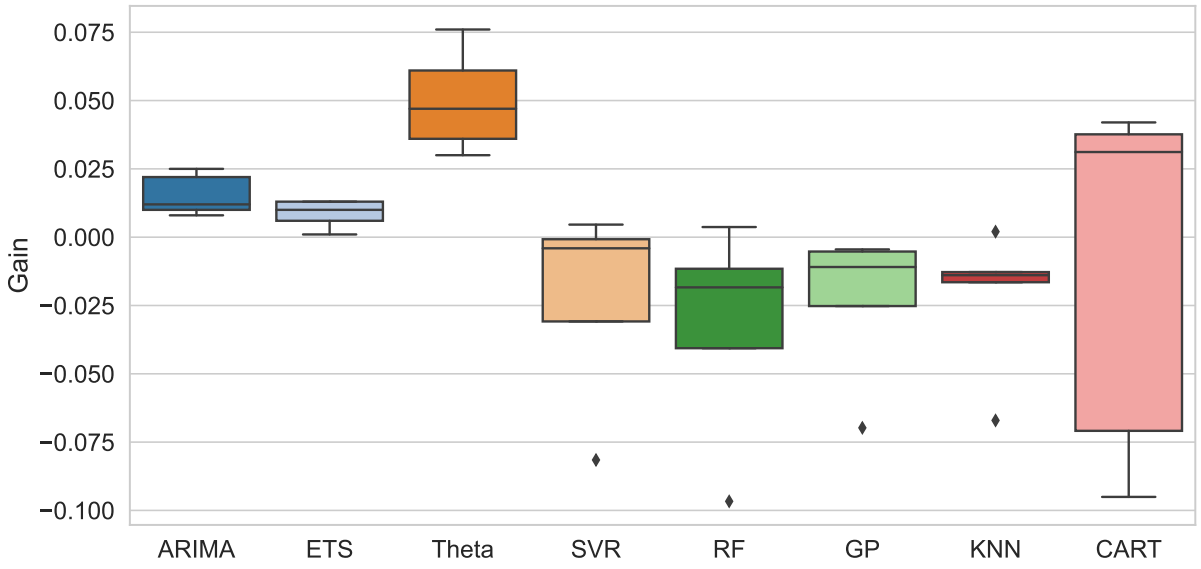


Figure 4.5: Boxplot of OWA gain from STL for each method.

4.4.2 Discussions

It is interesting to note from the results in Fig. 4.5 that CART performs the worst among all these methods, which is easy to understand as CART is a single forecaster. Its ensemble method Random Forest performs much better in terms of the precision of forecasting. At the same time, it consumes the most time.

The initial objective of this study was to determine whether STL decomposition can be helpful as a preprocessing step for TSF methods. Our results confirm using STL de-

composition as a preprocessing method can effectively improve the econometric methods' performance, which is consistent with [92] using M1-Competition data, but, for machine learning methods, it can be harmful.

A possible explanation for this might be extracting the seasonal information from the series can affect the features to be modeled. For econometric models, their intrinsic ability to handle the seasonality might be worse than the STL decomposition. For the machine learning models, it could be easier to model seasonal data. Further research is required to confirm this hypothesis.

4.5 Conclusions

The present study in this chapter was designed to investigate the effect of using STL decomposition as a preprocessing step on different forecasting strategies. The results show some vast differences between these methods. Among all tested models, the STL decomposition-based Theta method is the best one. In the meantime, the STL decomposition can benefit the econometric methods by providing a more robust decomposition procedure than their intrinsic mechanism. The machine learning methods tested in this experiment failed to outperform most econometric methods but still have some potential for improvement, e.g., combined with the econometric methods. More research is required in the future.

Chapter 5

Deep learning with multi-step forecasting strategies

Contents

Multivariate time series forecasting problem has attracted enormous research in recent years, and many deep learning models have been proposed and claimed to be effective in different tasks. We find that many of these models were tested in a simple one-step-ahead strategy, which does not apply to real scenarios requiring multi-step forecasting.

In this chapter, we investigate the performance of the three DL models which are known to be the milestones for the TSF problem (i.e., DA-RNN, LSTNet, and TPA-LSTM), for the MTSF problem, under five forecasting strategies for multi-step forecasting, namely, One-Step-Ahead, Recursive, Direct, MIMO, and MISMO strategies.

We conducted our experiments on six datasets, whose statistical nature varies in different aspects, with four forecasting horizons: 3, 6, 12, and 24. They are NASDAQ 100 Stock Data, Beijing PM2.5 Data, Electricity, Exchange Rate, Solar Energy, and Traffic.

The rest of this chapter is organized as follows. In Sec. 5.1, we give a simple introduction to this investigation. In Sec. 5.2, we present a concise description of all the involved deep learning models and the forecasting strategies. Sec. 5.3 presents how we organized and conducted the experiments. We present the comparison results and discussions based on these results in Sec. 5.4. Sec. 5.5 gives the conclusion. This chapter can also be found in the corresponding conference version of the paper [101].

5.1 Introduction

Recall that there are three crucial DL models in the field of MTS forecasting, i.e., *Dual-stage Attention-based Recurrent Neural Network (**DA-RNN**)* [54] for introducing attention mechanism into MTS analysis for the first time, *Long- and Short-term Time-series network (**LSTNet**)* [55] for combining CNN and RNN for MTS data, and *Temporal Pattern Attention Long Short-Term Memory (**TPA-LSTM**)* [56] for introducing the *Temporal Attention Pattern* concept for selecting relevant variables.

Nevertheless, although multi-step forecasting was claimed to be conducted in their original papers, only a one-step-ahead strategy was actually applied according to their descriptions for problem formulation. In this strategy, the authors generated a single-step-ahead forecast and fed the model with the new actual data to generate the following step. This strategy is intuitive but can only apply to limited cases where multi-step forecasting is not required.

For multi-step forecasting in real life where we do not possess future values, the *Recursive* and the *Multi-Input Multi-Output (MIMO)* strategies were often considered, and several machine learning models were proven to be applicable with these strategies to many tasks [102], [103]. To verify the applicability of deep learning models on multi-step tasks, we conducted several experiments in which we:

1. Implemented these three models using multi-step forecasting strategies.
2. Evaluated and compared their performance over different horizons.
3. Tested their applicability for multi-step forecasting.

5.2 Methods

This section reviews the previously mentioned multi-step forecasting strategies and deep learning models.

5.2.1 Multi-step Forecasting Strategies

A TSF problem can be transformed into a supervised learning task that ML/DL methods can do. A commonly used approach is to formulate a training set by lagging and stacking the historical series several times.

For a one-step forecasting problem, we can construct a training set $\{X, Y\}$ of shape $[(N-n), n]$ and $[(N-n), 1]$ where N is the total length of the series, and n is the number of times we lag the series, often named *window length*:

$$X = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ y_2 & y_3 & \cdots & y_{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{N-n} & y_{N-n+1} & \cdots & y_{N-1} \end{bmatrix}, Y = \begin{bmatrix} y_{n+1} \\ y_{n+2} \\ \vdots \\ y_N \end{bmatrix}, \tag{5.1}$$

where each row in X represents a training example, while its target corresponds to the element in the same row in Y.

Nonetheless, as Y is a vector, (5.1) only describes the strategy for one-step-ahead forecasting. Four strategies extending the framework to tackle the multi-step forecasting problem are discussed in the following content.

Recursive

In the *Recursive* strategy [102], a single model f is trained and used recursively to generate multi-step forecasting by taking the predicted values as the input for future time steps.

$$y_{t+1} = f(y_t, y_{t-1}, ..., y_{t-n+1}) + w_{t+1},$$

with $t \in \{n, n+1, ..., N-1\}$ and w_{t+1} is a noise term.

Direct

In the *Direct* strategy [104], H models f_h are trained independently:

$$y_{t+h} = f_h(y_t, y_{t-1}, ..., y_{t-n+1}) + w_{t+h},$$

with $t \in \{n, n+1, ..., N-H\}$ and $h \in \{1, 2, ..., H\}$ where H is the forecast horizon. Every model outputs a prediction of one value. The predictions from the H models are then concatenated to form an H-step forecasting.

The *Direct* strategy has no accumulated error but requires a long computational time due to enormous models to learn. Moreover, it requires more complex models than the *Recursive* strategy to model the dependency between two distant points [94].

Multi-Input Multi-Output (MIMO)

The *Recursive* strategy is intuitive but suffers from accumulated errors. The *MIMO* strategy [103] was proposed to alleviate this problem.

The *MIMO* strategy learns a single multiple output model F:

$$[y_{t+H}, y_{t+H-1}, ..., y_{t+1}] = F(y_t, y_{t-1}, ..., y_{t-n+1}) + \mathbf{w},$$

with $t \in \{n, n+1, ..., N-H\}$. $F : \mathbb{R}^n \mapsto \mathbb{R}^H$ is a vector-valued function and $\mathbf{w} \in \mathbb{R}^H$ is a noise vector.

The *MIMO* strategy extends (5.1) into the following format:

$$X = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ y_2 & y_3 & \cdots & y_{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{N-n-H+1} & y_{N-n-H+2} & \cdots & y_{N-H} \end{bmatrix}, Y = \begin{bmatrix} y_{n+1} & y_{n+2} & \cdots & y_{n+H} \\ y_{n+2} & y_{n+3} & \cdots & y_{n+H+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{N-H+1} & y_{N-H+2} & \cdots & y_N \end{bmatrix}.$$

The rationale of the *MIMO* strategy is that it outputs all future values in one vector during the forecasting stage. Meanwhile, it models the dependency between the values that characterizes the time series [94].

Multi-Input Several Multi-Output (MISMO)

Another multiple-output strategy is called MISMO [105]. It is a trade-off between *Direct* and *MIMO*. It can be described in the following equation:

$$[y_{t+m\times s}, y_{t+m\times s-1}, ..., y_{t+(m-1)\times s+1}] = F_m(y_t, y_{t-1}, ..., y_{t-n+1}) + \mathbf{w}_m,$$

with $m \in \{1, 2, ..., M\}$, $F_m : \mathbb{R}^n \mapsto \mathbb{R}^s$, and $\mathbf{w}_m \in \mathbb{R}^s$. *MISMO* learns an H-step forecast with M *MIMO* models ($M = \frac{H}{s}$), each model outputs an s-step forecast ($s \in \{1, 2, ..., H\}$). During the forecasting stage, the M forecasting of s values are concatenated to form the final results.

A *MISMO* model can boil down to *Direct* or *MIMO* model:

- If $s = 1$, *MISMO* = *Direct*.
- If $s = H$, *MISMO* = *MIMO*.

This provides a convenient trade-off between controlling the dependencies among future values and preserving the flexibility of the predictor [94].

5.2.2 Deep Learning Models

In this section, we present the three deep learning models previously mentioned.

DA-RNN

DA-RNN was proposed by Qin *et al.* [54] at the International Joint Conference on Artificial Intelligence (IJCAI) in 2017. It is a sequence-to-sequence model [87] combined with the attention mechanism.

Unlike the usual attention models for natural language processing, which include the attention mechanism only at the decoder stage, DA-RNN includes a dual attention mechanism at both the encoder and decoder stages. Fig. 5.1 and Fig. 5.2 illustrate the two attention mechanisms inside DA-RNN.

As multivariate time series are taken into consideration, in DA-RNN's paper, authors call exogenous series *driving series* and series to be predicted are referred to as *target series*.

Input Attention Mechanism As shown in Fig. 5.1, DA-RNN starts by treating different time series rather than time steps. It reads in n driving series of length T, i.e., $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^n\}^\top = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\} \in \mathbb{R}^{n\times T}$, with an LSTM network in favor of its benefit of handling the vanishing gradient problem and better capturing the long-term dependencies of time series. The target series is denoted by $\{y_1, y_2, ..., y_{T-1}\}$ with $y_t \in \mathbb{R}$. The prediction is thus $\hat{y}_T$.

Inside the LSTM layer, an *Input Attention Layer* calculates the importance of the k-th input series as:

$$e_t^k = \mathbf{v}_e^\top \tanh(\mathbf{W}_e[\mathbf{h}_{t-1}, \mathbf{s}_{t-1}] + \mathbf{U}_e\mathbf{x}^k), \quad 1 \leq k \leq n,$$

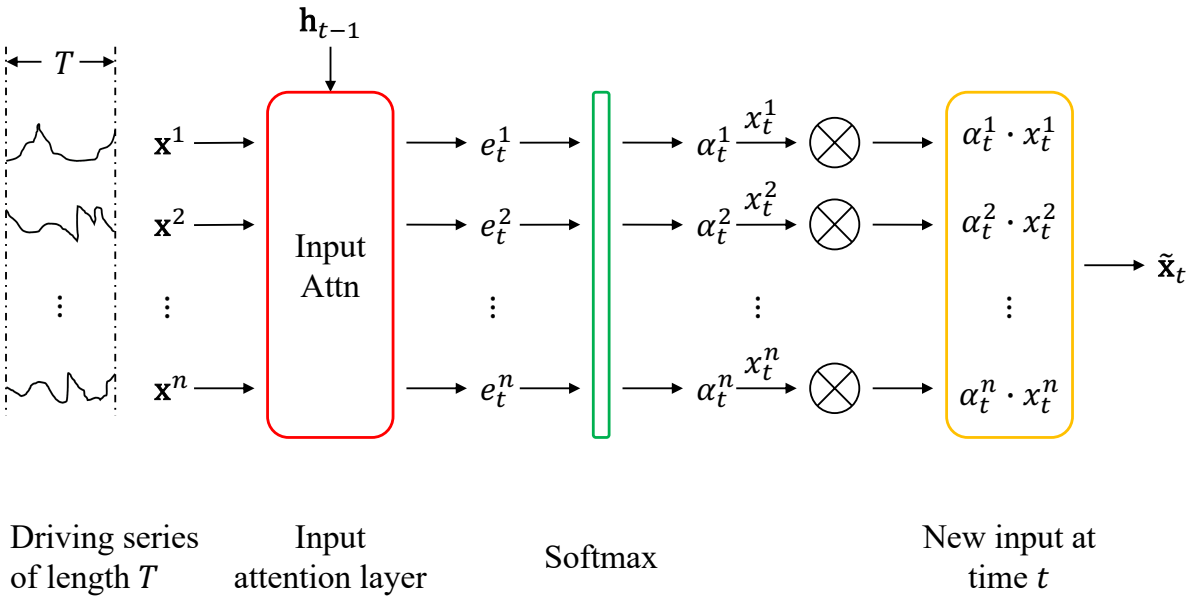


Figure 5.1: DA-RNN's Input Attention Mechanism.

where $\mathbf{v}_e \in \mathbb{R}^T$, $\mathbf{W}_e \in \mathbb{R}^{T\times 2m}$ and $\mathbf{U}_e \in \mathbb{R}^{T\times T}$ are parameters to learn. $\mathbf{h}_{t-1}, \mathbf{s}_{t-1} \in \mathbb{R}^m$ are respectively the encoder hidden and cell states in the m-unit LSTM.

Attention weights are then calculated by applying a softmax function on the importance e_t^k:

$$\alpha_t^k = \text{softmax}\left(e_t^k\right) = \frac{\exp(e_t^k)}{\sum_{i=1}^n \exp(e_t^i)}.$$

Then, the driving series with relevant ones attended are extracted:

$$\tilde{\mathbf{x}}_t = (\alpha_t^1 x_t^1, \alpha_t^2 x_t^2, ..., \alpha_t^n x_t^n)^\top.$$

Temporal Attention Mechanism DA-RNN cooperates with another attention layer called *Temporal Attention Mechanism* as shown in Fig. 5.2 to automatically select relevant encoder hidden states across all time steps from the encoder output.

Specifically, the temporal attention mechanism contains two LSTMs. The first LSTM serves inside the encoder as $f_1(\cdot)$. The relevant driving series is passed consecutively to update the hidden state in this LSTM layer:

$$\mathbf{h}_t = f_1(\mathbf{h}_{t-1}, \tilde{\mathbf{x}}_t).$$

Then a *Temporal Attention Layer* calculates the importance of the i-th time step:

$$l_t^i = \mathbf{v}_d^\top \tanh(\mathbf{W}_d[\mathbf{d}_{t-1}, \mathbf{s}'_{t-1}] + \mathbf{U}_d \mathbf{h}_i), \quad 1 \le i \le T,$$

where $\mathbf{v}_d \in \mathbb{R}^m$, $\mathbf{W}_d \in \mathbb{R}^{m\times 2p}$ and $\mathbf{U}_d \in \mathbb{R}^{m\times m}$ are parameters to learn. $\mathbf{d}_{t-1}, \mathbf{s}'_{t-1} \in \mathbb{R}^p$ are respectively the decoder hidden and cell states in the p-unit LSTM. The attention weight β_t^i is then calculated by:

$$\beta_t^i = \text{softmax}\left(l_t^i\right) = \frac{\exp(l_t^i)}{\sum_{j=1}^T \exp(l_t^j)}.$$

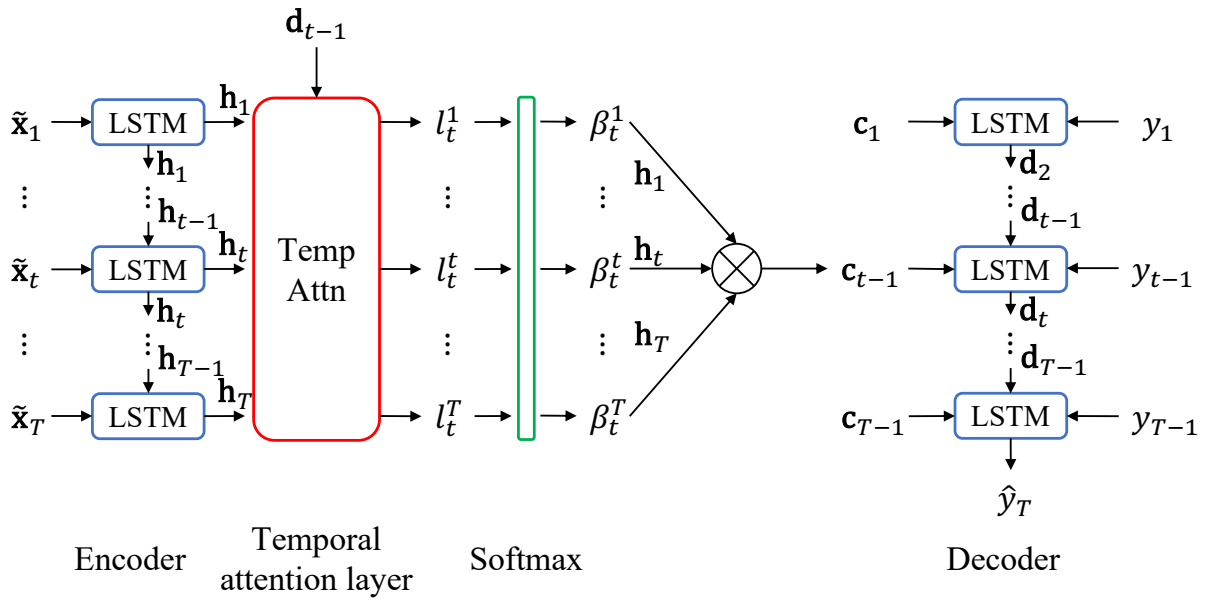


Figure 5.2: DA-RNN's Temporal Attention Mechanism.

Since β_t^i measures the importance of the i-th encoded hidden state $\mathbf{h}_i$ for the final prediction, the context vector $\mathbf{c}_t \in \mathbb{R}^m$ is then calculated as a weighted sum of $\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T\}$:

$$\mathbf{c}_t = \sum_{i=1}^{T} \beta_t^i \mathbf{h}_i.$$

The context vector should then be passed into an RNN by which the decoder's hidden state is calculated. In DA-RNN, this procedure is separated into two steps.

Firstly, the context vector and the target series are concatenated and mapped to the size of the decoder input $\tilde{y}_{t-1}$:

$$\tilde{y}_{t-1} = \tilde{\mathbf{w}}^\top [y_{t-1}, \mathbf{c}_{t-1}] + \tilde{b},$$

where $\tilde{\mathbf{w}} \in \mathbb{R}^{m+1}$ and $\tilde{b} \in \mathbb{R}$ are parameters to learn.

Secondly, the decoder input $\tilde{y}_{t-1}$ is employed to update the decoder hidden state $\mathbf{d}_t$:

$$\mathbf{d}_t = f_2(\mathbf{d}_{t-1}, \tilde{y}_{t-1}),$$

where $f_2(\cdot)$ is the other LSTM.

Once the current decoder's hidden state is computed, the model can output the final prediction. In DA-RNN, two linear functions are exploited to calculate $\hat{y}_T$:

$$\hat{y}_T = \mathbf{v}_y^\top (\mathbf{W}_y [\mathbf{d}_T, \mathbf{c}_t] + \mathbf{b}_w) + b_v,$$

where $\mathbf{v}_y \in \mathbb{R}^p$, $\mathbf{W}_y \in \mathbb{R}^{p \times (p+m)}$, $\mathbf{b}_w \in \mathbb{R}^p$, and $b_v \in \mathbb{R}$ are parameters to learn.

To summarize, DA-RNN includes two attention mechanisms at both the encoder and the decoder phases. The input attention mechanism measures the importance of the k-th input series at time t, and the temporal attention layer measures the importance

of the i-th hidden state from the encoder output. This dual-stage attention mechanism enables the model to capture the interdependencies among the relevant series, as well as the long-term dependencies at the same time. Although DA-RNN was initially designed for forecasting one target time series, it is easy to extend it for multiple target series by simply substituting y_{t-1} and $\tilde{y}_{t-1}$ with their multidimensional versions $\mathbf{y}_{t-1}$ and $\tilde{\mathbf{y}}_{t-1}$.

LSTNet

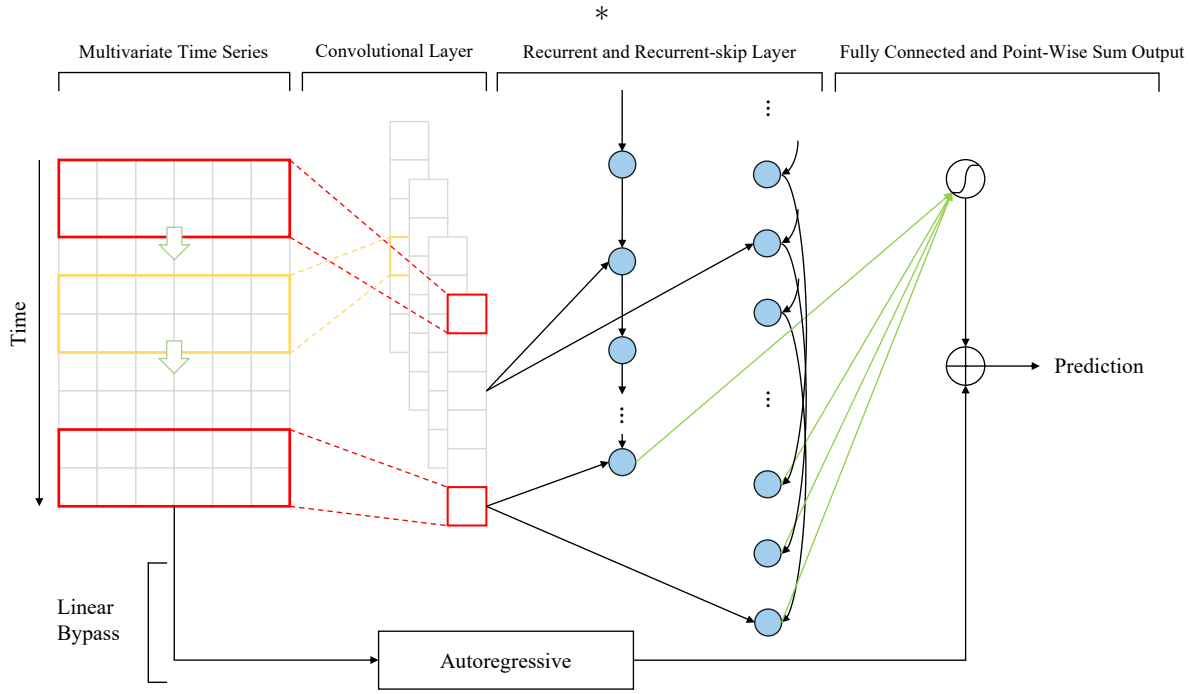


Figure 5.3: Graphical illustration of LSTNet.

LSTNet has attracted much attention since its first appearance at the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR) in 2018, proposed by Lai *et al.* [55]. LSTNet is not under an encoder-decoder structure, but by combining CNN and RNN, LSTNet generates relatively good results for MTS data. Unlike DA-RNN, LSTNet does not distinguish the driving and target series. It generates forecasting for all series entries.

In LSTNet, the n-dimensional MTS of length T is noted as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$ where $\mathbf{x}_t \in \mathbb{R}^n$. The object for an h-step forecasting is thus $\{\mathbf{x}_{T+1}, \mathbf{x}_{T+2}, ..., \mathbf{x}_{T+h}\}$. The input matrix at time t in a w-length window is formulated as $\mathbf{I}_t = \{\mathbf{x}_{t-w+1}, \mathbf{x}_{t-w+2}, ..., \mathbf{x}_t, \} \in \mathbb{R}^{n \times w}$.

The author designed four essential components inside LSTNet to ensure its forecasting effectiveness.

Convolutional Layer The first layer of LSTNet is a 1D convolutional layer of d_c kernels. Each convolutional kernel is of size $\omega \times n$ where ω is the kernel size, and n represents the dimension of the MTS data. The k-th kernel with ReLU activation function convolves

through the input matrix $\mathbf{I}_t$ and generates:

$$h_k^C = \text{ReLU}(W_k \circledast \mathbf{I}_t + b_k),$$

where $\circledast$ is the convolution operation and W_k and b_k are parameters to learn. The whole output matrix of the convolutional layer H^C is of size $d_C \times T$. In practice, this 1D convolutional layer can be implemented with either a 1D or a 2D convolutional function with the proper kernel size, padding, and stride parameter.

After the convolution, the output matrix is passed into two recurrent layers.

Recurrent Layer The first recurrent layer is a regular Gated Recurrent Unit (GRU) [106] layer but with a ReLU activation function for each neuron. It is similar to LSTM but simpler, where the hidden state is computed as:

$$\text{Reset gate} \qquad r_t = \sigma(W_r[h_{t-1}, x_t] + b_r), \tag{5.2}$$

$$\text{Update gate} \qquad u_t = \sigma(W_u[h_{t-1}, x_t] + b_u), \tag{5.3}$$

$$\text{New gate} \qquad n_t = \text{ReLU}(W_{xn}x_t + r_t \odot (W_{hn}h_{t-1}) + b_n), \tag{5.4}$$

$$\text{Hidden state} \qquad h_t = (1 - u_t) \odot h_{t-1} + u_t \odot n_t, \tag{5.5}$$

where $\odot$ is the Hadamard product, W_r, W_u, W_{xn}, W_{hn}, b_r, b_u, and b_n are weights and biases to learn. The final output of this recurrent layer is denoted by h_t^R.

Recurrent-skip Layer The second recurrent layer is an elaborate *Recurrent-skip Component*. It is designed to address the vanishing gradient problem for very long-term sequences and also performs an alignment for series seasonality.

The recurrent-skip component is developed with temporal skip connections to extend the span of the information flow. It adds *skip-links* from the current hidden states to the hidden states in adjacent periods.

$$\text{Reset gate} \qquad r_t = \sigma(W_r[h_{t-p}, x_t] + b_r),$$

$$\text{Update gate} \qquad u_t = \sigma(W_u[h_{t-p}, x_t] + b_u),$$

$$\text{New gate} \qquad n_t = \text{ReLU}(W_{xn}x_t + r_t \odot (W_{hn}h_{t-p}) + b_n),$$

$$\text{Hidden state} \qquad h_t = (1 - u_t) \odot h_{t-p} + u_t \odot n_t,$$

where parameters share the same notations with those in (5.2)-(5.5), they are different weights and biases. h_{t-p} is the hidden state in the adjacent period of p and p is thus the number of hidden cells skipped through. p's value is easy to determine for series with a simple and clear seasonality, e.g., 12 for monthly data and 24 for hourly data.

In practice, this recurrent-skip layer is a standard GRU layer rather than an explicitly redesigned one. The input of this layer is the output of the convolutional layer, the same as the previous recurrent layer, but rearranged to have a shape of $p \times n_p$ for each dimension, where n_p is the number of seasons in one input window. Every entry in the input matrix represents values at the same position in different periods.

In this way, skip-links allow LSTNet to look at the information in adjacent periods and thus enable the information to flow inside the RNN.

After the Recurrent and Recurrent-skip layers, the computation results are bundled by a fully connected layer. The Recurrent-skip layer output is h_i^S where $i \in [0, p-1]$. The output of the fully connected layer is thus:

$$h_t^D = W^R h_t^R + \sum_{i=0}^{p-1} W_t^S h_{t-i}^S + b,$$

where W^R, W_t^S, and b are parameters to learn. h_t^D is the output of the upper part of the neural network in Fig. 5.3 at time t.

Autoregressive Layer To address the scale-changing problem, where the output scale does not vary with the input, the author adds an autoregressive layer to insert linearity into the model. The autoregressive layer is implemented by another fully connected layer. The output of the i-th series is computed by:

$$h_{t,i}^L = \sum_{k=0}^{q^{\text{ar}}-1} W_k^{\text{ar}} \mathbf{x}_{t-k,i} + b^{\text{ar}},$$

where $W_k^{\text{ar}} \in \mathbb{R}^{q^{\text{ar}}}$ and $b^{\text{ar}} \in \mathbb{R}^n$ are parameters to learn and shared by all dimensions. q^{ar} is the input window length.

The final forecasting of LSTNet at time step t is an integration of the neural network and the autoregressive layer:

$$\hat{\mathbf{X}}_t = h_t^D + h_t^L.$$

To summarize, LSTNet leverages CNN and RNN for the MTSF problem. Firstly, LSTNet convolves on the preprocessed series to capture the interdependencies among multivariate series. Then, a GRU is stacked to model the long-term dependencies. An elaborate Recurrent-Skip component is devised to address the gradient vanishing problem for very long-term sequences, which allows the neurons to look at the hidden state in adjacent periods. A fully connected layer then integrates the outputs of the Recurrent layer and the Recurrent-skip layer. The authors also adopt an autoregressive component to deal with the violate scale changing in the series.

TPA-LSTM

Shih, Sun, and Lee [56] proposed TPA-LSTM in 2019. It ameliorates MTS forecasting by leveraging CNN, RNN, and an attention mechanism and focusing on different time steps for different time series, which contributes to a different idea from the aforementioned DA-RNN and LSTNet. TPA-LSTM has four different layers, as shown in Fig. 5.4.

Recurrent Layer The first component in TPA-LSTM is an LSTM layer that extracts the hidden state matrix. Given an n-dimensional MTS $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$ where $\mathbf{x}_t \in \mathbb{R}^n$, an LSTM layer $f(\cdot)$ with m hidden neurons calculates the hidden and cell states at time step t by:

$$h_t, c_t = f(h_{t-1}, c_{t-1}, \mathbf{x}_t),$$

where $h_t, c_t \in \mathbb{R}^m$. Given the historical hidden state matrix $H = \{h_1, h_2, ..., h_{t-1}\} \in \mathbb{R}^{m \times (t-1)}$, the LSTM layer generates the hidden state vector h_t at time step t. Once the

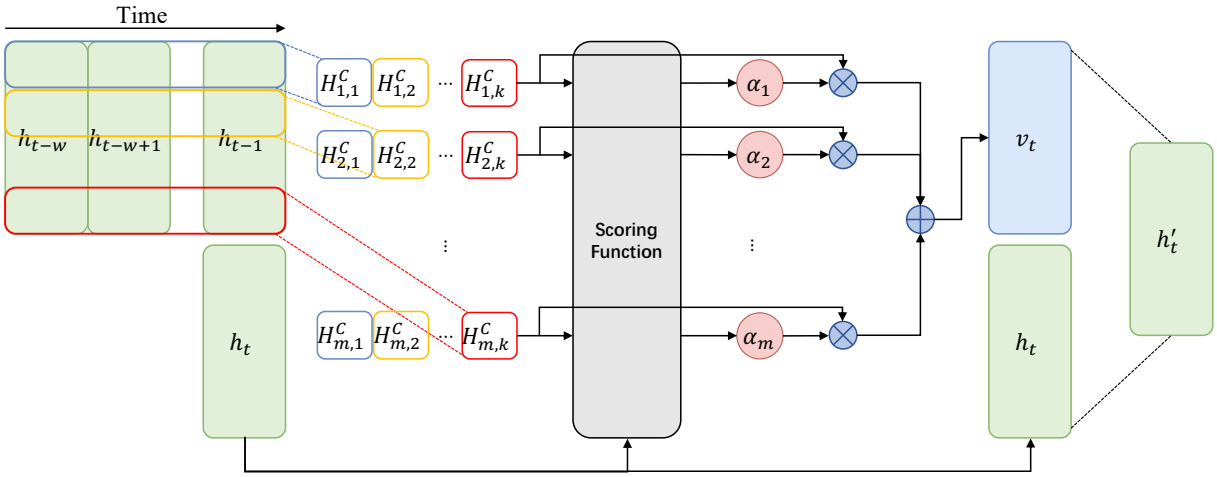


Figure 5.4: Graphical illustration of TPA-LSTM.

hidden state matrix H is acquired by the recurrent layer, it is passed into the following convolutional layer.

Convolutional Layer A CNN layer convolves its kernels on the row vectors of H to enhance the learning ability by detecting temporal patterns inside each series. Specifically, k kernels C_i of size $1 \times w$ are exploited, where w is the sliding window length. The i-th row of the convolutional results matrix generated by the j-th kernel is formularized as:

$$H^C_{i,j} = \sum_{l=1}^{w} H_{i,(t-w-1+l)} \circledast C_{j,l}, \quad 1 \leq i \leq m,$$

where $\circledast$ denotes the convolution operation and $H^C \in \mathbb{R}^{n \times k}$ is the convolutional result. It should be noted that the convolutional layer only operates on the historical hidden state matrix $H = \{h_1, h_2, ..., h_{t-1}\} \in \mathbb{R}^{m \times (t-1)}$. The current hidden state h_t is not involved. Every row in H^C represents the temporal pattern of the corresponding series.

Attention Layer The scoring function to measure the relevance in Fig. 5.4 is formularized as $g : \mathbb{R}^k \times \mathbb{R}^m \mapsto \mathbb{R}$:

$$g(H^c_i, h_t) = (H^C_i)^\top W_a h_t,$$

where $H^c_i \in \mathbb{R}^k$ is the i-th row vector of H^c and $W_a \in \mathbb{R}^{k \times m}$ is the parameter to learn. This step includes the current hidden state $h_t \in \mathbb{R}^m$ calculated by the recurrent layer. The attention weights α_i is then calculate by:

$$\alpha_i = \sigma(g(H^C_i, h_t)).$$

Note that here the author used the Sigmoid function instead of the Softmax to calculate the attention weights to benefit from letting more variables participate in forecasting. But this turns out to be no significant improvement in their ablation study, notably for the non-polyphonic time series datasets.

Once the attention weights are calculated, the row vectors of H^c are weighted to obtain the context vector $v_t \in \mathbb{R}^k$:

$$v_t = \sum_{i=1}^{m} \alpha_i H_i^C.$$

Then the context vector and the current hidden state are integrated:

$$h_t^D = W_h^D(W_h h_t + W_v v_t),$$

where W_h, W_h^D, and W_v are weights to learn.

Autoregressive Layer The author also leverages an autoregressive layer as per LSTNet for the scale-changing problem. The output of the i-th series is computed by:

$$h_{t,i}^L = \sum_{k=0}^{q^{\text{ar}}-1} W_k^{\text{ar}} \mathbf{x}_{t-k,i} + b^{\text{ar}},$$

The final result is the combination of h_t^D and h_t^L:

$$\hat{\mathbf{X}}_t = h_t^D + h_t^L.$$

To summarize, TPA-LSTM uses LSTM to deal with the preprocessed series to extract a hidden state matrix whose rows and columns represent the corresponding series and time steps. A CNN layer detects the temporal patterns of every series by convolving the kernel with the row vector of the hidden state matrix. After that, an attention layer is applied. It calculates the corresponding attention weights using the sigmoid function and generates the context vector. Finally, combining the results from an autoregressive module as per LSTNet, the model integrates the hidden state and the context vector to yield the final forecasting.

5.3 Experiment

The novelty of this work is that we combined the three deep learning models with the five multi-step strategies mentioned earlier to perform real multi-step forecasting.

5.3.1 Datasets

In our experiments, six datasets are selected to evaluate these deep learning models for MTS multi-step forecasting. The statistics of these datasets are listed in Tab. 5.1.

- Electricity[1]: Hourly electricity consumption of 321 clients from 2012 to 2014. Series in this dataset are complex seasonal.
- Exchange Rate[2]: Daily exchange rates of eight countries, i.e., Australia, British, Canada, China, Japan, New Zealand, Singapore, and Switzerland, from 1990 to 2016. Series in this dataset are nonseasonal.

[1] `https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`
[2] `https://github.com/laiguokun/multivariate-time-series-data`

- NASDAQ 100 Stock Data[3]: Stock price by minute from July 26 to December 22, 2016, of 81 corporations under NASDAQ 100 which are used as the driving series and the NASDAQ Index 100 used as target series. Series in this dataset are nonseasonal.
- Solar Energy[4]: 10-minute-level solar power production data from photovoltaic plants in Alabama State in 2006. Series in this dataset are seasonal.
- Traffic[5]: Hourly data from the California Department of Transportation describing the road occupancy rates on San Francisco Bay area freeways from 2015 to 2016. Seasonal data.
- Beijing $PM_{2.5}$ Data[6]: Hourly data of the $PM_{2.5}$ data of US Embassy in Beijing from 2010 to 2014, which is used as the driving and target series. Meteorological data from Beijing Capital International Airport are also included as driving series as well. Series in this dataset are seasonal.

If not specified, all the series in the datasets are harnessed as target series. We split our datasets into training, validation, and test sets in chronological order by the ratio of 8:1:1.

Table 5.1: Dataset Description.

Dataset	**Length**	**Dimension**	**Frequency**	**Seasonality**
Electricity	26304	321	1 hour	Complex
Exchange-Rate	7588	8	1 day	Nonseasonal
NASDAQ-100	40560	82	1 minute	Nonseasonal
Solar-Energy	52560	137	10 minutes	Seasonal
Traffic	17544	862	1 hour	Seasonal
Beijing-$PM_{2.5}$	43800	8	1 hour	Seasonal

5.3.2 Parameter Settings and Evaluation Metric

For simplicity, we took the same parameterization reported in [55] and [56] for LSTNet and TPA-LSTM on Electricity, Exchange-Rate, Solar-Energy, and Traffic. For DA-RNN, we followed the same parameter settings in [54] on NASDAQ-100. For other situations, the tunable parameters were selected based on the results from the validation set. The source codes of the aforementioned models are publicly available according to their papers.

Concretely, for LSTNet on NASDAQ-100 and Beijing $PM_{2.5}$, we set the window size w as 60 and 168, respectively. The periodicity pattern for *Recurrent-skip* was set to 30 and

[3] https://cseweb.ucsd.edu/~yaq007/NASDAQ100_stock_data.html
[4] https://www.nrel.gov/grid/solar-power-data.html
[5] https://pems.dot.ca.gov/
[6] https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data

24, and the AR components were both characterized to 24. The recurrent and convolution layer's hidden dimensions were set to 100, while the CNN kernel size was 6. Apart from the same parameterization of LSTNet, we set the number of the hidden state features to 12 on both NASDAQ-100 and Beijing $PM_{2.5}$ Dataset for TPA-LSTM.

For DA-RNN, on Beijing $PM_{2.5}$, we set the input window size to 10. On Electricity, Traffic, Exchange-Rate, and Solar-Energy, the window size was set to 24, 24, 10, and 144, respectively. Meanwhile, we fixed the encoder's and decoder's hidden dimensions to 64.

We performed a 128-minibatch training and a dropout after each layer as per LSTNet with a dropout rate of 0.2. The Adam optimizer [107] was used for all models with a learning rate of 0.0003. Furthermore, unlike the original normalization settings reported in LSTNet and TPA-LSTM, which causes information leakage, we normalized the training, validation, and test sets using the max-min values on their own.

We used the Root Relative Squared Error (RSE) as our evaluation metric with a slight difference from the one in [55], which concentrates more on the errors of each series:

$$\text{RSE} = \frac{1}{K}\sum_{i=1}^{K}\frac{\sqrt{\sum_{t=1}^{H}(y_{t,i}-\hat{y}_{t,i})^2}}{\sqrt{\sum_{t=1}^{H}(y_{t,i}-\overline{y_{1:H,i}})^2}}, \tag{5.6}$$

where H is the forecasting horizon, and K is the number of series in the datasets. y_t is the ground truth at time t. $\hat{y}_t$ is the forecast produced by the model, and $\overline{y}$ represents the mean of y.

5.4 Results and Discussions

We present our results in Tab. 5.2 with the best results highlighted in bold and the following contents.

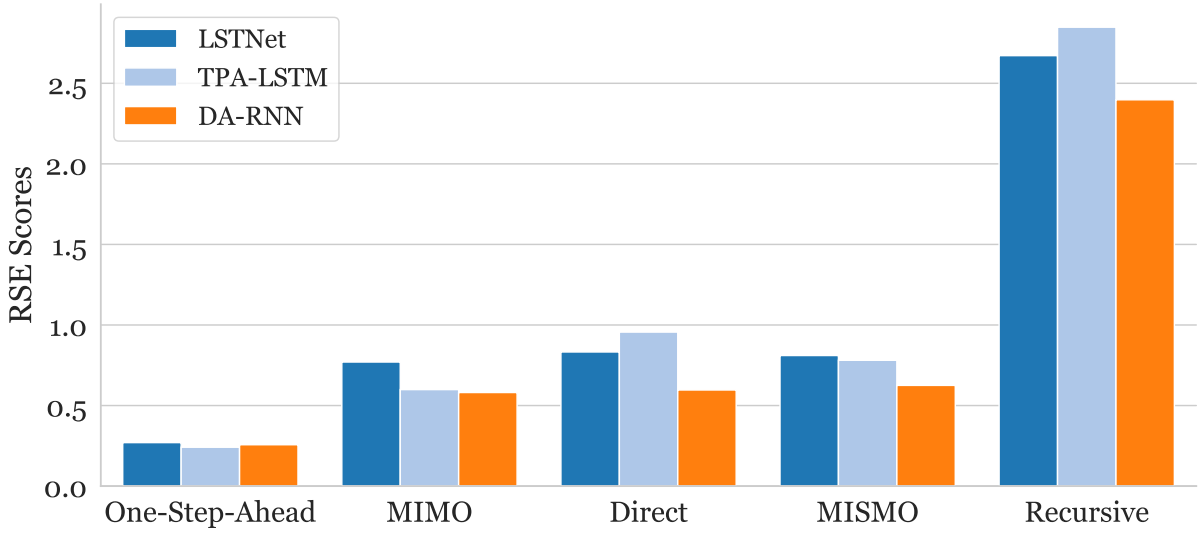


Figure 5.5: Average RSEs over the horizon for different strategies.

Table 5.2: Forecasting RSEs for different models on different forecast horizons with different strategies

Strategy		One-Step			MIMO			MISMO			Direct			Recursive		
Dataset	Horizon	LSTNet	TPA-LSTM	DA-RNN	LSTNet	TPA-LSTM	DA-RNN	LSTNet	TPA-LSTM	DA-RNN	LSTNet	TPA-LSTM	DA-RNN	LSTNet	TPA-LSTM	DA-RNN
Electricity (Complex)	3	0.0852	**0.0823**	0.0858	0.9020	**0.4310**	0.4748	1.1250	1.0426	**0.5466**	**1.0004**	1.0861	0.5565	**1.9427**	2.2879	2.3300
	6	0.0896	0.0920	**0.0882**	1.1232	0.5387	**0.5039**	1.1355	1.4672	**0.6498**	**1.0947**	1.2263	0.5769	**1.9981**	2.2502	2.2161
	12	0.0951	0.0945	**0.0923**	1.2349	0.6626	**0.5631**	1.4238	1.6061	**0.9293**	1.5452	**1.4134**	0.7658	2.8920	**2.7278**	2.7953
	24	0.1022	**0.1011**	0.1019	1.3857	0.9764	**0.7004**	1.6976	1.6916	**1.0344**	1.7132	1.6400	**1.1655**	3.7038	4.0929	**3.2050**
Exchange (Non seasonal)	3	0.0233	0.0184	**0.0173**	0.2469	**0.1224**	0.1248	0.1956	0.1320	**0.0970**	0.1611	0.2744	**0.0781**	1.5907	3.1972	**1.0445**
	6	0.0295	0.0244	**0.0233**	0.2929	0.1404	**0.1203**	0.2216	0.1456	**0.1066**	0.2231	0.3002	**0.1123**	1.6755	4.2372	**1.0606**
	12	0.0370	0.0342	**0.0338**	0.3984	0.1616	**0.1441**	0.3848	0.1595	**0.1276**	0.3759	0.3221	**0.1484**	2.0769	4.5510	**1.2025**
	24	0.0452	0.0452	**0.0429**	0.4023	0.1855	**0.1718**	0.4035	0.1894	**0.1623**	0.5369	0.3795	**0.1985**	2.2096	4.7817	**1.5694**
NASDAQ-101 (Non seasonal)	3	0.2580	**0.1266**	0.1301	0.9425	**0.4761**	0.4867	0.9560	**0.4520**	0.4968	1.0422	1.4679	**0.3322**	7.8994	5.4117	**5.2139**
	6	0.2618	**0.1327**	0.1480	0.8904	**0.4888**	0.5159	1.1064	0.6158	**0.5604**	1.0614	1.5486	**0.3452**	7.9054	6.7827	**6.7231**
	12	0.2915	**0.1493**	0.1505	0.9340	0.5289	**0.4683**	1.0191	**0.6704**	0.6961	1.0959	**1.5798**	0.4192	8.1166	**6.8759**	6.9342
	24	0.3266	**0.1622**	0.1627	1.0992	**0.5809**	0.6408	1.0772	**0.6962**	0.7122	1.1316	**1.8613**	0.5697	9.2435	**7.3209**	8.7724
Solar (Seasonal)	3	0.1900	0.1815	**0.1590**	0.2955	0.2723	**0.2502**	0.3328	0.3003	**0.2375**	0.2583	0.3097	**0.2231**	0.3940	0.3893	**0.3263**
	6	0.2601	0.2417	**0.2309**	0.3705	0.3363	**0.3299**	0.3330	0.3446	**0.3199**	0.3318	0.3808	**0.2949**	0.4707	0.5135	**0.4166**
	12	**0.3129**	0.3336	0.4233	**0.3630**	0.4950	0.4729	**0.4301**	0.4727	0.4660	**0.4384**	0.5095	0.4513	**0.5466**	0.6701	0.7990
	24	**0.4525**	0.4609	0.5752	**0.4450**	0.5033	0.6903	**0.6092**	0.7049	0.6848	**0.6587**	0.7015	0.7582	**0.7218**	0.7620	0.8921
Traffic (Seasonal)	3	0.4923	0.4609	**0.4348**	**0.7032**	0.7123	0.8329	**0.7777**	1.0408	0.8587	**0.7929**	0.9896	0.8284	**0.7417**	1.2104	1.1012
	6	0.5003	**0.4855**	0.5016	**0.7805**	0.8166	0.8849	**0.8390**	1.0890	0.9128	1.1102	**1.1072**	1.0031	1.2999	**1.2282**	1.3874
	12	0.5125	**0.4960**	0.6285	**0.8181**	1.1805	1.0014	1.1601	1.3395	**0.9963**	**1.2054**	1.1868	1.0544	**1.2037**	1.3951	1.5951
	24	0.5299	**0.5201**	0.5922	**0.9947**	1.2482	1.1108	1.2979	**1.1905**	1.2994	**1.2823**	1.3005	1.3125	1.3574	**1.2777**	1.7765
Beijing PM2.5 (Seasonal)	3	0.2868	**0.2691**	0.2722	0.5544	**0.4527**	0.4997	0.4416	0.5191	**0.4095**	0.4433	**0.5297**	0.4026	0.7581	**0.6940**	0.7004
	6	0.3533	0.3480	**0.3363**	0.7246	0.7192	**0.6203**	0.6509	0.6539	**0.6117**	0.5731	0.6560	**0.6243**	1.2652	1.2198	**1.0327**
	12	0.4418	**0.4332**	0.4333	1.0816	**0.9236**	0.9941	0.8122	0.9228	**0.7823**	0.8023	**0.8934**	0.8497	1.6615	**1.6505**	1.7788
	24	0.5019	**0.4972**	0.5001	1.4984	1.4232	**1.3388**	**1.0221**	1.2964	1.3063	**1.1022**	1.2779	1.2477	**2.4471**	2.8298	2.6719
Winning count		2	**12**	10	6	**7**	11	5	4	**15**	8	**6**	10	7	**7**	10

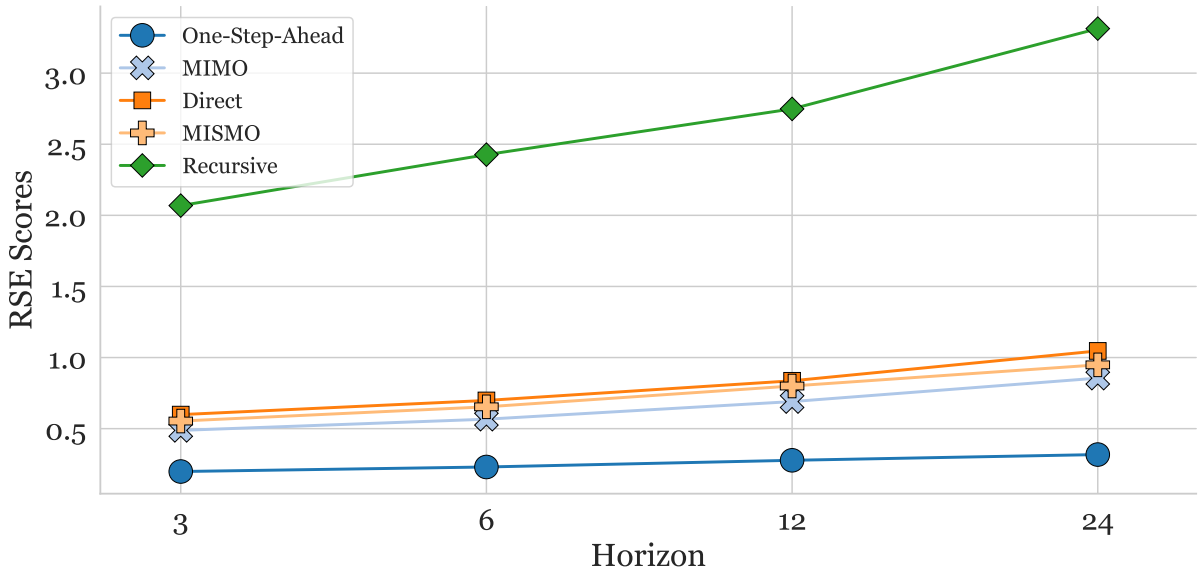


Figure 5.6: Average RSEs for different forecasting horizons.

Tab. 5.2 represents the RSEs of each model on six datasets with different seasonalities under five strategies, i.e., *One-Step-Ahead*, *Recursive*, *Direct*, *MIMO* and *MISMO*. The best results are printed in boldface.

The first question is whether the mentioned multi-step strategies can be applied to deep learning models for multi-step forecasting. Fig. 5.5 illustrates the average RSEs of different forecasting strategies. As the figure shows, the *Recursive* strategy performs the worst while the other strategies perform better with tolerable errors w.r.t. the *One-Step-Ahead* strategy. Less accumulated errors for the *Direct*, *MIMO*, and *MISMO* strategies are reported in Fig. 5.6 as the slopes of their curves at the middle are relatively smaller than that of the *Recursive* strategy on top.

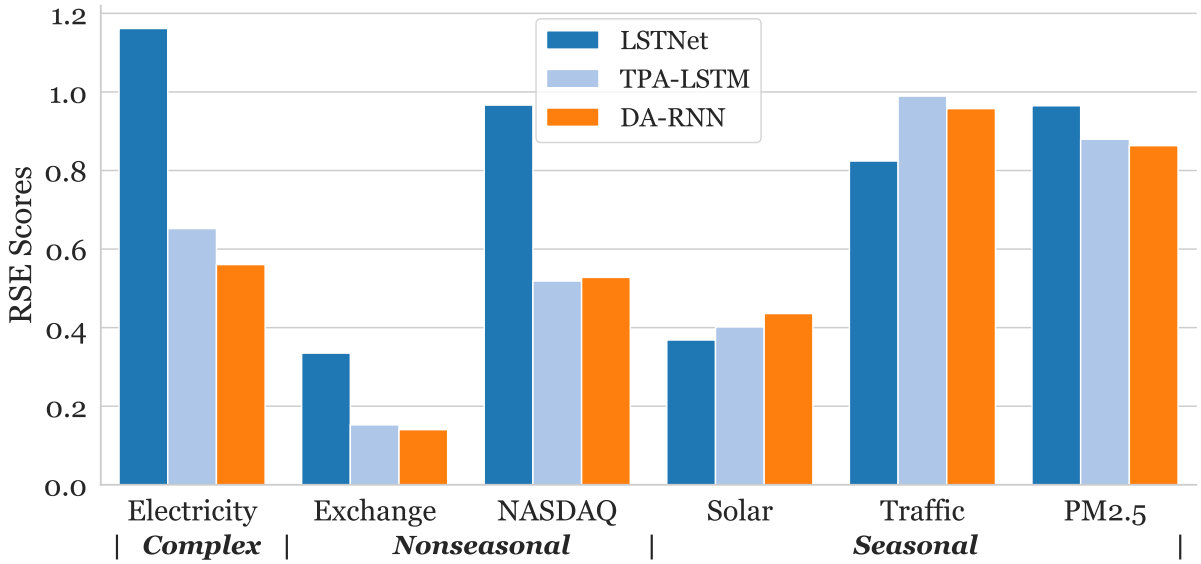


Figure 5.7: Average RSEs for different datasets in *MIMO* strategy.

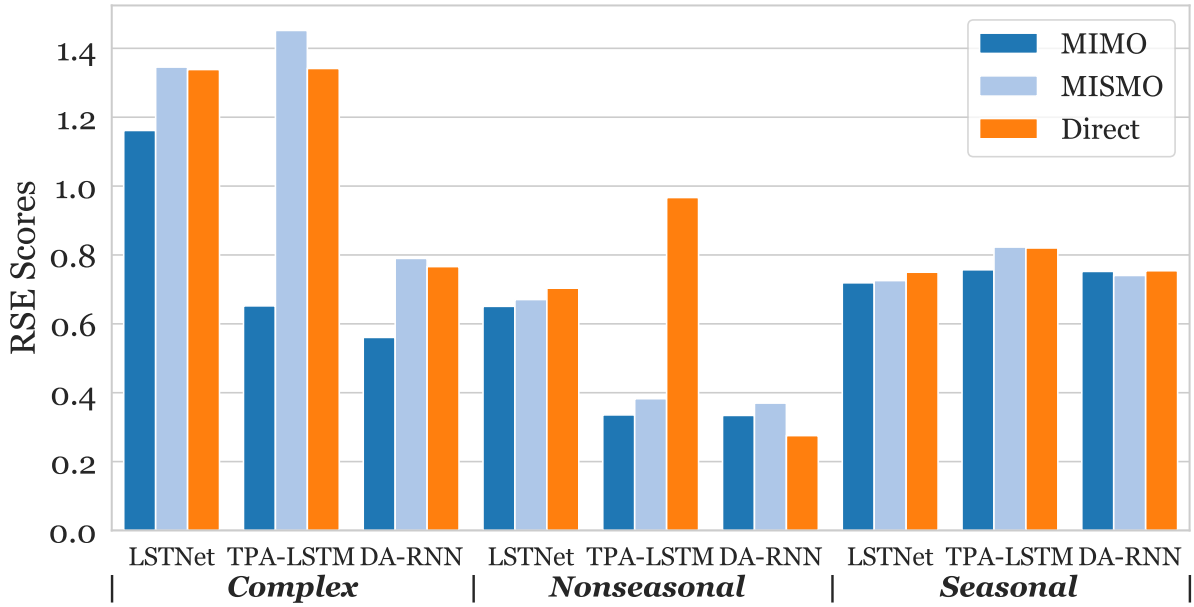


Figure 5.8: Average RSEs for different datasets in *MIMO*, *MISMO*, and *Direct* strategy.

Fig. 5.7 presents the average of RSEs over four horizons for different datasets in *MIMO* strategy. One obvious finding from Fig. 5.7 is that although all three models can capture the seasonal pattern, the performance of LSTNet falls behind TPA-LSTM and DA-RNN when facing series with zero or complex seasonalities. One explanation could be that the Recurrent-Skip component in LSTNet dedicated to capturing seasonal patterns is not applicable in this situation, as the periodicity pattern needs to be specified as a hyperparameter. Inversely, for seasonal data, periodicity specification is favorable for LSTNet in the case of Solar and Traffic, while less promising for Beijing $PM_{2.5}$ whose seasonality is less evident.

While TPA-LSTM uses a CNN to capture the temporal patterns, DA-RNN uses the attention mechanism in its decoder to put more importance on relevant time steps. So it is also interesting to note from Fig. 5.7 that DA-RNN performs slightly better in many cases than TPA-LSTM. This means the CNN component in TPA-LSTM is weaker than the attention mechanism in DA-RNN's decoder in capturing temporal patterns in long sequences.

Surprisingly, from Fig. 5.8, we noticed that using the MISMO or Direct strategy can potentially damage the model's performance, especially for TPS-LSTM. It shows massive error growth when combined with MISMO and Direct for series with complex seasonal and nonseasonal data. We attribute this to the break of seasonality's continuity brought by this kind of combination, damaging the model's intrinsic ability to capture complex seasonal patterns.

Furthermore, we noticed that a well-designed attention mechanism might help with the input scale variation. This accords with our observations that although DA-RNN does not include the AR component to respond to the changing scale, which LSTNet and TPA-LSTM both use, it still gives, in general, the best results.

5.5 Conclusions

This investigation aimed to determine whether the deep learning methods are suitable for dealing with the real MTS multi-step forecasting problem. The results gave a positive answer: by combining with the *MIMO/MISMO* strategy, deep learning models are competent to carry out real multi-step forecasting tasks. In the meantime, our experiments also revealed several interesting findings on their performances dealing with data seasonality. These could help us select the proper DL models for different tasks. Furthermore, other strategies dedicated to multi-step forecasting [94] are worthy of future research as well.

Chapter 6

Deep Learning Transformer-based Forecasting: Rankformer & STLformer

Contents

The challenge of time series forecasting has been the focus of research in recent years, with Transformer-based models using various self-attention mechanisms to uncover long-range dependencies. However, complex trends and nonlinear serial dependencies presented in some specific datasets may not always be captured properly. To address these issues, in this chapter, we propose a novel Transformer-based model, namely Rankformer, leveraging the rank correlation function and decomposition architecture for long-term time series forecasting tasks. We also present STLformer, an updated version of Rankformer that utilizes the STL decomposition architecture to improve forecasting performance. Rankformer and STLformer outperform four state-of-the-art Transformer and two RNN models across multiple datasets and forecasting horizons. This chapter can also be found in the corresponding conference version of papers [108], [109].

6.1 Introduction

TSF has been dominated for a few decades by econometric methods such as ARIMA, ETS, and Theta method [14], [32], [81], [110], [111]. In the past few years, deep learning has been applied to time series forecasting and achieved great success [48], [51], [112]. The most popular deep learning models for time series forecasting include Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer models. CNNs and RNNs have been widely exploited in forecasting tasks due to their ability to capture sequential/temporal dependencies in the time series [101]. Some representative works include LSTNet [55], DeepAR [113], and TCN [114].

Since its first birth in 2017, Transformer models have become increasingly popular and applied successfully in various fields, including machine translation, computer vision, and text generation, to list a few [63]–[66]. In the time series domain, Informer [72] was the first work that introduced Transformer for time series forecasting with a ProbSparse self-attention calculation and a self-attention distilling mechanism to handle the quadratic computational complexity. Autoformer [73] substitutes the self-attention block with an AutoCorrelation mechanism to discover the period-based dependencies and adopts a decomposition structure to separate the long-term stationary trend and the seasonal patterns. Other Transformer models were also applied to time series forecasting tasks, such as Reformer [115], which employed locally sensitive hashing self-attention, and LogTrans [69], which uses a heuristic method to reduce the complexity of the self-attention mechanism.

Nevertheless, the formerly mentioned Transformer models have not been able to exploit the long-range dependencies in time series fully, especially the nonlinear serial dependencies. Informer [72] applied the ProbSparse self-attention mechanism to reduce the computational complexity, but the hidden long-range dependency was not extracted properly. Autoformer [73] used the AutoCorrelation mechanism to discover the period-based dependencies. However, the AutoCorrelation used in the model is based on the Pearson correlation function, which supports only linear correlation, while in some time series, the long-term dependencies are nonlinear. Also, Autoformer relies solely on simple moving averages for decomposition, which may not accurately extract seasonal patterns, resulting in suboptimal modeling for seasonal and trend parts and final results.

This chapter presents Rankformer and STLformer, two novel Transformer-based models for long-term time series forecasting. Rankformer leverages the rank correlation function for dependency discovery, while STLformer is the first of its kind to incorporate the Seasonal-Trend decomposition using LOESS (LOcall Estimated Scatterplot Smoothing) architecture into the Transformer framework. By leveraging the Rank Correlation and the STL decomposition, our models are able to capture and model the trend and seasonal patterns present in certain time series more precisely, resulting in improved forecasting performance. Extensive experiments on four benchmark datasets demonstrate the superiority of Rankformer and STLformer over other Transformer models for four different forecasting horizons.

Overall, our contribution lies in the following aspects:

1. Introducing the Rank Correlation into the dependency discovery process.
2. Combining the Transformer architecture with STL decomposition.
3. Demonstrating their effectiveness for long-term time series forecasting.

The rest of the chapter is organized as follows. Sec. 6.2 introduces the proposed Rankformer and STLformer models. We then present the experimental setups and configurations in Sec. 6.3. The comparison and discussions based on the results are given in Sec. 6.4. Finally, Sec. 6.5 concludes the paper.

6.2 Methods

In this section, we introduce the architecture of Rankformer and STLformer, as well as their key components, i.e., Rank Correlation, Multi-Level Decomposition, and STL Decomposition modules.

6.2.1 Rankformer/STLformer Architecture

As shown in Fig. 6.1, Rankformer/STLformer is an encoder-decoder model per Autoformer [73]. For Rankformer, the encoder is composed of a stack of N identical layers, each containing one multi-head Rank Correlation (*RankCorr*) block, two Multi-Level Decomposition (*MLDecomp*) blocks, and one Feed-Forward (*FF*) block. The decoder is a stack of M identical layers, each of which is composed of two multi-head RankCorr blocks, three MLDecomp blocks, and one FF block. Combining the outputs of the last MLDecomp block and the refined trend-cyclical part in the decoder composes the final prediction. The STLformer substitutes the MLDecomp block with an STL Decomposition (*STLDecomp*) block. The architecture of Rankformer/STLformer is detailed in the following contents.

Encoder

With the RankCorr and ML/STLDecomp blocks, the encoder decomposes the series into seasonal and trend-cyclical parts. With the latter being neglected during the modeling process, the encoder mainly models the seasonal component. The output of the l-th encoder layer can be summarized as $\mathcal{X}_{\text{en}}^{l} = \text{Encoder}(\mathcal{X}_{\text{en}}^{l-1})$ and the process in one encoder layer is expressed as follows:

$$\begin{aligned}\mathcal{S}_{\text{en}}^{l,1}, \_ &= \text{ML/STLDecomp}\big(\text{RankCorr}(\mathcal{X}_{\text{en}}^{l-1}) + \mathcal{X}_{\text{en}}^{l-1}\big),\\ \mathcal{S}_{\text{en}}^{l,2}, \_ &= \text{ML/STLDecomp}\big(\text{FF}(\mathcal{S}_{\text{en}}^{l,1}) + \mathcal{S}_{\text{en}}^{l,1}\big),\end{aligned}$$

where $\mathcal{S}_{\text{en}}^{l,i}$ denotes the seasonal component after the i-th MLDecomp block and $\mathcal{X}_{\text{en}}^{l} = \mathcal{S}_{\text{en}}^{l,2},\ l \in \{1, 2, ..., N\}$.

Decoder

The decoder in Rankformer/STLformer has two streams, i.e., the trend-cyclical stream and the seasonal stream. While the seasonal stream continuously refines the seasonal part of the time series, the trend-cyclical stream focuses on modeling the trend-cyclical component. With a similar notation as per encoder, we can summarize the process in one

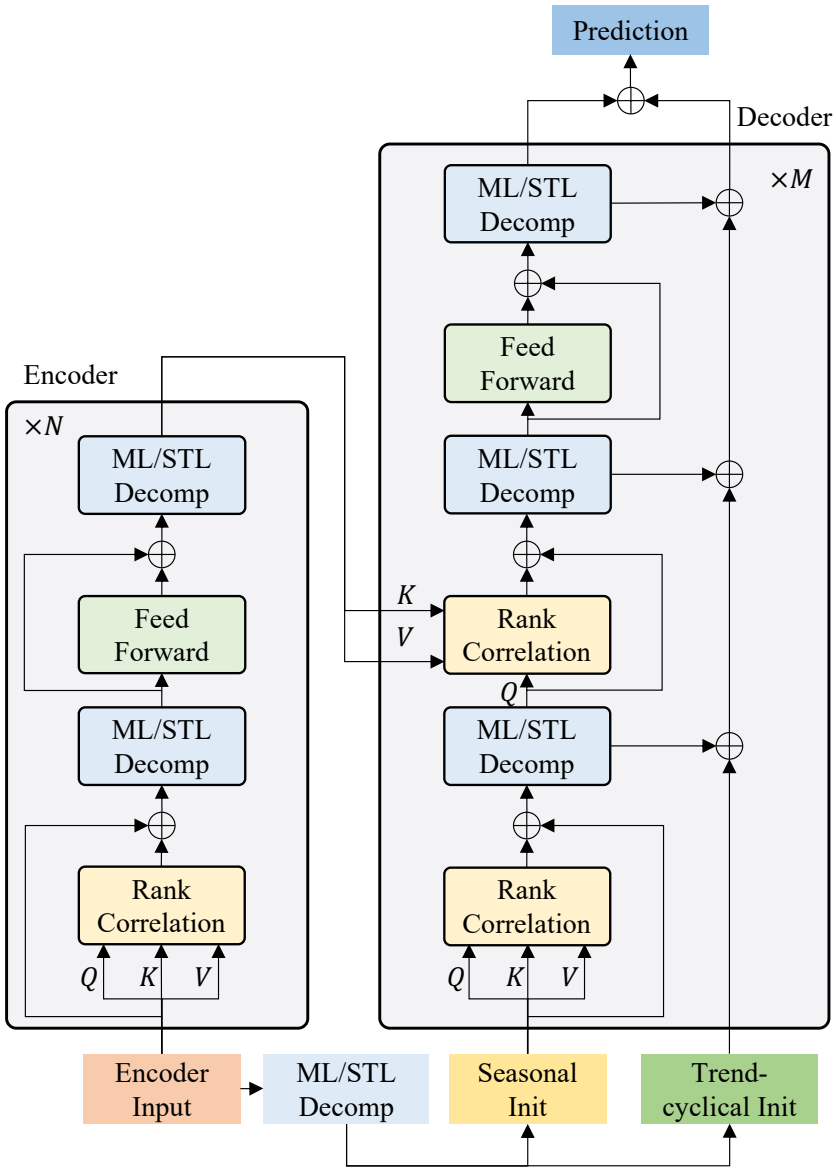


Figure 6.1: The architecture of Rankformer/STLformer.

decoder layer as $\mathcal{X}_{\text{de}}^{l}, \mathcal{T}_{\text{de}}^{l} = \text{Decoder}(\mathcal{X}_{\text{de}}^{l-1}, \mathcal{T}_{\text{de}}^{l-1})$ and formalize it as follows:

$$
\begin{aligned}
\mathcal{S}_{\text{de}}^{l,1}, \mathcal{T}_{\text{de}}^{l,1} &= \text{ML/STLDecomp}\Big(\text{RankCorr}(\mathcal{X}_{\text{de}}^{l-1}) + \mathcal{X}_{\text{de}}^{l-1}\Big), \\
\mathcal{S}_{\text{de}}^{l,2}, \mathcal{T}_{\text{de}}^{l,2} &= \text{ML/STLDecomp}\Big(\text{RankCorr}(\mathcal{S}_{\text{de}}^{l,1}, \mathcal{X}_{\text{en}}^{N}) + \mathcal{S}_{\text{de}}^{l,1}\Big), \\
\mathcal{S}_{\text{de}}^{l,3}, \mathcal{T}_{\text{de}}^{l,3} &= \text{ML/STLDecomp}\Big(\text{FF}(\mathcal{S}_{\text{de}}^{l,2}) + \mathcal{S}_{\text{de}}^{l,2}\Big), \\
\mathcal{T}_{\text{de}}^{l} &= \mathcal{T}_{\text{de}}^{l-1} + W_{l,1}\mathcal{T}_{\text{de}}^{l,1} + W_{l,2}\mathcal{T}_{\text{de}}^{l,2} + W_{l,3}\mathcal{T}_{\text{de}}^{l,3},
\end{aligned}
$$

where $\mathcal{S}_{\text{en}}^{l,i}$ and $\mathcal{T}_{\text{de}}^{l,i}$ are the seasonal and trend-cyclical components respectively, and $W_{l,1}$, $W_{l,2}$, $W_{l,3}$ are trainable weights. The outputs of the l-th decoder layer are two fold: the refined seasonal patterns $\mathcal{X}_{\text{de}}^{l} = \mathcal{S}_{\text{de}}^{l,3}$, and the multiple level trend-cyclical patterns $\mathcal{T}_{\text{de}}^{l}$, where $l \in \{1, 2, ..., M\}$.

Model Inputs and Outputs

We denote the input length as I, the output length as O, and the model dimension as d. There are three inputs for Rankformer:

- The encoder input are the last I time steps in the time series: $\mathcal{X}_{\text{en}} \in \mathbb{R}^{I \times d}$.
- The seasonal stream input concatenates the latter half of the encoder's decomposed input and a length-O placeholder with zeros: $\mathcal{X}_{\text{de,S}} = \text{concat}(\mathcal{X}_{\text{en,S}}, \mathcal{X}_0) \in \mathbb{R}^{(\frac{I}{2}+O) \times d}$.
- The trend-cyclical stream input also consists of the latter half of the decomposed $\mathcal{X}_{\text{en}}$ and a placeholder filled by the average of $\mathcal{X}_{\text{en}}$: $\mathcal{X}_{\text{de,T}} = \text{concat}(\mathcal{X}_{\text{en,T}}, \mathcal{X}_{\text{avg}}) \in \mathbb{R}^{(\frac{I}{2}+O) \times d}$.

The relationship between the inputs can be formalized as follows:

$$\begin{aligned} \mathcal{X}_{\text{en,S}}, \mathcal{X}_{\text{en,T}} &= \text{ML/STLDecomp}\left(\mathcal{X}_{\text{en}}\left[\frac{I}{2} : I\right]\right), \\ \mathcal{X}_{\text{de,S}} &= \text{concat}(\mathcal{X}_{\text{en,S}}, \mathcal{X}_0), \\ \mathcal{X}_{\text{de,T}} &= \text{concat}(\mathcal{X}_{\text{en,T}}, \mathcal{X}_{\text{avg}}). \end{aligned}$$

The final output of the model is a combination of the seasonal and the trend-cyclical streams in the decoder: $W_{\text{S}}\mathcal{X}_{\text{de}}^{M} + \mathcal{T}_{\text{de}}^{M}$, where W_{S} is a trainable weight to project $\mathcal{X}_{\text{de}}^{M}$ into the target dimension.

6.2.2 RankCorrelation Block

The Pearson correlation coefficient, also known as Pearson's ρ, is widely used to measure the linear correlation between two variables. Given two random variables X and Y, Pearson's ρ defined as follows:

$$\rho_p(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}}. \tag{6.1}$$

The AutoCorrelation Function (ACF) adopts the Pearson correlation function to measure the correlation between two distant time points in a stationary time series y_t:

$$\text{ACF}(k) = \rho_p(y_{t-k}, y_t) = \frac{\text{Cov}(y_{t-k}, y_t)}{\sqrt{\text{Var}(y_{t-k})\text{Var}(y_t)}},\ k = 0, 1, 2, ..., \forall t.$$

However, in some time series, the long-term dependencies are not linear. In this case, the nonlinear process can exhibit more complex autocorrelation structures than linear ones and thus result in an erroneous dependencies measurement based on Pearson's ρ. To address this issue, we propose to use the Rank Correlation Function (RCF), more generally known as Spearman's ρ [116], to measure the nonlinear correlation. Spearman's ρ is defined as follows:

$$\rho_s(X, Y) = \frac{\text{Cov}(R(X), R(Y))}{\sqrt{\text{Var}(R(X))\text{Var}(R(Y))}},$$

where $R(X)$ and $R(Y)$ are the ranks of X and Y in (6.1). ρ denotes the usual Pearson correlation coefficient but is applied to the rank variables, which is leveraged to compute the Ranked ACF ($RACF$). ρ_s is defined in $[-1, 1]$, where -1 indicates a perfect negative monotonic relationship, 0 indicates no monotonic relationship, and 1 indicates a perfect positive monotonic relationship. ρ_s is invariant to monotonic transformations of the variables and is robust to outliers. Therefore, it is more suitable for stationary time series with nonlinear serial dependencies. Our RACF is defined as:

$$\text{RACF}(k) = \rho_s(y_{t-k}, y_t),\ k = 0, 1, 2, ..., \forall t.$$

In our implementation, the RACF is computed by exploiting the FFT, which accelerates the Fourier transform to $\mathcal{O}(N \log N)$, and the Wiener-Khinchin theorem, which states that the ACF of a stationary time series can be computed by the Fourier transform of its power spectrum. The ranking procedure is supported by the `torchsort`[1] library, which offers an efficient $\mathcal{O}(N \log N)$ sorting operator [117]. Thus, the RACF is computed by ranking the time series and then computing the ACF of the ranked time series. The total computational complexity of calculating the RACF is thus $\mathcal{O}(N \log N)$.

6.2.3 Multi-Level Decomposition Block

In Rankformer, we adopted a multi-level decomposition block to decompose the input time series into the seasonal and trend-cyclical components. The block consists of multiple moving average (MMA) filters with varying kernel sizes to yield different trend-cyclical components. The MLDecomp block is formalized as follows:

$$\begin{aligned} \mathcal{X}_{\text{seasonal}}, \mathcal{X}_{\text{trend-cyclical}} &= \text{MLDecomp}\left(\mathcal{X}\right), \\ \mathcal{X}_{\text{trend-cyclical}} &= \sum_{k=1}^{K} W_{\text{decomp},k} \cdot \text{MMA}(\mathcal{X}_{\text{input}}, k), \\ \mathcal{X}_{\text{seasonal}} &= \mathcal{X} - \mathcal{X}_{\text{trend-cyclical}}, \end{aligned} \tag{6.2}$$

where K is a set of kernel sizes, $W_{\text{decomp},k}$ is a trainable weight tensor. MMA denotes multiple moving average filters, and $\mathcal{X}_{\text{trend-cyclical}}$ and $\mathcal{X}_{\text{seasonal}}$ denote the trend-cyclical and seasonal components, respectively. The output of the MLDecomp block is a weighted sum of the trend-cyclical components.

6.2.4 STL Decomposition Block

The key difference between STLformer and Rankformer is substituting the Multi-Level Decomposition block in Rankformer with the STL Decomposition block, which utilizes LOESS regression.

LOESS was first proposed by Cleveland [26]. It is a nonparametric robust locally weighted regression method for smoothing a scatterplot, $(x_i, y_i), i = 1, ..., n$, in which the fitted value at x_k is the value of a polynomial fit to the data using weighted least squares, where the weight for (x_i, y_i) is large if x_i is close to x_k and small if it is not. It splits the data into several small sections, performs weighted linear regressions on different sections,

[1] `https://github.com/teddykoker/torchsort`

and connects the center of these curves to form the complete regression curve. Specifically, LOESS is defined by the following sequence of operations for a given time series:

1. For one data point, often called the *focal point*, select k nearest points around it to form a local window. Every focal point has a corresponding local window.

2. Calculate the weights of every point in the window through a weight function W, which is conventionally a Tricube function as follows:

$$T(x) = \begin{cases} (1 - |x|^3)^3, & \text{for } |x| < 1, \\ 0, & \text{for } |x| \geq 1. \end{cases}$$

3. Fit a weighted linear regression in the window. For n focal points, we have n weighted linear regressions.

4. Connect the center points of the n weighted regressions to form the final fitted curve.

The time complexity of LOESS mainly involves traversing the entire dataset to select the k nearest points for each point to form the local window, which leads to an $\mathcal{O}(N^2)$ complexity. This issue can be resolved by using a k-d tree for acceleration, which can rapidly find the nearest neighbors of a data point. Implementing a k-d tree can reduce the time complexity of the LOESS algorithm to $\mathcal{O}(N \log N)$ [26].

In 1990, Cleveland *et al.* [25] proposed the famous STL decomposition method, which leverages LOESS to estimate the trend and seasonal components, contributing to a versatile and robust method for decomposing time series.

Our model adopted the idea of STL decomposition and implemented the k-d tree LOESS to decompose the input time series into seasonal and trend-cyclical components. The STLDecomp block employs the LOESS regression to fit a locally smoothed trend-cyclical component. We formalize the STLDecomp block as follows:

$$\begin{aligned} \mathcal{X}_{\text{seasonal}}, \mathcal{X}_{\text{trend-cyclical}} &= \text{STLDecomp}\left(\mathcal{X}\right), \\ \mathcal{X}_{\text{trend-cyclical}} &= \text{LOESS}\left(\mathcal{X}\right), \\ \mathcal{X}_{\text{seasonal}} &= \mathcal{X} - \mathcal{X}_{\text{trend-cyclical}}, \end{aligned}$$

where LOESS is the LOESS regression function. STLDecomp serves as an interchangeable block of MLDecomp in STLformer.

6.3 Experiments

This section presents our experimental settings and results.

6.3.1 Datasets

Rankformer and STLformer were tested with other state-of-the-art methods on four well-known datasets:

- Electricity Transformer Temperature (ETT)[2]: Oil temperature and six power load features recorded every 15 minutes from July 2016 to July 2018 in two Chinese counties. Seasonal data.
- Exchange-Rate[3]: Daily exchange rates of eight countries, i.e., Australia, British, Canada, China, Japan, New Zealand, Singapore, and Switzerland, from 1990 to 2016. Nonseasonal.
- Weather[4]: 10-minute level local climate data containing 21 meteorological features for 2020 collected by Max-Planck-Institut für Biogeochemie, Jena. Complex seasonal data.
- Influenza-Like Illness (ILI)[5]: Weekly ILI patients data from the U.S. Centers for Disease Control and Prevention between 2002 to 2021, containing the ratio of ILI patients and the total number of patients. Seasonal data.

All datasets were separated into train/validation/test sets in chronological order with a 7/1/2 split, except for the ETT dataset, which was split into 6/2/2, as per Autoformer [73] and Informer [72]. The datasets' statistics are listed in the first four rows of Tab. 6.1.

We also evaluated the significance of the nonlinearity in the serial dependencies by performing Engle's Lagrange Multiplier Test [118] on the four datasets. It assesses the significance of autoregressive conditional heteroskedasticity (ARCH) effects in a time series. A significant result reveals nonlinear serial dependencies in the series. The test results are listed in the last two rows of Tab. 6.1.

Table 6.1: Dataset Description

Dataset	ETT	Exchange	Weather	ILI
Length	69680	7588	52696	966
Dimension	7	8	21	7
Seasonality	Seasonal	Nonseasonal	Complex	Seasonal
Sampling Frequency	15 min	1 day	10 min	1 week
Engle's test p-value	$<$ 2.2e-16	$<$ 2.2e-16	$<$ 2.2e-16	0.8126
ARCH effect	Significant	Significant	Significant	Insignificant

6.3.2 Experimental Settings

For both Rankformer and STLformer, we kept the same number of encoder-decoder layers settings as Autoformer: two encoder layers and one decoder layer. Rankformer and

[2] https://github.com/zhouhaoyi/ETDataset
[3] https://github.com/laiguokun/multivariate-time-series-data
[4] https://www.bgc-jena.mpg.de/wetter/
[5] https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html

STLformer were trained using the Mean Square Error loss and the Adam optimizer [107] with an initial learning rate of 10^{-4}. The batch size was set to 32. Models were trained for ten epochs with a learning rate scheduler that reduces the learning rate by a factor of 0.5 when the validation loss plateaus. Both Rankformer and STLformer were implemented in PyTorch [119] and trained on a single NVIDIA Tesla V100 GPU.

6.4 Results and Discussions

We evaluated Rankformer and STLformer against the following state-of-the-art models: Autoformer [73], Informer [72], LogTrans [69], Reformer [115], LSTNet [55], and LSTM [120]. We used the Mean Square Error (MSE) and the Mean Absolute Error (MAE) as the evaluation metrics, and we fixed the input length to 36 for ILI and 96 for others as per Autoformer. The results are presented in Tab. 6.2. The best results are highlighted in bold, and the second-best results are highlighted with underscores.

6.4.1 Results of Rankformer

Overall, Rankformer outperforms the other methods on the ETT, Exchange-Rage, and Weather datasets and is slightly weaker than Autoformer on the ILI dataset. Particularly, under the Input-96-Output-96 setting, Rankformer yields **13.3%** MSE reduction on ETT and **17.5%** on Exchange-Rate, compared to Autoformer. It also outperforms Autoformer on the Weather dataset, but the difference is not significant.

The results on the Exchange-Rate dataset are particularly impressive. Despite the fact that Exchange-Rate is a very challenging dataset without any notable periodicity, Rankformer still gives the best improvement over Autoformer. We attribute this to the nonlinear serial dependencies in the dataset being captured more properly by Rankformer than by Autoformer.

On the contrary, due to the high linear correlation in the ILI dataset, Rankformer is not able to outperform Autoformer. In fact, the p-value of Engle's Lagrange Multiplier test of the ILI dataset is 0.8126 ($\gg 0.05$), which means that there are statistically significant linear serial dependencies inside the ILI series that can be handled more appropriately by Autoformer.

6.4.2 Results of STLformer

In general, STLformer achieved superior performance compared to the other methods on the ETT, Exchange-Rate, and Weather datasets while being slightly less accurate than Autoformer and Rankformer on the ILI dataset. STLformer also performed slightly better than Autoformer on the Weather dataset but also marginally poorer than Rankformer.

On the Exchange-Rate dataset, despite the absence of any significant periodicity in the dataset, STLformer still achieved the best performance boost on average for both Rankformer (**4.84%**) and Autoformer (**21.45%**). We attribute this to STLformer's STL Decomposition Block being better at extracting the trend and seasonal patterns, thus resulting in better handling of the dataset's nonlinear serial dependencies. STLformer also achieved decent improvements on the ETT dataset, with an average boost of 4.80% for Rankformer and 9.74% for Autoformer, compared to their respective performances

Table 6.2: Forecasting results for different models on different forecast horizons

Models		**STLformer** [109]		**Rankformer** [108]		Autoformer [73]		Informer [72]		LogTrans [69]		Reformer [115]		LSTNet [55]		LSTM [120]	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETT	96	**0.209**	**0.298**	0.221	0.302	0.255	0.339	0.365	0.453	0.768	0.642	0.658	0.619	3.142	1.365	2.041	1.073
	192	**0.268**	**0.330**	0.275	0.333	0.281	0.340	0.533	0.563	0.989	0.757	1.078	0.827	3.154	1.369	2.249	1.112
	336	**0.339**	0.373	0.342	0.377	**0.339**	**0.372**	1.363	0.887	1.334	0.872	1.549	0.972	3.160	1.369	2.568	1.238
	720	**0.414**	**0.415**	0.419	0.416	0.422	0.419	3.379	1.388	3.048	1.328	2.631	1.242	3.171	1.368	2.720	1.287
Exchange	96	**0.151**	**0.279**	0.162	0.290	0.197	0.323	0.847	0.752	0.968	0.812	1.065	0.829	1.551	1.058	1.453	1.049
	192	**0.238**	**0.356**	0.251	0.365	0.300	0.369	1.204	0.895	1.040	0.851	1.188	0.906	1.477	1.028	1.846	1.179
	336	**0.419**	**0.478**	0.428	0.486	0.509	0.524	1.672	1.036	1.659	1.081	1.357	0.976	1.507	1.031	2.136	1.231
	720	**1.098**	**0.813**	1.157	0.837	1.447	0.941	2.478	1.310	1.941	1.127	1.510	1.016	2.285	1.243	2.984	1.427
Weather	96	0.264	0.333	**0.263**	**0.332**	0.266	0.336	0.300	0.384	0.458	0.490	0.689	0.596	0.594	0.587	0.369	0.406
	192	0.310	0.365	**0.298**	**0.356**	0.307	0.367	0.598	0.544	0.658	0.589	0.752	0.638	0.560	0.565	0.416	0.435
	336	0.350	0.394	**0.350**	**0.390**	0.359	0.395	0.578	0.523	0.797	0.652	0.639	0.596	0.597	0.587	0.455	0.454
	720	0.433	0.440	0.430	0.435	**0.419**	**0.428**	1.059	0.741	0.869	0.675	1.130	0.792	0.618	0.599	0.535	0.520
ILI	24	3.690	1.353	3.556	1.319	**3.483**	**1.287**	5.764	1.677	4.480	1.444	4.400	1.382	6.026	1.770	5.914	1.734
	36	3.012	1.133	**2.821**	**1.112**	3.103	1.148	4.755	1.467	4.799	1.467	4.783	1.448	5.340	1.668	6.631	1.845
	48	3.134	1.198	2.907	1.144	**2.669**	**1.085**	4.763	1.469	4.800	1.468	4.832	1.465	6.080	1.787	6.736	1.857
	60	3.531	1.308	3.232	1.239	**2.770**	**1.125**	5.264	1.564	5.278	1.560	4.882	1.483	5.548	1.720	6.870	1.879

with their original decomposition blocks. This is less than the performance boost on the Exchange-Rate dataset. We observed that the ETT dataset has a significant seasonality, where a standard moving average can effectively separate the trend and seasonal patterns. At the same time, Exchange-Rate is nonseasonal, which requires a more sophisticated decomposition method to help the model focus on modeling different patterns. This explains why STLformer achieved a better performance boost on the Exchange-Rate dataset.

For the Weather dataset, STLformer outperformed Autoformer but not Rankformer. We think as the Weather dataset exhibits a complex seasonality, Rankformer's Multi-Level Decomposition block is more suitable for the dataset than STLformer's STL-based decomposition block. For ILI, both STLformer and Rankformer filed to outperform Autoformer, and there is no performance boost from STL decomposition. The p-value of Engle's Lagrange Multiplier test on the ILI dataset was 0.8126, indicating statistically significant linear serial dependencies that can be handled more appropriately by the AutoCorrelation Block adopted in Autoformer.

6.4.3 Complexity Analysis and Model Comparison

Thanks to the FFT and Wiener-Khinchin theorem, Rankformer achieves an $\mathcal{O}(N \log N)$ complexity. It is not only a huge advantage in the computing speed compared to the original Transformer's $\mathcal{O}(N^2)$ complexity but also brings the convenience of nonlinear serial dependencies measurement to Autoformer without augmenting the time complexity. As for STLformer, we adopted a k-d tree implementation for the LOESS regression, which can be done in $O(N \log N)$ time. These advantages make Rankformer and STLformer much more efficient than the Transformer, especially when dealing with long sequences, and also more appropriate for forecasting time series with nonlinear serial dependencies.

Rankformer, STLformer, and Autoformer are very similar in terms of their measurement of correlation. The only difference is that Rankformer/STLformer uses a rank-based ACF, i.e., the RACF, while Autoformer uses a value-based ACF, which means, with an optimized sorting and ranking operator, the RACF can be easily integrated into Autoformer and enables it for nonlinear time dependencies measurement.

In terms of the trend and seasonal patterns discovery, STLformer employed a more sophisticated decomposition method, i.e., STL decomposition, while Rankformer and Autoformer adopted a multi-level moving average decomposition. We think our STL Decomposition Block could be preferred for most datasets, while the Multi-Level Decomposition Block is more suitable for datasets with complex seasonality.

6.5 Conclusion

In this chapter, we propose a novel method, Rankformer, for forecasting time series, especially those with nonlinear serial dependencies, and an updated version called STLformer regarding the decomposition structure. Rankformer is based on the Transformer architecture and uses a rank-based ACF to measure the nonlinear serial dependencies, while STLformer leverages the STL decomposition to extract trend and seasonal patterns better. We show that Rankformer outperforms Autoformer, a state-of-the-art method with linear serial dependencies measurement, on three real-world datasets. At the same time,

experiments on two datasets demonstrate that STLformer performs better than the previous Rankformer, which solely corporates moving averages for decomposition, proving the effectiveness of the STL decomposition on certain datasets. We also show that Rankformer and STLformer are more efficient than the original Transformer in terms of both computing speed and memory usage. Furthermore, we show that the RACF and STL Decomposition Block can be easily integrated into other Transformer-based models to improve their performance. In fact, in most nonlinear serial dependencies cases, the series shows an ARCH effect, which can be captured by the RACF. In future work, we plan to explore the robust LOESS for the presence of perturbations and a multi-level LOESS for complex seasonality. We would also like to investigate the possibility of integrating other nonlinear serial dependencies measurement methods, such as Generalized ARCH (GARCH), to further enhance the performance of Rankformer and STLformer.

Chapter 7

A Web Application Prototype

Contents

In this chapter, we present the prototype developed for ATTILA's prediction project. The prototype is a web application that allows the user to upload a dataset and select a forecasting model. The application then generates the forecasted values, renders them in the browser, and calculates the corresponding error metrics.

In Sec. 7.1, we explain the motivation behind the development of the prototype and the requirements demanded. Sec. 7.2 presents in detail the prototype in terms of its user interface, technical architecture, core functionalities, and several different aspects. Finally, Sec. 7.3 concludes the chapter and presents the future work.

7.1 Introduction

The main objective behind the prototype is to provide an application that allows the user, mainly the managers of an agency, to easily perform forecasting on their agencies' data, in order to obtain an overview of the future performance and can then help them better manage the agency. The prototype is also a proof of concept for the ATTILA project.

Specifically, the application needs to provide the user with the following functionalities:

- **Upload a dataset.** The user should be able to upload a dataset to the application.
- **Select an agency.** The user should be able to select an agency from the uploaded dataset.

- **Select a forecasting model.** The user should be able to select a forecasting model from a list of available models.
- **Select a forecasting mode.** The user should be able to choose whether to perform training or a prediction.
- **Select a forecasting horizon.** The user should be able to select the forecasting horizon.
- **Visualize the forecasted values.** The application should be able to visualize the forecasted values in the browser.
- **Calculate the error metrics.** The application should be able to calculate the error metrics for the forecasted values.
- **Download the forecasted values.** The user should be able to download the forecasted values in an Excel file.

The developed prototype is a web application that fulfills the aforementioned requirements. In the following sections, this web application will be presented in detail.

7.2 The Prototype: A Web Application

7.2.1 User Interface

The user interface of the prototype has three pages:

- **Home page.** The home page is the first page that the user sees when accessing the application. It contains a brief introduction to the application and two buttons to upload the required dataset files. It also allows the user to select the agency and the indicator to be forecasted, with two toggleable dropdowns. The home page is shown in Fig. 7.1.
- **Statistics page.** The Statistics page is the page that the user sees after uploading the dataset files. It contains a table that shows the statistics of the uploaded dataset, including the name of the selected agency, the name of the selected indicator, and the number of data points as well as the final validated date. It also allows the user to select the forecasting model, the forecasting mode (`training` or `prediction`), and the forecasting horizon. The Statistics page is demonstrated in Fig. 7.2.
- **Result page.** The user sees the result page after the forecasting is done. It contains an interactive chart that demonstrates the forecasted values and the actual values, as well as the prediction interval. This chart allows the user to hover, zoom, and pan the data points. If the user has selected the `training` mode, it will also display the forecasting error. This page also provides a button that allows the user to download the forecasted values in an Excel file. The result page is shown in Fig. 7.3.

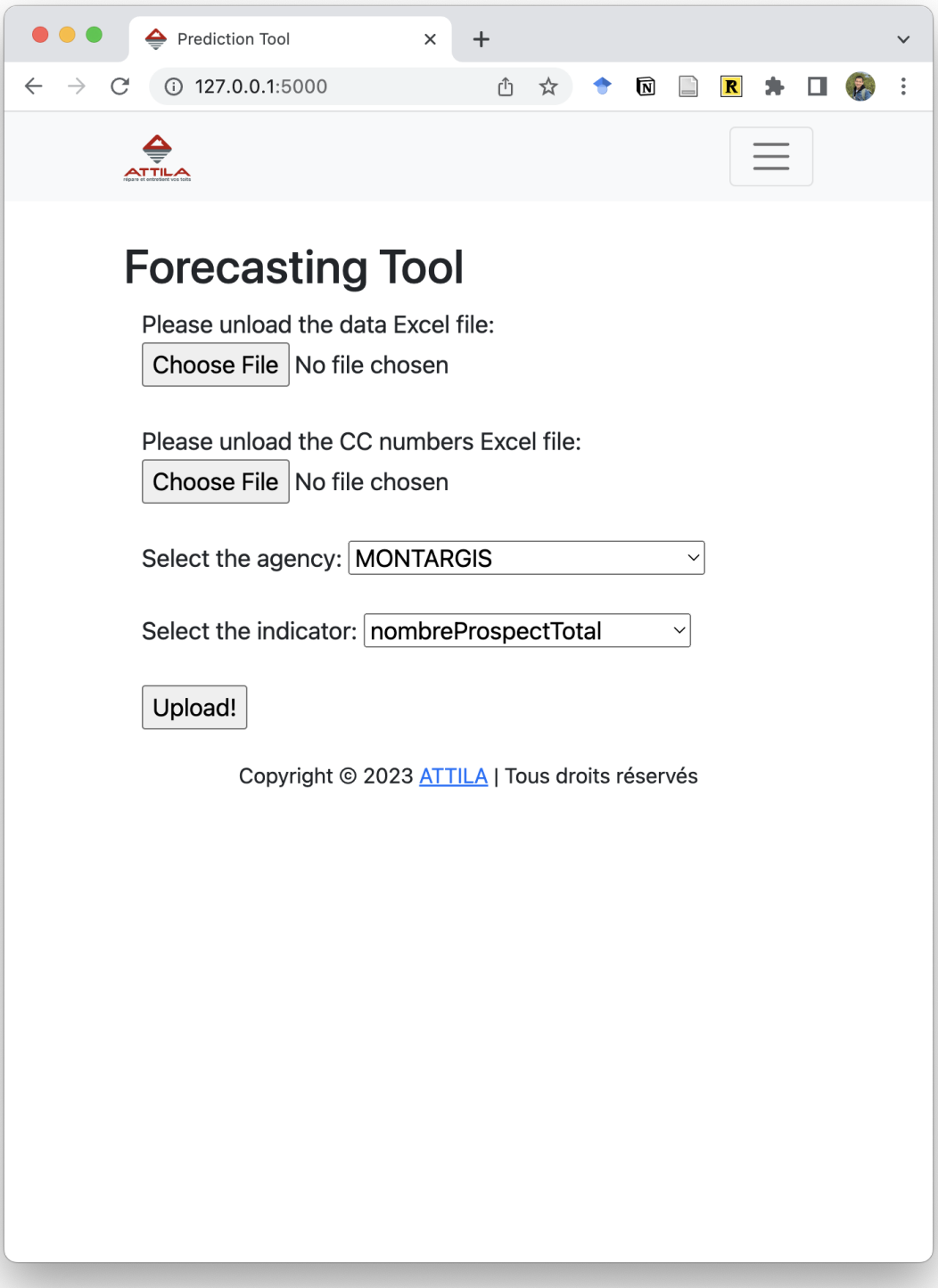


Figure 7.1: Home page.

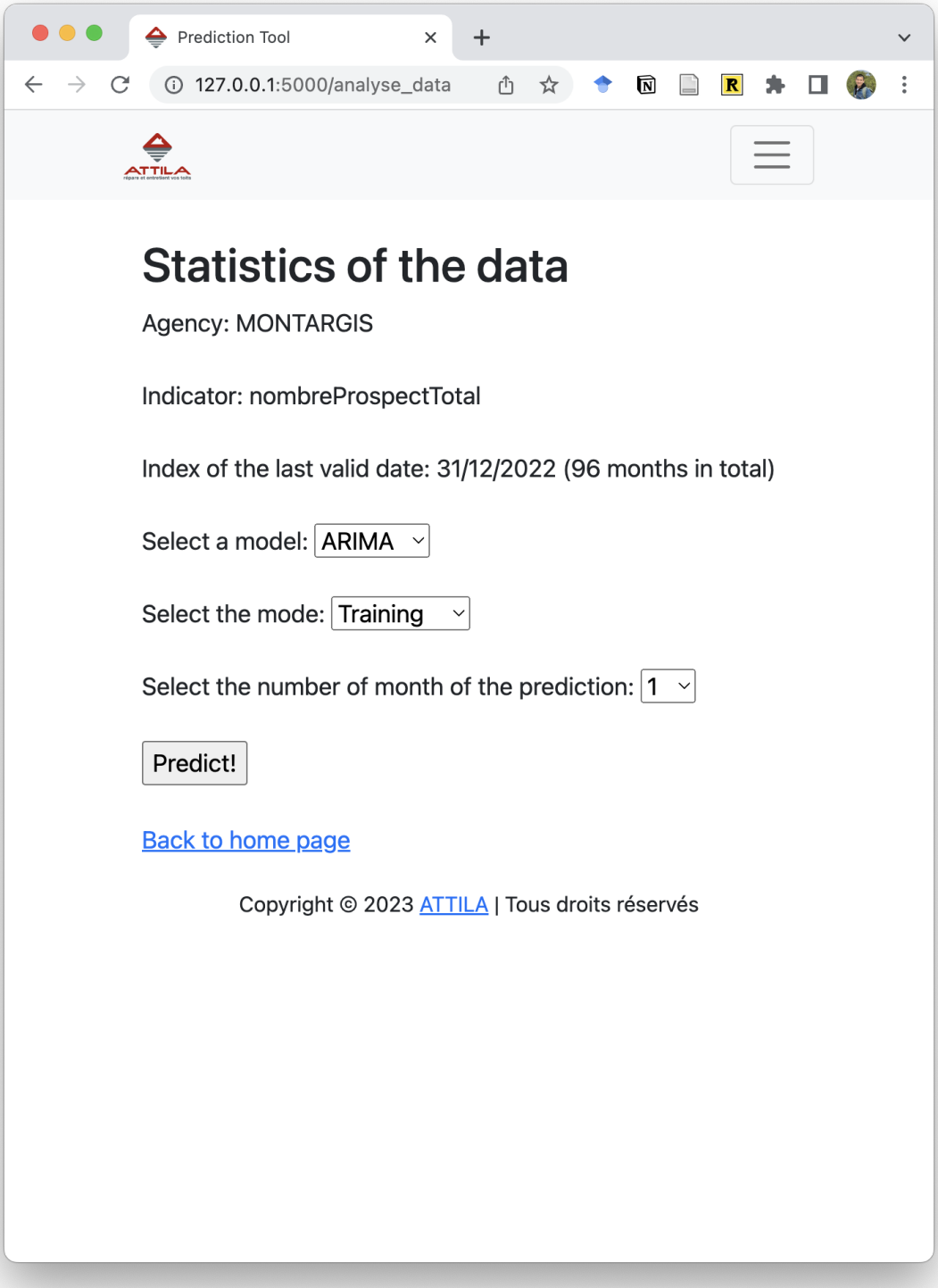


Figure 7.2: Statistics page.

Figure 7.3: Result page.

7.2.2 Technology Stack and Technical Architecture

Technology Stack

The prototype is a web application developed using the Flask framework, which allows the user to develop web applications using the Python programming language. It is open-sourced and available on GitHub. The Flask framework is also free to use and is distributed under the BSD-3-Clause license.

Figure 7.4: The libraries used in the prototype.

In our Flask application, several frameworks and libraries are incorporated to create a well-structured and functional web application, including:

1. **Flask Web framework**[1]**.** It provides the foundation for our web application by offering essential features such as routing, template engine, and server support.

2. **Jinja2 templating engine**[2]**.** Jinja is a fast, expressive, extensible templating engine. It is Flask's default templating engine and allows for dynamic HTML pages in the application.

3. **Bootstrap**[3]**.** It is a popular front-end framework that simplifies page layout and design by providing pre-defined CSS styles and components. In our application, we adopted the **Bootstrap-Flask**[4] extension, which integrates Bootstrap into Flask.

4. **Plotly**[5]**.** Plotly is a library for creating interactive charts that can be embedded within the Flask application to provide visually appealing and informative data visualizations.

[1] `https://flask.palletsprojects.com/en/2.3.x/`
[2] `https://jinja.palletsprojects.com/en/3.1.x/`
[3] `https://getbootstrap.com/`
[4] `https://bootstrap-flask.readthedocs.io/en/stable/`
[5] `https://dash.plotly.com/`

5. **Data processing libraries.** We employed several data processing libraries, such as pandas [121] and NumPy [122], to enable the analysis and manipulation of data to produce the forecasting results.

6. **Scientific libraries.** We also used several scientific libraries, such as SciPy [123], scikit-learn [98], sktime [22], statsmodels [15], and XGBoost [124] to implement the forecasting algorithms.

Fig. 7.4 summarizes the libraries used in the prototype.

The Model-View-Controller Design Pattern

The aforementioned components come together to form a typical Flask web application, which follows the Model-View-Controller (MVC) design pattern. There are three components in this pattern. In our Flask application, they can be described as follows:

1. **Model.** This represents the data processing and business logic components of the application, such as the time series forecasting algorithms and data manipulation.

2. **View.** This represents the HTML pages built using Jinja2 templates and Bootstrap, which present data to the user and collect user input.

3. **Controller.** This is the Flask routes and view functions that handle user requests, invoke the appropriate model components to process data, and return the results to the views for presentation.

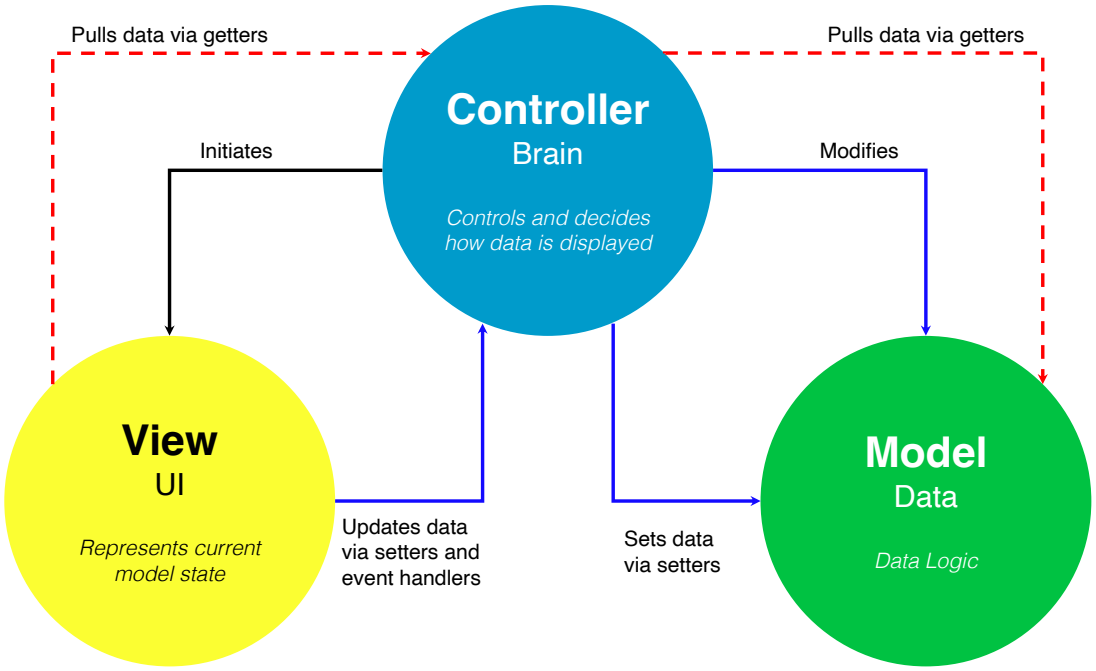


Figure 7.5: The MVC design pattern.

The architecture of the MVC design pattern can be illustrated by Fig. 7.5. By following the MVC design pattern, the application can be separated into three distinct components,

which makes it easier to maintain and extend the application. For example, if a new feature is required, the developer can simply add a new route and view function to the Controller, and add a new HTML page to the view, without the need to modify the model component.

7.2.3 Core Functionalities and Algorithms

As mentioned in Sec. 7.2.1, the prototype provides three core functionalities: data uploading, data preprocessing, and forecasting. In this section, we will describe the technical details of these functionalities and the forecasting algorithms involved.

Data Uploading

The data uploading functionality allows the user to upload several Excel files containing the time series data to be analyzed. The uploaded file is then processed by the Flask server and stored in the `uploads` folder. When the user uploads a file, the following components and steps in the Flask web application are called in sequence:

1. The user selects a file and clicks the `<Upload!>` button on the home page.
2. The browser sends the file to the specified route through an HTTP POST request to the Flask server.
3. Once the Controller component receives the request, it invokes the Model component. The Model will store the uploaded file in the `uploads` folder, preprocess the data and store the processed data in the `data` folder in CSV format.
4. Once the processing is complete, the Model component will return the results to the Controller component.

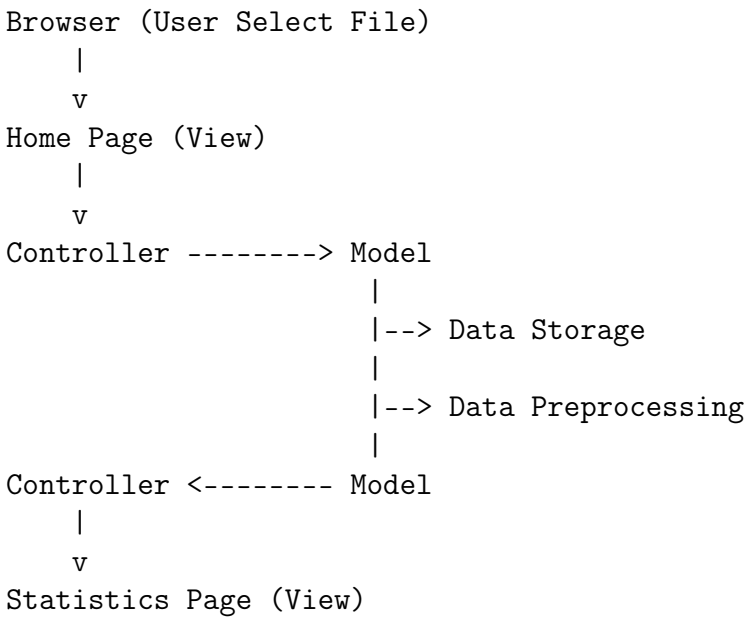


Figure 7.6: The data uploading process.

5. The Controller then passes the results to the View component, which will navigate to the Statistics page and display the results to the user.

The data uploading process can be illustrated by the chart in Fig. 7.6.

Data Preprocessing

In this part, for confidential reasons, we will omit the details of the data and the data preprocessing procedure, and focus on the implementation details of the data preprocessing functionality.

The data preprocessing functionality allows the user to preprocess the uploaded data before performing forecasting. All the preprocessing steps are hidden from the user and performed automatically by the Flask server. The user can view the results of each step on the Statistics page. This process will call the following components and steps in the Flask web application in sequence:

1. Assuming the user has already uploaded the Excel file and the Controller has already passed the uploaded file to the Model for processing.

2. The Model starts performing preprocessing on the uploaded data. The preprocessing steps include data cleaning, missing data handling, data transformation, feature engineering, and data splitting. In this step, the Model component will invoke the data processing libraries to perform the preprocessing steps and adjust the data structure and format as needed by different forecasting algorithms.

3. Once the preprocessing is complete, the Model component will return the results to the Controller.

4. The Controller update the View based on the results returned by the Model. The user can view the results on the Statistics page.

5. After preprocessing, the user can click the `<Predict!>` button on the Statistics page to perform forecasting.

The data preprocessing procedure can be illustrated by Fig. 7.7.

Forecasting

The forecasting functionality allows the user to perform forecasting on the preprocessed data by selecting different forecasting algorithms and settings. Similar to previous functionalities, the forecasting functionality will also sequentially call these following steps:

1. User clicks the `<Predict!>` button on the Statistics page.

2. The web browser sends an HTTP POST request to the Flask server, passing the selected forecasting algorithm and settings.

3. The Controller in the server receives the request and passes the selected forecasting algorithm and settings to the Model.

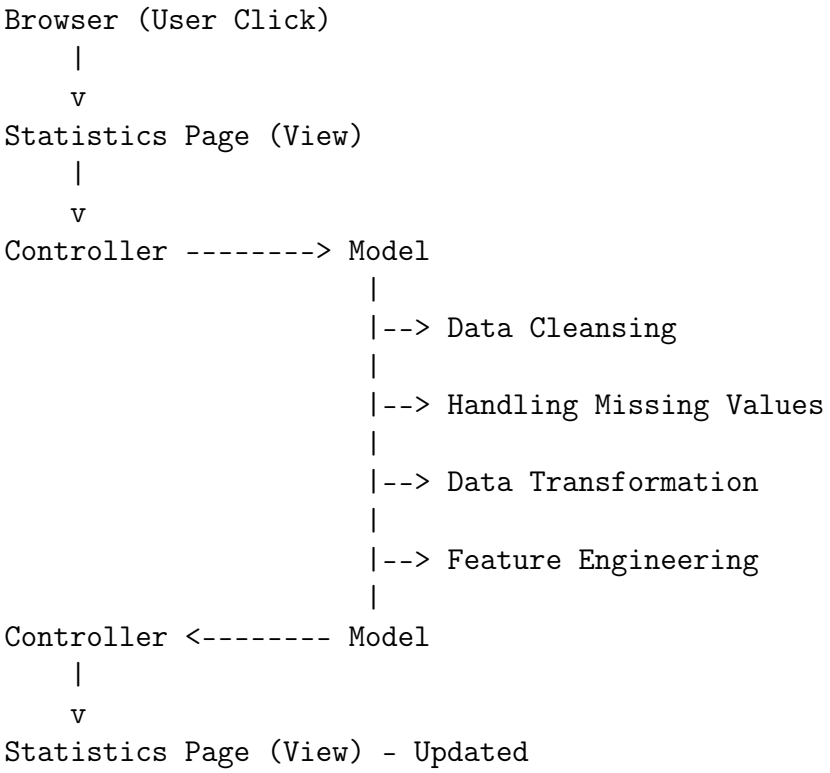


Figure 7.7: The data preprocessing procedure.

4. The Model invokes the pre-trained forecasting model or trains a real-time forecasting model and generates prediction values based on the preprocessed data.

5. Once the prediction values are generated, the Model returns the results to the Controller.

6. The Controller passes the results to the View, which will display the results on the Result page.

7. The user can check the forecasting results in the Result page, and continue to interact with the application as needed.

This procedure can be illustrated by the chart in Fig. 7.8.

We also provide a `<Save>` button on the Result page, allowing the user to download the forecasting results in the form of an Excel file. Here is a sequence of steps that will be called when the user clicks the `<Save>` button:

1. Once this button is clicked, the browser will send an HTTP GET request to the Flask server, which will invoke the Controller.

2. The Controller converts the results into Excel format and saves it in the temporary memory in the form of a `BytesIO` object.

3. The Controller then sets the response headers, including the file name and file type, and sends the file content back to the browser as a response.

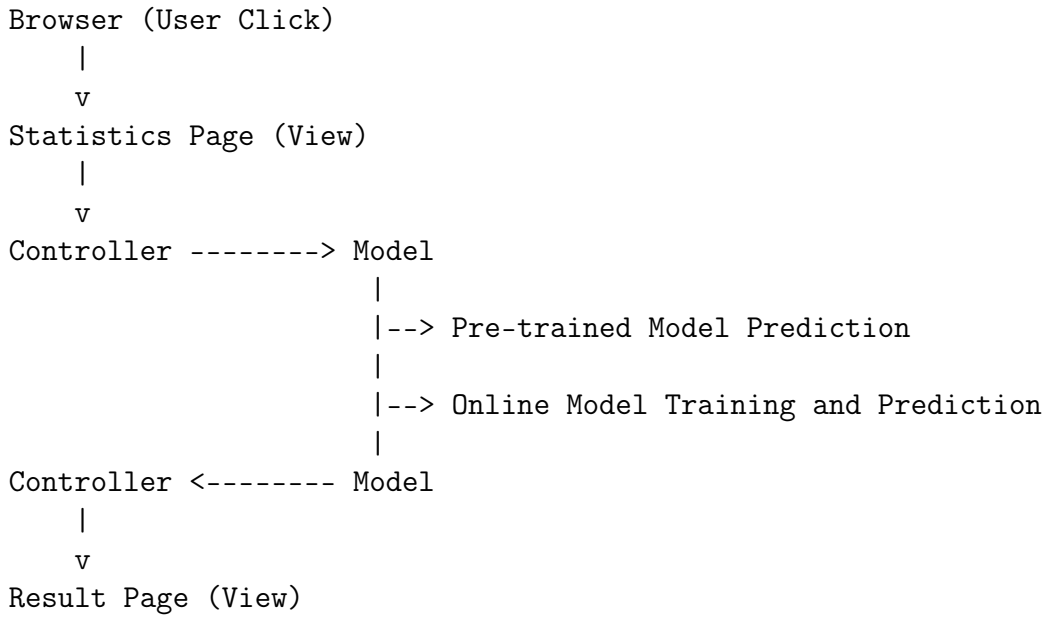


Figure 7.8: Forecasting procedure.

4. Based on the response headers, the browser will then prompt the user to save the file.

Fig. 7.9 illustrates this procedure.

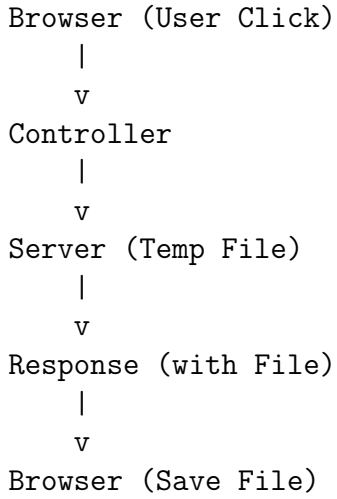


Figure 7.9: Saving results procedure.

Through these different processes, the MVC architecture works together to facilitate the interaction between data processing and the user interface. During this process, Flask acts as a web framework, responsible for handling HTTP requests/responses, as well as dispatching the corresponding Controller logic.

Algorithms and Settings

The application provides different forecasting algorithms for the user. For now, the following algorithms are implemented:

- **Statistical Algorithms**: ARIMA, ETS, Theta, and Prophet.
- **Machine Learning Algorithms**: SVR, k-NN, Random Forest, Gaussian Process, and Gradient Boosting.

For all implemented algorithms, two modes are offered: training and prediction. In the training mode, the last few data points (selected by the user) will be used as the test set, and the rest of the data will be used as the training set. In this mode, the model's performance will be evaluated by the Mean Absolute Percentage Error (MAPE), in order to help the user to choose the most fitted model for the selected indicator. While in the prediction mode, all data points will be used as the training set, and the application will generate prediction values for the future.

The application offers automatic parameterization for all implemented algorithms. Currently, the volume of the company's data is insufficient for training deep learning models. This is the reason why this functionality is not offered in the application. In the future, a transfer learning strategy may be considered to add appropriate deep learning methods to the application.

7.2.4 Deployment and Maintenance

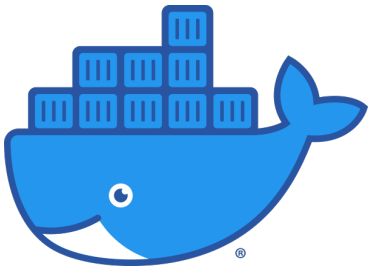


Figure 7.10: Docker®.

For delivery, the application is packed in a Docker[6] container, a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings, allowing the user to run the application on any compatible machine with a single command.

The application can also be deployed on a cloud server or integrated as a functionality of the company's intranet. As it is developed following an MVC design pattern, the application can be easily maintained, updated, or extended.

7.3 Conclusion

In this chapter, we have presented the design and implementation of a prototype in the form of a web application.

Specifically, we introduced the requirements of the functionalities at the beginning of this chapter. Then we presented in detail the design of the application, including the

[6] `https://www.docker.com/`

user interface, the technology stack, the architecture, core functionalities, and forecasting algorithms included. Finally, we discussed the deployment and maintenance of the application.

The application is built upon a web-based user interface, which is developed using Python, along with several support libraries. Specifically, we used Flask as the web framework, Bootstrap for the front end, and Plotly for visualization. The application is designed following an MVC design pattern, which allows more convenient maintenance and extension of the application. The application is also packed in a Docker container, which allows the user to run the application on any compatible machine with a single command.

Chapter 8

Conclusions and Outlook

This chapter summarizes the thesis and proposes some future research directions.

8.1 Summary of the Thesis

In this thesis, we have presented several contributions to the TSF problem, mainly in the econometric and deep learning domains. We have also developed a prototype web application to showcase the practicality of implementing our suggested models in real-world situations. We want to summarize them separately in this section.

For Econometric Time Series Forecasting

In our work, we undertook a comprehensive and in-depth review of state-of-the-art econometric time series analysis and forecasting techniques. Our examination spanned a variety of methodologies, including AutoRegressive Integrated Moving Average (ARIMA), Exponential Smoothing State Space Model (ETS), and Vector AutoRegressive (VAR) models, along with their respective variations. This review delved into the underlying mathematical principles and theoretical foundations that govern these sophisticated approaches, elucidating their relevance and applications.

Further, we also explored several time series decomposition methods, such as additive and multiplicative decomposition. Additionally, we introduced the Seasonal-Trend decomposition utilizing LOESS, known as STL, and provided a detailed study of it. Moreover, we expounded on the Theta method, which is a decomposition-based method for analyzing time series, along with its generalizations and alternative versions, to offer a broader understanding of these approaches and their applicability in the field.

On the foundation of the aforementioned review, we conducted a thorough investigation of various forecasting techniques, integrating them with STL decomposition and comparing their performance. The comparison encompasses three econometric methods and five machine learning models, with extensive experiments carried out on the M3-Competition datasets. Our findings reveal that employing STL decomposition as a preprocessing step for the monthly industrial M3-Competition dataset proves advantageous for statistical forecasting methods yet detrimental for their ML counterparts. Furthermore, the synergistic combination of STL and the Theta method surpasses the performance of alternative approaches under evaluation.

For Deep Learning Time Series Forecasting

Our exploration focused primarily on the key deep learning models utilized for TSF tasks, including MLPs, CNNs, RNNs, and Attention Mechanism. Moreover, we delved into the main challenges encountered in TSF and drew comparisons between traditional machine learning and deep learning models. We also revisited the Transformer model and offered an extensive overview of the Transformer model and its derivatives for the TSF problem, encompassing 14 distinct Transformer-based models. We examine their enhancements over the standard Transformer and their precursors.

We investigated the application of three DL models, DA-RNN, LSTNet, and TPA-LSTM, for multivariate TSF (MTSF) problems, examining five forecasting strategies for multi-step forecasting: One-Step-Ahead, Recursive, Direct, Multi-Input Multi-Output (MIMO), and Multi-Input Several Multi-Output (MISMO) strategies. Our findings indicate that these models consistently struggle with accumulated errors under the Recursive strategy, hindering their ability to execute actual multi-step forecasting tasks. Nevertheless, combining them thoughtfully with MIMO/MISMO strategies can mitigate this issue, thus enabling one-step-ahead deep learning models for multi-step forecasting.

Furthermore, we proposed the Rankformer, a new Transformer-based model for the TSF problem, which leverages the rank correlation function and a decomposition architecture for long-term TSF tasks. We also proposed the STLformer, an advanced iteration of the Rankformer that employs the STL decomposition to enhance forecasting performance. Notably, both the Rankformer and STLformer outperformed four state-of-the-art Transformers and two RNN models across multiple datasets and forecasting horizons.

Product Delivery

Additionally, we designed a prototype web application for ATTILA Gestion. Starting with the introduction of functional requirements, we delved into a comprehensive presentation of the application's design. This includes the user interface, technology stack, architecture, core functionalities, and integrated forecasting algorithms. Built on a web-based user interface using Python and several supporting libraries like Flask for the web framework, Bootstrap for the front end, and Plotly for visualization, the application adheres to an MVC design pattern. This design enables convenient maintenance and expansion of the application. The application is containerized using Docker for user-friendly deployment and execution on any compatible machine with a singular command, simplifying both its use and upkeep. It also offers a GUI that allows users to interact with the models and visualize the outcomes seamlessly. It is also deployable and extensible to ATTILA's existing infrastructure. The application is currently in the testing phase and will be deployed in the near future with more features and models integrated.

8.2 Challenges, Open Problems, and Future Perspectives

This section presents several research challenges and open problems that can be considered for future TSF problems research. They are data, model, and task requirements, with possible solutions discussed in the following subsections.

8.2.1 Data Requirements

Time series data can present different characteristics to which TSF models should adapt. We here present four main types of time series data characteristics that can pose challenges for TSF models: (i) non-stationarity; (ii) multivariate and high-dimensional time series; (iii) rare events, missing values, and anomalies; and (iv) hierarchical and grouped time series.

Non-stationarity

Real-world time series data often exhibit non-stationary behavior, with changing patterns, trends, and seasonality over time [1], [14], [125]–[127]. Meanwhile, for many of the time series models, the model assumptions are violated when non-stationary data is used. This leads to the estimators no longer having nice properties such as asymptotic normality and sometimes even consistency, which limits their applicability in many scenarios.

For econometric models, (S)ARIMA and ETS [110] can model non-stationarity series, but they have a linearity assumption and a fixed model structure. The lack of support for multivariate and missing data also limits their applicability. Other models like cointegration models [128], VAR models [110], Fractionally Integrated Models [129], [130], and GARCH models [131] can also model non-stationary series. However, they are limited either by their assumptions, e.g., linear relationships and conditional normality, or by their computational complexity and estimation difficulties.

In recent years, deep models have shown promising results in various aspects, yet they are also limited in their ability to model non-stationary time series. These models are based on statistics, which are exactly what changes in non-stationary time series data, making the problem even more intractable. Most of the solutions for deep models proceed with stationarization as a preprocessing step, such as Adaptive Norm [132], RevIN [133], and DAIN [134]. Liu *et al.* [76] proposed Non-Stationary Transformer to solve this problem with Series Stationarization and De-stationary Attention but still assumes a linear activation condition. Except for the Non-Stationary Transformer, to the best of our knowledge, we are not aware of any other deep models that can directly model non-stationary time series data.

As some econometric models exhibit good performance in modeling non-stationary time series, it is worth exploring how to combine the advantages of econometric models and deep models to model non-stationary time series data.

Multivariate and high-dimensional time series

Many real-world applications involve multivariate time series (MTS) with complex dependencies among variables. While many models have been proposed to model MTS, they also have various limitations.

VAR models are the most widely used models for econometric models for MTS. Vector Error Correction Models (VECMs) [135] enables VAR for non-stationary time series. Both VAR and VECMs assume linear relationships. VECMs also require cointegrated series, which can be challenging to test and establish in practice. Multivariate GARCH [136] extends GARCH to a multivariate setting but can be computationally demanding, especially when the number of variables is large. They also focus primarily on the conditional

covariance structure and may not capture other forms of non-linear dependence. Panel data models [137] are designed to handle data that combines both cross-sectional and time series dimensions but require the assumption of a specific structure for the unobserved heterogeneity across cross-sectional units, which may not always hold in practice.

Most deep models can handle multivariate data by design, but they also come with limitations of their own. Due to error accumulation, deep models may struggle to produce accurate long-term forecasts in a multi-step-ahead setting. They do not typically offer built-in variable selection mechanisms, which can lead to poor performance if irrelevant variables are included. While DL models can capture non-linear relationships, they may struggle to model complex interdependencies between variables in MTS. When recurrent models such as LSTMs and GRUs are employed, they can limit parallelization and lead to slow training and inference times, especially when applied to large-scale MTS data.

To sum up, developing models that can efficiently capture these dependencies and scale with the increasing dimensionality of the data is still an ongoing challenge.

Missing values, rare events, and anomalies

Time series data often contain missing values, rare events, or anomalies that can significantly impact forecasting accuracy. Although many different techniques have been proposed to address these issues, they are still open problems.

Econometric models typically require complete data and are not designed to handle missing values. Some techniques and statistical methods, such as imputation and deletion, may help but can also introduce bias and a loss of information. In addition, econometric models typically focus on understanding and forecasting the underlying structure and relationships in the data rather than explicitly addressing rare events and anomalies, which can lead to poor performance in these scenarios.

DL models are increasingly designed to deal with missing data and anomalies in recent years, such as imputation with autoencoders and sequence-sequence models [138]. Most deep models can be used for imputing missing values or detecting anomalies by learning the underlying patterns and structure. Nonetheless, these deep learning approaches are not specifically designed for handling missing rare events, values, and anomalies. Some models can also individually handle missing values, rare events, or anomalies, e.g., RobustSTL [29] and RobustTAD [11].

To the best of our knowledge, no model can handle missing values, rare events, and anomalies simultaneously, leaving an open problem and an area of active research.

8.2.2 Model Requirements

Model interpretability and explainability

While being emphasized on their performance, most DL models, not limited to TSF problems, are considered "black boxes" due to their complex inner workings.

Bringing interpretability to ML is not a new idea [139]–[141], and many TS models have integrated interpretability by design. Lundberg and Lee [142] presented a unified framework for interpreting predictions called SHAP that show improved computational performance and/or better consistency with human intuition. Assaf and Schumann [143] and Assaf *et al.* [144] demonstrated that CNN could be used for explaining predictions

of MTS data and then proposed MTEX-CNN that can simultaneously visualize the network's attention over both time and feature dimensions. Wang *et al.* [145] proposed an adversarially regularized CNN for shapelet-based TS classification. Fauvel, Masson, and Fromont [146] presented a performance-explainability framework to benchmark ML methods for MTS classification.

However, most of these models are designed for TS classification or problems from other domains, e.g., image, and there is still a growing need for models that are not only accurate but also interpretable and explainable for TSF problems.

Model selection and hyperparameter tuning

Choosing the best model and tuning its hyperparameters for a specific time series problem can be challenging and time-consuming.

Hyndman and Khandakar [17] developed the well-known `forecast` package for R language, which can automatically select the best model and tune its hyperparameters for a given time series problem. Löning *et al.* [22] replicated some functionalities from the `forecast` package for a Python implementation as the `sktime` library. Wang *et al.* [147] combined `auto-sklearn` [148] for automated window selection and `tsfresh` [149] for automatic feature extraction to build an automated TSF pipeline. There are also some other automatic ML/DL frameworks for TSF, such as `Auto-TS`[1] and `Auto-Pytorch-TS` [150].

However, most of the implemented models are either econometric ones or may not be able to handle large-scale time series data. The aforementioned AutoML/AutoDL frameworks cannot always guarantee an optimized model and hyper-parameterization for a given time series problem. Thus, developing automated or semi-automated techniques for model selection and hyperparameter optimization is an ongoing area of research.

8.2.3 Task Requirements

Long-term forecasting

Accurately predicting long-term trends in time series data is difficult due to the accumulation of errors over time and the inherent uncertainty in distant future predictions.

Most econometric methods are not designed for long-term forecasting, in which they cannot include distant past observations in the model. In recent years, many DL models, especially the Transformer-based models, have been proposed for long-term forecasting problems, such as LSTNet [55], Informer [72], Autoformer [73], FEDformer [75], to name a few. However, these models are either benchmarked on a small number of ideal standard datasets or do not accept short-term input, i.e., they require large amounts of training data.

Although there are already many models for long-term forecasting, there is still a need for models that can handle short-term input and perform practically well on real-world datasets.

[1] `https://github.com/AutoViML/Auto_TS`

Online forecasting and continual learning

In many real-world scenarios, time series data is a continuous input, which is more generally known as *data stream*, and models must adapt to new data simultaneously. This is known as online forecasting [151].

Much research has been conducted on this topic. Liu *et al.* [152] proposed an online ARIMA method for TSF tasks by reformulating the ARIMA model into a full information online optimization task. Gultekin and Paisley [153] considered the TSF problem that can be modeled as matrices and use low-rank matrix factorization for predicting future values. Matsubara and Sakurai [154] proposed RegimeCast for co-evolving TS streams, but it handles only the sudden discontinuity (regime shifts) rather than gradual shifts. Boulegane, Bifet, and Madhusudan [155] presented Streaming-ADE and dynamically updated the base models when concept drift was detected. Zuo, Zeitouni, and Taher [156] adopted an incremental shapelet extraction for stable concepts and an adjusted concept drift detection method for unstable concepts for TS classification problems.

Nonetheless, to the best of our knowledge, there is very little research on online forecasting for deep models. Developing efficient online learning algorithms for TSF is still a critical challenge.

Uncertainty quantification

Quantifying the uncertainty associated with forecasts is crucial in many applications, as it helps users make more informed decisions [157]. Wang *et al.* [158] proposed a deep uncertainty quantification by simultaneously implementing single-value forecasting and uncertainty quantification for weather forecasting tasks. Cui, Hu, and Zhu [159] used the maximum mean discrepancy for a calibrated regression to provide well-calibrated and sharp prediction intervals.

However, most of the research dedicated to uncertainty quantification is designed for image processing and computer vision tasks. Very little research has been done on uncertainty quantification for TSF problems. Developing methods for reliable uncertainty quantification in TSF remains an open problem.

8.2.4 Others

Evaluation metrics and benchmarking

Performance comparisons among different models can be challenging due to the variety of evaluation metrics and benchmarks used in the literature.

The Mean Absolute Percentage Error (MAPE) is one of the most widely used measures of forecasting accuracy. Another widely used one is the symmetric Mean Absolute Percentage Error (sMAPE). Nonetheless, both MAPE and sMAPE can produce infinite or undefined values when the actual and/or the forecasted values are zero [99]. Hyndman and Koehler [100] proposed the Mean Absolute Scaled Error (MASE). However, it can still be dominated by one large erroring value.

Kim and Kim [160] proposed in 2016 the Mean Arctangent Absolute Percentage Error (MAAPE) to overcome the problem of division by zero fundamentally using bounded

influences for outliers. It is defined as follows:

$$\text{MAAPE} = \frac{1}{h}\sum_{t=n+1}^{n+h} \arctan\left(\left|\frac{e_t}{y_t}\right|\right),$$

where $e_t = y_t - \hat{y}_t$. y_t is the actual value at time t and $\hat{y}_t$ is the forecast value at time t. n is the number of data points available in-sample, and h is the forecast horizon.

Chen, Twycross, and Garibaldi [161] invented the Unscaled Mean Bounded Relative Absolute Error (UMBRAE), a scale-independent, outlier-resistant, and more interpretable error measure. The definition of UMBRAE is as follows:

$$\text{UMBRAE} = \frac{\sum_{t=n+1}^{n+h} \frac{|e_t|}{|e_t|+|e_t^*|}}{h - \sum_{t=n+1}^{n+h} \frac{|e_t|}{|e_t|+|e_t^*|}},$$

where e_t^* is the error of a benchmark model, e.g., the naïve model.

Establishing standardized evaluation protocols and benchmarks can help facilitate model comparison and drive the development of better forecasting models.

Stones from other hills

As an old Chinese proverb goes, "Stones from other hills may serve to polish the jade of this one". Some of these problems in one certain requirement can also be attributed to other requirements and can also correlate with other problems. For example, one can very often have to deal with non-stationarity in the data and/or incorporate rare events when performing long-term forecasting tasks. These problems can also be correlated with other research areas, such as computer vision, natural language processing, and speech recognition. Therefore, we can also draw inspiration from these fields to tackle the challenges in TSF.

Bibliography

[1] R. S. Tsay, *Analysis of Financial Time Series*, 3rd ed. John Wiley & Sons, 2010.

[2] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning : A systematic literature review: 2005–2019," *Appl. Soft Comput.*, vol. 90, p. 106 181, 2020.

[3] L. Longo, M. Riccaboni, and A. Rungi, "A neural network ensemble approach for GDP forecasting," *J. Econ. Dyn. Control*, vol. 134, p. 104 278, 2022.

[4] A. Zeroual *et al.*, "Deep learning methods for forecasting COVID-19 time-Series data: A Comparative study," *Chaos Solit. Fractals*, vol. 140, p. 110 121, 2020.

[5] H. Qi *et al.*, "COVID-19 transmission in Mainland China is associated with temperature and humidity: A time-series analysis," *Sci. Total Environ.*, vol. 728, p. 138 778, 2020.

[6] H. Li *et al.*, "Air pollution and temperature are associated with increased COVID-19 incidence: A time series study," *Int. J. Infect. Dis.*, vol. 97, pp. 278–282, 2020.

[7] X. Shi *et al.*, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *Proc. NeurIPS*, 2015.

[8] J. M. Rožanec, B. Fortuna, and D. Mladenić, "Reframing Demand Forecasting: A Two-Fold Approach for Lumpy and Intermittent Demand," *Sustainability*, vol. 14, no. 15, p. 9295, 2022.

[9] J. Zuo *et al.*, "Graph convolutional networks for traffic forecasting with missing values," *Data. Min. Knowl. Disc.*, vol. 37, no. 2, pp. 913–947, 2023.

[10] P. Gloaguen *et al.*, "Scalable clustering of segmented trajectories within a continuous time framework: Application to maritime traffic data," *Mach. Learn.*, 2021, early access.

[11] J. Gao *et al.*, "RobustTAD: Robust Time Series Anomaly Detection via Decomposition and Convolutional Neural Networks," in *Proc. MileTS*, 2020.

[12] H. Akaike, "A new look at the statistical model identification," *IEEE Trans. Autom. Control*, vol. 19, no. 6, pp. 716–723, 1974.

[13] G. Schwarz, "Estimating the Dimension of a Model," *Ann. Stat.*, vol. 6, no. 2, pp. 461–464, 1978.

[14] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed. OTexts, 2021.

[15] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with python," in *Proc. SCIPY*, 2010.

[16] T. Smith *et al.* "Pmdarima: ARIMA estimators for Python." (2017), [Online]. Available: `http://alkaline-ml.com/pmdarima/`.

[17] R. J. Hyndman and Y. Khandakar, "Automatic Time Series Forecasting: The **forecast** Package for *R*," *J. Stat. Soft.*, vol. 27, no. 3, pp. 1–22, 2008.

[18] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *Int. J. Forecast.*, vol. 20, no. 1, pp. 5–10, 2004.

[19] E. S. Gardner and E. Mckenzie, "Forecasting Trends in Time Series," *Manage. Sci.*, vol. 31, no. 10, pp. 1237–1246, 1985.

[20] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Manage. Sci.*, vol. 6, no. 3, pp. 324–342, 1960.

[21] R. J. Hyndman *et al.*, *Forecasting with Exponential Smoothing*. Springer-Verlag Berlin Heidelberg, 2008.

[22] M. Löning *et al.*, "Sktime: A unified interface for machine learning with time series," in *Proc. NeurIPS*, 2019.

[23] H. Lütkepohl, *New Introduction to Multiple Time Series Analysis*, 1. ed., corr. 2. print. Springer, 2007.

[24] B. Pfaff, "VAR, SVAR and SVEC Models: Implementation Within R Package vars," *J. Stat. Softw.*, vol. 27, pp. 1–32, 2008.

[25] R. B. Cleveland *et al.*, "A Seasonal-Trend Decomposition Procedure Based on Loess," *J. Off. Stat.*, vol. 6, no. 1, pp. 3–73, 1990.

[26] W. S. Cleveland, "Robust Locally Weighted Regression and Smoothing Scatterplots," *J. Am. Stat. Assoc.*, vol. 74, no. 368, pp. 829–836, 1979.

[27] G. E. P. Box and D. R. Cox, "An Analysis of Transformations," *J. R. Stat. Soc., A: Stat. Soc.*, vol. 26, no. 2, pp. 211–252, 1964.

[28] K. Bandara, R. Hyndman, and C. Bergmeir, "MSTL: A Seasonal-Trend Decomposition Algorithm for Time Series with Multiple Seasonal Patterns," *Int. J. Oper. Res.*, vol. 1, no. 1, 2022.

[29] Q. Wen *et al.*, "RobustSTL: A Robust Seasonal-Trend Decomposition Algorithm for Long Time Series," in *Proc. AAAI*, 2019.

[30] Q. Wen *et al.*, "Fast RobustSTL: Efficient and Robust Seasonal-Trend Decomposition for Time Series with Complex Patterns," in *Proc. ACM SIGKDD*, 2020.

[31] X. Wang, K. Smith, and R. Hyndman, "Characteristic-Based Clustering for Time Series Data," *Data Min. Knowl. Disc.*, vol. 13, no. 3, pp. 335–364, 2006.

[32] V. Assimakopoulos and K. Nikolopoulos, "The theta model: A decomposition approach to forecasting," *Int. J. Forecast.*, vol. 16, no. 4, pp. 521–530, 2000.

[33] K. Nikolopoulos *et al.*, "The theta model: An essential forecasting tool for supply chain planning," in *Proc. ICAR*, 2011.

[34] R. J. Hyndman, "A brief history of forecasting competitions," *Int. J. Forecast.*, M4 Competition, vol. 36, no. 1, pp. 7–14, 2020.

[35] E. Spiliotis, V. Assimakopoulos, and S. Makridakis, "Generalizing the Theta method for automatic forecasting," *Eur. J. Oper. Res.*, vol. 284, no. 2, pp. 550–558, 2020.

[36] F. Petropoulos and K. Nikolopoulos, "Optimizing Theta Model for Monthly Data:" in *Proc. ICAART*, 2013.

[37] K. Nikolopoulos and D. D. Thomakos, *Forecasting with the Theta Method: Theory and Applications*. Wiley, 2019.

[38] J. A. Fiorucci *et al.*, "Models for optimising the theta method and their relationship to state space models," *Int. J. Forecast.*, vol. 32, no. 4, pp. 1151–1161, 2016.

[39] R. J. Hyndman and B. Billah, "Unmasking the Theta method," *Int. J. Forecast.*, vol. 19, no. 2, pp. 287–290, 2003.

[40] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[41] J. B. Yang *et al.*, "Deep convolutional neural networks on multichannel time series for human activity recognition," in *Proc. IJCAI*, 2015.

[42] R. Wen *et al.*, "A Multi-Horizon Quantile Recurrent Forecaster," in *Proc. NeurIPS*, 2018.

[43] R. Sen, H.-F. Yu, and I. S. Dhillon, "Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting," in *Proc. NeurIPS*, vol. 32, 2019.

[44] M. Liu *et al.*, "Time Series is a Special Sequence: Forecasting with Sample Convolution and Interaction," in *Proc. NeurIPS*, 2022.

[45] P. Esling and C. Agon, "Time-series data mining," *ACM Comput. Surv.*, vol. 45, no. 1, 12:1–12:34, 2017.

[46] H. I. Fawaz *et al.*, "Deep learning for time series classification: A review," *Data Min. Knowl. Disc.*, vol. 33, no. 4, pp. 917–963, 2019.

[47] V. Kuznetsov and Z. Mariet, "Foundations of Sequence-to-Sequence Modeling for Time Series," in *Proc. AISTATS*, 2019.

[48] K. Benidis *et al.*, "Deep Learning for Time Series Forecasting: Tutorial and Literature Survey," *ACM Comput. Surv.*, vol. 55, no. 6, 121:1–121:36, 2022.

[49] H. Hewamalage, C. Bergmeir, and K. Bandara, "Recurrent Neural Networks for Time Series Forecasting: Current status and future directions," *Int. J. Forecast.*, vol. 37, no. 1, pp. 388–427, 2021.

[50] B. Lim and S. Zohren, "Time Series Forecasting With Deep Learning: A Survey," *Phil. Trans. R. Soc. A.*, vol. 379, no. 2194, p. 20 200 209, 2021.

[51] J. F. Torres *et al.*, "Deep Learning for Time Series Forecasting: A Survey," *Big Data*, vol. 9, no. 1, pp. 3–21, 2021.

[52] S. Makridakis *et al.*, "Statistical, machine learning and deep learning forecasting methods: Comparisons and ways forward," *J. Oper. Res. Soc.*, pp. 1–20, 2022.

[53] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *Proc. ICLR*, 2015.

[54] Y. Qin *et al.*, "A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction," in *Proc. IJCAI*, 2017.

[55] G. Lai *et al.*, "Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks," in *Proc. ACM SIGIR*, 2018.

[56] S.-Y. Shih, F.-K. Sun, and H.-y. Lee, "Temporal pattern attention for multivariate time series forecasting," *Mach. Learn.*, vol. 108, no. 8-9, pp. 1421–1441, 2019.

[57] Y. Wang *et al.*, "Deep Factors for Forecasting," in *Proc. ICML*, 2019.

[58] D. S. De O. Santos Júnior, J. F. L. De Oliveira, and P. S. G. De Mattos Neto, "An intelligent hybridization of ARIMA with machine learning models for time series forecasting," *Knowl.-Based Syst.*, vol. 175, pp. 72–86, 2019.

[59] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *Int. J. Forecast.*, p. 11, 2020.

[60] G. Dudek, P. Pełka, and S. Smyl, "A Hybrid Residual Dilated LSTM and Exponential Smoothing Model for Midterm Electric Load Forecasting," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 1–13, 2021.

[61] F.-K. Sun, C. I. Lang, and D. S. Boning, "Adjusting for Autocorrelated Errors in Neural Networks for Time Series Regression and Forecasting," in *Proc. NeurIPS*, 2021.

[62] Y. Sun *et al.*, "Memory Augmented State Space Model for Time Series Forecasting," in *Proc. IJCAI*, 2022.

[63] A. Vaswani *et al.*, "Attention is All you Need," in *Proc. NeurIPS*, 2017.

[64] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL*, 2019.

[65] T. B. Brown *et al.*, "Language Models are Few-Shot Learners," in *Proc. NeurIPS*, 2020.

[66] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proc. ICLR*, 2021.

[67] N. Wu *et al.*, "Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case," in *Proc. ICML*, 2020.

[68] B. Lim *et al.*, "Temporal Fusion Transformers for interpretable multi-horizon time series forecasting," *Int. J. Forecast.*, vol. 37, no. 4, pp. 1748–1764, 2021.

[69] S. Li *et al.*, "Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting," in *Proc. NeurIPS*, 2019.

[70] S. Wu *et al.*, "Adversarial Sparse Transformer for Time Series Forecasting," in *Proc. NeurIPS*, vol. 33, 2020.

[71] Y. Lin, I. Koprinska, and M. Rana, "SSDNet: State Space Decomposition Neural Network for Time Series Forecasting," in *Proc. ICDM*, 2021.

[72] H. Zhou *et al.*, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," in *Proc. AAAI*, 2021.

[73] H. Wu *et al.*, "Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting," in *Proc. NeurIPS*, 2021.

[74] S. Liu *et al.*, "Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting," in *Proc. ICLR*, 2022.

[75] T. Zhou *et al.*, "FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting," in *Proc. ICML*, 2022.

[76] Y. Liu *et al.*, "Non-stationary Transformers: Exploring the Stationarity in Time Series Forecasting," in *Proc. NeurIPS*, 2022.

[77] Y. Nie *et al.*, "A Time Series is Worth 64 Words:Long-Term Forecasting with Transformers," in *Proc. ICLR*, 2023.

[78] Y. Zhang and J. Yan, "CrossFormer: Transformer Utilizing Cross-Dimension Dependency for Multivariate Time Series Forecasting," in *Proc. ICLR*, 2023.

[79] G. Woo *et al.*, "ETSformer: Exponential Smoothing Transformers for Time-series Forecasting," in *Proc. ICLR*, 2023.

[80] A. Shabani *et al.*, "Scaleformer: Iterative Multi-scale Refining Transformers for Time Series Forecasting," in *Proc. ICLR*, 2023.

[81] **Z. Ouyang**, P. Ravier, and M. Jabloun, "STL Decomposition of Time Series Can Benefit Forecasting Done by Statistical Methods but Not by Machine Learning Ones," *Eng. Proc.*, vol. 5, no. 1, p. 42, 2021.

[82] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Proc. NeurIPS*, 2013.

[83] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks* (Studies in Computational Intelligence). Springer Berlin Heidelberg, 2012, vol. 385.

[84] E. Yurtsever *et al.*, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[85] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. NeurIPS*, 2012.

[86] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, 2012.

[87] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Proc. NeurIPS*, 2014.

[88] G. P. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model," *Neurocomputing*, vol. 50, pp. 59–175, 2003.

[89] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2009.

[90] N. K. Ahmed *et al.*, "An empirical comparison of machine learning models for time series forecasting," *Econom. Rev.*, vol. 29, no. 5, pp. 594–621, 2010.

[91] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and Machine Learning forecasting methods: Concerns and ways forward," *PLOS ONE*, vol. 13, no. 3, e0194889, 2018.

[92] M. Theodosiou, "Forecasting monthly and quarterly time series using STL decomposition," *Int. J. Forecast.*, vol. 27, no. 4, pp. 1178–1195, 2011.

[93] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The M4 Competition: 100,000 time series and 61 forecasting methods," *Int. J. Forecast.*, vol. 36, no. 1, pp. 54–74, 2020.

[94] G. Bontempi, S. B. Taieb, and Y.-A. Le Borgne, "Machine learning strategies for time series forecasting," in *Proc. eBISS*, 2012.

[95] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[96] S. Roberts *et al.*, "Gaussian processes for time-series modelling," *Phil. Trans. R. Soc. A.*, vol. 371, no. 1984, p. 20 110 550, 2013.

[97] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2020.

[98] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[99] S. Makridakis, "Accuracy measures: Theoretical and practical concerns," *Int. J. Forecast.*, vol. 9, no. 4, pp. 527–529, 1993.

[100] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *Int. J. Forecast.*, vol. 22, pp. 679–688, 2006.

[101] **Z. Ouyang**, P. Ravier, and M. Jabloun, "Are Deep Learning Models Practically Good as Promised? A Strategic Comparison of Deep Learning Models for Time Series Forecasting," in *Proc. EUSIPCO*, 2022.

[102] A. Sorjamaa *et al.*, "Methodology for long-term prediction of time series," *Neurocomputing*, vol. 70, no. 16-18, pp. 2861–2869, 2007.

[103] G. Bontempi, "Long term time series prediction with multi-input multi-output local learning," in *Proc. ESTSP*, 2008.

[104] H. Cheng *et al.*, "Multistep-Ahead Time Series Prediction," in *Proc. PAKDD*, 2006.

[105] S. B. Taieb *et al.*, "Long-term prediction of time series by combining direct and mimo strategies," in *Proc. JICNN*, 2009.

[106] K. Cho *et al.*, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proc. SSST*, 2014.

[107] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. ICLR*, 2014.

[108] **Z. Ouyang**, M. Jabloun, and P. Ravier, "Rankformer: Leveraging Rank Correlation for Transformer-based Time Series Forecasting," in *Proc. IEEE SSP*, 2023.

[109] **Z. Ouyang**, M. Jabloun, and P. Ravier, "STLformer: Exploit STL decomposition and Rank Correlation for Time Series Forecasting," in *Proc. EUSIPCO*, 2023.

[110] G. E. P. Box *et al.*, *Time Series Analysis: Forecasting and Control*, 5th ed. John Wiley & Sons, Inc., 2015.

[111] Y. Yaslan and B. Bican, "Empirical mode decomposition based denoising method with support vector regression for time series prediction: A case study for electricity load forecasting," *Measurement*, vol. 103, pp. 52–61, 2017.

[112] H. Hewamalage, C. Bergmeir, and K. Bandara, "Recurrent Neural Networks for Time Series Forecasting: Current status and future directions," *Int. J. Forecast.*, vol. 37, no. 1, pp. 388–427, 2021.

[113] D. Salinas *et al.*, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *Int. J. Forecast.*, vol. 36, no. 3, pp. 1181–1191, 2020.

[114] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," in *Proc. ICLR*, 2018.

[115] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The Efficient Transformer," *Proc. ICLR*, 2020.

[116] C. Spearman, "The Proof and Measurement of Association between Two Things," *Am. J. Psychol.*, vol. 15, no. 1, pp. 72–101, 1904.

[117] M. Blondel *et al.*, "Fast Differentiable Sorting and Ranking," in *Proc. ICML*, 2020.

[118] R. F. Engle, "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica*, vol. 50, no. 4, pp. 987–1007, 1982.

[119] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. NeurIPS*, 2019.

[120] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[121] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proc. SciPy*, 2010.

[122] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[123] P. Virtanen *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nat. Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[124] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proc. ACM SIGKDD*, 2016.

[125] M. B. Priestley, "Evolutionary Spectra and Non-Stationary Processes," *J. R. Stat. Soc. Ser. B Methodol.*, vol. 27, no. 2, pp. 204–229, 1965.

[126] M. B. Priestley and T. S. Rao, "A Test for Non-Stationarity of Time-Series," *J. R. Stat. Soc. Ser. B Methodol.*, vol. 31, no. 1, pp. 140–149, 1969.

[127] R. Dahlhaus, "Fitting time series models to nonstationary processes," *Ann. Stat.*, vol. 25, no. 1, pp. 1–37, 1997.

[128] R. F. Engle and C. W. J. Granger, "Co-Integration and Error Correction: Representation, Estimation, and Testing," *Econometrica*, vol. 55, no. 2, pp. 251–276, 1987.

[129] C. W. J. Granger and R. Joyeux, "An Introduction to Long-Memory Time Series Models and Fractional Differencing," *J. Time Ser. Anal.*, vol. 1, no. 1, pp. 15–29, 1980.

[130] C. W. J. Granger, "Long memory relationships and the aggregation of dynamic models," *J. Econom.*, vol. 14, no. 2, pp. 227–238, 1980.

[131] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *J. Econom.*, vol. 31, no. 3, pp. 307–327, 1986.

[132] E. Ogasawara *et al.*, "Adaptive Normalization: A novel data normalization approach for non-stationary time series," in *Proc. IJCNN*, 2010.

[133] T. Kim *et al.*, "Reversible Instance Normalization for Accurate Time-Series Forecasting Against Distribution Shift," in *Proc. ICLR*, 2022.

[134] N. Passalis *et al.*, "Deep Adaptive Input Normalization for Time Series Forecasting," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 31, no. 9, pp. 3760–3765, 2020.

[135] S. Johansen, "Statistical analysis of cointegration vectors," *J. Econ. Dyn. Control*, vol. 12, no. 2-3, pp. 231–254, 1988.

[136] L. Bauwens, S. Laurent, and J. V. K. Rombouts, "Multivariate GARCH models: A survey," *J. Appl. Econ.*, vol. 21, no. 1, pp. 79–109, 2006.

[137] J. M. Wooldridge, *Econometric Analysis of Cross Section and Panel Data*. The MIT Press, 2010.

[138] G. Boquet *et al.*, "A variational autoencoder solution for road traffic forecasting systems: Missing data imputation, dimension reduction, model selection and anomaly detection," *Transp. Res. Part C Emerg. Technol.*, vol. 115, p. 102 622, 2020.

[139] A. Bibal and B. Frénay, "Interpretability of Machine Learning Models and Representations: An Introduction," in *Proc. ESANN*, 2016.

[140] R. Guidotti *et al.*, "A Survey of Methods for Explaining Black Box Models," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1–42, 2019.

[141] Y. Wang, "Interpretable time series classification," Ph.D. dissertation, University of Rennes 1, 2021, 116 pp.

[142] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Proc. NeurIPS*, 2017.

[143] R. Assaf and A. Schumann, "Explainable Deep Neural Networks for Multivariate Time Series Predictions," in *Proc. IJCAI*, 2019.

[144] R. Assaf *et al.*, "MTEX-CNN: Multivariate Time Series EXplanations for Predictions with Convolutional Neural Networks," in *Proc. ICDM*, 2019.

[145] Y. Wang *et al.*, "Adversarial Regularization for Explainable-by-Design Time Series Classification," in *Proc. ICTAI*, 2020.

[146] K. Fauvel, V. Masson, and É. Fromont, "A Performance-Explainability Framework to Benchmark Machine Learning Methods: Application to Multivariate Time Series Classifiers," in *Proc. IJCAI-PRICAI*, 2021.

[147] C. Wang *et al.*, "Towards Time-Series Feature Engineering in Automated Machine Learning for Multi-Step-Ahead Forecasting," *Eng. Proc.*, vol. 18, no. 1, p. 17, 2022.

[148] M. Feurer *et al.*, "Efficient and Robust Automated Machine Learning," in *Proc. NeurIPS*, 2015.

[149] M. Christ *et al.*, "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)," *Neurocomputing*, vol. 307, pp. 72–77, 2018.

[150] D. Deng *et al.*, "Efficient Automated Deep Learning for Time Series Forecasting," in *Proc. ECML/PKDD*, 2023.

[151] O. Anava *et al.*, "Online Learning for Time Series Prediction," in *Proc. COTL*, 2013.

[152] C. Liu *et al.*, "Online ARIMA Algorithms for Time Series Prediction," in *Proc. AAAI*, 2016.

[153] S. Gultekin and J. Paisley, "Online Forecasting Matrix Factorization," *IEEE Trans. Signal Process.*, vol. 67, no. 5, pp. 1223–1236, 2019.

[154] Y. Matsubara and Y. Sakurai, "Regime Shifts in Streams: Real-time Forecasting of Co-evolving Time Sequences," in *Proc. ACM SIGKDD*, 2016.

[155] D. Boulegane, A. Bifet, and G. Madhusudan, "Arbitrated Dynamic Ensemble with Abstaining for Time-Series Forecasting on Data Streams," in *Proc. Big Data*, 2019.

[156] J. Zuo, K. Zeitouni, and Y. Taher, "Incremental and Adaptive Feature Exploration over Time Series Stream," in *Proc. Big Data*, 2019.

[157] M. Abdar *et al.*, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," *Inf. Fusion*, vol. 76, pp. 243–297, 2021.

[158] B. Wang *et al.*, "Deep Uncertainty Quantification: A Machine Learning Approach for Weather Forecasting," in *Proc. ACM SIGKDD*, 2019.

[159] P. Cui, W. Hu, and J. Zhu, "Calibrated Reliable Regression using Maximum Mean Discrepancy," in *Proc. NeurIPS*, 2020.

[160] S. Kim and H. Kim, "A new metric of absolute percentage error for intermittent demand forecasts," *Int. J. Forecast.*, vol. 32, no. 3, pp. 669–679, 2016.

[161] C. Chen, J. Twycross, and J. M. Garibaldi, "A new accuracy measure based on bounded relative error for time series forecasting," *PLOS ONE*, vol. 12, no. 3, e0174202, 2017.

Zuokun OUYANG

Prédiction de Séries Temporelles : De l'Économétrie à l'Apprentissage Profond

Résumé :
La prédiction des séries temporelles (TSF) est un problème fondamental dans de nombreux domaines, dont la finance, l'économie et la météorologie. Une prédiction précise des valeurs futures d'une série temporelle peut fournir des informations précieuses sur le processus sous-jacent, permettant une meilleure prise de décision et une planification plus efficace.
Dans cette thèse, nous avons exploré en profondeur le problème de la TSF, apportant plusieurs contributions dans les domaines économétriques et de l'apprentissage profond. Notre exploration des méthodes de prédiction économétriques comprenait une revue approfondie des techniques de pointe, y compris, mais sans s'y limiter, les modèles AutoRegressive Integrated Moving Average (ARIMA), Exponential Smoothing State Space Model (ETS) et Vector AutoRegressive (VAR), ainsi que leurs variations respectives. Nous nous sommes également penchés sur les méthodes de décomposition des séries temporelles, y compris la décomposition canonique et la décomposition saisonnière-tendance utilisant LOESS (STL). De plus, nous discutons de la méthode Theta, une méthode de décomposition pour l'analyse des séries temporelles. Cette revue complète a formé la base de notre investigation expérimentale, où nous avons évalué la performance de ces techniques de prédiction avec la décomposition STL comme étape de prétraitement en utilisant les ensembles de données du M3-Competition.
Simultanément, notre enquête a parcouru le paysage complexe des modèles d'apprentissage profond (DL) pour la TSF. Nous avons exploré des modèles clés tels que les MLP, les CNN, les RNN et le mécanisme d'attention. Nous avons ensuite approfondi les défis et les goulots d'étranglement inhérents à la TSF. Une vue d'ensemble détaillée du modèle Transformer et de ses dérivés a fourni des informations sur leurs améliorations par rapport au Transformer standard et à leurs précurseurs. Cela nous a amenés à évaluer trois modèles d'apprentissage profond, DA-RNN, LSTNet et TPA-LSTM, pour les problèmes de prédiction multi-étapes de TSF. Nos résultats mettent en évidence les pièges de certaines stratégies de prédiction multi-étapes, en particulier la stratégie Récursive, et proposent une combinaison réfléchie de stratégies MIMO/MISMO comme solution. En faisant progresser le domaine de la TSF, nous présentons deux nouveaux modèles basés sur Transformer, Rankformer et STLformer, conçus pour les tâches de prédiction à long terme de TSF en combinant les mesures économétriques avec des modèles profonds, démontrant une performance supérieure par rapport aux modèles existants de pointe sur plusieurs ensembles de données et horizons de prédiction.
Un autre aspect de notre contribution est une application web prototype qui démontre pratiquement l'implémentation de nos modèles. L'application, développée avec Python et plusieurs bibliothèques d'assistance telles que Flask, Bootstrap et Plotly, adhère à un modèle de conception de type Modèle-Vue-Contrôleur (MVC), garantissant une maintenance pratique et une expansion potentielle. De plus, l'application est conteneurisée à l'aide de Docker, facilitant un déploiement et une exécution conviviaux sur n'importe quelle machine compatible. L'application offre une interface utilisateur intuitive pour interagir avec les modèles et visualiser les résultats. Cette application est actuellement en phase de test et sera déployée dans un futur proche avec l'intégration de fonctionnalités et de modèles supplémentaires.

Mots clés : Séries Temporelles, Prédiction, Econométrie, Apprentissage Profond, Transformer, Application Web

Time Series Forecasting: From Econometrics to Deep Learning

Abstract:
Time series forecasting (TSF) is a fundamental problem in various fields, including finance, economics, and meteorology. An accurate prediction of future values of a time series can provide valuable insights into the underlying process, enabling better decision-making and planning.
In this thesis, we thoroughly explored the TSF problem, making several contributions in both econometric and deep learning domains. Our exploration of econometric forecasting methods included an in-depth review of state-of-the-art techniques, including but not limited to AutoRegressive Integrated Moving Average (ARIMA), Exponential Smoothing State Space Model (ETS), and Vector AutoRegressive (VAR) models, along with their respective variations. We also delved into time series decomposition methods, including canonical decomposition and Seasonal-Trend decomposition utilizing LOESS (STL). In addition, we discuss the Theta method, a decomposition-based method for analyzing time series. This comprehensive review formed the basis of our experimental investigation, where we assessed the performance of these forecasting techniques with STL decomposition as a preprocessing step using the M3-Competition datasets.
Simultaneously, our investigation navigated the complex landscape of deep learning (DL) models for TSF. We explored key models such as MLPs, CNNs, RNNs, and Attention Mechanism. We then delved into the challenges and bottlenecks inherent to TSF. A detailed overview of the Transformer model and its derivatives provided insights into their enhancements over the standard Transformer and their precursors. This led us to evaluate three deep learning models, DA-RNN, LSTNet, and TPA-LSTM, for multi-step TSF problems. Our findings highlight the pitfalls of certain multi-step forecasting strategies, particularly the Recursive strategy, and propose a thoughtful combination of MIMO/MISMO strategies as a solution. Further advancing the TSF field, we introduce two novel Transformer-based models, Rankformer and STLformer, designed for long-term TSF tasks by combining the econometric measures with deep models, demonstrating superior performance compared to existing state-of-the-art models across multiple datasets and forecasting horizons.
Another aspect of our contribution is a prototype web application that practically demonstrates our models' implementation. The application, developed with Python and several supporting libraries such as Flask, Bootstrap, and Plotly, adheres to a Model-View-Controller (MVC) design pattern, ensuring convenient maintenance and potential expansion. Additionally, the application is containerized using Docker, facilitating user-friendly deployment and execution on any compatible machine. The application offers an intuitive user interface for interacting with the models and visualizing outcomes.

Keywords: Time Series, Forecasting, Econometrics, Deep Learning, Transformer, Web Application

Université d'ORLÉANS

PRISME Laboratoire
12 rue de Blois, 45100 Orléans, France

INSA CENTRE VAL DE LOIRE