Modeling and Interpreting High-Dimensional Spaces

by

Mohammadhossein Toutiaee

(Under the Direction of John A. Miller)

Abstract

Curse of dimensionality in modeling occurs when the subject of study is analyzed in high-dimensional data, while it cannot be easily identifiable in low dimensional spaces. But migrating to higher dimensional spaces causes challenges in modeling and interpretation of the subject. Advances in machine learning, particularly in the form of neural networks, supposedly tackles the challenge of modeling, but such techniques require a plethora of input data for training. Additionally, those techniques can be opaque and brittle when they highly become performant as a result of learning in complex spaces. It is not directly understandable as to why and when they work well, and why they may fail entirely when faced with new cases not seen in the training data. In this dissertation, we tackle those issues by proposing two techniques. (1) In the case of modeling, we propose a novel method that can help unlock the power of neural networks on limited data to produce competitive results. With extensive experiments, we demonstrate that our proposed method can be effective on limited data, and we test and evaluate our method on intermediate length time-series data that may not be suitable for simple neural networks due to lack of data with high-dimensional features. (2) In the interpretation context, we propose a new framework for 2-D interpreting (features and samples) black-box machine learning models via a metamodeling technique. Our interpretable toolset can explain the behavior and verify the properties of black-box models, by which we study the output and input relationships of the underlying machine learning models. We show how our method facilitates the analysis of a black-box, aiding practitioners to demystify its behavior, and in turn, providing transparency towards learning better and more reliable models.

Index words: Responsible Data Science, Time-Series Analysis, Machine Learning, Metamodeling, High-Dimensional Spaces, Knowledge Graphs

Modeling and Interpreting High-Dimensional Spaces

by

Mohammadhossein Toutiaee

M.Sc., The University of Ottawa, Canada, 2013
M.Sc., The University of Georgia, 2020

A Dissertation Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the
Degree

Doctor of Philosophy

Athens, Georgia

2021

Modeling and Interpreting High-Dimensional Spaces

by

Mohammadhossein Toutiaee

Dedicated to my parents, who have loved me, raised me, and taught me encouragement.

Acknowledgments

First, I would like to thank my major professor, Dr. John A. Miller. He is truly an example of a scholar with diligence and integrity. He has inspired me to become an independent researcher and helped me realize the power of critical reasoning. He also demonstrated what a brilliant and hard-working scientist can accomplish. He has provided me constructive criticism which helped me develop a broader perspective on my dissertation.

My sincere thanks must also go to my committee members: Professors Hamid Arabnia, Khaled Rasheed, Budak Arpinar and Yuan Ke. They generously gave their time to offer me valuable comments toward improving my work. In particular, I would like to express my deep gratitude and respect to Prof. Arabnia, who has always supported me through the ups and downs of my Ph.D. journey.

I am most grateful to my lab collaborators for lending me their expertise and intuition to my work: Indrajeet Javeri and Yogesh Chaudhari and all the other current and former AIMS Lab grad students and visitors that I know.

I also extend my gratitude to Dr. Ke's research group members whom I have the pleasure of working with: Dr. Nicole Lazar and Xiaochuan Li.

I owe a special thanks to my family. My wife, Delaram, who has always been my best friend and great companion, loved, supported, encouraged, and helped me get through this long journey in the most positive way. I deeply thank my parents, for always encouraging me to explore my own path, their unconditional trust, timely encouragement, endless patience, and for everything they have given me in my life - all of the support and love. It was their love that raised me up again when I got weary. To my brothers, Ehsan and Amirhesan, for helping me feel confident and brave even from afar.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Every second, large volumes of data are being generated, and many organizations and institutions seek to collect them by various ways. The data usually appears in the format of *record*, consisting of a set of datum, each of which contains a number of fields that hold values. Thus the data naturally form a table or matrix. Or it appears in the form of *text*, consisting of documents that can represent words, sentences or even paragraphs of free flowing text. It can also appear as images, where each pixel in an image represents a feature of the subject. Another format could be time-series data, where data is collected at different points in time and each time point represents a feature, and because data points in time-series are collected at adjacent time periods, there might be correlations between the observations. While collecting such forms of data appears to be an easy task, extracting useful knowledge from them is much harder.

In literature, n usually denotes the number of samples, the number of rows of the table or the number of images, and d or m will denote the number of values in each record or sample, which are called attributes or features.

High-dimensional data refers to the kind of datasets that have two of the following properties:

- The datasets have many features, either collected naturally or as a result of generating new features.
- The number of features reflects the intrinsic complexity of data, and the number of combinations between features reflects the number of abstract representations arising from the system being modeled. For ex-

ample, by definition, an image is considered a record with one feature per pixel. While this definition is true, the surface representation of the image does not necessarily pair with the real underlying complexity of the collections of images. To put it another way, the representations can be artificially chosen which create apparent complexity that is not present visually there.

The advantage of roaming in high-dimensional spaces is capturing patterns that are not apparent in the data. But this exploration requires special algorithms that traditional methods do not provide. With the great success of Machine Learning applications in many domains, practitioners are more interested in applying Artificial Intelligence techniques in such high-dimensional spaces. However, it has become increasingly apparent that while machine learning models are highly performant, they have difficulties in providing predictions when the available data does not suffice. In other words, machine learning methods, in particular deep neural network models, require a plethora of data for training, and if a sufficient amount of data is not fed into them, they tend to be less effective in performing the learning task.

Most machine learning models benefit from regularization techniques to force the learning process to converge to the asymptotic optimal solution. The main concern is how accurate the asymptotic prediction would be in the context of parameters estimation in the new high-dimensional setting. And if there is a chance to bound the error estimation between the estimator $\hat{\theta}$ and the unknown parameters θ^* with high probability, what is the convergence rate for our problem? (Negahban et al., 2012) addressed this question by introducing the notions that are more or less the extension of existing work, which together form a theorem as a solution to the aforementioned question.

1.1 Motivation and Research Objectives

Modeling data in high-dimensional spaces introduces challenges in modeling and interpretation of the subject being modeled. This dissertation seeks to propose solutions for those challenges as we employ machine learning methods for high-dimensional data.

In the context of modeling, we presented a new approach to enrich the space parameters to compensate for data deficiency. We applied and evaluated our methods on two forms of high-dimensional applications, intermediate length time-series data and undersampled video cases, to demonstrate the usefulness of our approaches.

In the interpretation challenge, since machine learning methods become rich in the feature space because of the extensive training process over the data, the issue of model interpretation arises. We introduced a new framework that can decipher rich machine learning models for explaining the behaviour of underlying features after training.

1.2 Contributions and Outlines

To be specific, two forms of datasets have been studied for the purpose of modeling. The first set of datasets were collected from COVID-19 time-series data. This form of data is considered as intermediate length time-series data, and producing accurate forecasts for the COVID-19 Pandemic has been challenging due to the length of the time-series data that are available. There is only a year's worth of data available for the United States. Pandemic modeling and forecasting are important for several reasons. Forecasting is important, so citizens and their governments know what to expect in the next few weeks.

Researchers apply various techniques in time-series analysis and the common modeling including statistical methods and machine learning approaches are studied in chapter 2.

In chapter 3, we studied pandemic modeling (COVID-19) by infusing eXogenous variables into the modeling process. We evaluated and compared several statistical and machine learning models to forecast the pandemic course in the United States, using national and state levels data. We studied the effectiveness of the mobility data in the COVID-19 prediction problem regarding the accuracy and discussed the benefits of including eXogenous variables such as hospitalization, ICU occupancy rate, and the count of patients who require a ventilator mask into a multivariate time-series forecast. The empirical advantages of including such eXogenous variables were justified by our experiments.

In chapter 4, we argued that neural networks can be powerful data modeling techniques in various data-driven problems, but their overall results have not been significantly better than the statistical models, especially for intermediate length times-series data. We reasoned that their modeling capacities are limited in cases where enough data may not be available to estimate the large number of parameters that these non-linear models require. We then presented the wavelet-ANN model that benefits from wavelet transformation functions, a neural network trainer and backcasting strategy for improving forecast values, and we empirically showed that our new method could outperform other competitors.

In chapter 5, we studied the second form of the dataset that appeared as video samples. We addressed data deficiency in the limited video samples via the transfer learning technique. We applied and evaluated our method on the traffic video data collected by (https://www.wsdot.wa.gov, n.d.). This dataset suffers from multiple issues. First, the number of video samples is limited for a certain number of classes, so the task of learning those particular classes is non-trivial. Also, the length and quality of videos are low. Especially, in the presence of severe weather such as "foggy" weather or in a situation where "corrupted" video frames exist in the data. These issues would add to the challenges in the learning tasks, and our proposed method is an attempt to classify videos under such difficulties.

The knowledge graph and its applications were addressed in chapter 6. This chapter described the idea of combining the knowledge graph with machine learning methods to help users improve the accuracy of the systems and expand the range of machine learning capabilities. There are many opportunities in machine learning where the knowledge graphs can be infused to obtain promising results, and a few use cases were provided in this chapter.

In chapter 7, we addressed the challenge of model interpretation by proposing a new framework for 2-D (i.e. features and samples) interpretation of black-box machine learning models. We introduced G-FORSE, a metamodel inheriting the characteristics of the Kriging process to support global interpretation using activeness parameter and a network between samples using correlation function, where both aspects can increase the level of transparency for interpretation.

In chapter 8, the summary of these studies was provided, in an effort to show that this dissertation adds to this growing body of machine learning research with novel approaches to defining, developing, and demonstrating the effectiveness of such methods for modeling and interpreting high-dimensional spaces.

Chapter 2

Literature Review

Time-series data appears in a form of sequence recorded at successive equally spaced points in time, so it is a sequence of discrete-time data. Each time point in the series denotes one feature, and because data points in time-series are collected at adjacent time periods, there might be correlations between features. A year-long time-series data with monthly observations may simply contain 12 features, while a traffic sensor data may record 31536000 $(365 \times 24 \times 60 \times 60)$ observations per second over a year. So, time-series data is considered as high-dimensional data since the time component adds to the complexity.

Since the sequence of observations is unlikely to be independent of one another in time-series data, it is a far more challenging task than just fitting a linear or nonlinear regression model. Therefore, time-series analysis requires special methods to extract meaningful statistics and other characteristics of the data.

This section reviews literature in the field of time-series forecasting via traditional and modern toolsets. The traditional methods refer to a family of models that have been practicing for many years. It can be observed that many studies have shifted focus to applying machine learning models in time-series analysis in very recent years.

2.1 Overview of Traditional Methods

Several common statistical techniques for univariate and multivariate time-series analysis may be applied in time-series data. Although these techniques have been introduced many years ago, they are still popular among the majority of researchers, trader companies and practitioners. The most common use of these techniques are provided in the following:

2.1.1 Random Walk

By definition, a candidate series follows a random walk if the first differences are random (non-stationary). Random Walk (RW) is a common technique in graphical models (Aldous & Fill, 2014), and it is widely used in webpage ranking, image segmentation and time-series analysis. The most practical usage of RW is in the financial market, where it states that the historical trend of a market cannot be used to forecast its future trend. Many studies showed that the RW method applies to most time-series data, especially when the samples have the same distribution and are independent of each other. A Gaussian Random Walk for variable y_t can be written by:

$$y_t = y_{t-1} + \epsilon_t \tag{2.1}$$

where ϵ_t follows Gaussian distribution.

2.1.2 Exponential Smoothing

This family of methods has been formed back in the 1950s, and they are closely related to simple forms of state space and Box-Jenkins models. This method is widely used for the forecasting processes. For one-step ahead forecasting of y_{t+1} in a univariate time-series data $y_t, y_{t-1}, \ldots$, this process had been originally introduced on a basis of exponentially weighted moving average, which is formulated as:

$$\hat{y}_{t+1} = (1-\lambda)\sum_{j=0}^{\infty} \lambda^j y_{t-j}, \quad 0 < \lambda < 1. \tag{2.2}$$

By deduction, the following recursion is obtained:

$$\hat{y}_{t+1} = (1-\lambda)y_t + \lambda\hat{y}_t \tag{2.3}$$

Since the above formula is efficient for both computation and storage, it is widely used in many multivariate forecasting simultaneously. This formula is called *exponential smoothing*. If one substitutes error of $\hat{y}_{t+1}$ by u_t, replaces t by $t-1$ and inserts it in the above formula, yielding:

$$y_t - u_t = (1-\lambda)y_{t-1} + \lambda(y_{t-1} - u_{t-1}), \tag{2.4}$$

that is:

$$\Delta y_t = u_t - \lambda u_{t-1} \tag{2.5}$$

Given that u_t is a series of i.i.d $N(0, \sigma_u^2)$ samples, the recursion formula is deduced to the simple ARIMA model.

2.1.3 Auto-Regressive of Lags p

Perhaps the simplest form of time-series modeling is the *autoregressive* family. This method has been practiced for many years and it is popular among statisticians and economists. This method takes one argument p, indicating the past lags p values. The autoregressive series have a certain number of properties:

- The auto-correlation function (ACF) exponentially decays as lags(p) increase.
- The partial autocorrelation function (PACF) spikes up to certain "p" lags, and it becomes nearly zero.

In addition to these properties, the AR series can be non-stationary in mean or variance, making it popular among other similar methods. The AR series can be formulated and estimated via following system of linear equations:

$$z_t = \phi_0 z_{t-p} + \phi_1 z_{t-p+1} + \ldots + \phi_{p-1} z_{t-1} + \epsilon_t$$

Multiplying by z_{t-k}, taking the expectation and Dividing by γ_0 produces:

$$\rho_k = \phi_0\rho_{k-p} + \phi_1\rho_{k-p+1} + \ldots + \phi_{p-1}\rho_{k-1}$$

The number of parameters or lags p are estimated via the above equation, and it can be used to generate p equations, or one matrix equation.

2.1.4 Auto-Regressive Integrated Moving Average

Box-Jenkins introduced an autoregressive integrated moving average (ARIMA) method for modeling time-series data in their book, back in 1970. They considered trend, seasonality and irregularity components in a univariate time-series y_t. They modeled such problems by removing the trend and seasonal via differencing process, so the result becomes a stationary time-series. Denote:

$$\begin{aligned} \Delta y_t &= y_t - y_{t-1}, & \Delta^2 y_t &= \Delta(\Delta y_t) \\ \Delta_s y_t &= y_t - y_{t-s}, & \Delta_s^2 y_t &= \Delta_s(\Delta_s y_t) \\ &\ldots, & &\ldots \end{aligned} \tag{2.6}$$

The differencing is continued until trend and seasonal effects disappear from the data, yielding a new variable:

$$y_t^* = \Delta^d \Delta_s^D y_t \quad \text{for} \quad d, D = 0, 1, \ldots, \tag{2.7}$$

which we model as a stationary ARMA(p, q) model formulated by:

$$y_t^* = \phi_1 y_{t-1}^* + \ldots + \phi_p y_{t-p}^* + \zeta_t + \theta_1\zeta_{t-1} + \ldots + \theta_p\zeta_{t-q}, \quad \zeta_t \sim N(0, \sigma_\zeta^2), \tag{2.8}$$

with non-negative integers p and q and ζ_t represents a serially independent series of $N(0, \sigma_\zeta^2)$ noises. The above formula can be simplified as:

$$y_t^* = \sum_{j=1}^{r} \phi_j y_{t-j}^* + \zeta_t + \sum_{j=1}^{r-1} \theta_j \zeta_{t-j}, \quad t = 1, \ldots, n, \tag{2.9}$$

where $r = max(p, q+1)$ and for which a certain number of coefficients are zero.

2.1.5 SARIMA

ARIMA model can be extended to a case when seasonality appears in time-series data, so a SARIMA$(p, d, q) \times (0, D, 0)_s$ is formulated by:

$$\phi.[-B^p, \dots, -B^1, 1](1 - B)^d (1 - B^s)^D y_t = \delta + \boldsymbol{\theta}.[B^q, \dots, B^1, 1]\epsilon_t \tag{2.10}$$

This family of models is essentially formulated by $((1 - B)^d)$ and $(1 - B^s)^D$ for respectively controlling the seasonal and regular differencing followed by an ARMA model. A more general case of SARIMA happens when the seasonal autoregressive vector $\phi^s \in \mathbb{R}^p$ and seasonal moving-average parameter vector $\boldsymbol{\theta}^s \in \mathbb{R}^Q$ is added to a SARIMA$(p, d, q) \times (P, D, Q)_s$ model, namely as:

$$\begin{aligned}[\phi.[-B^p, \dots, -B^1, 1]][\phi^s.[-B^{sp}, \dots, -B^s, 1]](1 - B)^d (1 - B^s)^D y_t = \\ \delta + [\boldsymbol{\theta}.[B^q, \dots, B^1, 1]][\boldsymbol{\theta}^s.[B^{sq}, \dots, B^s, 1]]\epsilon_t\end{aligned} \tag{2.11}$$

This form of SARIMA model is also known as *multiplicative* SARIMA model since The whole expression is multiplied by the polynomial term $[\phi.[-B^p, \dots, -B^1, 1]]$.

2.1.6 SARIMAX

Another popular time-series method is Seasonal Autoregressive Integrated Moving Average (SARIMA). This (Arunraj et al., 2016) model is defined as below

$$\varphi_p(B)\Phi_P(B^s)\nabla^d\nabla_s^D y_t = \theta_q(B)\Theta_Q(B^s)\varepsilon_t, \tag{2.12}$$

where y_t is a variable to forecast, $t = 1, 2, \dots$, $\varphi_p(B)$ is a regular AR polynomial of order p, $\theta_q(B)$ is a regular MA polynomial of order q, $\Phi_P(B^s)$ is a seasonal AR polynomial of order P, and $\Theta_Q(B^s)$ is a seasonal MA polynomial of order Q. The differencing operator ∇^d and the seasonal differencing operator ∇_s^D eliminate the non-seasonal and seasonal non-stationarity, respectively.

2.1.7 Time-Series Regression

A univariate time-series with variable y_t can be modeled by the regression model as:

$$y_t = X_t\beta + \epsilon_t, \quad \epsilon_t \sim N(0, H_t), \tag{2.13}$$

for t=1,...,n, where X_t is the $1 \times k$ regressor vector with exogenous variables, β is the $k \times 1$ vector of regression coefficients and H_t is the known variance. The generalized least squares estimator of the regression coefficient vector β is given by:

$$\hat{\beta} = \left(\sum_{t=1}^{n} X_t' H_t^{-1} X_t\right)^{-1} \sum_{t=1}^{n} X_t' H_t^{-1} y_t \tag{2.14}$$

In this setting, Kalman Filter can be applied to effectively estimate $\hat{\beta}$ in a recursive way.

2.1.8 Vector Auto-Regressive

Vector Auto-Regressive model (Zivot & Wang, 2006) is another popular approach to model and predict multivariate time-series. For a p dimensional response vector of interest, say:

$$\mathbf{y}_t = (y_{1,t}, y_{2,t}, \ldots, y_{n,t})^{\mathrm{T}}, \tag{2.15}$$

a vector auto-regressive model of order q, i.e. VAR(q), is defined as

$$\mathbf{y}_t = \delta + \Phi^{(0)}\mathbf{y}_{t-q} + \Phi^{(1)}\mathbf{y}_{t-q+1} + \ldots + \Phi^{(q-1)}\mathbf{y}_{t-1} + \epsilon_t \tag{2.16}$$

where $\delta \in \mathbb{R}^n$ is an intercept vector, $\Phi^{(s)} \in \mathbb{R}^{n\times n}(s = 0, \ldots, p-1)$ are regression coefficient matrices, and $\epsilon_t \in \mathbb{R}^n$ is an error vector.

2.1.9 Minimax Concave Penalty

Since time-series data appears in a high-dimensional format, a penalized linear regression approach can be utilized for forecasting. The method is formulated as follows:

$$Q(\boldsymbol{\beta} \mid \mathbf{X}, \mathbf{z}) = \frac{1}{2N}\|\mathbf{z} - \mathbf{X}\boldsymbol{\beta}\|^2 + \sum_{j=1}^{d} P_\gamma\left(\beta_j; \lambda\right), \qquad (2.17)$$

where $\mathbf{z} \in \mathbb{R}^n$ is the vector of response variables, $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the design matrix of all predictors and $\boldsymbol{\beta} = (\beta_1, ..., \beta_d)^{\mathrm{T}}$ is a vector of unknown regression coefficients. Besides, $P_\gamma(\cdot; \lambda)$ is the Minimax Concave Penalty (MCP) (C.-H. Zhang et al., 2010) which satisfies

$$P_\gamma(\beta; \lambda) = \begin{cases} \lambda|x| - \frac{\beta^2}{2\gamma}, & \text{if } |\beta| \le \gamma\lambda, \\ \frac{1}{2}\gamma\lambda^2, & \text{if } |\beta| > \gamma\lambda. \end{cases} \qquad (2.18)$$

We extensively discuss about this method next.

2.2 Machine Learning Approaches

The deep learning revolution, driven especially by incredible achievements in image recognition technology, encourages a predominately data-driven approach to machine learning and data science techniques. One challenge in using machine learning methods for time-series data is they require a plethora of data and training a deep neural network on intermediate length time-series data is more challenging. This issue will be elaborated on in later sections.

Selected machine learning techniques are provided in the following, although this list can be extended by many other methods:

2.2.1 Neural Network for Time-Series

With the extension of Neural Network models in many applications, we can forecast time-series data with such sophisticated methods. Perhaps a simple architecture of a neural network for univariate time-series data is modeled by a p^{th} order Auto-Regressive AR(p) model. It is a three-layer (one hidden) neural network, where the input layer has a node for each of the p lags, the hidden layer also has p nodes, and the output layer has 1 node:

$$\begin{aligned} \mathbf{x}_t &= [y_{t-p}, \ldots, y_{t-1}] \\ y_t &= \mathbf{w}.\mathbf{f}(\mathbf{\Phi x_t}) + \epsilon_t \end{aligned} \tag{2.19}$$

The $p \times p$ matrix Φ bears the parameters/weights connecting the input and hidden layers, while the p dimensional vector $\mathbf{w}$ bears the parameters/weights connecting the hidden and output layers. There is only one activation function (vectorized $\mathbf{f}$) for the hidden layer.

The time-series data which is a representation of time can be modeled by neural networks such that it provides an implicit functional representation of the time domain. The input layer in neural networks constructs short-term memory via the use of the time-delay approach. Time-delay neural network (TDNN) (Bromley et al., 1993) was studied by many researchers, and it is a simple case of such architecture. In most cases, a single hidden layer is used in the neural networks for modeling time-series data with a single-output node. For p input lag features, q hidden nodes in the hidden layer, and one output node, the total number of parameters in a three-layer neural network is $q(p + 2) + 1$.

2.2.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) has been proposed many years ago to fix some issues that a simple Recurrent Neural Network model may face during the training process, and many experiments have proved that this particular method is actually effective.

2.2.3 Gated Recurrent Unit

In recent years, some groups of researchers have improved the LSTM model by simplifying its architecture, and they introduced Gated Recurrent Unit (GRU) which became popular among the researchers. It adds a second gate, a third parameter matrix and a third bias vector. The two gates in a GRU are the *reset gate* and the *update gate*. The equations below show how information flows through a gated recurrent unit (Cho et al., 2014; Yu et al., n.d.; G.-B. Zhou et al., 2016):

$$
\begin{aligned}
\mathbf{x_t} &= [y_{t-1}, \dots, y_{t-p}] && \text{input} && (2.20)\\
\mathbf{r_t} &= f_1(\Psi[\mathbf{h_{t-1}}, \mathbf{x_t}] + \beta_r) && \text{reset gate} && (2.21)\\
\mathbf{u_t} &= f_2(\Phi[\mathbf{h_{t-1}}, \mathbf{x_t}] + \beta_u) && \text{update gate} && (2.22)\\
\tilde{\mathbf{h}}_\mathbf{t} &= f_3(\Theta[r_t \odot \mathbf{h_{t-1}}, \mathbf{x_t}] + \beta_h) && \text{candidate state} && (2.23)\\
\mathbf{h_t} &= (\mathbf{1} - \mathbf{u_t}) \odot \mathbf{h_{t-1}} + \mathbf{u_t} \odot \tilde{\mathbf{h}}_\mathbf{t} && \text{state} && (2.24)\\
y_t &= \mathbf{h_t}(0) + \epsilon_t && \text{output} && (2.25)
\end{aligned}
$$

The parameters consist of three weight/parameter matrices $\mathbf{\Phi} \in \mathbb{R}^{p\times 2p}$, $\mathbf{\Psi} \in \mathbb{R}^{p\times 2p}$ and $\mathbf{\Theta} \in \mathbb{R}^{p\times 2p}$, as well as three bias vectors $\beta_z \in \mathbb{R}^p$, $\beta_r \in \mathbb{R}^p$ and $\beta_h \in \mathbb{R}^p$. The first two activation functions default to *sigmoid*, while the third activation function defaults to *tanh*.

The vanishing gradient issue in GRU is managed by an update gate and a reset gate. The update gate and reset gate control information that flows into and out of memory, respectively.

2.2.4 Convolutional-LSTM

The Convolutional LSTM or ConvLSTM (Shi et al., 2015) is constructed upon the fully connected LSTM (FC_LSTM), where it extends the idea of having convolutional structures in both the input-to-state and state-to-state transitions. In comparison with other simple RNN network architectures, such as LSTM or GRU, this combined architecture encourages the process to determine the future state of a certain cell in the grid by the inputs and past states of its local neighbors. By stacking multiple ConvLSTM layers and forming an encoding-forecasting structure, it is possible to build an end-to-end trainable model for spatio-temporal casting. Like other RNN models, it has feedback connections, making it useful to process entire sequences of data. The ConvLSTM is formulated as follow:

$$\begin{aligned} i_t &= \sigma(W_{xi}X_t + W_{hi}H_{t-1} + W_{ci} \odot C_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}X_t + W_{hf}H_{t-1} + W_{cf} \odot C_{t-1} + b_f) \\ C_t &= f_t \odot C_{t-1} + i_t \odot tanh(W_{xc}X_t + W_{hc}H_{t-1} + b_c) \\ o_t &= \sigma(W_{xo}X_t + W_{ho}H_{t-1} + W_{co} \odot C_t + b_o) \\ H_t &= o_t \odot tanh(C_t) \end{aligned} \tag{2.26}$$

2.2.5 Encoder-Decoder-Attention-LSTM

The encoder-decoder model uses an LSTM based encoder that learns vector representations from input time-series. These encoded representations have to be of a fixed length. This representation is then fed into an LSTM based decoder whose job is to reconstruct the time-series using the current hidden state and the output of the previous time step (Malhotra et al., 2016). Since the encoded representations have to be of a fixed length, capturing all the information from input series can be challenging (Bahdanau et al., 2014). It is also difficult to capture long term dependencies since previous memories are erased and the model is refreshed with new observations (C. Fan et al., 2019). To combat this problem, attention mechanisms can be incorporated in the decoder. An alignment score is generated for each time step of the encoder and the decoder uses a weighted sum. The alignment scores are then normalized using the softmax function to obtain attention weights. A context vector is then built as the weighted sum of hidden values of the encoder and the attention weights. This context vector is used to compute the current hidden state of the decoder using the hidden state of the previous step and the output of the previous step. Once the hidden state is computed for each time step, the output of the decoder can be computed using softmax on weighted hidden state (Gangopadhyay et al., 2018).

2.2.6 Graph Convolutional Network-LSTM

One of the interesting topics in network science and knowledge graph is Graph Neural Network (GNN) models, which are gradually receiving tons of attention from practitioners. A Graph Convolutional Network (GCN) is a special case of GNN, where it takes the advantage of convolutional layers for aggregating the neighbors' information at each node. In GCN models the entities or nodes are connected by some edges, where those edges can represent distance, correlation and causality (Przymus et al., 2017). The GCN_LSTM captures the spatial dependency of the underlying graph network by the GCN component, and the dynamic dependency of the graph is captured by the LSTM, together forming a hybrid architecture that takes the advantage of combining two complex deep frameworks.

The main idea behind Graph Convolutional Network (GCN) models is that a given input signal (node) can be enriched via information propagation from its neighbors to improve a future prediction task. The neighbors are often defined by constructing a network of inputs where nodes and connections represent features and relations between them. This study (Gilmer et al., 2017) formulated an oracle definition upon the message-passing framework that was inspired by many previously proposed methods. In such settings, the spatial dependency is obtained by the 2-layer GCN architecture through the following formula:

$$f(X, A) = \sigma(\hat{A}\, Relu(\hat{A}\, X\, W_0)\, W_1) \tag{2.27}$$

where X denotes the feature matrix, A denotes the adjacency matrix, W_0 and W_1 respectively denote the weight matrix in the first and second layer.

2.2.7 Transformers

In addition to using popular recurrent neural networks to model our sequential data, we attempt to use other classes of transduction models which are transformer models. Transformer models have been successful in modeling sequential data, such as language data, by achieving state-of-the-art results on machine translations (Vaswani et al., 2017). The transformer models replace the recurrent and the convolutional neural networks by solely relying on multiheaded self-attentions to capture the temporal and complex global dependencies (Vaswani et al., 2017). Such models also allow for parallelization which is ideal for modeling long sequences (Vaswani et al., 2017). The architecture of the transformer models consists of stacked encoders and decoders. The encoder consists of positional embedding, multiheaded self-attentions, and a feed-forward layer (Wu et al., 2020). The decoder consists of positional encoding, multiheaded attention applied to the encoded sequence, masked multiheaded attention applied to the output sequence, and a feed-forward layer (Wu et al., 2020). The transformer models have gained attention recently to model time-series data, such as forecasting seasonal influenza epidemics.

2.3 Generative Adversarial Networks

2.3.1 Introduction

Most machine learning models attempt to minimize a cost function for learning some tasks, so that they would be able to make low to little mistakes on the training set. Now in adversarial machine learning, there are two players of two costs, where each player tries to minimize its cost function and encourages the other one to maximize itself. One question of interest is why is an adversarial network required versus just providing a pool of data for training some tasks? One purpose of using such frameworks is to produce imaginary samples by providing the model an opportunity to practice and perform useful tasks. Generative Adversarial Networks (GAN) learn to produce realistic images via a fictitious competition between an image *generator* and an image *discriminator*, where the generator tries to produce real images, the discriminator collects points for detecting fake images and blocking them. Additionally, there might be some real-world adversary, such that an intelligence aims to "fool" our system into predicting incorrectly, an adversary that can make money by causing other systems to make mistakes, from search engines to face recognition, to self-driving cars are few examples out of many that the system may be vulnerable to be tricked.

In the remaining sections, GAN will be discussed in detail with emphasis on its motivations, assumptions, strengths, and weakness. In the following, some prominent research that has utilized GANs in time-series data are reviewed through intuitions and mathematical formulations.

2.3.2 Generative Adversarial Networks

(Goodfellow et al., 2014) are machine learning models that can imagine new samples. A generative model $\mathcal{G}$ trained on training data $\mathcal{X}$ sampled from some true distribution $\mathcal{D}$ is one which, given some standard random distribution $\mathcal{Z}$, produces a distribution $\mathcal{D}'$ which is close to $\mathcal{D}$, according to a pre-defined metric function.

Generative Adversarial Networks (Goodfellow et al., 2014) are machine learning models that can imagine new samples. If the inputs are a set of images, for instance, they can output entirely new images that are realistic, even though they have never been seen before. Most of the applications for GANs so far have been for images, and the outputs of GANs are totally imaginary. In fact, GAN is drawing a sample from the probability distribution over all hypothetical images matching that description. Thus, GAN can be kept running to produce more images.

Along with several other kinds of generative models, GANs use a differentiable function represented by a neural network as a *generator* network (G). The generator network takes *random noise* as input, then runs that noise through a differentiable function to transform the noise and reshape it to have a recognizable structure. The output of the generator network is a realistic image. The choice of random input noise determines which image will be generated out of the generator network.

Running the generator network with many different input noise values produces many different realistic output images. The goal is for these images to be fair samples from the distribution over real data.

Certainly, the generator network does not start producing realistic images, it has to be trained. The training process for a generative model is very different from the training process for a supervised learning model. For a supervised learning model, we show the model an image of a "traffic light", for example, and we label it in such a way that this is a traffic light. For a generative model, on the other hand, there is no output to associate with each image. We just show many images to the model and ask it to generate more images that come from the same probability distribution.

Most generative models are trained by adjusting parameters to maximize the probability that the generator net will generate the training data set. Since it can be difficult to compute this probability, most generative models approximate it, where a second network, called *discriminator* (D), learns to guide the generator. The discriminator is just a regular neural net classifier.

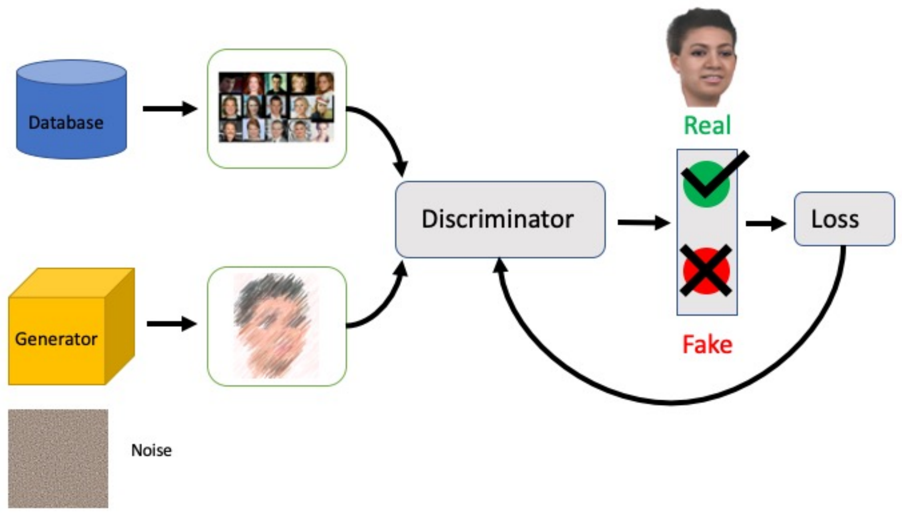


Figure 2.1: GAN architecture.

During the training process, the discriminator is shown real images from training the data half the time, and fake images from the generator the other half of the time. Over time, the generator is forced to produce more realistic outputs to fool the discriminator. The generator takes random noise values z from a prior distribution P_z and maps them to output values x via function $G(z)$. Wherever the generator maps more values of z, the probability distribution over x, represented by the model, becomes denser. The discriminator outputs high values wherever real data density is greater than the density of generated data. The generator changes the samples it produces to move uphill along the function learned by the discriminator. Eventually, the generator's distribution P_z matches the real distribution P_{data}. And the discriminator has to output a probability of one-half everywhere because every point is equally likely to be generated by the real data set as to be generated by the model.

The objective function of D and G are respectively defined as:

$$\max_{D} \quad \mathbb{E}_{x\sim P_{data}}[logD(x)] + \mathbb{E}_{z\sim P_z}[log(1 - D(G(z)))], \tag{2.28}$$

$$\min_{G} \quad \mathbb{E}_{z\sim P_z}[log(1 - D(G(z)))]. \tag{2.29}$$

During the training process, the discriminator is shown real images from training the data half the time, and fake images from the generator the other half of the time. Over time, the generator is forced to produce more realistic outputs to fool the discriminator. The generator takes random noise values z from a prior distribution P_z and maps them to output values x via function $G(z)$. Wherever the generator maps more values of z, the probability distribution over x, represented by the model, becomes denser. The discriminator outputs high values wherever real data density is greater than the density of generated data.

Thus, the GAN (Figure 2.1) is formulated as a $\min_G \max_D V(G, D)$, namely as:

$$V(G, D) = \mathbb{E}_{x\sim P_{data}}[logD(x)] + \mathbb{E}_{z\sim P_z}[log(1 - D(G(z)))]. \tag{2.30}$$

Many variants on GAN are extended by machine learning enthusiasts, while some are very prominent. DCGANs (Radford et al., 2015), Conditional Generative Adversarial Nets (Mirza & Osindero, 2014), Progressively Growing of GANs for Improved Quality, Stability, and Variation (Karras et al., 2017), Bi-GAN (Brock et al., 2018), Style-GAN (Karras et al., 2019), CycleGAN (Zhu et al., 2017), WGAN-GP (Gulrajani et al., 2017), Pix2Pix (Isola et al., 2017), Stack-GAN (H. Zhang et al., 2017), DRAGAN (Kodali et al., 2017) and many others. Among which, we elaborate the discussion on "WGAN" and "DRAGAN" to argue as to how the weaknesses of vanilla GAN can be improved by proposing new loss, and then we touch on "CGAN" which is the core basis of some work in tabular settings, particularly in time-series data.

2.3.3 Issues with GAN

GAN suffers from instability in the training step as a result of nonlinearities in the last layer of discriminator, leading to generates a small range of outputs. GAN applies the softplus nonlinearity to the output of the last matrix multiplication (matmul) in the discriminator when computing the loss. So GAN is relying on tuning the hyper-parameters to alleviate that effect, which is almost impossible in the high-dimensional setting. Another issue with GAN is vanishing the gradients which happens when the discriminator beats the generator, so the generator learns nothing. These issues are the main motivations of the following two important work that will be discussed in the following.

2.3.4 Wasserstein GAN

One alternative to remove the nonlinearities of the output of the last matmul in the discriminator is using a new loss function. WGAN (Arjovsky et al., 2017) introduced a new loss function using Wasserstein distance, which encourages a restriction on the range of the weights, to guarantee Lipschitz continuity. The proposed loss function computes the Wasserstein distance between two distributions, such that the function is continuous and differentiable. The WGAN loss function is defined as follows:

$$\min_{G} \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim P_{data}}[D(x)] - \mathbb{E}_{z \sim P_z}[D(G(z))], \tag{2.31}$$

where $\mathcal{D}$ includes all $1-$Lipschitz functions in the subspace. The authors of WGAN believe that the gradient of the discriminator outperforms GAN, leading to generate better quality of images. In the paper, the authors suggested to use a small value to enforce weight clipping for the Lipschitz effects. Another group of researcher proposed the idea (Gulrajani et al., 2017) of incorporating a gradient regularization in the loss function of WGAN to achieve comparable results. However, some researcher at Google Brain (Lucic et al., 2018) refused the idea of using the gradient penalty, and they showed that WGAN-GP shows inconsistently in generating results in some situations.

2.3.5 On Convergence and Stability of GANs

DRAGAN (Kodali et al., 2017) is yet another GAN fixer, aiming to overcome the gradient vanishing problem by injecting a gradient penalty into GAN. The authors observed that GAN encounters many local peaks within GAN's loss function (1) during the training phase, leading to spike gradients of the discriminator in the vicinity of some data points. They suggested that this issue can be alleviated to some degree by incorporating a gradient penalty on the loss function, namely as:

$$\min_{G} \max_{D} \{(1) - \lambda \mathbb{E}_{\hat{x} \sim p_d + N(0,c)}[(||\nabla D(\hat{x})||_2 - 1)^2]\}, \tag{2.32}$$

The authors of the paper concluded that the proposed loss function enables GAN to be trained faster with fewer mode collapses, encouraging the generator produces better results with effective performance. They evaluated their work on different GANs' architectures and loss functions.

2.3.6 Conditional Generative Adversarial Network

CGAN (Mirza & Osindero, 2014) improved the traditional GAN in such a way that samples are generated via conditioning on $\mathbf{y}$. This conditioning term enables GAN to produce targeted outputs, while this is not the case in the traditional GAN. The method is founded by supporting supplementary information from the real data points for the generator to provide a direction a head of time for producing new samples. Technically, the directive information denoted by $\mathbf{y}$ is provided for both generator and discriminator as extra input layers. This extra piece of information is appeared in the loss function as follows:

$$\min_{G} \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim P_{data}}[logD(\mathbf{x}|\mathbf{y})] - \mathbb{E}_{z \sim P_z}[log(1 - D(G(\mathbf{z}|\mathbf{y})))], \tag{2.33}$$

where p_z and p_d are the distributions of the random noise and samples data, respectively. The WGAN loss function can be modified in a similar way by conditioning $\mathbf{y}$ in the loss function:

$$\min_{G} \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim P_{data}}[D(\mathbf{x}|\mathbf{y})] - \mathbb{E}_{z \sim P_z}[D(G(\mathbf{z}|\mathbf{y}))]. \tag{2.34}$$

It is worth mentioning that the way CGAN is learning on a single category is not the same as training several GANs on a single subject, where the former is much harder than the latter in the context of learning difficulty. This difficulty happens due to the fact that trained CGAN can contain one batch of weights of a single subject with the ability of producing different samples if the subjects' distributions are given. CGAN can also be trained on continuous conditions, which it makes it equipped to be utilized in the tabular data such as time-series analysis in particular.

2.3.7 GAN in Time-Series Data

Before proceeding, it is worth mentioning some of the challenges GAN method would encounter for simulating tabular data, where a group of researchers have collected in their article (Xu et al., 2019). They aimed to generate columns of a dataset $\mathbf{T}$ containing N_c continuous columns $\{C_1, \dots, C_{N_c}\}$ and N_d continuous columns $\{D_1, \dots, D_{N_d}\}$, where each column is considered to be a random variable. These columns are jointly distributed by $\mathbb{P}(C_{1:N_c}, D_{1:N_d})$, and $\mathbf{T}$ is partitioned into training and test sets. But real-world tabular data may contain mix of discrete and continuous, and GANs should apply both *softmax* and *tanh*. GANs have been developed purposely for Gaussian distributions where min-max transformation is applicable, while the continuous values in tabular data are usually not forming a Gaussian trend and vanishing gradient may occur as a result of min-max transformation. Also GANs are not able to capture all the modes in a higher dimension, while most tabular data appears as the multi-modal distribution of continuous columns. Additionally, the main levels in a high-dimensional data with categorical columns appear in the majority of observations ($\%90$), so the dataset suffers from imbalanced labels in data, leading to poor learning in the discriminator. This deficiency for minor levels discourage the network to be learned fairly, thus detecting the rare events (e.g. accidents in the traffic data, anomalies in normal systems, etc.) is challenging. To overcome this issue they introduced *conditional GAN* and *training-by-sampling*, hoping to manage imbalance training data, then the original distribution is reconstructed by:

$$\mathbb{P}(row) = \sum_{k \in D_{i*}} \mathbb{P}_{\mathcal{G}}(row | D_{i^*} = k^*) \mathbb{P}(D_{i^*} = k) \tag{2.35}$$

where k^* is the value from the i^*th discrete column D_{i^*} that must be matched by the generated samples $\hat{r}$.

To help the learner evenly represent all possible values in discrete columns, an assessor is required so that it estimated the difference between the learned conditional distribution $\mathbb{P}_{\mathcal{G}}(row|cond)$ and the real data $\mathbb{P}(row|cond)$. The paper instructed practitioners on how to learn imbalance levels via six steps, which they called it training-by-sampling.

These are some of the main concerns the authors have raised in the context of tabular data modeling, some of them may be managed elegantly by modified GANs.

2.3.8 Time-Series GANs

TimeGAN (Yoon et al., 2019) introduced the idea of combining the adaptability of the unsupervised GAN method with supervised autoregressive models. The authors believe that TimeGAN takes the temporal correlations in time-series data into account, which have not been considered in any of the existing methods. Thus this feature encourages the network to maintain the dynamics of the training data in sampling. TimeGAN has been proposed via connecting four components in three separate parts. The *autoencoding* part containing an embedding function and the recovery function ought to be trained jointly with the sequence generator and sequence discriminator. This enables TimeGAN to concurrently learn to encode, generate and iterate across the time steps. To achieve the "jointly learning" within the components, the embedding $e(.)$ and recovery $r(.)$ functions should reconstruct $\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{1:T}$ of the original data $\mathbf{s}, \mathbf{x}_{1:T}$ accurately from their representations by minimising the reconstruction loss:

$$\mathcal{L}_R = \mathbb{E}_{s,x_{1:T} \sim p}[||\mathbf{s} - \tilde{\mathbf{s}}||_2 + \sum_t ||\mathbf{x}_t - \tilde{\mathbf{x}}_t||_2] \tag{2.36}$$

The generator is supposed to produce synthetic embedding $\hat{\mathbf{h}}_s$, $hat\mathbf{h}_{1:T}$ similar to the training data $\mathbf{h}_s$, $\mathbf{h}_{1:T}$ by minimizing the unsupervised loss:

$$\mathcal{L}_U = \mathbb{E}_{s,x_{1:T}\sim p}[logy_S + \sum_t logy_t] + \mathbb{E}_{s,x_{1:T}\sim \hat{p}}[log(1-\hat{y}_S) + \sum_t log(1-\hat{y}_t)] \tag{2.37}$$

while the discriminator attempts to trick the generator by maximizing the $\mathcal{L}_U$ to provide the generator with the binary adversarial feedback of incorrect classifications $\hat{y}_S, \hat{y}_{1:T}$. To boost the generator, the work introduced a supervised loss $\mathcal{L}_S$ in addition to unsupervised loss $\mathcal{L}_U$, where it enables the generator to capture the conditional distributions in the data by optimizing the likelihood function via:

$$\mathcal{L}_S = \mathbb{E}_{s,x_{1:T}\sim p}[\sum_t ||\mathbf{h}_t - g_{\mathcal{X}}(\mathbf{h}_S, \mathbf{h}_{t-1}, \mathbf{z}_t)||_2] \tag{2.38}$$

where $g_{\mathcal{X}}(\mathbf{h}_S, \mathbf{h}_{t-1}, \mathbf{z}_t)$ estimates $\mathbb{E}_{z_t\sim\mathcal{N}}[\hat{p}(\mathbf{H}_t|\mathbf{H}_S, \mathbf{H}_{1:t-1}, \mathbf{z}_t)]$ with one sample $\mathbf{z}_t$. Overall, the article presented that in generating realistic time-series data, the improvements of TimeGAN architecture over prior work were particularly pronounced.

2.3.9 Simulation Time-Series Data by CGAN

Very little prior work has been done on GANs for time-series data analysis tasks. One reason may be GANs architectures are not relying on the *likelihood function*, so there is nothing to maximized, and consequently, no convergence is guaranteed. Another challenge in using GANs in the tabular data is that it requires a great number of iterations in the training step, leading to the use of the computation resources extensively. On the other hand, GANs are able to approximate any *posterior* distributions without having the closed-form function or making assumptions on the prior, provided that the posterior distribution has zero support. This approximation can be greatly pronounced in classification problems when the label of the events are available. Thus, having conditions on the samples can improve the quality of prediction and simulation, and CGAN is one of the best frameworks for such purposes. CGAN enables users to produce new samples from the same distribution upon which the event is conditioned.

The following is a review of GANs and their applications in tabular data, by collecting some prominent work in the subject. We argue that research directions in GANs can be useful in the context of simulation and approximation. We discuss how GAN networks can support supervised learning, anomaly detection and help detect underlying trends in time-series analysis, laying the cornerstone for more complex patterns of learning.

Some work in the time-series domain evaluated the performance of CGAN in various examples. One study (Fu et al., 2019) applied CGAN on a mixture of Gaussian distributions with destabilized parameters μ and Σ. They employed CGAN to generate a Gaussian Mixture Model (GMM) and compared it to a similar case with T-mixture distributions for performance evaluation. They trained and tested the simulations on both GMMs categorical and continuous, and they showed that one can benefit from CGAN in extrapolation tasks when the conditions are continuous. They extended the usage of CGAN for *Vector Autoregressive* (VAR), by simulating autoregressive and GARCH-type time-series samples. A VAR(p) model indicates a multivariate autoregressive time-series model of order p. They simplified the problem by assuming that the training data is governed by a simple VAR(1) with the following parameter:

$$\mathbf{X}_t = c + a\mathbf{X}_{t-1} + \epsilon_t, \tag{2.39}$$

This VAR model is founded on the assumption that the variables rely on the historical data and on random noises, thus the current time can be conditioned on 1-time-lag values, fitting to the purpose for what CGAN has been essentially designed. The CGAN uses the previous value as a condition to generate the present value. The paper continued the discussion to use CGAN for region switching time-series modeling, where the region switching in time-series data is defined by different dependent relationships from different time periods. The authors then extended the idea to apply CGAN on the GARCH time-series, which has been proposed initially for the structure with dependent variance, and the model is calibrated through the following equation:

$$\sigma^2 = c + \sum_{i=1}^{p} a_i X_{t-i}^2 + \sum_{i=1}^{q} b_i \sigma_{t-i}^2, \quad X_t \sim N(0, \sigma_t^2), \tag{2.40}$$

where $\{a_i, b_i, i = 1, \dots, p\}$, $\{X_t, t = 1, \dots, T\}$, c is the constant and ϵ_t is random noise. Unlike VAR models that emphasize more on the mean values, GARCH models emphasize the variance of the model, thus CGAN acknowledges the condition on the variance of data.

2.3.10 Time-Series Generation with Recurrent Conditional GANs

In this work (Esteban et al., 2017) *Recurrent Network* and GAN have intersected to produce realistic medical images by conditioning recurrent neural network (RNN) on auxiliary information. The authors showed by sample likelihood and maximum mean discrepancy that the RCGANs can generate new samples that are useful for supervised training.

The study started by defining RGAN and RCGAN, which both follow the architecture of vanilla GAN with one difference and that is both the generator and the discriminator have been replaced by RNN. Thus RGAN should be able to take two inputs, one for random seed and the other for conditioning on the generated sequence with extra data. Similarly, the discriminator RNN in RCGAN takes the generated sequence with additional input to produce samples as imaginary or real for every time step. Basically, the discriminator in this setting ought to minimize the difference between the sequence labels and the predictions per time-step via optimizing the cross-entropy metric function $CE(\mathbf{a}, \mathbf{b})$. The discriminator loss for the sequence and the label denoted by $\{X_n, \mathbf{y}_n\}$ where $X_n \in \mathbb{R}^{T \times d}$ and $\mathbf{y}_n \in \{0, 1\}^T$ is defined as:

$$D_{\mathcal{L}}(X_n, \mathbf{y}_n) = -CE(RNN_D(X_n), \mathbf{y}_n). \tag{2.41}$$

The generator, similar to GAN, then aims to make the discriminator predict its outputs as true, thus it tries to minimize the negative $CE(RNN_D(X_n), \mathbf{y}_n)$ between the discriminators' outcomes on produced sequences and the 1s vector, denoted by $\mathbf{1}$ for the true label, namely as:

$$G_{\mathcal{L}}(Z_n) = D_{\mathcal{L}}(RNN_G(Z_n), \mathbf{1}) = -CE(RNN_D(RNN_G(Z_n)), \mathbf{1}), \tag{2.42}$$

where Z_n is sampled independently from the latent distribution $\mathbf{Z}$ in a subspace of $Z_n \in \mathbb{R}^{T\times m}$. Similarly, in RCGAN the auxiliary information $\mathbf{c}_n$ is added at every time-step in the inputs to each RNN, thus it encourages RNN to reuse the conditional information in training.

One of the authors' contributions in this work was to generate realistic ICU data used by medical sectors. Since privacy in medical data might be jeopardized by adversaries, the authors also proposed a *differential private* RGAN model to avoid privacy violation via:

$$P[\mathcal{M}(D) \in S] \leq e^{\epsilon} P[\mathcal{M}(D') \in S] + \delta, \tag{2.43}$$

where $\mathcal{M}(D)$ denotes the GAN training on D, S is the outcomes of GANs and P is the probability of randomness in the mechanism $\mathcal{M}$. They noted that while maintaining the GAN private is critical in medical sectors, making private GANs' results comparable to non-private GANs is challenging.

2.3.11 Anomaly Detection with GAN for Time-Series Data

Anomaly detection is applied in many domains, and time-series data modeling–as one of the popular models with temporal ordering structure–is contributing to the domain significantly. Of interest to classical analysis is to measure when the underlying system deviates from its normal behavior, and this is the main focus of anomaly detection in a complex system. But the problem becomes more challenging when the system has temporal (dynamic) patterns, while there is no obvious solution to capture those possible patterns. Those patterns can be temporary or permanent, thus the problem is nontrivial. Researchers attempt to control the normality of a system by mimicking it, thus it enables them to be vigilant if an anomaly happens. This strategy is the foundation of GAN-AD (D. Li et al., 2018) and is the motivation behind the acquisition and analysis of intrusion events in data streams.

The authors of the paper have proposed two networks in GAN that one learns the normal behavior of the data, and another detects rare events due to intrusion incidents being targeted against the system in an unsupervised fashion. This GAN should be able to directly utilize both the discriminator and the generator to capture those incidents. They claimed that their new framework is able to achieve comparable results. The networks they have selected for GAN are Long Short Term-Recurrent Neural Networks (LSTM-RNN), which is heavily used in modeling time-series data. The loss function they used in the framework is the original one used by GAN. Both networks ought to be trained for imitating the underlying behavior of the system. Once the generator is prepared to produce samples that resemble the normal data, then anything unusual in the testing phase can be reported as anomalies. They formulated their anomaly detection framework for multivariate times-series as follows. $\mathbf{X} = \{x^{(t)}, t = 1, \ldots, T\}$ is an m-dimensional time-series data with length T, then GAN learns the normal time-series dataset denoted by $\mathbf{X}^{real}$ to generate "fake" sample denoted by $\mathbf{X}^{gs}$, appears realistic data. Thus GAN-AD reports anomalies if the testing time-series dataset $\mathbf{X}^{tes}$ deviates from $\mathbf{X}^{real}$. They adopted PCA to reduce the dimension of the data before inputting it into GAN, and the GAN-AD computes the anomaly scores by:

$$S_t^{tes} = \lambda Res(X_t^{tes}) + (1-\lambda) D_{rnn}(X_t^{tes}) \tag{2.44}$$

The anomaly events are labeled by a pre-defined function to detect whether the i^{th} variable of the testing time-series set $\mathbf{X}^{tes}$ at time i has deviated or not via:

$$A_t^{tes,i} = \begin{cases} 1, & \text{if } H(S(x_t^{tes,i}), 1) > \tau \\ 0, & \text{else} \end{cases} \tag{2.45}$$

then the anomaly is reported if the cross-entropy error $H(.,.)$ for the anomaly score is higher than a certain threshold τ.

Chapter 3

Improving Time-Series Forecasting by eXogenous Variables[1]

[1]M Toutiaee et al., Submitted to MiLeTS 2021

Abstract

In this work, we study the pandemic course in the United States by considering national and state levels data. We propose and compare multiple time-series prediction techniques which incorporate auxiliary variables. One type of approach is based on spatio-temporal graph which forecasts the pandemic course by utilizing a hybrid deep learning architecture and human mobility data. Nodes in this graph represent the state level deaths due to COVID-19, edges represent the human mobility trend and temporal edges correspond node attributes across time. The second approach is based on a statistical technique for COVID-19 mortality prediction in the United States that uses the SARIMA model and exogenous variables. We evaluate these techniques on both state and national levels COVID-19 data in the United States and claim that the SARIMA and MCP models generated forecast values by the exogenous variables can enrich the underlying model to capture complexity in respectively national and state levels data. We demonstrate significant enhancement in the forecasting accuracy for a COVID-19 dataset, with a maximum improvement in forecasting accuracy by **64.58%** and **59.18%** (on average) over the GCN_LSTM model in the national level data, and **58.79%** and **52.40%** (on average) over the GCN_LSTM model in the state level data. Additionally, our proposed model outperforms a parallel study (AUG-NN) by **27.35%** improvement in the accuracy on average.

3.1 Introduction

The outbreak of the COVID-19 pandemic from early 2020 until today has resulted in over 170M infected individuals and in over 3.54M deaths worldwide ("WHO Coronavirus (COVID-19) Dashboard", 2021). The ability to forecast the number of infections and deaths is vital to policy makers since they can manage healthcare resources, control disease upsurges, and take preventive actions when necessary to ensure public health safety. As the number of mortality and morbidity in the U.S. continued to rise within 2020, many states enforced the lockdown policy, practiced remote working and imposed social distancing to slow the spread of COVID-19. These policies also affect individual mobility.

On the other hand, the number of cases in hospitals, ICUs and ventilators also increase as a result of the pandemic. Therefore, mobility and hospitalization patterns both at the national and local levels can provide useful measures for predicting the pandemic course, especially when this data is included in the pandemic analysis (Kapoor et al., 2020).

3.2 Related Work

A great amount of research has been conducted since the beginning of the COVID-19 pandemic on forecasting the number of people affected. Early studies such as the one proposed in (Barmparis & Tsironis, 2020) used the SIR model for forecasting the infection rate in different countries. Other work (Fazeli et al., 2020) used ARIMA to predict the daily death rate in different states in the United States. (Javeri et al., 2021) introduced the AUG-NN model that enriches neural network models by augmentation which resulted in significant improvements in the accuracy. They reported the forecast values on the national level data.

Time-series forecasting using Graph Neural Networks (GNN) has been introduced in various domains in the past, however, it is less studied in epidemic disease. For example, (Zhao et al., 2019) used a Temporal Graph Convolutional Network (T-GCN) for traffic prediction while (Matsunaga et al., 2019) used Graph Neural Networks for Stock Market Prediction. GNN based approach for COVID-19 prediction discussed in (Kapoor et al., 2020), generated forecast values by using spatio-temporal mobility data. Although their work is impressive, they reported COVID-19 forecasting on a very small scale, i.e. top 20 most populated counties in the United States.

In this work, we extend the current research of pandemic modeling by proposing novel approaches for predicting daily death cases in the United States on both national and state levels. We introduce a spatio-temporal graph convolutional network that can capture complex dynamics by including mobility patterns across different states and a statistical model that generates forecasts by eXogenous variables. With extensive experiments among proposed methods, we demonstrate the power of eXogenous variables combined with lagged variables within the predictive models and conclude with an analysis of eXogenous variables and their potential in monitoring virus spread.

3.3 Aggregate Mobility Data

The mobility data used in the study is obtained from COVID-19 U.S. Flows (Kang et al., 2020). This dataset consists of dynamic human mobility patterns across the United States in the form of the daily and weekly population flows at three geographic scales: census tract, county and state. The spatio-temporal data is obtained by analyzing, computing, and aggregating the millions of anonymous mobile phone users' visit trajectories to various places provided by "Safegraph". To be specific, we use the daily and weekly state-to-state daily population flow (GeoDS, 2021) starting from January 19, 2020 to January 19, 2021. The data files consist of the unique identifiers, latitudes and longitudes for the origin and destination states, date, visitor flows (estimated number of visitors detected by SafeGraph between two geographic units) and the population flow (estimated population flows between two geographic units, inferred from visitor flows).

3.4 COVID-19 Dataset

The ScalaTion COVID-19 dataset available on Github comprises data about COVID-19 cases, hospitalizations, deaths, etc. in the United States, both at the national [2] and state [3] levels.

The national level data spans from January 13, 2020 to March 7, 2021, which consists of 420 days. We ignore the first 44 days due to missing values and start from February 26, 2020, when the first COVID-19 death in the U.S. was recorded. As a result, the national level time-series studied in this paper contains 376 days. We used the first 236 days as our training set and the rest 140 days as the test set. The state level data spans from January 19, 2020 to January 19, 2021 (3.4), which consists of 367 days. Similarly, we ignore the first 70 days and start

[2]https://github.com/scalation/data/blob/master/COVID/CLEANED_35_Updated.csv

[3]https://github.com/scalation/data/blob/master/COVID/Until%201-14-21/USCOVID_BY_STATE.csv

from March 29, 2020 to avoid missing values. Then, the state level time-series studied in this paper contains 297 days for each state. We used the first 185 days as the training set and the rest 112 days as a test set.

3.5 SARIMAX

Weekly seasonality of daily death increase can be observed using Auto-Correlation Function (ACF), as shown in Figure 3.1. Then, we propose to fit the daily death increase by a Seasonal Autoregressive Integrated Moving Average (SARIMA) (Arunraj et al., 2016) model defined as below

$$\varphi_p(B)\Phi_P\left(B^s\right)\nabla^d\nabla_s^D y_t = \theta_q(B)\Theta_Q\left(B^s\right)\varepsilon_t, \tag{3.1}$$

where y_t is a variable to forecast, i.e., the logarithm of *deathIncrease*, $t = 1, 2, \ldots,$ $\varphi_p(B)$ is a regular AR polynomial of order p, $\theta_q(B)$ is a regular MA polynomial of order q, $\Phi_P\left(B^s\right)$ is a seasonal AR polynomial of order P, and $\Theta_Q\left(B^s\right)$ is a seasonal MA polynomial of order Q. The differencing operator ∇^d and the seasonal differencing operator ∇_s^D eliminate the non-seasonal and seasonal non-stationarity, respectively.

The SARIMA with eXogenous factor (SARIMAX) model is an extension of the SARIMA model in (4.12), which has the ability to include exogenous variables, such as hospitalization and ICU occupancy rate. The SARIMAX model can be defined as:

$$\varphi_p(B)\Phi_P\left(B^s\right)\nabla^d\nabla_s^D y_t = \theta_q(B)\Theta_Q\left(B^s\right)\varepsilon_t + \sum_{i=1}^{n}\beta_i x_t^i, \tag{3.2}$$

where $\{x_t^1, \ldots, x_t^n\}$ are the n exogenous variables defined at time t with coefficients $\{\beta_1, \ldots, \beta_n\}$. Further, we apply a log transformation to categorical variables.

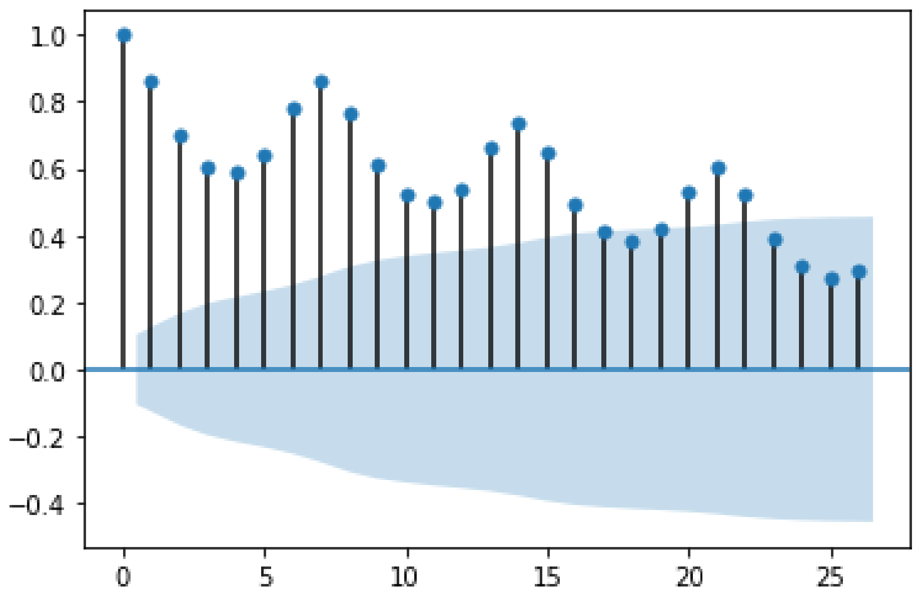


Figure 3.1: ACF plot of the daily death increase

3.6 Minimax Concave Penalty

We also consider a penalized linear regression approach to predict the daily death increase by historical data and exogenous explanatory variables. Denote $z_t = \log y_t - \log y_{(t-7)}$ the weekly log-return of *deathIncrease* at time t and $\mathbf{x_t}$ the exogenous hospitalization variables at time t. We linearly regress z_t on $\{z_{t-h}, \ldots, z_{t-(h+k-1)}\}$ and $\{\mathbf{x}_{t-h}, \ldots, \mathbf{x}_{t-(h+k-1)}\}$, where $k = 14$ and $1 \leq h \leq 14$ is a horizon parameter. The *horizon* refers to the number of days into the future for which forecast values are to be generated. Since the state level hospitalization variables are highly correlated, we first implement a sure independent screening (J. Fan & Lv, 2008) procedure to reduce the dimensionality of $\mathbf{x_t}$. As a result, only the top 7 exogenous variables are included in the following penalized regression model.

$$Q(\boldsymbol{\beta} \mid \mathbf{X}, \mathbf{z}) = \frac{1}{2N}\|\mathbf{z} - \mathbf{X}\boldsymbol{\beta}\|^2 + \sum_{j=1}^{d} P_\gamma\left(\beta_j; \lambda\right), \tag{3.3}$$

where $\mathbf{z} \in \mathbb{R}^n$ is the vector of response variables, $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the design matrix of all predictors and $\boldsymbol{\beta} = (\beta_1, ..., \beta_d)^{\mathrm{T}}$ is a vector of unknown regression coefficients. Besides, $P_\gamma(\cdot; \lambda)$ is the Minimax Concave Penalty (MCP) (C.-H. Zhang et al., 2010) which satisfies

$$P_\gamma(\beta; \lambda) = \begin{cases} \lambda|x| - \frac{\beta^2}{2\gamma}, & \text{if } |\beta| \leq \gamma\lambda, \\ \frac{1}{2}\gamma\lambda^2, & \text{if } |\beta| > \gamma\lambda. \end{cases} \tag{3.4}$$

3.7 Vector Auto-Regressive model

Vector Auto-Regressive model (Zivot & Wang, 2006) is another popular approach to model and predict multivariate time-series. For a p dimensional response vector of interest, say:

$$\mathbf{y}_t = (y_{1,t}, y_{2,t}, \ldots, y_{n,t})^{\mathrm{T}}, \tag{3.5}$$

a vector auto-regressive model of order q, i.e. VAR(q), is defined as

$$\mathbf{y}_t = \delta + \Phi^{(0)}\mathbf{y}_{t-q} + \Phi^{(1)}\mathbf{y}_{t-q+1} + \ldots + \Phi^{(q-1)}\mathbf{y}_{t-1} + \epsilon_t \tag{3.6}$$

where $\delta \in \mathbb{R}^n$ is an intercept vector, $\Phi^{(s)} \in \mathbb{R}^{n\times n}(s = 0, \ldots, p-1)$ are regression coefficient matrices, and $\epsilon_t \in \mathbb{R}^n$ is an error vector.

Similar to the data pre-processing procedure described in Section 3.6, a weekly log-return has been taken to both the response vector and exogenous hospitalization variables to remove the seasonality.

3.8 Random Walk

By definition, a candidate series follows a random walk if the first differences are random (non-stationary). Random Walk (RW) is a common technique in graphical models (Aldous & Fill, 2014), and it is widely used in webpage ranking, image segmentation and time-series analysis. The most practical usage of RW is in the financial market, where it states that the historical trend of a market cannot be used to forecast its future trend. Many studies showed that the RW method is applicable to most time-series data, especially when the samples have the same distribution and are independent of each other. A Gaussian Random Walk for variable y_t can be written by:

$$y_t = y_{t-1} + \epsilon_t \tag{3.7}$$

where ϵ_t follows Gaussian distribution.

3.9 GCN_LSTM

GCN: The main idea behind Graph Convolutional Network (GCN) models is that a given input signal (node) can be enriched via information propagation from its neighbors to improve a future prediction task. The neighbors are often defined by constructing a network of inputs where nodes and connections represent features and relations between them. This study (Gilmer et al., 2017) formulated an oracle definition upon the message-passing framework that was inspired by many previously proposed methods. In such settings, the spatial dependency is obtained by the 2-layer GCN architecture through the following formula:

$$\boldsymbol{m}_i^{(l+1)} = \sum_{j \in \mathcal{N}(i)} \mathcal{F}^{(l)}\left(\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_j^{(l)}\right), \quad \boldsymbol{h}_i^{(l+1)} = \mathcal{G}^{(l)}\left(\boldsymbol{h}_i^{(l)}, \boldsymbol{m}_i^{(l+1)}\right) \tag{3.8}$$

where $\mathcal{F}^{(l)}$ denotes message function, $\mathcal{G}^{(l)}$ denotes node update function, $\boldsymbol{m}_i^{(l)}$ denotes the messages propagated between nodes, and $\boldsymbol{h}_i^{(l)}$ denotes the node representations. This process is implemented in two steps: first, the information flow along with the neighbors; and second, the information is aggregated to determine the updated hidden representations.

$$f(X, A) = \sigma(\hat{A}\; Relu(\hat{A}\, X\, W_0)\, W_1) \tag{3.9}$$

where X denotes the feature matrix, A denotes the adjacency matrix, W_0 and W_1 respectively denote the weight matrix in the first and second layer.

3.9.1 Mobility Network Graph:

In pandemic modeling, we usually analyze the global model through multiple time-series data obtained at the local level, which denotes the spreading dynamics in each region. The forecasting task is usually defined as a regression model that takes in a time-series of $t-k, \ldots, t-1, t$ and emits a single value $t+1$. In a general case, the output can be a series of future time points $t+1, t+2, \ldots$ as generated forecast values. However, the regression model requires an adjustment for modeling human mobility across regions. Mobility data forms a spatial graph, where a region i is denoted by nodes, and every node can be connected to other nodes $j, k, z, \ldots$, and weighted edges represent the strength of relations between the nodes.

3.9.2 Binary Graph

: We constructed a binary adjacency matrix to feed into the trainer. This adjacency matrix has been created by the following steps: (1) the average of mobility data along the time point has been obtained, and (2) if the number of movements from origins to a destination is among the top 20%, the corresponding cell was assigned 1, and 0 otherwise. (3) Finally, the matrix was corrected to be a full-rank matrix by an orthonormal set obtained via the Gram—Schmidt process. We used "matlib" package in R for this purpose.

3.9.3 GCN_LSTM:

Disease epidemic forecasting is a quintessential example of spatio-temporal problems for which we present a deep neural network framework that captures the number of deaths using spatio-temporal data. The task is challenging due to two main inter-twined factors: (1) the complex spatial dependency between time-series of each state, and (2) non-linear temporal dynamics with changing non-pharmaceutical interventions (NPI) such as mobility trends.

We attempt to populate a temporal knowledge graph using a mobility pattern, since people moving around the regions with similar epidemic patterns may contribute more to the forecasting process. The people traveling serve as

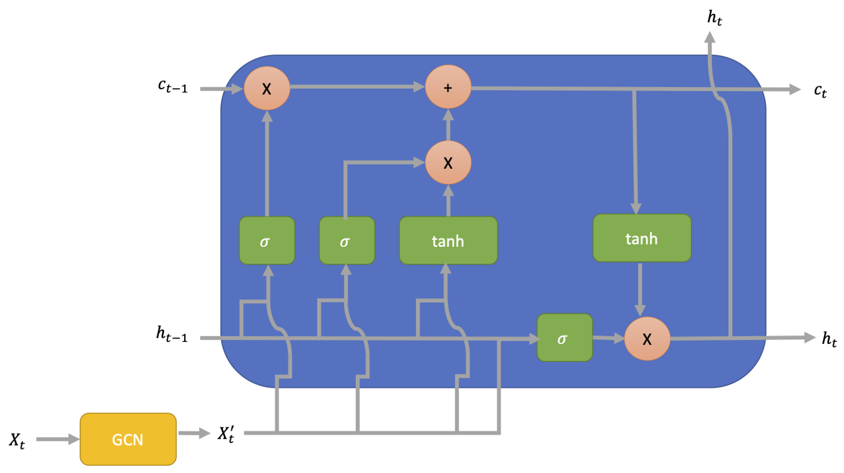


Figure 3.2: An overview of GCN_LSTM architecture.

the ground truth for training a GCN for identifying the underlying graph between sub-regions. Next, the constructed graph embedding from the GCN model is used to feed into an LSTM model to forecast the pandemic in the future. Notice that the graph embedding provides knowledge about the forecasting system, and the LSTM provides a direction for how to leverage the GCN output in the COVID-19 forecasting task.

3.10 Rolling Validation for Multiple Horizons

Classical multi-folds cross-validation approaches are not directly applicable to time-series data due to the existence of serial dependence. Instead, we follow a rolling-validation scheme in this study. To be specific, we reserve the first 60% of days in the time-series as the training set and use the rest 40% of days to make a rolling window forecast. For example, a forecast with horizon parameter $1 \leq h \leq 14$ is obtained using the model trained in the training, an input from a rolling window contains the information from $t-13$ to time t, and an out-of-sample forecast for time $t+h$. Then, we move the rolling window forward by two weeks and repeat the above process. The two weeks ahead forecast policy is suggested by CDC. The forecast accuracy is measure by the symmetric mean absolute percentage error (sMAPE) which is the smaller the better.

For the purpose of simplicity and efficiency in models, the process of rolling forward to forecast the next value in the test set often involves retraining the model by including the first value in the test set in the training set. The first value is then removed from the training set, so the size of the training set remains the same. We trained our architecture only once on the training set to generate the out-of-sample forecast values.

3.11 Hyperparameters and Architectures

3.11.1 GCN_LSTM

The GCN_LSTM architecture used in this experiment consists of two GCN layers followed by one LSTM layer. The model is built using StellarGraph library Data61, 2018. The sizes of each of these layers vary for each horizon. The size of GCN layers falls between 10 and 32 while the size of LSTM layers falls between 150 and 300. Information from 10 previous lags is used for forecasting future instances.

3.11.2 SARIMAX

The exogenous variables used in SARIMAX are the number of daily cases in hospitals and ICUs. The specific model we fit is a SARIMAX$(4, 1, 4) \times (3, 1, 1, 7)$ model with a constant trend.

3.11.3 MCP

We fit an MCP model and consider the serial dependence among daily new cases in hospitals. Each day's new hospitalized count is dependent on the previous 14 days of observations. The regularized parameters in MCP are selected by a multi-fold cross-validation. Since the number of predictors is large, a sure Independence screening procedure is used to remove e uninformative predictors. As a result, 7 predictors are included in the MCP model.

3.11.4 VAR

Similar to SARIMAX model, VAR model contains multiple predictors including "inIcuCurrently", "hospitalizedCurrently", "hospitalizedCumulative" and "onVentilatorCurrently" in addition to "deathIncrease" variable. The hyper parameters are selected by the Bayesian Information Criterion (BIC).

3.11.5 Baseline

We compare the performance of all the statistical models and the GCN_LSTM model with a baseline Random Walk (RW) model. For each model, we make 14 days ahead rolling window forecast as introduced in the rolling-validation section3.10. The forecast performance is measured by the sMAPE score at each horizon $1 \leq h \leq 14$.

The national level forecast results are reported in Table 3.2. The SARIMAX model performs the best on this dataset with the lowest sMAPE score. RW has slightly better results than other models at horizons $h = 7$ and 14.

Table 3.3 shows the forecast results on the state level data. For each state in the U.S., we make 14 days ahead rolling window forecast with horizons $1 \leq h \leq 14$. Then, we aggregate the state level forecast into a forecast for the national level daily death increase. The sMAPE scores on this aggregated forecast are then used to compare all models. MCP model performs the best on this experiment while RW has the worst performance in most scenarios.

Table 3.4 reports the sMAPE scores for different models on a set of representative states. We notice that the scores for each model are much higher than the aggregated results reported in Table 3.3. Also, we find that the GCN_LSTM model performs the worst when the forecasts for these representative states are considered.

3.12 Mortality Prediction Performance

In Tables 3.2 and 3.3, we compare the forecasting performance of SARIMAX with a range of models. We report the sMAPE score for the predicted "deathIncrease". Table 3.2 shows the performance of models over the national level data, and Table 3.3 shows the performance over aggregated state level data for the United States mortality. All performance values were reported for 14 horizons. According to Table 3.2, the SARIMAX model achieves the best sMAPE score at each horizon and outperforms other models on the national level forecast. The best results were achieved by evaluating several combinations, i.e. hospitalizedIncrease, hospitalizedCurrently, etc.

On the other hand, the MCP model performs best on the state level forecast which is evidenced by the results in Table 3.3. This trend is due to the fact that we included more lags ($k = 14$) in MCP compared to SARIMAX, and MCP tends to manage overfitting better than SARIMAX since it uses a regularized term. Although SARIMAX appears to be more accurate than MCP in forecasting national level data, it is inferior to MCP in forecasting state level data, owing to the fact that the global combination might not be the best option for all the states.

Further, we discovered a general pattern that the forecast results can be improved by including exogenous variables. That may explain the success of SARIMAX and MCP in our study. Interestingly, introducing additional variables resulted in worse performance for the SARIMAX model. Table 3.4 compares different models for selected states.

3.13 Conclusion

In this paper, we evaluate and compare several statistical and machine learning models to forecast the pandemic course in the United States, using national and state levels data. We studied the effectiveness of the mobility data in the COVID-19 prediction problem regarding accuracy and discussed the benefits of including exogenous variables. The advantage of SARIMAX and MCP, over other competitive methods such as GCN_LSTM, is they can incorporate exogenous variables such as hospitalization, ICU occupancy rate, and the count of patients who require a ventilator mask into a multivariate time-series forecast. The empirical advantages of including such exogenous variables are justified by our experiments.

Our work adds to this growing body of epidemic disease modeling with a novel approach to combine the multivariate time-series analysis with human mobility data and exogenous variables. We expect the findings in this paper can motive future studies for selecting and incorporating important exogenous variables into time-series modeling to enhance the prediction.

Table 3.1: COVID-19 Dataset: State level

Column	**Description**
date	date: January 19, 2020 to January 19, 2021
death	The cumulative number of deaths
deathIncrease	The number of new deaths on the given date
inIcuCumulative	The cumulative number of patients in ICU
inIcuCurrently	The number of net total patients in ICU on the given date
hospitalizedIncrease	The number of new hospitalizations on the given date
hospitalizedCurrently	The number of net total hospitalizations on the given date
hospitalizedCumulative	The cumulative number of hospitalizations
negative	The cumulative number of negative test results
negativeIncrease	The number of new negative test results on the given date
onVentilatorCumulative	The cumulative number of patients on ventilator
onVentilatorCurrently	The number of net total patients on ventilator on the given date
positive	The cumulative number of positive test results
positiveIncrease	The number of new positive test results on the given date
states	The number of states reporting COVID-19 cases on the given date
totalTestResults	The cumulative number of total test results
totalTestResultsIncrease	The number of total test results on the given date
recovered	The cumulative number of recovered cases

Table 3.2: Multi-Horizon (h) Rolling Forecasts for the Unites States (National Level): Competitive Models (sMAPE).

Horizon	RW	GCN	**SARIMAX**	SARIMA	MCP	VAR
$h = 1$	29.20	28.87	12.77	16.02	15.61	15.08
$h = 2$	48.40	33.81	13.81	19.36	17.13	16.09
$h = 3$	53.50	35.97	13.87	19.13	17.85	16.55
$h = 4$	54.20	32.26	13.88	19.26	18.02	16.53
$h = 5$	50.10	29.83	13.89	20.40	18.02	16.62
$h = 6$	32.20	36.76	13.64	20.49	17.87	16.50
$h = 7$	18.20	34.04	13.50	20.45	17.86	16.75
$h = 8$	31.30	34.29	14.92	22.70	23.03	18.61
$h = 9$	47.40	48.39	15.92	24.92	22.94	19.58
$h = 10$	52.10	54.37	16.33	25.77	23.24	19.84
$h = 11$	53.60	32.07	16.27	25.75	23.39	19.86
$h = 12$	49.90	28.39	16.45	26.27	23.16	20.10
$h = 13$	34.20	35.63	17.12	27.00	22.91	19.79
$h = 14$	24.30	32.84	17.05	26.94	22.97	20.17
Average	41.33	35.54	**14.50**	22.46	20.29	18.01

Table 3.3: Multi-Horizon (h) Rolling Forecasts for the Unites States by State Level Data: Competitive Models (sMAPE).

Horizon	RW	GCN	SARIMAX	SARIMA	**MCP**	VAR
$h = 1$	29.66	24.99	18.22	15.41	16.28	15.40
$h = 2$	50.17	34.42	18.01	16.36	16.09	15.42
$h = 3$	53.98	34.75	18.42	17.31	16.17	16.26
$h = 4$	54.30	34.42	17.53	17.84	16.38	16.32
$h = 5$	51.71	30.41	17.58	17.95	16.61	15.90
$h = 6$	33.89	30.81	18.73	17.76	16.51	16.17
$h = 7$	17.44	35.74	18.04	17.84	16.63	16.65
$h = 8$	31.54	34.62	18.64	21.21	18.17	19.65
$h = 9$	47.21	38.88	18.64	21.89	17.82	20.67
$h = 10$	51.13	44.04	18.52	22.98	19.86	21.07
$h = 11$	52.70	44.15	19.02	23.24	19.40	22.36
$h = 12$	49.94	43.34	29.08	23.62	17.98	20.36
$h = 13$	34.91	38.21	26.88	23.63	18.19	20.31
$h = 14$	23.30	44.46	23.10	23.33	18.32	20.13
Average	41.56	36.66	20.03	20.81	**17.55**	18.33

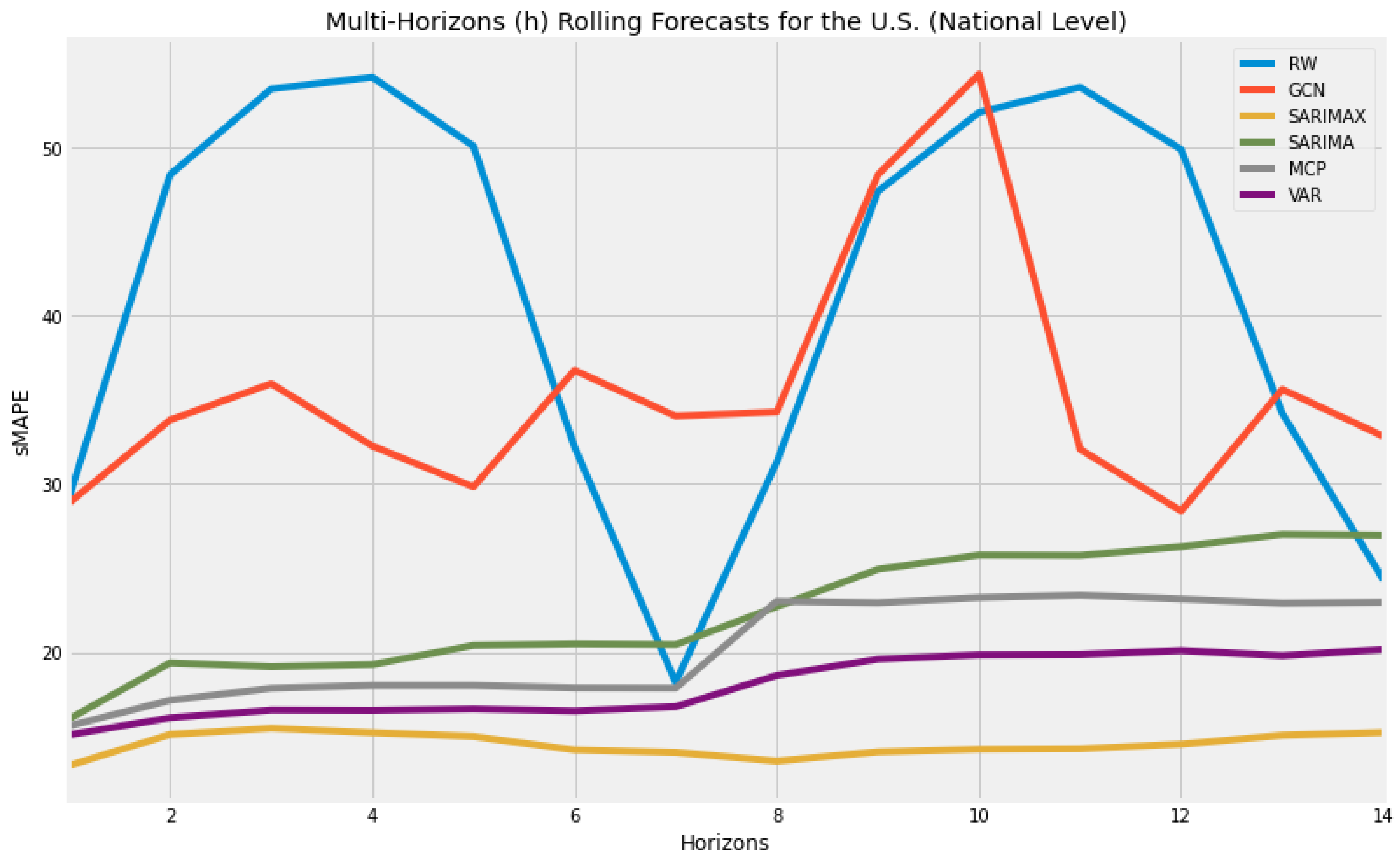


Figure 3.3: Multi-horizons (h) forecasts for the U.S. by the national level data.

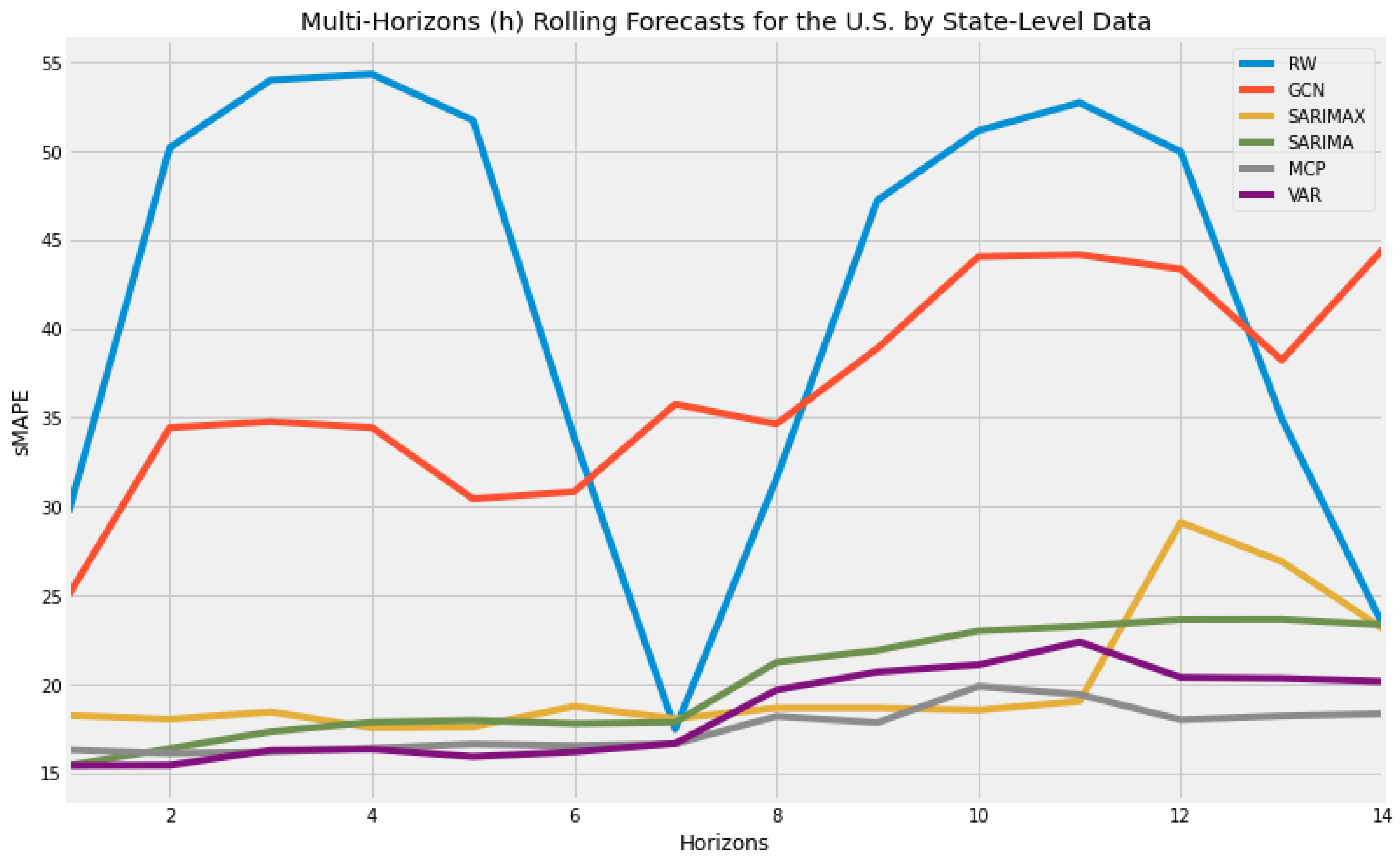


Figure 3.4: Multi-horizons (h) forecasts for the U.S. by state level data.

Table 3.4: Multi-Horizon (h) Rolling Forecasts for Selected States: Random Walk, SARIMA, GCN_LSTM, SARIMAX, VAR and MCP (sMAPE).

State	CA						GA						IL					
Horizons	RW	SR	GCN	**SRX**	VAR	MCP	RW	**SR**	GCN	SRX	VAR	MCP	RW	SR	GCN	**SRX**	VAR	MCP
$h = 1$	54.10	38.60	55.63	35.63	42.02	35.10	68.80	54.40	58.74	59.54	51.87	53.13	41.60	27.40	61.76	30.49	32.72	32.99
$h = 2$	68.50	39.40	57.86	38.02	41.06	34.42	82.00	51.50	62.68	57.52	51.93	55.08	53.00	29.00	64.67	29.25	31.63	32.73
$h = 3$	76.20	40.80	64.93	36.44	41.11	33.93	87.00	52.80	68.37	59.93	51.09	52.49	54.90	30.10	68.11	30.05	31.67	32.43
$h = 4$	75.90	41.70	68.76	37.50	41.06	33.70	87.60	52.60	70.51	56.92	51.06	53.57	55.10	31.00	70.67	29.44	31.38	32.97
$h = 5$	66.10	41.70	68.31	35.31	41.32	34.34	83.30	52.20	70.14	56.42	51.18	53.45	52.10	32.30	71.32	29.17	31.07	32.84
$h = 6$	55.60	40.60	65.53	38.44	41.40	34.98	69.10	51.60	69.05	56.36	51.13	53.62	43.20	31.70	70.93	30.76	31.57	31.94
$h = 7$	41.40	40.90	60.42	40.54	41.48	34.73	48.80	52.50	62.67	54.57	51.54	53.77	31.00	31.40	70.04	30.60	31.47	31.78
$h = 8$	59.60	47.30	55.34	37.00	42.62	41.94	69.70	56.30	59.24	61.46	61.13	63.82	40.40	35.20	70.57	30.52	34.45	36.89
$h = 9$	68.80	47.90	58.54	37.55	43.28	42.44	79.70	56.80	63.7	61.21	60.27	60.75	51.10	37.00	73.42	31.34	33.21	34.82
$h = 10$	74.70	49.90	66.69	42.21	43.44	42.46	82.40	57.60	67.85	61.98	60.85	60.81	54.50	38.10	75.41	31.89	33.47	39.67
$h = 11$	74.50	50.60	68.88	36.79	43.40	43.02	83.70	57.30	72.01	58.91	60.37	62.04	56.20	38.50	76.99	29.48	34.41	36.63
$h = 12$	67.80	51.40	67.8	39.08	43.75	43.10	79.50	57.00	73	59.55	60.56	59.43	52.70	39.90	78.53	30.19	33.87	35.09
$h = 13$	57.40	49.70	64.91	37.79	43.93	42.87	66.10	55.80	71.01	59.46	60.45	60.58	42.40	41.10	78.42	31.59	34.64	35.03
$h = 14$	48.60	50.40	59.42	34.93	44.37	43.80	55.50	56.20	62.24	58.79	60.74	61.46	36.20	39.80	77.51	32.84	35.68	34.83
Average	63.51	45.06	63.07	**37.66**	42.45	38.63	74.51	**54.61**	66.52	58.76	56.01	57.43	47.46	34.46	72.03	**30.54**	32.95	34.33

State	TX						NY						PA					
Horizons	RW	SR	GCN	SRX	**VAR**	MCP	RW	SR	GCN	**SRX**	VAR	MCP	RW	SR	GCN	**SRX**	VAR	MCP
$h = 1$	58.80	29.00	36.01	27.88	28.19	29.53	26.50	25.70	35.04	20.76	24.06	24.62	57.20	46.40	70.86	45.86	46.30	42.96
$h = 2$	76.40	32.60	57.29	25.92	27.50	29.66	25.20	28.20	36.56	20.47	24.46	25.79	69.70	47.20	74.13	46.85	44.87	42.62
$h = 3$	84.30	33.30	70.94	32.02	27.76	29.43	24.50	32.80	37.84	20.25	24.56	25.45	78.50	47.00	77.8	44.13	45.24	42.97
$h = 4$	84.30	33.00	74.1	26.16	27.58	28.57	27.30	37.50	39.69	21.92	24.28	24.93	79.00	46.60	80.19	44.50	46.56	41.88
$h = 5$	75.60	32.60	74.4	34.15	27.86	28.39	26.00	40.40	41.71	22.99	24.65	25.08	75.80	46.90	80.9	45.30	46.23	41.69
$h = 6$	60.70	32.60	62.65	36.06	27.59	28.42	30.00	28.70	42.72	23.61	24.41	25.80	61.60	46.30	80.57	44.05	46.26	42.61
$h = 7$	27.20	32.70	47.52	28.22	27.36	28.43	27.90	31.90	44.08	23.31	25.22	27.32	44.30	46.80	79.94	43.93	45.99	42.14
$h = 8$	57.60	39.40	38.89	33.59	33.47	31.64	31.10	40.10	45.72	33.50	31.00	30.09	55.10	51.30	80.92	46.75	51.03	51.61
$h = 9$	74.20	41.40	58.74	30.72	32.79	32.58	33.40	44.20	47.83	31.02	31.54	29.33	69.30	52.70	82.98	46.59	51.03	55.01
$h = 10$	79.90	42.20	72.86	30.80	32.76	32.26	34.20	49.40	47.65	42.86	30.72	27.41	72.50	52.90	84.92	44.69	49.29	57.48
$h = 11$	79.60	41.90	76.34	30.62	32.68	32.37	37.60	55.00	49.62	29.57	30.49	30.28	76.00	53.60	87.59	46.79	48.18	55.89
$h = 12$	72.00	41.60	76.26	41.85	32.58	32.11	37.60	57.80	52.95	28.17	30.20	32.02	71.40	55.80	87.83	46.22	49.90	54.43
$h = 13$	60.30	41.50	64.47	35.27	32.52	32.17	40.30	46.30	53.56	28.31	30.89	32.96	61.80	54.60	87.33	44.91	48.56	52.90
$h = 14$	33.80	41.40	50.34	27.48	32.13	32.25	42.00	51.30	55.03	32.02	31.59	32.92	48.90	55.10	87.83	45.80	46.71	54.30
Average	66.05	36.80	61.49	31.48	**30.20**	30.56	31.69	40.66	45.00	**27.05**	27.72	28.14	65.79	50.23	81.70	**45.46**	47.58	48.46

Chapter 4

Improving Intermediate Length Time-Series Data by Hybrid Models

Neural Networks are commonly used to capture nonlinearities in time-series data that classical statistical methods are not fully capable of capturing complex patterns. Earlier research was around combining traditional methods with neural networks to capturing linear trends by the traditional part, and the neural networks were trained to learn the remaining non-linearities. Automated Neural Networks took the center stage of forecasting in time-series data after they were ranked the 9th best model in the M3 forecasting competition (Makridakis & Hibon, 2000) back in 2000, whilst they had been abandoned prior to the competition. More recently, several neural network models have been a part of the M4 in 2018 and M5 in 2020 time-series forecasting competitions (Makridakis et al., 2018), (Makridakis et al., 2020a) and have been competitive and, in some cases, better than the traditional statistical models.

However, the most challenging part of the training neural networks is still remaining in short or intermediate-length time-series data such that these networks are not fully able to learn the trend from the limited number of samples. Intermediate length time-series forecasting is one of such areas that neural networks tend to be less competitive than other models. However, multiple studies in time-series forecasting proved that neural networks can be competitive in longer length time-series data, e.g., (Peng et al., 2018; Peng et al., 2019; Qin et al., 2017).

This chapter shows a methodology for modeling intermediate length time-series data that are comparable to the M4 competition average of one thousand data points. Neural networks are often considered as powerful methods for longer time-series data, but they tend to generate worse results than any traditional methods for shorter time-series. COVID-19 datasets collected daily by several public and private institutes are categorized as this intermediate length group. Moreover, a volatile trend in such data makes it challenging for time-series forecasting. Finally, the importance of having accurate forecasts for the pandemic cannot be underestimated.

This chapter provides a comparative study among classical methods and neural networks. In particular, the results presented in this chapter reveal that neural networks perform on par with the traditional statistical methods. However, this work shows that a simple technique in neural networks can improve the results of out-of-sample forecasts.

This chapter also seeks to apply Automated Machine Learning (AutoML) on building the neural network architecture via architecture optimization to identify an optimal and efficient method for the given training task and dataset. Neural Architecture Search (NAS) coupled with data augmentation would boost the performance significantly, as the results presented in this chapter supports that claim.

4.1 Background

Earlier work in time-series modeling by neural networks were around the idea of feeding neural networks by features generated by statistical methods. This includes the top-performing models in M4 competition such as the hybrid method of exponential smoothing and recurrent neural networks (Smyl, n.d.) and the FFORMA (Montero-Manso et al., 2020). These models seek to provide the neural networks with more information to help them capture the non-linear patterns better within the data and thus, enhancing the generalization performance of these models. The data augmentation method used in this work incorporates the forecast values generated from a statistical toolset to construct a time-series that is dual the length of the original time-series, encouraging the neural networks to train on more samples.

Modeling the pandemic course using the COVID-19 dataset seems to be challenging because the length of the time-series data does not suffice. Only now is a year's worth of data available for the United States. The shortness of the time-series data is particularly a problem for machine learning models that require extensive training sets in order to optimize their large sets of model parameters. This work leverages two main categories of methods, i.e. statistical and machine learning models, to study COVID-19 modeling. The first group of models tends to provide a robust and easy-to-implement set of toolsets that are commonly used in time-series data, and they should be used in this analysis as a baseline. The second group, that of machine learning, supports more intense pattern matching that has the potential to produce more accurate forecasts. While using sophisticated machine learning methods generates better results, the overhead of more data, hyper-parameters calibration, architecture optimization and interpretability can introduce adversity.

Pandemic modeling is vital for both policy makers and people because they know what to expect in the next few weeks. The COVID-19 Pandemic is unique in the sense that the last pandemic with this (or more) significance was the 1918 H1N1 Influenza Pandemic (Morens et al., 2010).

4.2 Dataset

COVID-19 Datasets: There are multiple sources for the COVID-19 pandemic in the United States which most of them fork over two primary data sources. Both sources collect information from the public health authorities throughout the country. First, The Johns Hopkins Coronavirus Resource Center (CRC) is a continuously updated source of COVID-19 data have their data repository stored at https://github.com/CSSEGISandData/COVID-19. Secondly, The COVID Tracking Project is a volunteer organization collecting COVID-19 data used by multiple organizations and research groups. This work utilizes the national COVID-19 dataset composed of data collected and stored at https://

covidtracking.com/data. In addition to having COVID-19 case data for testing, hospitalization, patient and death outcomes from all over the United States, the dataset records other information that might be useful for other studies.
COVID-19 Datasets - United States: The ScalaTion COVID-19 dataset https://github.com/scalation/data/tree/master/COVID contains data about COVID-19 cases, hospitalizations, deaths and several more columns in the United States. This dataset is in a CSV file with 420 rows and 18 columns. This work seeks to analyze the mortality rate or daily death increase column using an eXogenous variable by forming a time-series for the neural networks and statistical models. The time-series includes daily death and hospitalization counts in the United States starting from January 13, 2020 to March 7, 2021. The first 44 day records were eliminated to form a set of observations with 376 days. Our models input time-series data starting February 26, 2020 when the first deaths due to COVID-19 in the United States were recorded.

This study took the entire dataset of the COVID tracking project as the data has stopped collecting new records on the 7th of March 2021. Ending the dataset on March 7, 2021 has the modeling advantage that vaccination effects may be ignored, but after this time, models should take vaccinations under consideration. Since the vaccination data does not suffice for this type of study, so we ignore including it at this point. In addition to the COVID Tracking project, there are other resources for further data such as the Centers for Disease Control and Prevention (CDC) ("United States COVID-19 Cases and Deaths by State over Time", n.d.) and National Center for Health Statistics (NCHS) ("Provisional Death Counts for Coronavirus Disease 2019 (COVID-19)", 2021).

4.3 Data Preprocessing

As part of the analysis, the time-series data was smoothed to eliminate the outliers that fall 3.5 standard deviations away from the rolling mean. The rolling mean is the 6 time points local average, such that we consider 3 time-points before and after the current point. Observations within 3.5 standard deviations are left unchanged, while the observations which are considered as outliers are replaced by the Kalman Smoothing using a Local Level Model (Commandeur & Koopman, 2007).

4.4 Lag Selection

In time-series modeling, it is common to use lagged features as inputs to the models. A lag value is the target value from any of its previous values. For example, the k^{th} lag value of y_t will be y_{t-k}. We need to define models such that they look for p previous lags while estimating their parameters. The choice of p needs to be determined before taking any model under consideration. Choosing a particular value for p depends on the characteristics and dynamics of the time-series and the model. We determine the optimal number of lags to be 22 using the Partial Auto-correlation function (PACF). Since, the PACF eliminates the effect of correlation with all earlier lags, significant correlations at lag 22 indicate we can use up to 22 lags for our modeling and also eliminate any in between the lags which are not significant contributors.

4.5 Related Work

Neural Networks have been popular among practitioners for competing with other statistical toolsets for a long time. Back in the year of 2000, the only rival in the M3 competition among other 23 statistical models was a neural network method (Makridakis & Hibon, 2000) The concept of Automated Artificial Neural Networks (ANN) was first introduced in this competition (Balkin & Ord, 2000) where an automated procedure for the selection of neural network architecture proved useful. That event encouraged the community to use AutoML techniques such as NAS in the context of time-series forecasting. However, the automatic architecture selection tends to be less effective than the other statistical models considered in the case of shorter length time-series.

But the community witnessed a greater number of neural network-based models, triggered by the M4 time-series competition, while some of them were able to outperform other statistical methods. One of the greatest impacts of such competitions was introducing the "hybrid" approach which interleaved both statistical and machine learning features (Makridakis et al., 2018, 2020b). The winner of the M4 competition (Smyl, n.d.) used a Dynamic Computational Graph Neural Network system to combine an Exponential Smoothing (statistical model) with Long Short-Term Memory (LSTM) networks to provide a single hybrid but powerful method to generate forecasting values 10% (in sMAPE) better than the combination benchmarks considered in the competition.

The second place team, FFORMA (Montero-Manso et al., 2020), utilized a combination of 7 statistical methods and 1 ML method, and the weights for averaging/combining such methods also used an ML algorithm. This encouraged the time-series research community to develop techniques that use a combination of machine learning models with time-series features as indicated in (Makridakis et al., 2018). In an effort to prove the usefulness of neural networks over intermediate length time-series, we compare such networks with statistical benchmarks and with extensive experiments, we show that using data augmentation can help boost the performance of such neural network models.

In addition to Fully Connected Neural Networks, a certain number of Recurrent Neural Network (RNN) based frameworks have also been introduced for mid-length time-series forecasting as referred by (Rangapuram et al., 2018; Y. Wang et al., n.d.). More recent studies such as (Hewamalage et al., 2021), show the application of RNNs on several short and intermediate length time-series where the results are better than statistical models for some datasets while the RNNs performing worse than ARIMA and ETS models on diverse datasets such as the M4, which consists of 48,000 time-series.

Some tweaking in the Convolutional Neural Networks (CNN) have resulted in broader percolation to earlier samples. This technique was cited by (Borovykh et al., 2017) and they used the dilated convolutions for that purpose. They claimed that CNN performs on par with the recurrent-type network, whilst requiring significantly fewer parameters and computation to achieve comparable performance.

"Attention Mechanisms" studied by many researchers in recent years allow the network to directly concentrate on important time points in the past. Application of such networks in longer length time-series shows improvements over

RNN-based competitors which has been reflected in (C. Fan et al., 2019; S. Li et al., 2019; Lim et al., 2019). This type of network often employs an encoder to summarize historical information and a decoder to integrate them with known future values (C. Fan et al., 2019; Lim et al., 2019). However, there has been little evidence for the performance of such networks on shorter and intermediate length time-series data.

Many studies in neural networks have been conducted in image and text domains, while intermediate length time-series data have been less studied by such networks. One reason is that such techniques require a plethora of input data for training, and mid-length time-series data would not suffice for training. In general, we expect that our new proposed hybrid technique for forecasting intermediate length time-series problems will provide interesting avenues for future studies.

4.6 Modeling Studies for COVID-19

There are many studies in pandemic modeling that forecast morbidity and mortality rates around the world. The study by (Kumar & Susan, 2020) attempted to forecast the spread of the disease in 10 largely affected countries including the United States. They applied ARIMA and Prophet time-series forecasting models on the number of positive cases and deaths, and they showed that ARIMA performed better than FBProphet (Taylor & Letham, 2018) on a scale of different error metrics such as MAPE and RMSE. Another similar study that applied an ARIMA is found in (Ilie et al., 2020) where they extensively tested and tuned different parameters to minimize the prediction error, and they discussed that ARIMA models are appropriate for generating forecasts for the pandemic crisis.

The GWO-LSTM model proposed by (Prasanth et al., 2020) was an attempt to forecast the daily new cases, cumulative cases, and deaths for India, the USA, and the UK. In their work, they showed that their hybrid architecture obtained a better performance compared to the baseline (ARIMA), and they calibrated the LSTM component by Grey Wolf Optimizer (GWO).

4.7 ARIMA

Box-Jenkins introduced an autoregressive integrated moving average (ARIMA) method for modeling time-series data in their book, back in 1970. They considered trend, seasonality and irregularity components in a univariate time-series y_t. They modeled such problems by removing the trend and seasonal via differencing process, so the result becomes a stationary time-series. Denote:

$$\begin{aligned} \Delta y_t = y_t - y_{t-1}, \qquad \Delta^2 y_t =& \Delta(\Delta y_t) \\ \Delta_s y_t = y_t - y_{t-s}, \qquad \Delta_s^2 y_t =& \Delta_s(\Delta_s y_t) \\ \ldots, \qquad\qquad & \ldots \end{aligned} \tag{4.1}$$

The differencing is continued until trend and seasonal effects disappear from the data, yielding a new variable:

$$y_t^* = \Delta^d \Delta_s^D y_t \quad \text{for} \quad d, D = 0, 1, \ldots, \tag{4.2}$$

which we model as a stationary ARMA(p, q) model formulated by:

$$y_t^* = \phi_1 y_{t-1}^* + \ldots + \phi_p y_{t-p}^* + \zeta_t + \theta_1 \zeta_{t-1} + \ldots + \theta_p \zeta_{t-q}, \quad \zeta_t \sim N(0, \sigma_\zeta^2), \tag{4.3}$$

with non-negative integers p and q and ζ_t represents a serially independent series of $N(0, \sigma_\zeta^2)$ noises. The above formula can be simplified as:

$$y_t^* = \sum_{j=1}^{r} \phi_j y_{t-j}^* + \zeta_t + \sum_{j=1}^{r-1} \theta_j \zeta_{t-j}, \quad t = 1, \ldots, n, \tag{4.4}$$

where $r = max(p, q+1)$ and for which a certain number of coefficients are zero.

4.8 SARIMA

ARIMA model can be extended to a case when seasonality appears in time-series data, so a SARIMA$(p, d, q) \times (0, D, 0)_s$ is formulated by:

$$\phi.[-B^p, \ldots, -B^1, 1](1-B)^d(1-B^s)^D y_t = \delta+\boldsymbol{\theta}.[B^q, \ldots, B^1, 1]\epsilon_t \quad (4.5)$$

This family of models is essentially formulated by $((1 - B)^d)$ and $(1 - B^s)^D$ for respectively controlling the seasonal and regular differencing followed by an ARMA model. A more general case of SARIMA happens when the seasonal autoregressive vector $\phi^s \in \mathbb{R}^p$ and seasonal moving-average parameter vector $\boldsymbol{\theta}^s \in \mathbb{R}^Q$ is added to a SARIMA$(p, d, q) \times (P, D, Q)_s$ model, namely as:

$$\begin{aligned}[\phi.[-B^p, \ldots, -B^1, 1]][\phi^s.[-B^{sp}, \ldots, -B^s, 1]](1 - B)^d(1 - B^s)^D y_t = \\ \delta + [\boldsymbol{\theta}.[B^q, \ldots, B^1, 1]][\boldsymbol{\theta}^s.[B^{sq}, \ldots, B^s, 1]]\epsilon_t\end{aligned} \quad (4.6)$$

This form of SARIMA model is also known as *multiplicative* SARIMA model since The whole expression is multiplied by the polynomial term $[\phi.[-B^p, \ldots, -B^1, 1]]$.

4.9 Artificial Neural Networks

Neural networks are powerful methods to learn complex patterns in the data, owing to the fact that they learn patterns via hidden layers and several neurons with non-linear activation functions in each hidden layer. If the number of hidden layers is properly optimized, a number of training samples are adequately available and other hyperparameters are correctly set, neural networks can serve as powerful data modeling techniques in the machine learning domain. However, such models tend to be less effective over intermediate length time-series data. This is the case in forecasting COVID-19 data since the length of the time-series is still shorter than ideal for training neural networks. Thus, we attempted to utilize a simpler architecture in this work, and we used AutoML to search over the space for better models, and we used data augmentation to compensate for data deficiency issues.

A simple type of Neural Network generalizes an AR(p) model. We use a three-layer (two hidden) neural network, where the input layer has a node for each of the p lags and the current time t. This architecture was used by (Javeri et al., 2021). For p input lag features, q hidden nodes in the hidden layer and one output node, the total number of parameters in a three-layer neural network is $q(p+2)+1$.

$$\begin{aligned} \mathbf{x}_t &= [y_{t-p}, \ldots, y_{t-1}, t] \\ \mathbf{y_t} &= W_3^t\, \mathbf{f_2}(W_2^t\, \mathbf{f_1}(W_1^t \mathbf{x_t} + \mathbf{b_1}) + \mathbf{b_2}) + \mathbf{b_3} + \epsilon_\mathbf{t} \end{aligned} \tag{4.7}$$

where $W_1 \in \mathbb{R}^{(p+1)\times n_h}$, $W_2 \in \mathbb{R}^{n_h \times n_h}$ and $W_3 \in \mathbb{R}^{n_h \times n_o}$ are the weight matrices and $b_1 \in \mathbb{R}^{n_h}$, $b_2 \in \mathbb{R}^{n_h}$ and $b_3 \in \mathbb{R}^{n_o}$ are the bias vectors. Since the last layer uses Identity function as the activation function, so there are only two activation functions (vectorized $\mathbf{f_1}$ & $\mathbf{f_2}$) for the hidden layers.

4.10 Gated Recurrent Unit (GRU)-based Autoencoders

One of the popular architectures among the practitioners is Autoencoders. The deep architecture inside enables Autoencoders to generate compressed feature vectors in between the layers used for reconstructing the input data. Autoencoders learn in such a way that multiple layers to transform the input data into a latent space represented by a lower dimensional space than the feature space. the latent space is transferred to the next layers to be decoded back to the input layer. This process enables us to enrich the feature space of the data, leading to a better modeling of the training samples. The latent or hidden space is also called the compressed knowledge representation of the original input data. One can benefit from the dimension reduction property providing by an autoencoder as the constrained knowledge representation forces the autoencoders to generate important features from the input samples required to model the given input. The intermediate feature vectors are usually fixed-length, however, their size can be calibrated, and choosing a suitable architecture is critical in the modeling process.

GRU-based autoencoders (GRU-AE) are autoencoder architectures that benefit from GRUs as the modeling layers in the encoder and decoder levels of the network. The sequential modeling property enables GRUs to generate the knowledge representation. Although autoencoders are mostly used to reconstruct the original input, (Javeri et al., 2021) used Fully Connected Layer added at the output of autoencoder to use such networks for modeling the time-series data. This model is one of the serious competitors in this study, and the results generated by this architecture were provided in the Table 4.2.

4.11 LSTM with Convolutional Layer

Using convolution layers inside LSTM cells is the idea that has been explored by researchers, and (Shi et al., 2015; Sutskever et al., 2014) reflected that in their studies. This type of architecture enables the training process to benefit from the advantages of both models while typically achieving results better than individual CNN and LSTM. Time-series data may be a good candidate for CNN models since local information is supposedly relevant for the analysis of the time-stamped data. However, this relevancy is short-term due to the constraint by the size of convolutional kernels. However, LSTM coupled with a convolution layer can store and output information during the training process by capturing the long-term time dependency of input data ConvLSTM replaces matrix multiplication with convolution operation at each gate in the LSTM cell, leading to capture informative features by convolution operations and provide the LSTM cell with sequenced data. This is the major difference between a classical LSTM and a ConvLSTM. Inside the LSTM equations, all weight matrix multiplications are converted into convolution operations. By stacking multiple ConvLSTM layers and forming an encoding-forecasting structure, it is possible to build an end-to-end trainable model for spatio-temporal forecasting.

4.12 Wavelet-based Neural Network Models

The wavelet transformation is commonly used in signal processing, and it can generate information in both the time and frequency domain with a higher resolution. Wavelets have appeared at the beginning of the eighties (Grossmann & Morlet, 1984) as a method for processing seismic data. This type of transformation is not available in traditional transformations.

A certain number of applications of wavelets in statistics are including but are not limited to denoising, nonparametric function estimation, data compression, and process synthesis. The wavelet-based neural network for time-series data can appear in two approaches. The first approach seeks to transform the signal into different sets so that they are empirically easy to process. Two types of wavelet transformation are Continuous Wavelet Transform (CWT) and Discrete Wavelet Transform (DWT). DWT considers the series as a range of integers, while CWT treats the time-series as the real axis.

In case of DWT, translation and dilation steps are uniformly discretized via following formula:

$$\psi_{m,n}(t) = a^{-m/2}\psi(a^{-m}t - n) \tag{4.8}$$

The above expression can be simplified for computational purposes, so Dyadic Wavelet Transform is utilized. In the dyadic case a is chosen to be equal to 2, yielding the following translation- dilation equation:

$$\psi_{j,n}(x) = 2^{-j/2}\psi(2^{-j}x - n) \tag{4.9}$$

where j denotes the level of scale (or dilation), and n denotes the translation (or shifting) where $j, n \in \mathbb{Z}$. The above transformation is a shift variant, and it requires the fixed series X of length 2^j. Therefore, Maximal Overlap Discrete Wavelet Transform (MODWT) is used which is appropriate for all sample sizes N, while translation invariant too.

Decomposition contains low-pass and high-pass filters through which wavelet detail and approximation sets are obtained by decomposition algorithm tree. This process encodes raw signals with useful and distinctive features that are informative for further analysis. Overall, the approximation step is the low-frequency or high-resolution part, and

In general, the approximation part is the low-frequency or high-resolution component, and the high-frequency or low-scale part is obtained by transformed details. Low and high frequency filters contribute to the prediction by different roles. The global trend can be captured by wavelet approximate component that is quite identical with the raw series, and as the level of approximation increases, the quality of the information in the original signal decreases. On the flip side, detail parts may be useful for detecting the information that is lost. Haar, Daubechies, and Morlet are only a few examples of wavelet basis functions that are commonly found in literature (Bultheel et al., 1995; Percival & Walden, 2000; Vidakovic, 2009).

Figure 4.1 illustrates how wavelet-ANN generates forecast values, similar to the first approach where TDNN is employed to capture individual detail and approximate part of the decomposed signals. One of the motivations of using ANN, wavelet details and approximate part is the presence of nonlinearities in sub-series. In the case of nonlinearities in the time-series, the MODWT can provide a simpler series by decomposing it into its sub-frequencies. This encourages the ANN to model the details and approximate parts more effectively so that the overall forecasting accuracy improves. One positive side-effect of using wavelet-ANN over a single ANN method is a tendency of wavelet decomposition to be more sensitive to the original nonlinear pattern by dividing the raw series into multiple sub-frequencies and further learning global and local features of the series. This procedure can also be named "wavelet denoising", as the approach attempts to retrieve back the localized information loss in forecasting.

Different wavelet functions can also be plugged in a neural network model for various objectives of function approximation or estimating outputs. Wavelet basis functions, therefore, serve as an activation function in the hidden neurons, bearing the concept of wavelet neural network (WNM).

For example, a feed-forward neural network, with one hidden layer and one output layer, may use the activation functions that are formulated on an orthonormal wavelet basis. These neurons are referred to as wavelet network (or wavelon), and the output for single-input wavelon is as follows:

$$\Psi_{\alpha,t}(x) = \Psi\left(\frac{x-t}{\alpha}\right) \tag{4.10}$$

where α and t are respectively scale and shift parameters. For single-input–single-output wavelet neural network, if the hidden layer contains λ wavelons, then the output is weighted sum of wavelon outputs, namely as:

$$y = \sum_{i=1}^{\lambda} w_i \Psi_{\alpha_i, t_i}(x) + \bar{y}, \tag{4.11}$$

where $\bar{y}$ is added to the expression to manage functions with nonzero mean (since the wavelet function w(x) is zero mean).

4.13 Fitting SARIMAX Model

Weekly seasonality of daily death increase can be observed using Auto-Correlation Function (ACF). Then, we propose to fit the daily death increase by a Seasonal Autoregressive Integrated Moving Average (SARIMA) (Arunraj et al., 2016) model defined as below

$$\varphi_p(B)\Phi_P\left(B^s\right)\nabla^d\nabla_s^D y_t = \theta_q(B)\Theta_Q\left(B^s\right)\varepsilon_t, \tag{4.12}$$

where y_t is a variable to forecast, i.e., the logarithm of *deathIncrease*, $t = 1, 2, \ldots$, $\varphi_p(B)$ is a regular AR polynomial of order p, $\theta_q(B)$ is a regular MA polynomial of order q, $\Phi_P\left(B^s\right)$ is a seasonal AR polynomial of order P, and $\Theta_Q\left(B^s\right)$ is a seasonal MA polynomial of order Q. The differencing operator ∇^d and the seasonal differencing operator ∇_s^D eliminate the non-seasonal and seasonal non-stationarity, respectively.

The SARIMA with eXogenous factor (SARIMAX) model is an extension of the SARIMA model in (4.12), which has the ability to include exogenous variables, such as hospitalization and ICU occupancy rate. The SARIMAX model can be defined as:

$$\varphi_p(B)\Phi_P\left(B^s\right)\nabla^d\nabla_s^D y_t = \theta_q(B)\Theta_Q\left(B^s\right)\varepsilon_t + \sum_{i=1}^{n}\beta_i x_t^i, \tag{4.13}$$

where $\{x_t^1, \ldots, x_t^n\}$ are the n exogenous variables defined at time t with coefficients $\{\beta_1, \ldots, \beta_n\}$. Further, we apply a $\log$ transformation to categorical variables.

4.14 Rolling Validation for Multiple Horizons

Since there are dependencies between time instances in the series, k-fold cross-validation is not an appropriate choice for generalization. A simple form of rolling validation divides a dataset into an initial training set and test set. In this study, we sampled 60% of the series as the training set and left the rest for the test set. Horizon $h = 1$ forecasting in the test set is generated such that the first day is forecasted based on the model produced by training on the training set. The sMAPE (i.e. symmetric mean absolute percentage error) value for validation is calculated via evaluating the difference between the actual and forecasted values in the test set.

The next forecast value in the test set is often used in the retraining step as the rolling window moves forward along the time. The size of the training set is fixed by removing the first sample from the training set. We adjust the frequency of retraining for the statistical models such that we forecast kt samples ahead in the test set before including them in the training set and retraining our model.

4.15 Backcasting Strategy

Backcasting in times-series refers to the process that the series is forecasted in reverse time. Backcasting can help practitioners recover the lost or weak information generated as a result of the forecasting process. Although backcasting may be used in smoothing the training set in time-series, leading to an increase in the signal-to-noise ratio, we used it for the generated forecast values to amplify the signals. With extensive experiments, we showed that time-series forecasting can be improved significantly by backcasting strategy, and the results were provided in Table 4.2.

4.16 Fitting of Wavelet-ANN Model

The Wavelet-ANN method was used in the backcasting process, after generating 14 days horizons. We implemented a non-decimated Haar wavelet transform to decompose the newly generated forecasting values into the multi-resolution level. For our case, maximum decomposition level $J_0 \leq \log_e N$ was set to 5. The wavelet transformed sub-series were good candidates to be fed into TDNN because they contain nonlinearity patterns. The number of input lag and hidden nodes were respectfully varied from 1 to 7 and 1 to 10 in the TDNN training process. To optimize the coefficients in the back-propagation step, Levenberg–Marquardt algorithm was used in the process. The Identity and logistic functions were used for the hidden nodes and output nodes, respectively. Every single series was combined with the prediction values by Haar filter reconstruction to forecast the original series. Because the output node produces single forecasting values, longer horizons forecasts were iteratively generated. The backcasting performance of the wavelet-ANN method was generated for an out-of-sample rolling validation period of 14 days, and sMAPE validation criteria were used for performance evaluation (Table 4.2).

4.17 Results

The experimental results for various statistical and neural network models are shown in Table 4.1 and Table 4.2, respectively. We collected Multi-Horizon rolling forecasts on daily deaths for the next 2 weeks i.e. from $h = 1$ through $h = 14$ and used sMAPE as our primary performance metric which is one of the standard performance metrics in time-series forecasting (Makridakis et al., 2018; Taieb, Hyndman, et al., 2012).

(Javeri et al., 2021) trained and evaluated several statistical models, and they optimized the auto-regressive non-seasonal order (p) of such models to obtain the best results. The (p) in Table 4.1 indicates the auto-regressive non-seasonal order for the model with the lowest sMAPE for the given horizon. The SARIMA$(p, 0, 0)_{\times}(3, 1, 1)_7$ uses no differencing and no MA component, whereas it includes $P = 3$ and $Q = 1$ seasonal components with 1 seasonal

Table 4.1: Multi-Horizon (h) Rolling Forecasts: SARIMAX vs. Statistical Models (sMAPE) by (Javeri et al., 2021).

Horizon	RW	AR	ARIMA	SARIMA	SARIMAX
$h = 1$	27.75	17.20 (8)	16.80 (12)	15.40 (1)	**13.25**
$h = 2$	43.49	19.20 (10)	19.10 (7)	17.30 (1)	**15.1**
$h = 3$	50.23	19.00 (10)	18.70 (9)	18.60 (4)	**15.47**
$h = 4$	50.44	19.30 (10)	18.70 (9)	18.80 (6)	**15.21**
$h = 5$	44.61	19.80 (9)	19.10 (9)	19.60 (1)	**14.98**
$h = 6$	30.31	19.60 (9)	18.70 (9)	19.20 (9)	**14.18**
$h = 7$	17.36	19.60 (12)	18.50 (8)	19.40 (9)	**14.03**
$h = 8$	29.92	23.60 (15)	21.30 (12)	22.10 (9)	**13.52**
$h = 9$	42.49	27.20 (11)	24.80 (10)	24.60 (9)	**14.06**
$h = 10$	47.77	27.40 (15)	25.00 (14)	24.90 (8)	**14.22**
$h = 11$	49.36	27.70 (15)	24.10 (15)	25.00 (9)	**14.25**
$h = 12$	44.91	28.00 (15)	24.50 (15)	25.10 (9)	**14.52**
$h = 13$	32.30	28.50 (9)	24.70 (15)	25.30 (9)	**15.05**
$h = 14$	24.05	29.30 (9)	24.10 (15)	25.40 (22)	**15.22**
AVG	38.21	23.24	21.29	21.47	**14.50**

differencing component and a seasonal period of 7 days. The highlighted values in each row indicate the best performing model for the given horizon having the lowest sMAPE score amongst all the proposed models. The SARIMA model performs well overall, however, the ARIMA model is competitive on several horizons, whereas the RW model having the lowest error for horizons 7 and 14.

Table 4.3 compares the performance of multiple configurations of neural network-based models that were studied by (Javeri et al., 2021), against the wavelet-ANN model. The neural network (NN) model refers to the feed-forward neural network, GRU-AE refers to the GRU-based Autoencoder, ConvLSTM refers to the Convolutional LSTM network and AUG-NN refers to the Augmented-Neural-Network run on the data augmented time-series with 750 samples, i.e. double the size of regular time-series with 376 samples. Except for the latter one, all other models were trained on the regular time-series with 376 samples. Since the augmented time-series has a resolution of half a day, the sMAPE values were collected for every even horizon output from $h = 2$ through $h = 28$

to give us the actual 14 days ahead forecasts. The wavelet-ANN model with backcasting strategy performed much better compared to the augmented and non-augmented ones. A certain number of strategies involved in the wavelet-ANN model that led to being a highly performant model: (1) The log transformation was applied to the input data to alleviate the variability of the data. (2) SARIMAX model was used in forecasting the future horizons, and "HospitalizedCurrently" was used as the eXogenous variable in the model. (3) The wavelet transformation was used in the backcasting step over the generated forecasting values, and the forecasted values were smoothed by a neural network model. The wavelet-ANN performed the best for both longer and shorter horizons and all the results along with the improvement percentage were provided in the Tables 4.2 & 4.3, respectively.

Table 4.3 shows the improvement in wavelet-ANN over all neural network models trained on the original time-series. In terms of improvement due to wavelet-ANN, on average we see a 28.53% improvement across all horizons with a maximum improvement of 35.63% observed at horizon 13 forecasts. While the NN model did not perform better than the SARIMA model, the wavelet-ANN model substantially outperformed SARIMA, AUG-NN and AUG-GRU-AE models.

4.18 Conclusion

In this work, we developed and presented hybrid methods to improve the quality of out-of-sample forecasts for intermediate length times-series data. We argued that neural network-based models are limited in cases where enough data may not be available to estimate the large number of parameters that these non-linear models require.

By incorporating multiple transformations in the analysis, we could increase the accuracy. When the wavelet transformation was paired with the backcasting model, the generated forecast values were improved significantly compared to original values, due to the fact that the wavelet functions tend to recover poor signals as a result of rolling forward forecasting. We also showed that the log transformation can be beneficial to the prediction, particularly when the series shows a volatile trend across time. However, with the backcasting strategy paired with hyperparameters optimization, the performance of such predictions is particularly pronounced, making them better than the best performant traditional models.

This work has demonstrated the viability of backcasting strategy with the wavelet transformation functions, suggesting that these research directions could prove useful.

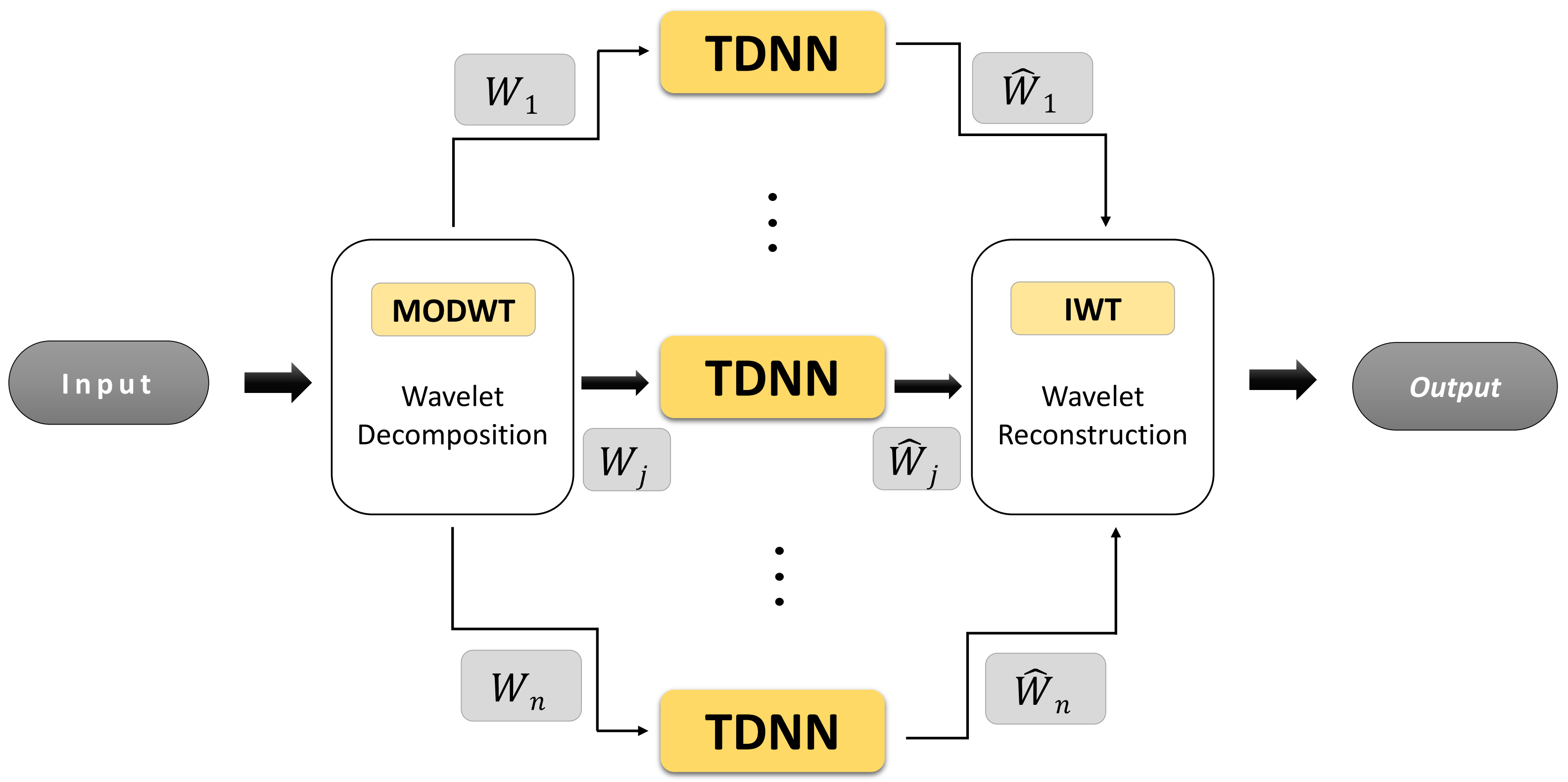


Figure 4.1: Wavelet Neural Network (Wavelet-ANN) Architecture

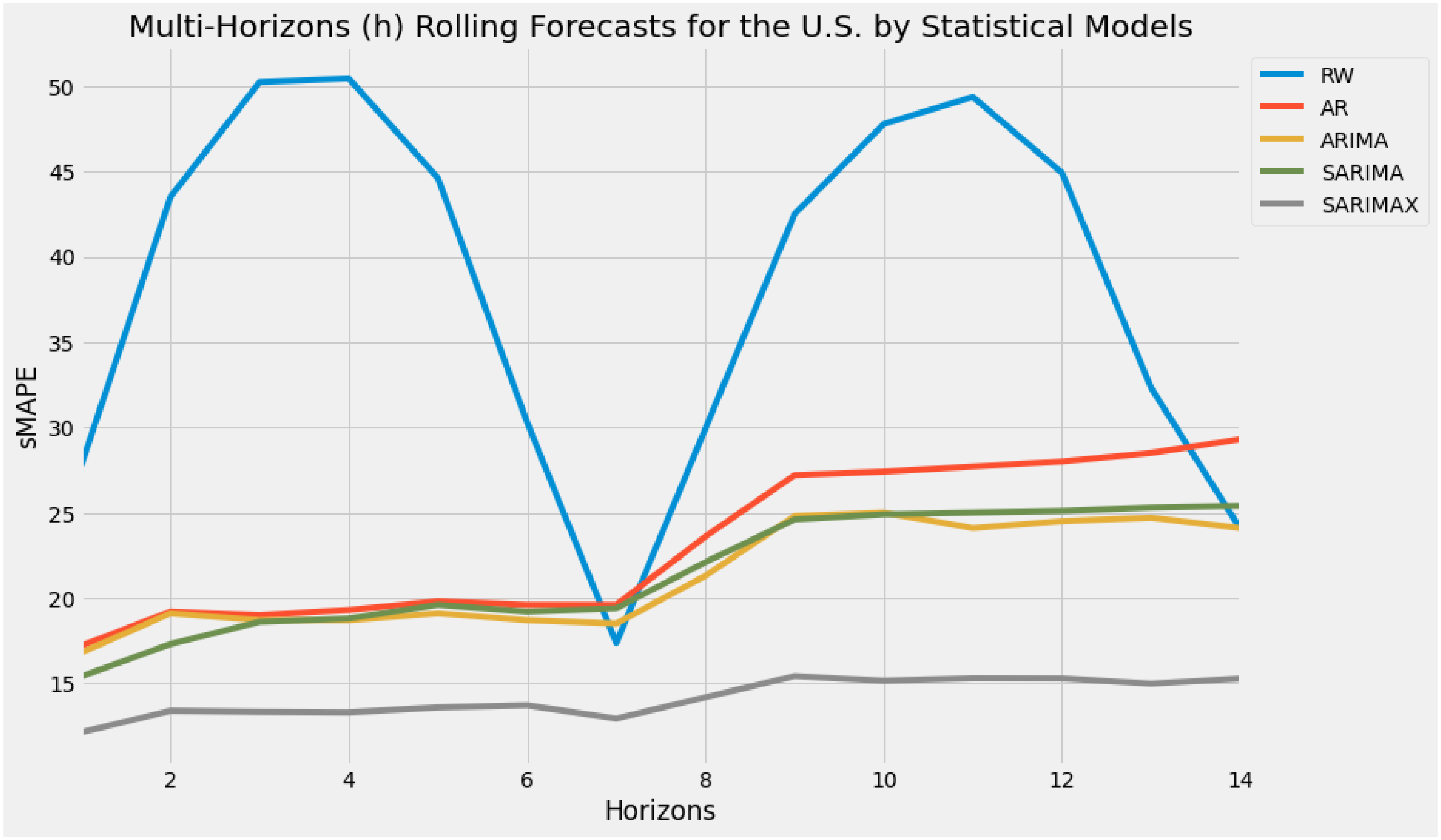


Figure 4.2: Multi-horizons rolling forecasts for the U.S. data by traditional methods. SARIMAX models obtained the best results in all horizons compared to (Javeri et al., 2021) results.

Table 4.2: Multi-Horizon (h) Rolling Forecasts: Wavelet-ANN vs. Neural Network models (sMAPE) by (Javeri et al., 2021).

Horizon	NN	AUG-NN	GRU-AE	AUG-GRU-AE	ConvLSTM	AUG-ConvLSTM	Wavelet-ANN	Improvement (%)
$h = 1$	15.90	14.36	18.87	15.32	18.59	16.37	**12.13 (8)**	**20.82%**
$h = 2$	18.43	15.73	19.48	17.37	17.88	17.38	**13.21 (8)**	**23.94%**
$h = 3$	18.62	16.79	18.75	17.91	18.05	17.01	**13.17 (8)**	**26.46%**
$h = 4$	19.06	17.82	19.38	17.69	18.59	16.88	**13.19 (8)**	**25.43%**
$h = 5$	23.45	18.43	19.92	18.61	20.77	18.90	**13.47 (6)**	**26.91%**
$h = 6$	20.18	18.72	21.08	18.81	22.32	19.65	**13.29 (4)**	**29.00%**
$h = 7$	22.77	19.35	22.18	18.06	24.09	22.79	**12.92 (8)**	**28.46%**
$h = 8$	22.27	20.13	25.21	19.35	26.02	23.81	**13.76 (6)**	**28.88%**
$h = 9$	24.83	21.15	24.29	20.63	26.28	22.87	**14.72 (6)**	**28.64%**
$h = 10$	24.69	21.94	24.52	20.64	26.80	22.40	**14.87 (6)**	**27.95%**
$h = 11$	23.13	22.57	23.77	20.84	26.81	23.08	**14.98 (10)**	**28.11%**
$h = 12$	25.49	23.82	24.13	23.30	28.44	25.20	**15.16 (12)**	**34.93%**
$h = 13$	26.47	24.46	25.25	23.06	30.72	28.25	**15.19 (11)**	**35.63%**
$h = 14$	25.97	24.28	28.74	21.76	32.96	30.49	**15.30 (11)**	**29.68%**
AVG	22.23	19.96	22.54	19.52	24.16	21.79	**13.95**	**28.53%**

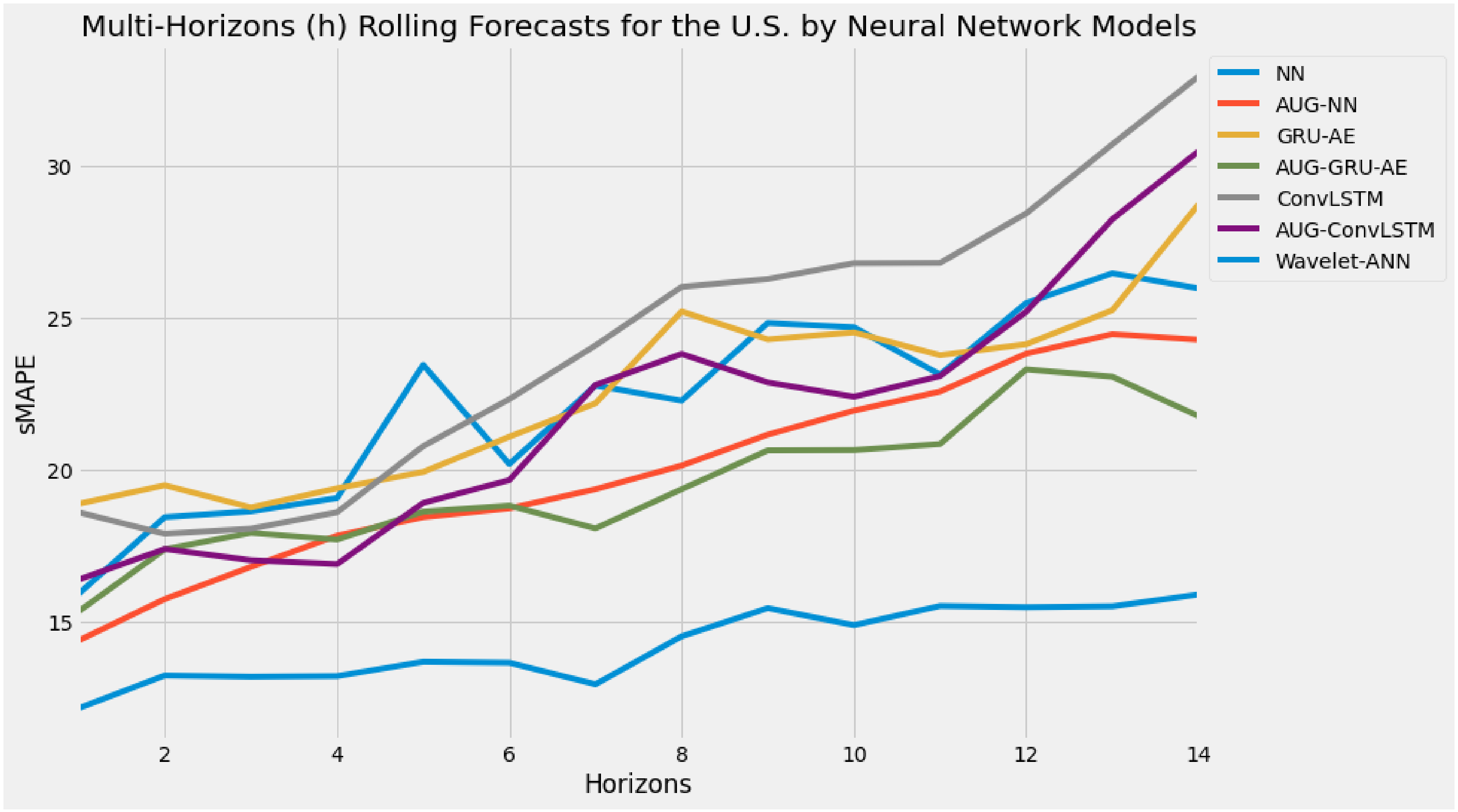


Figure 4.3: Multi-horizons rolling forecasts for the U.S. data by neural network models. The wavelet-ANN achieved the best results in all horizons compared to (Javeri et al., 2021) results.

Table 4.3: New Improvement Due to the Wavelet-ANN compared to SARIMAX

Horizon	SARIMAX	AUG-NN	Wavelet-ANN	Improvement (%)
$h = 1$	13.25	14.35	12.13	**8.45%**
$h = 2$	15.10	17.19	13.21	**12.51%**
$h = 3$	15.47	16.09	13.17	**14.86%**
$h = 4$	15.21	18.50	13.19	**13.28%**
$h = 5$	14.98	18.27	13.47	**10.08%**
$h = 6$	14.18	18.34	13.29	**6.27%**
$h = 7$	14.03	17.93	12.92	**7.91%**
$h = 8$	13.52	20.32	13.73	-
$h = 9$	14.06	19.78	14.72	-
$h = 10$	14.22	21.98	14.87	-
$h = 11$	14.25	22.91	14.98	-
$h = 12$	14.52	23.54	15.16	-
$h = 13$	15.05	23.48	15.19	-
$h = 14$	15.22	23.31	15.30	-
AVG	14.50	19.71	13.95	**3.80%**

Chapter 5

Improving Classification Performance by Transfer Learning in Undersampled Videos

5.1 Introduction

Video data is considered as a super high-dimensional space, because it enriches spaces both at the frame and time spaces. A video is a sequence of images (called frames) captured and eventually displayed at a given frequency. The number of frames in a video are highly relying on the format of the video, but a video usually contains 8 to 16 frames of sequential images to form one second of video. The space in the video is super rich, because video data contains pixels at the frame level and multiple frames at the time level.

Recent and advances in Artificial Intelligence (AI), especially in the form of deep neural networks, have opened many new possibilities in this domain. However, those sophisticated techniques require enough data for training, and if the length of the video is not high enough or the quality of the video is poor, the learning process may fail.

One of the important tasks in the video is classification. Similar to image classification, we should apply the classification tasks to a sequence of images.

But as it was mentioned earlier, if the space is not rich enough, the classification may fail. This incident is common when we have a limited number of video samples, or the videos were recorded poorly. In recent years, many approaches have been brought to bear on such high-dimensional, undersampled problems, including data augmentation, but this technique might not be applied in the video domain due to a lack of consistency between samples.

To address this issue, we propose a novel application of "Transfer Learning" to classify video-frame sequences over different classes. Transfer learning enriches the space of learning by pre-weighted parameters trained previously over other datasets. One positive side-effect of transfer learning is using a pre-weighted model that does not require training a fresh deep model.

We applied and evaluated our method on the traffic video data collected by (https://www.wsdot.wa.gov, n.d.). This dataset suffers from multiple issues. First, the number of video samples is limited for a certain number of classes, so the task of learning of those particular classes is non-trivial. Also, the length and quality of videos are low. Especially, in the presence of severe weather such as "foggy" weather or in the situation where "corrupted" video frames exist in the data. These issues would add to the challenges in the learning tasks. In the following, we demonstrate how our method can classify the videos successfully under such difficulty.

Traffic Management System (TMS)(Chan & Vasconcelos, 2005a) is a field in which the technology is integrated to improve the flow of vehicle traffic and safety. Real-time traffic data from cameras, speed sensors and loop detectors are just a few means of monitoring, out of many, to manage traffic flows. Among those, CCTV camera or video surveillance technology is being used more, since it is less expensive and more manageable compared to other ones. Moreover, a fully automatic monitoring system is in the interest of certain researchers.

The majority of the existing framework in monitoring traffic uses sophisticated equations with a substantial number of parameters and coefficients. A group of researcher in (Chan & Vasconcelos, 2005a) and (Chan & Vasconcelos, 2005b) described methods that are heavily based on motion analysis and object segmentation using "auto-regressive stochastic" technique and "KL-SVM" classifier, respectively. Both articles showed techniques that are estimated, tuned and utilized once the objects are segmented.

One drawback of such techniques is they detect objects in videos when the quality of the frames are visually accurate and the frames are not degraded due to the severe weather conditions, nor distorted due to the corrupted signal. Another issue is that videos should be fed into the proposed models for the video classification tasks, and this requires new weights and parameters estimation specified during the training and tuning process.

Last, as Figure **??** depicts, Medium and Heavy classes are very similar to one another in terms of appearance and number of vehicles in the videos. These difficulties affect the classifier negatively, and traffic mode prediction based on the Entity Detection approach in videos would fail, and distinguishing between Medium and Heavy classes becomes more challenging. The latter experiment will also be discussed and elaborated in this study along with cases of how they would stall.

In the end, this work will address those aforementioned problems by suggesting a novel approach, so that video classification case is able to be processed while running over the "foggy" weather or "corrupted" video frames, and the classification in all traffic modes would be achievable with high accuracy. Additionally, training a new model from scratch over the samples would not be required in this setting. These are the contributions studied via a series of experiments that are elaborated and addressed in the following sections.

5.2 Previous Work

Statistical Methods:

Statistical methods have been practicing by many researchers in the context of classification problems. (Chan & Vasconcelos, 2005b) presented the "Dynamic Texture Model" based on the KL classification framework which extracts motion information from the video sequence to determine the motion classes. (Chan & Vasconcelos, 2005a) proposed an "Auto-Regressive Stochastic Processes" and it was claimed that it would not require segmentation or tracking in the videos to capture the traffic flow. (Vaghasia, 2018) presented a hybrid approach that combines the ARIMA model with fuzzy wavelet transform to manage noise attached to a dataset that was previously studied. (Peng & Miller, 2019) compared different statistical and machine learning models including seasonal ARIMA, seasonal VARMA, exponential smoothing and regression, Support Vector Regression, feed-forward Neural Networks, and Long Short-Term Memory Neural Networks. Both later articles attempted to forecast traffic flow in both the short and long terms. One disadvantage of all aforementioned techniques previously studied is they are heavily relying on estimating parameters in a fully supervised fashion, which requires extensive parameters setting and model fitting.

CNN Methods:

Many deep learning frameworks and architectures are being utilized by researchers for different applications and domains and have achieved remarkable results in various computer vision tasks. (Szegedy et al., 2015) developed a deep convolutional neural network architecture for image classification tasks by utilization of the computing resources inside the network. (Simonyan & Zisserman, 2014b) showed very deep convolutional networks for large scale image classification. (Bertasius et al., 2015) presented a multi-scale deep network for image segmentation task. (He et al., 2016) empirically claimed that their residual networks are easier to optimize while keeping the accuracy relatively high. (Xie et al., 2017) described a highly modularized network architecture with fewer hyper-parameters to set by repeating a building block that aggregates a set of transformations. (Toshev & Szegedy, 2014) applied Deep Neural Networks for human pose estimation. (Bertasius et al., 2017) introduced Random Walk Networks (RWNs) for the purposes of object localization boosting and the segmentations that are spatially disjoint. With the advent of the CNN era, many scientists research further to deploy DNNs into the video-content datasets. (Yue-Hei Ng et al., 2015) and (Simonyan & Zisserman, 2014a) implemented a video classification task using stacked video frames as input to the network. Karpathy et al., 2014 studied the performance of CNNs in large-scale video classification and they achieved the highest transfer learning performance by retraining the top 3 layers of the network. (Srivastava et al., 2015) compared and analyzed different proposed models based on LSTMs.

5.3 Video Format and Preparation

The traffic video dataset contains 254 video samples of highway traffic in Seattle, recorded from a single fixed traffic camera (https://www.wsdot.wa.gov, n.d.). The collection is divided into three classes, namely as Low, Medium and High traffic road congestion (Figure 5.1). Since the number of Low traffic jam labels is considerably greater than the two others, the "unbalanced" samples happened in the dataset. The video samples also suffer from "poor" quality in the resolution of the recorded frames. Corrupted frames and sudden "jumps" between the frames are observed across the video frames. Moreover, a certain number of videos have been recorded in the "precipitation" weather conditions where the fog and rain affected on the visibility and clarity of the videos (Figure 5.1).

These are the issues that have negative effects on the prediction. A certain number of videos have been recorded in the foggy or rainy climate, so the visibility is drastically low in those videos. This is what causes the classification task more challenging since the model should ignore noises due to water droplets spots and steam conditions during the prediction process.

5.4 Model Selection

YOLO

YOLO (You Look Only Once) (Redmon et al., 2016) developed an object detection framework as a regression problem to spatially separated bounding boxes and associated class probabilities. YOLO9000, a real-time object detection network that is the improved version of the YOLO detection model, proposed by the same scientists, has fewer challenges than we had primarily by Google API. We deployed YOLO architecture using pre-trained weights to recognize objects across a video sample. Then we count the number of objects per frame (specifically vehicles) to classify each video into Low, Medium and Heavy on aggregated frames level, while the traffic flow is consistent across the frames.

Several objects in the videos are quite clear, especially when there was "low" traffic congestion. So, YOLO is able to label other objects in videos including trees, lanes and persons. The object "Person" is a false class YOLO specifies, since there is no person in the videos (Figure 5.2).

However, object detection tasks in Medium and Heavy classes are challenging, so YOLO fails to recognize more distinct objects as it does when traffic flow is Low (Figure 5.2). This drawback is also seen in other Network architecture such as Google Video Intelligence Service which we address in the next section.

Google Video Intelligence API

Another framework we analyzed performance is Google Cloud Video Intelligence API. There are cloud-based service providers such as Amazon, Google, Microsoft, BigML, and others who have developed MLAAS (Machine Learning As A Service) platforms, so that individual users and commercial companies are able to employ pre-trained models to solve their problems without managing any hosts or servers. Also, they can benefit from Machine Learning Engine models running on powerful machines contain CPU and GPU, so the problem solving will be accelerated tremendously. GUI in Google Video API is user-friendly, so this can be considered as an advantage. It is worth noting that Google API recognizes traffic congestion directly from the videos, however, it does not specify the intensity of traffic.

We observed that on the videos with Heavy or Medium traffic flow, the performance of YOLO architecture over Google API was particularly pronounced. So, Google detects vehicles less than YOLO within the same video samples. In comparison with YOLO, Google API performs poorly in the object detection tasks, and it fails when video samples are in foggy or rainy conditions.

5.5 Fresh Convolutional Neural Network

We built Convolutional Neural Network (CNN) models that could identify whether a given traffic flow video is predicted as Low, Medium or Heavy. Ultimately, we developed and trained these CNN models from scratch to see how they would perform on new unseen data samples.

These models were created in Keras framework, as a sequential model. The first layer is a "Convolutional Layer" (Conv2D), and this is a 2-dimensional convolutional layer. The number of output filters in the convolution is 32, with 3×3 "Kernel Size". We used a "Relu" activation function, with $224 \times 224 \times 3$ specified in the first layer of sequential model for height, width and channel dimensions of frames, respectively. Each convolutional layer is followed by a "Max Pooling" layer with 2×2 pool size.

We created CNNs with 5, 6 and 7 convolutional layers, and we have a flattened layer taking the output from the previous layer and flattening it into a one-dimensional tensor fed into a dense layer that has 3 nodes. The last layer contains 3 nodes since this will be the output layer that categorizing videos as Low, Medium or Heavy. We used the activation function of "softmax" in this last layer. The models were trained with "Adam" optimizer, learning rate = 0.00005, categorical cross-entropy loss function and an array with the single string accuracy as metrics.

5.6 Transfer Learning

Using a pre-trained model for prediction is a growing technique practiced by many researchers. That model would achieve great results if the input data is classified into the categories used by the original model. However, none of the current pre-trained CNN architectures classify Highway Traffic Videos into specific "Mode" categories. Thus, one approach by which the scientists could obtain reasonable results is to Transfer the knowledge from one domain aspect to another. This is what people call *Transfer Learning*. Recent studies suggest that early layers of deep learning models identify simple "patterns", while later layers would identify more complex patterns. (Geirhos et al., 2018) showed that the ImageNet-trained CNNs are heavily biased in favor of recognizing textures rather than shapes. Thus, the later layers in CNNs are complex representations of image textures.

VGG19

Very deep convolutional networks, also known as VGG, for large-scale image classification tasks is employed for labels prediction in traffic flow (Simonyan & Zisserman, 2014b). The model increases depth using an architecture with very small (3×3) convolution filters which encourages higher performance in the localization and classification tasks.

Model Setup

Researchers in (Simonyan & Zisserman, 2014b) explained how well their CNN classifier model performed on classifying the ImageNet challenge dataset. This section shows how VGG19 will be applied on a completely new type of dataset (CCTV videos of traffic flow) which does not contain classes similar to those included in ImageNet.

VGG was originally has been trained on images and now it is trained on videos. Video samples contain 3 classes labeled as Low (L), Medium (M) and Heavy (H) traffic flow, and each class is constructed of images of frames extracted from 5-second videos. This dataset is available on (https://www.wsdot.wa.gov, n.d.) website as videos. We extracted the data like videos and extracted 10 frames per second. Having extracted and organized all frames from video samples, we created the directory iterators for train, validation and test sets. We built the new model that contains all of the VGG19 layers up to its 5 to the last layer, with an added output layer containing 3 output nodes that correspond to each of the traffic mode classes.

observed that removing 2 fully connected layers (fc1 and fc2) would increase the accuracy of prediction significantly, while decreasing the number of "Trainable Parameters" and "Total Parameters" by 96%. This DNN has been developed with Keras functional API framework. One question of interest is how many layers should be trained on the new dataset. One may still want to keep the most of what the original VGG19 model has already learned from ImageNet data by freezing the weights in the majority of layers. One solution is to implement the "Brute-Force" approach on a different number of layers, so one may find the optimal number as to unfreeze for training.

We evaluated the performance of VGG19 on traffic videos by testing all the combinations of the last 5 layers, and in our experiments, VGG tends to yield reasonably well accuracy with training the last 5 layers, which is "block5_conv1 (Conv2D)" layer with output shape of (None, 14, 14, 512) and 2,359,808 parameters. All model configurations and results are provided in Table 5.1. Under multiple settings, it achieved state-of-the-art results across several highly competitive configurations with the highest accuracy of 96.5%.

5.7 Results

Among 4 study cases, YOLO and Google Video Intelligence API are based on object detection techniques. In other words, these techniques detected the number of objects (vehicles) per frame across a video and attempted to classify a given video based on an average number of vehicles they recognize. As Figure 5.2 shows, both classifiers are not powerful enough to detect all the vehicles in a video, so classification merely based on the number of vehicles in the samples is not an appropriate idea. One reason could be video samples have not been recorded in high quality. Another reason is both classifiers were not able to segment between several vehicles when they are moving in a group, and they detected them mistakenly as a truck or a bus.

Figure 5.3 shows that in the Train plot, the average number of vehicles between Medium and Heavy modes are very close to one another, and the line in Heavy mode falls below the Medium line which is not correct. Interestingly, the Heavy line (Blue) is below the Medium line (Orange) almost entirely. This technique may be useful when one would attempt to predict class labels between the "Low" and "non-Low" modes, as Low lines (Grey) in Figure 5.3 indicate lower waving as opposed to the other two lines (non-Low). In our experiments, YOLO tends to yield better performance in detecting vehicles than Google Video Intelligence API by segmenting objects more reliably, with fewer false negatives.

Also, we observed that YOLO is more robust against severe conditions than Google Video Intelligence API, where the weather is foggy or rainy in videos. However, Google Video Intelligence API has a richer GUI and users can benefit

Table 5.1: VGG19 and CNNs Table of Outputs

Model	Hyper Parameters	Train	Validation	Test(CV)
VGG	{'trained_layers': last 5, 'batch_size': 100, 'steps_per_epoch: 45', 'epochs': '70', 'pre_process': MobileNet}	100%	85.81%	**96.50%**
VGG	{'trained_layers': last 5, 'batch_size': 100, 'steps_per_epoch: 16', 'epochs': '70', 'pre_process': MobileNet}	100%	85.91%	**96.50%**
VGG	{'trained_layers': last 5, 'batch_size': 287, 'epochs': '70', 'pre_process': MobileNet}	100%	87.42%	95.42%
VGG	{'trained_layers': last 5, 'batch_size': 300, 'epochs': '70', 'pre_process': MobileNet}	100%	87.53%	95.27%
CNN	{'Conv2D_layers': 5 layers, 'activation': Relu, 'epochs': '70', 'num_dense_nodes': 409}	100%	91.5%	92.84%
CNN	{'Conv2D_layers': 6 layers, 'activation': tanh, 'epochs': '70', 'num_dense_nodes': 449}	88.38%	83.08%	82.67%
CNN	{'Conv2D_layers': 7 layers, 'activation': Relu, 'epochs': '70', 'num_dense_nodes': 87}	100%	91.67%	91.98%
			state-of-the-art	94.50%

The results from the VGG-Transfer-Learning approach under multiple configurations. The test results are calculated by the average of 10-fold cross-validation. Several batch sizes were included in the experiments to achieve higher performance. Conventionally, "steps per epoch" is set to an integer number obtained through division of train size over batch size. For batch size = 100, we set steps per epoch to 45 (simple division) and 16 (arbitrary number), to compare both settings. The table shows the highest accuracy is obtained by setting the batch size to 100 and steps per epoch to 16.

from uploading the video on Google Storage to run models faster than a local system, simply because Google runs the models in a big data framework and highly scalable infrastructure. CNN and VGG-Transfer-Learning appear to be effective ways to classify videos' contents. The results presented in Table 5.1 reveal that VGG19 outperforms all the fresh CNNs in the prediction task. In the same table, the CNN with 5 convolutional layers gained the highest accuracy (92.84%).

Although CNN-5 achieved relatively high accuracy in classification (92%), VGG19 obtained 96.5% accuracy which is 4.5% more than CNN-5. CNN-5 misclassified 77 frames videos as Medium mode, while they are Heavy, without any misclassification between Low and Medium or Low and Heavy. In VGG19, on the other hand, several hyper-parameters were tested and evaluated. We observed that when the batch size decreases from 300 to 100, the number of misclassifications will also drop significantly (50%). The misclassification happened when predicting between Medium and Heavy modes. Under multiple settings, it achieved state-of-the-art results with training the last 5 layers and 70 epochs (Table 5.1).

5.8 Model Performance

Tables 5.2 & 5.3 are vividly showing that both CNN-5 and VGG19 classifiers performed perfectly well in classifying Low vs. non-Low classes, as it shows values 100% for Sensitivity and Specificity. However, both models are less efficient in predicting between Medium and Heavy classes. Although Sensitivity was recorded very high for CNN (100%) and VGG19 (97.54%), Specificity needs an improvement, especially in the CNN-5 model. In other words, the table reports that CNN-5 would be able to identify 83% of Heavy Traffic flow video cases as Heavy correctly and classify 17% as Medium Traffic flow incorrectly. Similarly, VGG can classify almost 91% of Heavy Traffic flow cases as Heavy correctly, and predict the rest (9%) as Medium incorrectly.

The results presented in Table 5.2 & 5.3 reveal that VGG-Transfer-Learning achieved the largest Accuracy, Sensitivity and Specificity results across several highly competitive hyper-parameters with nearly perfect evaluation metrics, while CNN-5 performed on par with VGG in Sensitivity criteria.

Table 5.2: VGG-Transfer-Learning Diagnostic Table.

Pair-Labels	#batches	Sensitivity	Specificity	Accuracy
Low-Medium	100	100%	100%	100%
	287	100%	100%	100%
	300	100%	100%	100%
Low-Heavy	100	100%	100%	100%
	287	100%	100%	100%
	300	100%	100%	100%
Medium-Heavy	100	97.12%	94.42%	95.74%
	287	97.54%	91.12%	94.12%
	300	97.54%	90.74%	93.91%

Table 5.3: Diagnostic Table for the Best CNN-5 (5 Conv2D).

Pair-Labels	Sensitivity	Specificity	Accuracy
Low-Medium	100%	100%	100%
Low-Heavy	100%	100%	100%
Medium-Heavy	100%	83.71%	89.71%

5.9 Why VGG19 Outperformed?

Transfer Values

To extract the transfer values, we excluded the last layer in VGG architecture, which is the "softmax" classification layer and we call it "Transfer Layer" and its output is "Transfer Values". We stored this layer on a hard disk since this is an expensive computation. The transfer values are nothing but arrays with 25,088 elements due to the output shape of VGG-Transfer-Learning architecture. Thus, the transfer values have 1396×25088 dimensions, with 1396 and 25088 represent the number of samples (video frames) and features, respectively. This new space contains pertinent information as to how they form classes in a high dimension space.

Dimension Reduction: PCA

As part of this study, we are interested in analyzing the transfer values, so we can learn how the VGG19 model is able to extract useful information and separate the 3-class labels we specify. The challenge is the transfer values are obtained in a very high-dimensional space (25088 and 12544 elements in VGG19 and CNN-5, respectively), so plotting is impossible. For ease of exposition, we applied a widely-used dimension reduction technique, called PCA. We called this method from the "Scikit-learn" package with n = 2 meaning all transfer values are reduced to arrays of length 2. The PCA process was implemented for the entire test size (1396 samples), each of them is an array with 25088 and 12544 values, the dimension size of VGG19 and CNN-5 networks, respectively. Ultimately, the PCA was reduced from 25088 and 12544 values to only 2 values. Figures 5.4a & 5.4b show PC1 and PC2 for the transfer values in VGG19 (top) and CNN-5 (bottom).

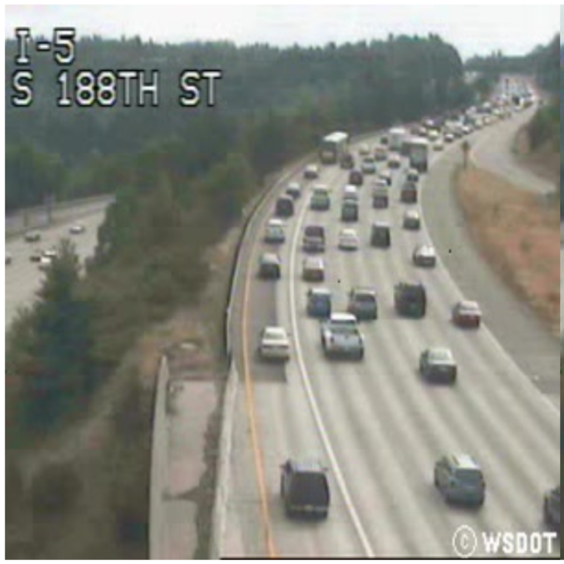
The plots illustrate visibly that class Low (green) is easily separable from the other two classes, while Medium and Heavy classes are mixing at some points. The class colors Red and Purple might be Medium and Heavy classes or vice versa. As the plots suggest, the VGG19-Transfer-Learning model that we train on traffic flow video samples predicted as expected, since the information extracted from video frames are properly separated into 3 classes with minor overlapping between Red and Purple. However, the transfer values in CNN-5 tend to be distributed more sparsely after dimension reduction, which resulted in facing difficulty for the classifier to perform classification. Figures 5.4a & 5.4b are in line with the results on the classification performance we provided in Tables 5.2 & 5.3. We benefit from the PCA technique, since it is linear and deterministic, but other dimension reduction techniques such as "t-SNE" (t-Distributed Stochastic Neighbor Embedding) which is non-linear and non-deterministic is also applicable, however, t-SNE would not perform well when the data samples are relatively large.

5.10 Conclusion

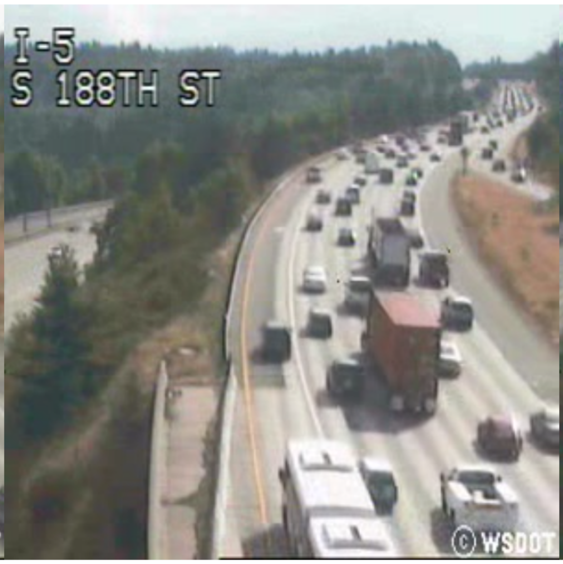
We presented several experiments in understanding the contents of videos. Two approaches, YOLO9000 and Google Video Intelligence API, were depending on object detection fashion, and the other two, CNN and VGG-Transfer-Learning, predicted classes through a convolutional paradigm. We showed that video contents understanding scales naturally to tens of frames and objects inside while exhibiting no implementation difficulties. Whilst the object detection approach follows a simple object counting rule in the frame-level, distinguishing between two similar classes (Medium and Heavy) is challenging.

In our experiments, YOLO outperformed Google Video Intelligence API in detecting more vehicles across video frames, while Google was more user-friendly. We also trained and evaluated a new CNN from scratch to compare it with another pre-trained model (VGG19). Although the CNN model performed astonishingly well, VGG showed higher accuracy in the prediction task due to its complex network architecture and a large number of examples fed as input.

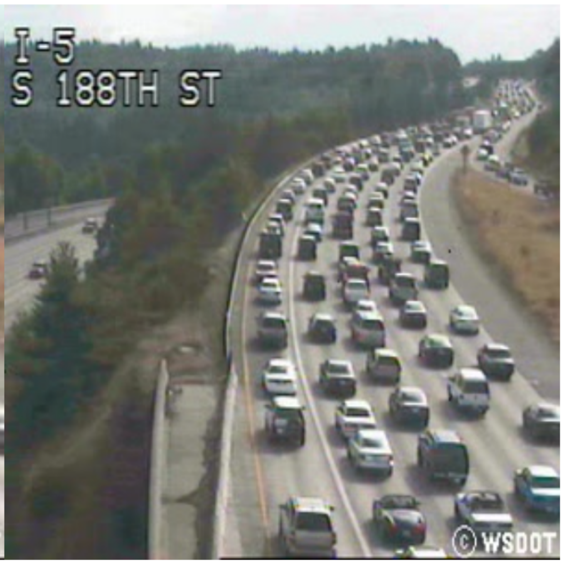
It is conceivable that more extensive hyper-parameter searches may further improve the performance of CNN on the Traffic database. Under multiple settings, VGG19 achieved the best results. In our experiments, both CNN architectures tended to yield consistent results in accuracy with a reducing number of parameters, without any signs of performance degradation. On the traffic video dataset, we observed that the simplicity of our approaches over prior work is particularly pronounced. The results presented in Table 5.1 revealed that our proposed solution (VGG19) performed better than the best performance from KL-SVM (98.50% > 94.5%), whilst requiring significantly fewer parameters tuning and computation to achieve supremacy.



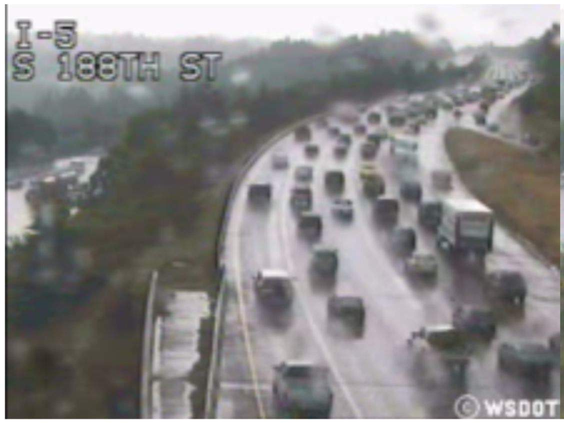
(a) Low



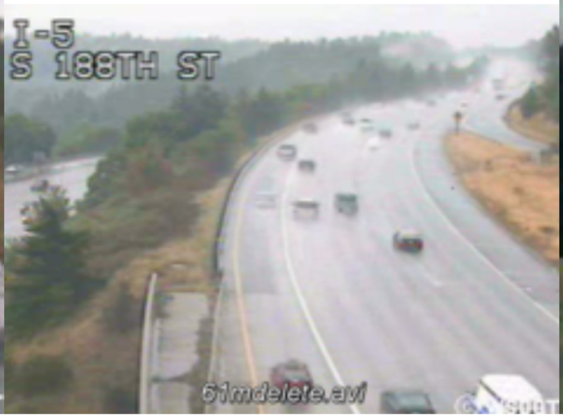
(b) Medium



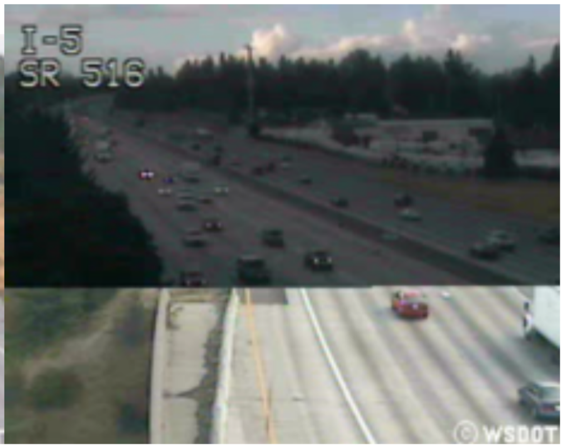
(c) Heavy



(d) Rainy



(e) Foggy



(f) Corrupted

Figure 5.1: (Top row) dataset contains 3 modes of traffic jam; Low, Medium and Heavy. (Bottom row) shows poor quality and visibility due to precipitation or corrupted signal in the number of video samples.

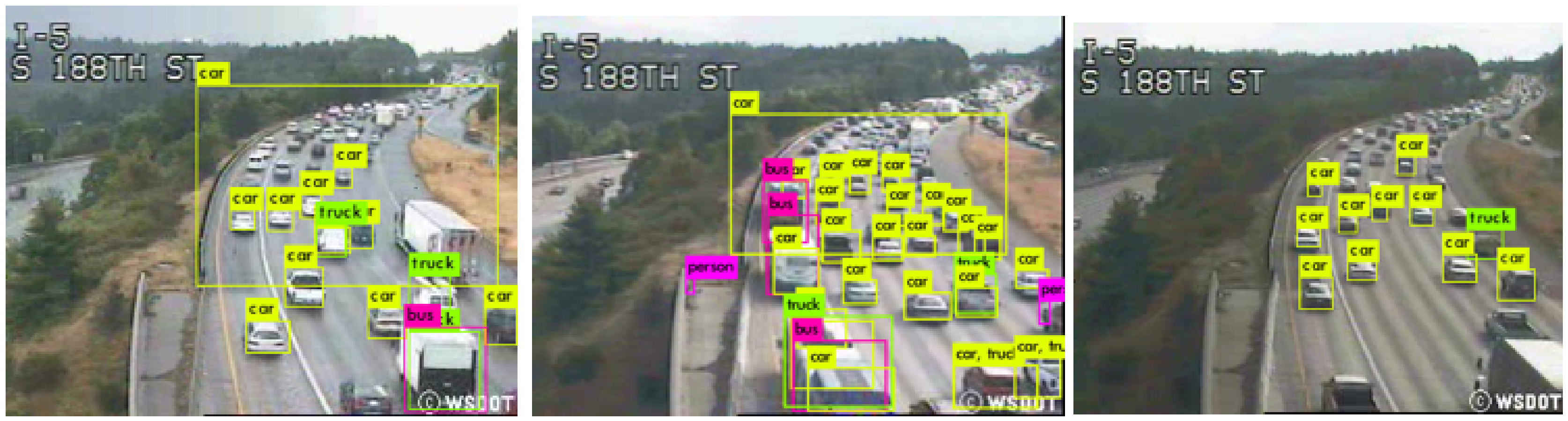


Figure 5.2: Object detection in Videos using YOLO network. The middle scene clearly shows that YOLO recognizes Person and Truck by mistake, and it cannot detect vehicles in farther distance.

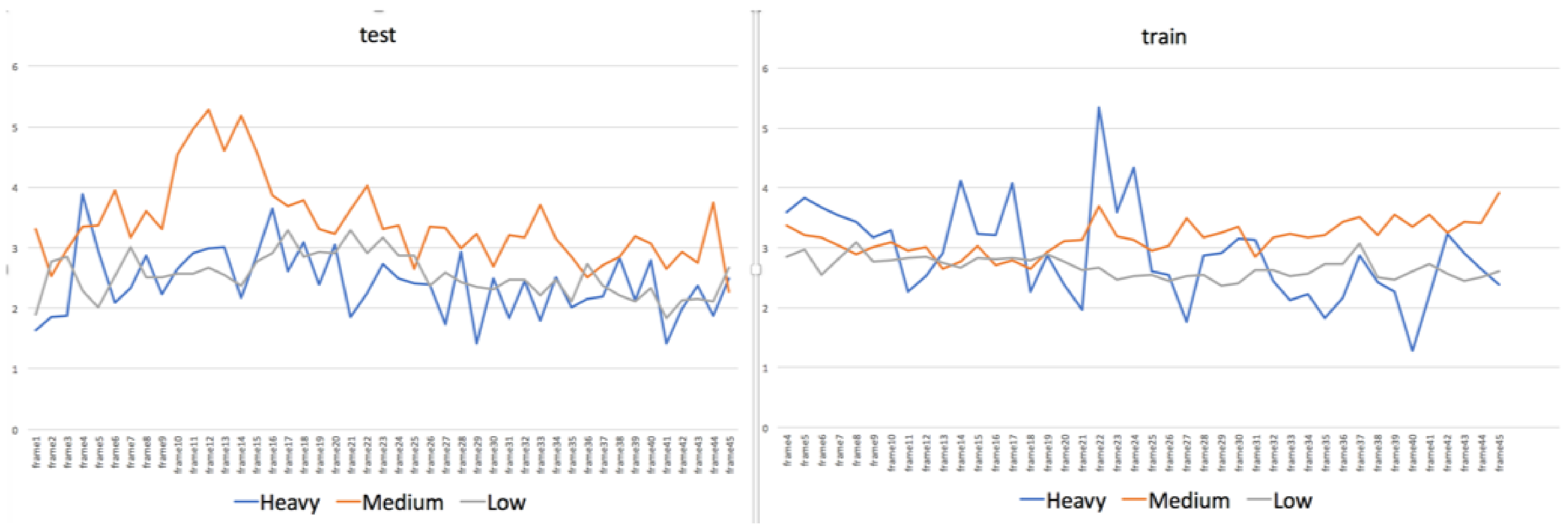
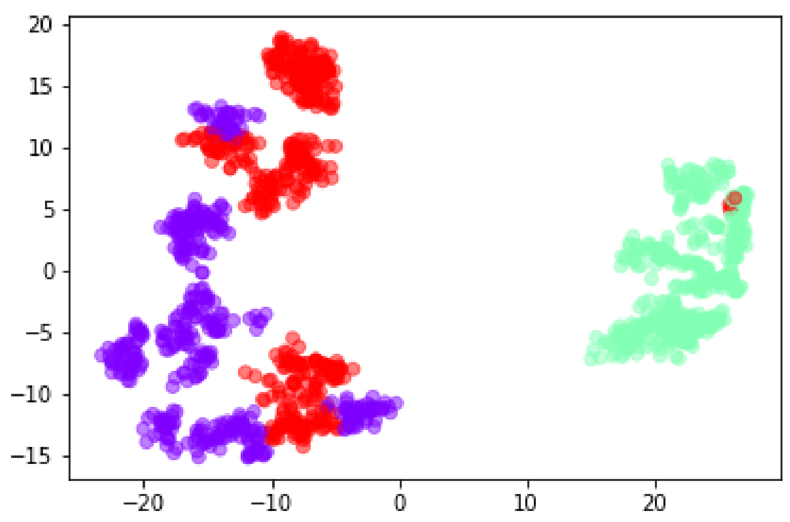
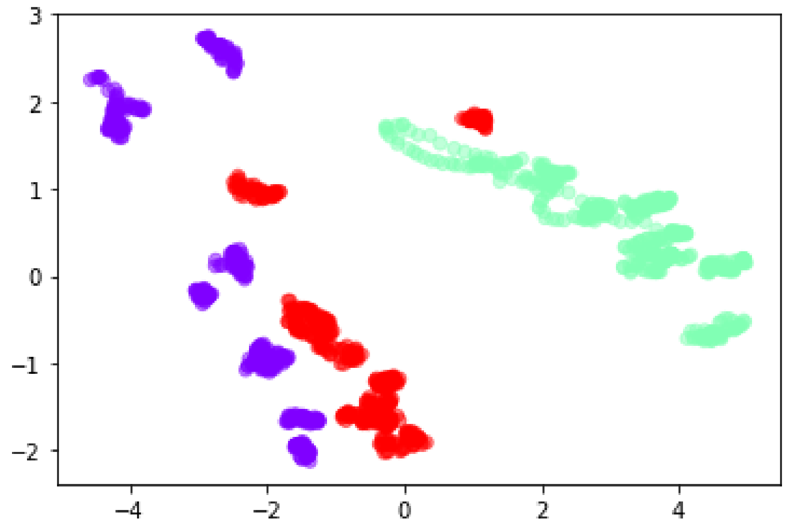


Figure 5.3: Different variability in the number of detected vehicles in the test and train subsets.



(a) The dimension reduction technique using PCA method to reduce the dimension size from 25088 to 2. Three colors refer to three class labels (L, M, H). This plot illustrates information flow throughout VGG network up to "bottleneck" layer.



(b) The dimension reduction technique using PCA method to reduce the dimension size from 12544 to 2. Three colors refer to three class labels (L, M, H). This plot illustrates information flow throughout CNN with 5 convolutional layers network up to "bottleneck" layer.

Chapter 6

Use of Knowledge Graphs in Time-Series Forecasting

6.1 Introduction

In the past few years, rich arrays of time-series data have become available and its analysis has also become increasingly important. It is predicted that in the next few years, the importance of time-series data will grow even more rapidly due to the massive production of such data. All this data requires competent time-series analysis with both statistical and machine learning methods.

In the past, time-series forecasting has been undoubtedly a challenging application for machine learning due to the shortness of the sample time-series, however, in recent years high-resolution time-series are becoming more prevalent and therefore, many machine learning approaches have shown success in forecasting.

On the other hand, graph-structured data have also become ubiquitous within many fields, where it can be used to model anything from recommendation systems to protein-protein structure networks. Such structures are complex enough to capture the interactions (edges) between the entity (nodes). Knowledge Graphs have become the backbone of many systems, but also have many applications when it comes to machine learning tools. Many machine learning applications use knowledge graphs to make predictions and discover patterns in a dataset.

An example where knowledge graph data is used to make predictions and forecast is in vehicle traffic forecasting where traffic flow and speed data is available via sensors located on roads and crossroads (Peng et al., 2019). These sensors collect high-resolution data as there are a number of them located in different parts of the roads. Another example that can be captured with knowledge graphs is epidemiological data. Such data is collected with less frequency where the time resolution could be daily. It is also captured over wider spatial regions, for example, the spatial resolution could be regional. In both of the applications given above, the data is spatial and temporal, so their spatial aspects can be captured properly with knowledge graphs to help generate accurate forecasts for traffic on a road system or hospitalization rates during a pandemic.

In addition, knowledge graphs are known to help build hybrid Statistics-Machine Learning models, therefore one can take advantage of this knowledge and apply it to the model to improve the accuracy of the system. Such kind of system will be more reliable and robust in forecasting since it has the capability to adapt to new events with the help of the knowledge graph data. The benefits of this approach are two-fold; first that we can build knowledge-directed systems by using the knowledge in building the machine learning models' architecture. Second, by infusing the knowledge as an exogenous source of data into the model.

Machine learning and knowledge graphs can be proven to have a close relationship. With the advantage of machine learning, we may help knowledge graphs to establish any missing links they may have by link prediction and making inferences as an example in (Makni et al., 2020; Ostapuk et al., 2019). On the other hand, however, the opposite has not been explored as much except for a few recent studies (Kursuncu, 2018).

One aspect that knowledge graphs can aid machine learning techniques is to establish some contextual meaning. Knowledge graphs can easily represent real-world concepts and describe the internal relations of a machine learning algorithm by observing and impacting its policy making processes. This will help eliminate any bias that may exist in the system. Recent work by (Sheth et al., 2019) proposed using knowledge graphs as the input to neural networks or even correcting the output once a prediction has been produced.

In the past two decades, neural networks have attracted much interest because of the success they have with their performance and predictions. This has inspired many researchers to use them for analyzing graph-structured data with graph neural networks (GNN). (Scarselli et al., 2008) originally proposed GNN as a method to learn graph-structured data using neural networks, this approach was then further extended to convolutional neural networks using spectral or spatial methods (Kipf & Welling, 2016) and have obtained successful performance in node classification tasks.

The issue with traditional GNN methods is that while they can obtain effective feature embeddings, they tend to lose the information associated with the interactions (edges). Many prediction tasks such as link prediction and community detection will benefit from information about the interactions. Since graphs usually have both the entity (nodes) and their interactions, learning both simultaneously would be beneficial. Furthermore, both the entities and their relations provide rich information which would facilitate entity classifications. (Jiang et al., 2019) have proposed a framework to learn both node and edge features on graphs. They proposed a framework to learn the node and edge embeddings consists of a convolution with an edge-node switching network. To put it another way, the role of a node and an edge can be switched. If G denotes the node adjacency matrix and $L(G)$ denotes the line graph of the adjacencies between entities in matrix G, the framework then conducts operations on both G and $L(G)$.

When using Deep Learning and other machine learning techniques, we need to be mindful of the training dataset contains ambiguity, bias, and sparsity (Kursuncu, Gaur, Castillo, et al., 2019) which can lead the model to fail. Another point of concern with such techniques even if they provide reasonable results is their limitation in explainability (Palmonari & Minervini, 2020). Furthermore, in the case that there have been some historically unforeseen events, the prediction reliability of such models is unreliable.

6.2 Related Work

Recent and rapid advances in Artificial Intelligence (AI), particularly in neural networks and knowledge representation learning, have opened many new possibilities in performing various tasks such as time-series forecasting, classification in pattern and sequence recognition, clustering, and filtering. On the parallel, Knowledge Representation has provided the ability to represent entities and relations with high reliability and reusability.

Researchers in (Kursuncu, Gaur, & Sheth, 2019) believe that separate knowledge graph and machine learning methods tend to learn unreliable concepts or relationships that appear misleadingly accurate on KG or representation space, yet do not provide adequate results when the dataset contains contextual and dynamically changing concepts and relations. Our idea is that combining the knowledge graph with machine learning methods will systematically assist in improving the accuracy of the systems and expand the range of machine learning capabilities. There are many opportunities in machine learning where the knowledge graphs can be infused to obtain promising results. A certain number of aspects of knowledge graphs that can be helpful are as follow:

- Data insufficiency: Machine learning models, particularly neural networks, require a large amount of training data. As described in (Javeri et al., 2021), neural networks with short and intermediate length data struggle to compete with other statistical models in time-series forecasting problems. In the case of sparse data, knowledge graphs can be beneficial to augment the training data.

- Zero-shot learning: Properly trained data is essential for a machine learning model to distinguish between two data points. With improperly trained data the model cannot distinguish, and this is called zero-shot learning. In this case, the induction from the machine learning model can be complemented with a deduction from the knowledge graph.

- Interpretation: With the prompt acceptance of sophisticated Artificial Intelligence (AI) models in the industry for solving problems, Machine Learning Interpretation (MLI) is not a fancy direction but a need. Knowledge Graph can be beneficial to explain a black-box model by mapping interpretations to certain proper nodes in the graph and providing a summary of the learning process.

Although knowledge graph (KG) can be applied in many domains, this section focuses only on applying KG in time-series data. (Javeri et al., 2021) studied building a neural network model by using data augmentation techniques for time-series forecasting and (Kursuncu, Gaur, & Sheth, 2019) worked on techniques to incorporate knowledge graphs in deep learning. They believe that infusion of knowledge within the hidden layers of neural networks 2 will become a critical and integral component of AI models that are integrated into deployed tools. In their work, they created a subgraph from the actual knowledge graph, because concrete problems often require a relevant portion of the full graph. They explored the idea of infusing the structural information of the subgraph before the output layers in an attempt to prove that combining knowledge graphs and neural networks greatly enhance the performance.

(Feng et al., 2019) studied stock prediction analysis by graph neural networks, where they infuse the firm knowledge graph into the predictive model. In their effort, they showed if a knowledge graph with company-relations data could be useful to improve the stock-market predictions. They applied a graph convolutional network which takes input features and their connections for training, including an additional LSTM layer inspired by (Zhao et al., 2019).

The CensNet framework introduced by (Jiang et al., 2019) is a general graph embedding framework that represents the nodes and edges into a latent space. This framework produces a novel approach that uses a line graph of the original undirected graph to switch the roles of nodes and edges. Their framework focuses on the idea that not just the nodes but also the edge contains some crucial information. In other words, the CensNet framework conducts the graph convolution operations on both the input graph and its line graph counterpart.

6.3 Knowledge Graph Use Cases

Although knowledge graph is in its early stage, a certain number of examples that knowledge graph can be applied are provided in the following:

Question-Answering: Perhaps question-answering is one of the common applications of the knowledge graph. With an extensive availability of information in the knowledge graph, question-answering may provide end-users with an effective and efficient way to retrieve information from Knowledge Graphs.
Storing Information of Research: Many companies tend to use knowledge graphs to store data generated from different levels of research that are useful for model reproducing, risk management, process evaluating, etc.
Recommendation System: One of the interesting applications of knowledge graph is the recommendation system. A certain number of entertainment companies try to store a large amount of information for recommendation purposes. This would provide them with finding relationships between each component of movies, and later, these relationships may be used to predict what viewers might be interested in watching next.
Supply Chain Management: Knowledge graph can also be applied in supply chain management. This can help companies manage their inventories cost-effectively regarding various components, staff involved, time, etc.

6.4 Conclusion

In this chapter, an overview of using knowledge graphs in machine learning models was provided, and a certain number of previous studies were described. Those aforementioned studies showed that when the information from the knowledge graph is used in the model, the performance of the model could be better because combining the knowledge graph with machine learning methods will systematically assist in improving the accuracy of the systems and expand the range of machine learning capabilities.

We also named a few use cases. Among those, recommendation systems and question-answering are on the top, and there are many other domains that the application of knowledge graph is still undiscovered.

Chapter 7

Metamodeling for Prediction and Interpretation

Complex Simulation or Deep Neural Network models may have very long run times or be hard to interpret, so a simpler Metamodel (model of the complex model) may be created. This section evaluates some famous metamodels that are applicable to machine learning interpretation and optimization. We will focus on their motivations, assumptions, strengths and weakness. Therefore, proposed metamodels are mostly explained through intuitions and experiment performance rather than mathematical formulations. With extensive experiments across regression and classification problems running on two datasets, we will show that how such metamodels can construct the rules, develop constraints and analyze frames for surrogating a target black-box model. We will expand the discussion on the efficiency, simplicity, accuracy and robustness of each model, and argue which metamodels could potentially be useful for machine learning interpretation and optimization tasks.

7.1 Metamodeling

Metamodeling is a branch of computer experiments in the design of experiments (DOE), where a metamodel can be estimated by sampling data points from a complex model that has been trained expensively by using extensive resources. Metamodeling (or surrogate) is utilized when an outcome of interest in a complex model cannot be directly identified easily. In this situation, traditional DOE methods would fail to capture the structure and relationships (geometry) of the model. Modeling geometry effects is vital when the purpose of the experiment is a study of how combinations of various inputs influence an output (response) measuring quantity or quality characteristics. This enables us to identify the relationship between the input (predictors X) and output (response y) variables in a complex machine learning process, by fitting a global metamodel to the underlying model. The aim of (Kianifar & Campean, 2020) was to present a fair evaluation on several metamodels widely used by practitioners, comparing different characteristics of the techniques regarding robustness, accuracy, efficiency, etc. Given the fact that each metamodel is able to focus on one or two aspects of a model, they showed that not all techniques are powerful in every aspect, so the metamodeling strategy should be selected carefully. The work provides a comprehensive evaluation of techniques for each aspect, and it concluded that Gaussian Process could outperform other metamodels in some aspects.

7.1.1 Evaluation

To assess the performance of different Metamodelling techniques we perform an empirical analysis on a set of Metamodel techniques namely as Linear/Logistic Regression(LR), Support Vector Regression (SVR), Random Forests (RF), Gaussian Process Regression (GPR) or Kriging, and Neural Networks(NN). This work by experiments shows which Metamodel is efficient for modeling. Also, we attempt to evaluate the proposed models on the simplicity and accuracy and conclude the best and competitive models for metamodeling tasks. In addition, other aspects of usability such as interpretability and usefulness for optimization are rated, and the findings are summarized as a table.

The ultimate goal of most Metamodeling techniques is to learn the characteristics of a complex model by highlighting the properties of the model itself. An underlying model abides by its metamodel in a way that a computer program abides by the grammar of the programming language in which it is coded.

With this goal in mind, we would like to evaluate the Metamodels by comparing them based on several factors such as but not limited to interpretability, accuracy, computational efficiency and robustness. This section provides a systematic analysis of the results of the study.

For ease of exposition, we decompose the evaluation techniques into two groups: (1) classification problem; and (2) regression problem. We trained 5 Metamodels (RF, SVR(M), NN, LR and Kriging) on two datasets including Boston house price and the diabetes diagnosis. The Boston house price dataset (Harrison Jr & Rubinfeld, 1978) is mostly used as a regression problem with a continuous response variable, and the diabetes diagnosis dataset (Dua & Graff, 2017) is presented as a classification problem with a binary response variable. Both datasets are publicly available.

7.1.2 Accuracy

Table 7.1 summarises the accuracy of five popular Metamodel techniques for two complex models, Neural Network and GBM on the classification problem. For each of these Metamodels the accuracy as well as the Kappa scores, in parenthesis, are presented. The Accuracy describes the amount of deviation of the Metamodels from the complex model outputs. The accuracy measure is the average results of 10 runs, each run is evaluated based on a 10-fold cross-validation resampling procedure. G-FORSE has outperformed all the Metamodels for Complex Neural Networks, and has a comparable accuracy rate for GBM. GLM, RF and G-FORSE have similar accuracy rates on the GBM model. Simple Neural Networks have slightly better accuracy in this category, while SVM has the least accuracy rate.

The other class of problem that we analyze here is the regression problem. Table 7.2 summarizes the Root Mean Square Error(RMSE) and R^2 metrics for the Metamodels used on the Boston Housing Dataset. G-FORSE has the least RMSE and the highest R^2 on both of the complex models, suggesting that it is the most accurate Metamodel in this class of problems. Random Forest is the next model with the highest RMSE, which is followed by Neural Network.

It is also seen that some of the metamodels have not been able to accurately model the Neural Network Complex model compared to GBM. There could be

Table 7.1: The average of the accuracy and Kappa scores obtained via five methods for two complex models (neural network and GBM) on Diabetes dataset.

		Algorithms - Accuracy(Kappa)	
		Neural Network	GBM
Metamodels	GLM	0.789(0.466)	0.955(0.902)
	SVM	0.789(0.463)	0.952(0.895)
	RF	0.789(0.482)	0.955(0.901)
	NN (simple)	0.786(0.463)	0.957(0.905)
	G-FORSE	0.841(0.574)	0.955(0.902)

several reasons to explain this phenomenon; first, the Housing dataset is skewed and the distribution has not been targeted here. The second reason is that the metamodels such as LM and SVR are not complex enough to model a complex architecture such as Neural Network. One may argue that metamodels should be simple, but we should note that there is a certain degree of trade-off between accuracy and interpretability.

Table 7.2: The average of the root mean squared error (RMSE) and R^2 obtained via five methods for two complex models on Housing dataset.

		Algorithms —– RMSE(R^2)	
		Neural Network	GBM
Metamodels	LM	2.607(0.619)	1.625(0.96)
	SVR	2.736(0.587)	1.588(0.96)
	RF	1.066(0.940)	1.480(0.97)
	NN (simple)	1.826(0.808)	1.750(0.95)
	G-FORSE	0.022(0.980)	1.44(0.98)

7.1.3 Robustness

In this section, we will evaluate the robustness of the metamodels on the same complex models as the previous section. We will use the standard deviations of the accuracy rates obtained from multiple runs of each metamodel to assess the robustness of the classification problem. Table 7.3 summarizes the findings on robustness for each Metamodel. We can see that the GLM metamodel has the smallest SD of accuracy, followed by NN and RF on Neural Network complex models. This suggests that GLM metamodels are potentially the most robust compared to the other metamodels. Although all metamodels have very similar SD of accuracy on the GBM complex model, the same three Metamodels, GLM, NN and RF have a slightly lower SD. Figure 7.1 also presents this by using box plots of the Accuracy and Kappa score on the Diabetes dataset. We also assess

Table 7.3: The standard deviation of the accuracy obtained via five methods for two complex models on Diabetes dataset.

		Algorithms - $\mathrm{SD}_{Accuracy}$	
		Neural Network	GBM
Metamodels	GLM	0.039	0.021
	SVM	0.045	0.022
	RF	0.042	0.021
	NN (simple)	0.041	0.021
	G-FORSE	0.047	0.022

the robustness of the regression problem and present the results in Table 7.4. The Standard Deviation of the RMSE is used to indicate the robustness of each Metamodel. Random Forest appears to be the most robust with the least SD of RMSE. LM and G-FORSE are the next two robust models and finally, NN and SVR follow. The same order of robustness of the metamodels is seen on both complex models.

Table 7.4: The standard deviation of the root mean squared error (RMSE) obtained via five methods for two complex models on the Housing dataset.

		Algorithms - SD_{RMSE}	
		Neural Network	GBM
Metamodels	LM	0.332	0.626
	SVR	0.451	0.726
	RF	0.263	0.398
	NN (simple)	0.373	0.531
	G-FORSE	0.347	0.694

Figure 7.2 presents the RMSE box plots for the Boston Housing dataset. It can be seen that Random Forest's boxplot is more concentrated compared to other metamodels suggesting that it is potentially more robust in modeling the complex models.

7.1.4 Efficiency

The running time is heavily impacted by the number of parameters involved in the algorithm. So if a metamodel has many hyperparameters to tune, then the running time would be relatively large. This run time could be even longer if any extra operations occur during the training stage. For example, a family of Gaussian process metamodels (including G-FORSE) is considered an expensive model, because they require to calculate the correlation between data points, leading to engage in high computations. This complexity is not the case in a linear regression where the computations are focusing on the variables' space. The neural network model is also notorious for having a high running time, due to exorbitant calculations that occur in the feedforward and backpropagation of the model.

Apart from a theory discussion behind the running time of the metamodels, we also evaluate the time spent on training and testing of the Metamodels over 100 runs on both classification and regression problems. In regards to both of the problem categories, LM and SVM have the fastest run times in seconds and G-FORSE is the slowest. This is the case in both of the complex models. Reporting run times in seconds has some flaws, since it is heavily hardware and software dependant, so it is suggested that the time efficiency using the time complexity of the Metamodels would also be studied and compared with the run times. Run times in seconds are reported in Tables 7.5 and 7.6.

7.1.5 Simplicity

In this section, we would like to compare the simplicity of the Metamodels. Simplicity refers to the ease-of-use of the model, as well as the simplicity to implement and tune the model (Østergård et al., 2018).

Table 7.5: Time spent to perform 100 runs of training and testing for the Metamodels on Diabetes Dataset (classification problem)

		Time spent in seconds (s)	
		Neural Network	GBM
Metamodels	GLM	1.445	1.580
	SVM	9.801	7.173
	RF	94.662	65.071
	NN (simple)	27.070	25.713
	G-FORSE	246.679	264.654

Table 7.6: Time spent to perform 100 runs of training and testing for the Metamodels on Boston Housing Dataset (regression problem)

		Time spent in seconds (s)	
		Neural Network	GBM
Metamodels	LM	1.034	1.111
	SVR	9.624	6.833
	RF	226.474	217.335
	NN (simple)	18.859	18.854
	G-FORSE	13.417	15.028

Neural Network and Random Forest are considered to be the most complex of the methods discussed in this study. They require several hyper-parameters to tune for optimization which could be complex and is not intuitive. On the other hand, LM is the simplest model because of the lack of parameters to tune, the only parameter that needs to be tuned is the number of variables in the model. SVM needs parameter tuning depending on the type of Kernel used in the model, so it is not easy to generally categorize it as a simple or non-simple model. And finally, G-FORSE needs p and θ parameters to be tuned, and more information can be found in the related article (Toutiaee & Miller, 2020).

7.1.6 Interpretability

An interpretable model helps us gain a good level of understanding of the factors which are (and not) included in the model and also account for the parameter importance and interaction effects. In this section, we compare and analyze the Metamodels based on their interpretability.

A major advantage of linear regression models is linearity: It enables the estimation to be simple and, most importantly, these linear equations are easily perceivable for humans. In LM Metamodels, a linear combination of the features with the weights is used to predict the outcome. The weights provide a transparent interpretation of the model architecture and thus make such linear models highly interpretable.

Yet another interpretable but powerful model is G-FORSE, approximating the original model through the Kriging process. G-FORSE aims to provide transparent information on the complex model or simulation, and this mission has been reflected in the method's manifest by the authors. G-FORSE is most straightforward to apply when the basis function is Gaussian product correlation, and it is very similar to a Gaussian process where the observations have Gaussian basis function described with two characteristics functions $\mu(x)$ (mean) and $\mathbb{C}(x, x')$ (covariance).

RF models obtain information on the feature importance. Each tree in the RF model may be intuitive and easy to interpret, however, when working with many trees in an aggregated model, it becomes difficult to interpret the interaction effects of the features and thus to interpret the model. The complexity of the model enables the RF to predict with quite high accuracy. One negative side effect of high performance in a model is the tendency of the model to be less interpretable. So inevitably there is a trade-off between accuracy and interpretability which one should take into account in the modeling process.

With SVM models, we capture the non-linear relations between variables by projecting their features into the kernel space. Interpreting the model is heavily relying on the kernel-basis function. This kernel could be chosen from analytical functions (e.g. family of Gaussian functions), which are transparent to the viewer, so interpretability is straightforward in terms of the parameters involved in the function. However, if the kernel is replaced with some black-box function trained by other learners, then there might be some difficulties in the transparency of SVM. So we classify such models in the gray area in between highly interpretable and not interpretable.

Simple Neural Networks with just one hidden layer may be easier to interpret compared to RF. However, it is worth noting that in some literature such as (Østergård et al., 2018), the authors believe that NN is possibly the least interpretable method since it provides very little insight into the structure of the approximated function.

7.1.7 Sensitivity Analysis

Sensitivity analysis is the study of the influence of uncertain input variables or a group of variables on the output. It is widely used in many applications and design problems. Since most complex model building cases and design problems are computationally expensive, metamodels are often surrogated for facilitating computations, sensitivity analysis and optimal solution direction. Sensitivity analysis is grouped into three main categories:
Global sensitivity analysis: It helps practitioners understand the distribution of the target outcome based on the input features. The Global sensitivity analysis is very difficult to achieve in practice. Any model that involves many parameters or weights is unlikely to fit into the memory of the average human. While Global sensitivity analysis is usually out of reach, there is a chance of understanding at least some models on a modular level. In linear models, the sensitivity analysis is defined as the weights, for regression trees it would be the splits and leaf node predictions.
Local sensitivity analysis: The local sensitivity analysis investigates a complex model on a single instance and examines what the model predicts for a particular input. LIME (Ribeiro et al., 2016) is one example in this group where it relies on the assumption that every model can be analyzed locally. Thus, the prediction may only depend linearly on some features, rather than having a complex dependence on them.
Hierarchical sensitivity analysis: This family of analyses focuses on how the input features can hierarchically influence the output of an original model. Very few prior studies have been done on this group in metamodeling.

Variable importance and sensitivity analysis are interchangeably used in the literature. The global sensitivity analysis provides a wider range of inputs analysis, thus variable importance could be considered as a special case of the sensitivity analysis where the influence of feature inputs is controlled for the prediction task. Many metamodels have been proposed for the purpose of global sensitivity analysis (Saltelli et al., 2008). This report focuses on five metamodels, namely Random Forest, Support Vector Regression (Machine), Linear Model, simple Neural Network and G-FORSE.

G-FORSE method relies upon the assumption that the surface of the underlying model being supported is a sample map of the Gaussian random field. G-FORSE estimates the black box information through a Kriging process (Kaymaz, 2005) by defining a model consisting of two parts: linear regression part and non-parametric stochastic part. The G-FORSE framework is most straightforward to apply when the basis function is Gaussian product correlation of the form $\psi(\boldsymbol{h}) = exp\left(-\sum_{j=1}^{k}\theta_j h^{p_j}\right)$. The core functionality of G-FORSE on the global sensitivity analysis is under calibration of the correlation parameters θ_j. Practically, we assume that $cor[Y(\boldsymbol{x}^{(i)}), Y(\boldsymbol{x}^{(l)})]$ reflects our expectation function (equation 1) and it is smooth and continuous in the defined space. Such assumptions provide some correlations between a set of random variables $\boldsymbol{Y}$ that are relying on parameters θ_j and p_j and the distance between points $|x_j^{(i)} - x_j^{(l)}|$. The likelihood function, which can be expressed in terms of the sample data, provides us the concentrated log-likelihood function which optimizes the locations of unknown parameters, and consequently, it enables us to determine the rank of importance of variables.

Neural Network and Random Forest are one of the rich studies, used in various domains. The main disadvantage of using such techniques is the tendency of both models to be more complex when they are trained for a prediction task, while a high accuracy is desirable. Although scholars utilize them for feature selection tasks, the explanation of why the selected features are chosen is not transparent due to the nonlinear relationships existing in the models. The complexity of the models discourages the practitioners to use them for sensitivity analysis. Another weakness of NN is it needs a huge amount of data to run, thus the users might abandon it for a simpler model in case of the low amount of data.

The features are selected in the random forest by measuring how much each feature can generate information. The information encodes how much variance or *Gini* index would be dropped compared to the top node, and this amount of information can be averaged across trees to determine the final importance of the variable.

Support Vector Regression is similar to other discussed methods, such that SVM is able to rank the input variables by their importance. Thus, it provides one with a range of informative features that contribute to the model jointly or individually. One positive side-effect of SVM is that the scholars are able to choose their desired kernels for fitting to the original model. Whilst SVM could become relatively difficult for explanation in the form of global sensitivity analysis, SVM naturally integrates the properties of optimization and sensitivity analysis, which is helpful for some problems.

Linear Regression is perhaps the most popular and simple metamodel for sensitivity analysis tasks. This is simply because the linear regression is easy to use and interpret. The feature importance is calculated by the absolute value of its t-statistic measurement, which is derived by:

$$t_{\hat{\beta}_j} = \frac{\hat{\beta}_j}{SE(\hat{\beta}_j)} \tag{7.1}$$

Equation (1) has two definitions of the feature importance. First, the feature is less important if the estimation is not reliable (i.e. higher variance). Second, the weight's magnitude indicates the importance of a feature in an estimated model. The linear model has the ability to perform the sensitivity analysis both locally and globally if accuracy is not important in the analysis.

7.1.8 Potential Optimization of System being Modeled

Model optimization is being practiced widely by practitioners. From "tuning the hyperparameters" of classical machine learning, to "meta-learning" in the recent revolutionized artificial intelligence. The common aspect of the two contexts is to optimize the underlying model by tweaking some parameters. Scholars categorize the meta-learning approaches into three groups:

- One-shot learning with memory augmented neural network (Santoro et al., 2016).
- Optimization as a model for few-shot learning (Ravi & Larochelle, 2016).
- Model agnostic meta-learning (Finn et al., 2017).

Although this branch of machine learning has received some attention among the scientist, Metamodeling of the advanced trainers has been less studied for the purpose of faster convergence.

Among the proposed Metamodels, those with the *interpolation* property are in the interest of this article. Since most machine learning methods, such as neural networks and random forest, *extrapolate* through training over a large number of data points, thus they are not helpful for optimization tasks. The interpolation encourages the Metamodel predictor to precisely pass through every data point and leads to zero error at each sample point. This phenomenon will be discussed in more detail in the following.

Fig 7.3 illustrates the G-FORSE model for predicting unseen points by use of few points. G-FORSE provides two piece of information $\hat{y}(\mathbf{x})$ and $s(\mathbf{x})$, which are the predicted response value and the prediction error, respectively. Thus, the reliable surface for each prediction response value is given, as a result of the positive-side effects of using such Gaussian family function. Those parameters $\hat{y}(\mathbf{x})$ and $s(\mathbf{x})$ are useful in optimizing the objective function.

The region of exploration for a system being calibrated is essential in the optimization problem. One of the important criteria widely used by scholars is the *Expected Improvement* (EI) criterion (Jones et al., 1998). Many unconstrained optimization problems are evaluated by the EI information, namely as:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ s.t. \quad & x_i^l \leq x_i \leq x_i^u, i = 1, 2, \ldots, n \\ & \mathbf{x} = [x_1, x_2, \ldots, x_n] \end{aligned}$$

where n is the number of design inputs and x_i^l and x_i^u denote lower and upper bounds, respectively. Since optimizing this unconstrained problem is challenging, one might benefit from using G-FORSE in approximating such objective function of the underlying model. EI information is captured by G-FORSE prediction and the error function, the next sample point is selected through the EI direction per iteration, and the underlying model is optimized accordingly. The EI criterion helps the metamodel find the optimal direction and narrow the search solution, despite having a plethora of local and global search regions. An unknown point $\mathbf{x}$ is defined by a normal distribution as follows:

$$Y(\mathbf{x}) \sim N(\hat{y}(\mathbf{x}), s(\mathbf{x})) \tag{7.2}$$

We aim to locate the current optimal solution as a result of minimizing the function f. Thus, we can consider the searching steps as a random variable such that:

$$I(\mathbf{x}) = \max(f_{min} - Y(\mathbf{x}), 0) \tag{7.3}$$

To fix the random variable $I(\mathbf{x})$, we utilize the property of *expectation* in the probability theory, namely as:

$$\begin{aligned} EI(\mathbf{x}) &= E[\max(f_{min} - Y(\mathbf{x}), 0)] \\ &= \int_{-\infty}^{f_{min}} (f_{min} - Y) \frac{1}{\sqrt{2\pi}(\mathbf{x})} exp\left(-\frac{\hat{y}(\mathbf{x}^2)}{2s(\mathbf{x})^2}\right) dY \end{aligned}$$

The integral in the equation (5) is calculated to obtain:

$$EI(\mathbf{x}) = (f_{min} - \hat{y}(\mathbf{x}))\Phi\left(\frac{f_{min} - \hat{y}(\mathbf{x}^2)}{s(\mathbf{x})}\right) + s(\mathbf{x})\phi\left(\frac{f_{min} - \hat{y}(\mathbf{x}^2)}{s(\mathbf{x})}\right) \tag{7.4}$$

where $\phi(.)$ and $\Phi(.)$ denote as the probability density function and the cumulative distribution function, respectively, of the standard normal distribution.

Equation (6) shows mathematically as to why a "regression" model would not be appropriate for interpolating values since the error of estimation should be zero around the known values by passing through all the data values. Whereas in regression, we generalize the points by minimizing the cost function. Thus, most regression models are excluded from optimization problems if they fail passing through data points. Among the studied metamodels in this work, G-FORSE and SVM (Huizan et al., 2008) are able to provide interpolation between the data values, thus scholars might consider them for further explorations in optimization problems.

7.2 Evaluation Summary

The experimental results on two datasets show that three of the five metamodels yield good overall accuracy. Among them, G-FORSE performs on par with the RF model, whilst providing interpretability for the original model being approximated. In addition, the SVR(M) has a modest running time and hence, is applicable to large problems with rich feature space. Moreover, the linear regression appears to be the fastest and simplest metamodel, in which the variables are the only parameters to be tuned linearly, and therefore, lower accuracy is expected. The discussed information is provided in the summary table 7.10.

Table 7.7: Challenges in Metamodels: a summary of the pros and cons of the different methods to metamodel the complex models for each of the major operations involving a method.

	RF	LR	SVR(M)	NN (simple)	G-FORSE
Accuracy	🙂	🙁	😐	🙂	😐
Robustness	🙂	🙂	🙁	😐	😐
Efficiency	🙁	🙂	😐	🙁	🙁
Simplicity	🙁	🙂	😐	🙁	😐
Interpretability	🙁	🙂	🙁	🙁	🙂
Sensitivity Analysis	🙁	🙂	🙁	🙁	🙂
Use for Optimization	🙂	🙁	🙂	🙁	🙂

7.3 Black-Box Interpretation

With the prompt acceptance of sophisticated Artificial Intelligence (AI) models in the industry for solving problems, Machine Learning Interpretation (MLI) is not a fancy direction but a need. Model explanation enables us to interpret and legitimize the outcomes of a predictive model to accredit Fairness, Accountability and Transparency (FAT) in making decisions. In other words, the theory and mechanics of algorithmic decisions should be deciphered such that the relationship between inputs and outputs is understandable to humans.

Determining FAT in predictive models is a major problem when a model is used for decision making. When using a complex gradient boosting model (GBM) with rich parameters to predict membership, for example, prediction cannot be acted upon as an interpretable algorithmic process due to the fact that thousands of pre-trained regression trees seem like a black-box to humans.

Apart from understanding the black-box process, the complex model should have variable relationships evaluated before launching it into production. To make this decision, humans require to validate what has actually been trained via an algorithmic model to what they perceive from that model. That is, they will not expect to see correlations between a group of variables and a target in a prediction problem when the model has excluded that group from the prediction process. In general, machine learning interpretation is evaluated based on Local Inference (single instance) and Global Inference (entire or part of data). However, real-world models are often significantly complex, and further, we do not have enough understanding of why and when they work well, and why they may fail completely when faced with new situations not seen in the training data. A transparent and simplified version (a surrogate or metamodel) of a complex model is an effective solution, in addition to such methods. In this case, a metamodel provides the best approximation to the underlying model by minimizing a metamodeling loss $\ell(g, f)$, where g and f are the metamodel and black-box, respectively.

One aspect of metamodeling that has been less studied is "black-box interpretation", and this part aims to provide thorough insights by introducing a new technique for machine learning interpretation through the Gaussian Process method. A metamodel is useful for interpretation because it is a simplified copy of the black-box model. A complex prediction model represents raw data and conforms to a metamodel.

In this section, we propose a metamodel for global interpretation as a solution to enable the prediction of FAT and optimize the metamodel by interpolation to explore the relationships between several explanatory variables and one or more response variables. Our main contributions are summarized as follows.

- The framework that can explore the relationships between several input variables and one or more target variables by metamodeling.

- The metamodel that can reveal the structure and relationships between data points, while simplifying the explanations of the complex model.
- A surrogate for a simple model such as Logistic Regression model that can suffer from instability of estimation as a result of Hauck-Donner effect.
- Comprehensive evaluation of metamodeling in interpreting a complex model on some datasets, where we evaluate the strength of a feature for prediction. In our experiments, the final fitted model obtained from our proposed approach is more interpretable and decipherable to the viewers. We also show how the mechanics and theory of metamodel characterized by the "meta" of metamodeling can be used to emulate the distribution of a black-box efficiently by employing the black-box's resources.

7.4 Interpretable Models

Until recently, most work on interpretable machine learning models focused on feature importance estimation. (Adadi & Berrada, 2018) reviewed a wide range of methods used for explainable artificial intelligence (XAI), and the work categorized the techniques into six groups where each technique can cover some groups. Although the authors have not presented metamodeling as a separate technique in their work, metamodeling can be grouped by "surrogate" or "model distillation" where the work reviewed them well. The work has been published under the assumption that LIME (which is based on linear regression), model distillation, surrogate" models and decision trees should be considered individually, while their functionalities are quite similar to each other in that all the techniques can explain the black-box by another understandable model. This class of models that turn black-box into white-box using another simpler model is the main motivation of this work to group them into a general group called "metamodels". This encourages a unified definition and a clearer connection between simpler models, and leads to incorporate more range of interpretable models.

7.4.1 Model-specific Interpretation

Integrated Gradients (Sundararajan et al., 2017), Simple Gradients (Simonyan et al., 2013), DeepLIFT (Shrikumar et al., 2017) and DeepSHAP (Fernando et al., 2019) are examples of machine learning interpretation for gradient based models. These methods tend to interpret the global variable selection where they select the global variables through instance-wise learning. The instance-wise variable selection has been extended by some recent studies, **L2X** (Chen et al., 2018) and **Invase** (Yoon et al., 2018), whilst they have three differences in the variable selection task: First, L2X closes the gap between the response Y and the selected input features X_S by maximizing a lower bound of the mutual information, while Invase uses KL divergence to close that gap. Second, Gumbel-softmax distribution (Jang et al., 2016) used by L2X, enables the back-propagation through subset sampling, however, Invase bypasses that backpropagation using actor-critic models (Peters & Schaal, 2008). And third, the number of selected variables for every sample should be fixed in advance, and this is imposed by Gumbel-softmax distribution, whereas the actor-critic methodology used by Invase would lift such limitations. This flexibility in Invase encourages relevant features to be selected freely and leads to directly inject L_0 regularization term for inducing sparsity. Both models are categorized as model-agnostic and non-additive, and they need training before applying.

Class Activation Maps (CAM) (B. Zhou et al., 2016) and Saliency maps (Simonyan et al., 2013) have been introduced for visualizing the predictions of deep convolutional network for images. The CAM method that has received more attention, introduced a method to explicitly show how a DNN discriminates between different regions of the trained image by revisiting the global average pooling layer introduced by (Lin et al., 2013). The Global Average Pooling (GAP) used in the CAM method enables the method to identify the group of features which contribute to the prediction. CAM is defined mathematically as:

$$S_c = \sum_{x,y} M_c(x,y) = \sum_{x,y} \sum_{k} w_k^c f_k(x,y) \tag{7.5}$$

where $f_k(x,y)$ is the activation of unit k reported by the last Conv layer, and w_k^c is the weight of class c for unit k. Intuitively, this expression is described as the summation of the similarities between the last convolutional feature map and the class-wise weights of the fully connected (FC) layer. The CAM hence sums all the visual patterns of $f_x(x, y)$ at different locations by multiplying the relevant weights w_k^c by them. The method then generates an image-level class label by passing the sample of regions in the input image through a softmax/sigmoid activator.

DeepLIFT relies on the assumption that what matters in a neural network model is not the gradient, which describes how y changes as x changes at the point x, but the slope, which describes how y changes as x differs from the baseline. So if we consider the slope instead of the gradient, then we can redefine the importance of a feature as the following equation:

$$x_i \times \frac{\partial Y}{\partial x_i} \rightarrow (x_i - x_i^{baseline}) \times \frac{Y - Y^{baseline}}{x_i - x^{baseline}} \tag{7.6}$$

As such, neural networks and their variants are good candidates for being explained by these methods, since the gradient can be used to explain the outcome of a neural network. Due to the approximation of the gradient in their estimation pipelines, they are relatively fast in computation.

DeepLIFT tends to define the concept of the multiplier by attributing the difference in output to certain neurons by using a multiplier, defined as $m_{\Delta x \Delta t} = C_{\Delta x \Delta t}/\Delta x$, where x and t are the input and target neurons, respectively. Δ denotes the difference between the neuron and the baseline. Equation (2) indicates that we measure change relative to the baseline. This formulation can be illustrated in Figure 7.5. Per DeepLIFT's idea, the input x had an importance value of -2. In other words, the input $x = -2$ changed the output of the model by -2 compared to the baseline. This example is solid evidence as to why inconsistency happens when we apply other conventional interpretable models. As it is shown in Figure 7.5, $ReLU(x = 2) = 2$,and $ReLU(x = -2) = 0$, so the input feature $x = -2$ has changed the output of the model by 2 compared to the baseline. This change in the output of the model has to be attributed to the change in x, since it is the only input feature to this model, but the gradient of $ReLU(x)$ at the point $x = -2$ is 0. This is indicating the contribution of x to the output is 0, which is obviously a contradiction.

For each layer, the slope is calculated by:

$$\text{slope} = \frac{y - y^{baseline}}{x - x^{baseline}} = \frac{\Delta y}{\Delta x}. \tag{7.7}$$

This is the slope that DeepLIFT calls "multiplier", and it denotes by m. Having defined a new gradient (slope), one can apply the chain rule and backpropagation property to estimate the changes relative to the baseline:

$$\frac{\partial F}{\partial x} = \frac{\partial Y}{\partial x}\frac{\partial F}{\partial Y} \rightarrow \frac{\partial F}{\partial x} = \frac{\partial Y}{\partial x}\frac{\partial F}{\partial Y} \tag{7.8}$$

The slope between input and output of a model can be obtained via backpropagating along these multipliers, and the expression is as follows:

$$\text{Feature Importance} = (x_i - x_i^{baseline})\frac{\Delta_i Y}{\Delta_i x}. \tag{7.9}$$

In DeepLIFT, and similarly in the gradient-based methods overall, choosing an appropriate baseline for a model is very challenging, and it might require domain expertise. One option to pick a baseline is to consider what the prior distribution of a trained model is. For example, a naive but effective approach is to take the average of the dataset, simply by finding the mean of all samples. This approach has been suggested and utilized in several articles, where it removes the need to be a domain expert to pick an appropriate baseline for the model being interpreted.

DeepLIFT redefines how gradients are calculated in the model is explained The model-specific property of DeepLIFT makes it less flexible for interpreting all types of models and hence, every deep learning model needs to be examined heavily into the nuts and bolts to implement this technique. However, one backward pass of the model enables the algorithm to calculate feature importance values rapidly.

Integrated Gradients (IG), yet another gradient-based method, is built over a simple idea that no prerequisite in network architecture modification for computing feature importance. The inventors claim that IG can be utilized in many ways: from rule or feature reasoning, to model debugging. Integrated Gradients also relies on a baseline that resembles the gradient with emphasis on measuring the difference between input and baseline. A part of the problem is that the gradient should acknowledge the difference between baseline and output being measured. We will discuss more on this issue in the following.

Integrated Gradients, as a self-explanatory method, computes the integral of gradients between desired input and a pre-defined baseline. This integral is approximated using *Riemann Sum* or *Gauss Legendre* quadrature due to not being solved analytically. The exact expression is represented as follows:

$$IG_i(x) ::= (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha \tag{7.10}$$

where α is the scaling coefficient and IG_i represents the Integrated Gradients of ith feature of X along the path from a given reference to input. The sensitivity and implementation invariance are two critical factors that the inventors identified for this method. More information regarding these axioms can be found in the original paper available in the references (Sundararajan et al., 2017). Although this concept contains more details that authors have incorporated in their article, we try to list a few but important properties they claim their attribution method is supposed to satisfy: completeness, linearity preservation, symmetry preservation and sensitivity. We discuss briefly each in the following.

A) *Completeness*: One important property that Integrated Gradients ought to meet is that the difference between the prediction scores of the input and the baseline should be on a par with the sum of the attributions from Integrated Gradients results. The paper claims that this property could be supported by the gradient theorem. Intuitively, this property can ensure the end-user having a reliable prediction from the model.

B) *Linearity preservation*: If a network F is a linear combination $aF_1 + bF_2$ of two networks F_1 and F_2, then a linear combination of the attributions for F_1 and $F - 2$, with weights a and b respectively, is the attribution for the network F. This property is desirable because the attribution method preserves any linear trend present within a network.

C) *Symmetry preservation*: Integrated gradients protects symmetry. Symmetrical property in a network means we expect to see symmetrical feature attributions if two input features are treated symmetrically in a network. For instance, suppose a function F contains three variables x_1, x_2, x_3, and $F(x_1, x_2, x_3) = F(x_2, x_1, x_3)$ for all values of x_1, x_2, x_3. Then F is symmetric in the two variables x_1 and x_2. If the variables have equal values in the input and baseline, i.e. $x_1 = x_2$, and $x'_1 = x'_2$, then symmetry retention enforces that $a_1 = a_2$.

D) *Sensitivity*: The inventors of IG defined two aspects of sensitivity:

- Non-zero attribution represents the baseline and the input feature differs only in one feature while having different predictions.
- No activity of a feature, no attributions for that feature.

One major issue with IG is that it is practically not as accurate as DeepLIFT. In comparison with the DeepLIFT method, IG tends to yield unreliable results based on our experiments for this purpose. IG seems to be inconsistent with our true GLM model's weights generated for our experiment. Another issue with IG is the time complexity where it requires $O(n^3)$ order of time to compute the attributions. Thus, it is slower than DeepLIFT. However, IG has motivated many other toolsets including but not limited to "Layer Conductance" (Dhamdhere et al., 2018b) and "Internal Influence" (Leino et al., 2018).

7.4.2 Model-agnostic Interpretation

The second group of interpretable methods is studied under sensitivity analysis. In this family of models, perhaps the most successful one is LIME (Ribeiro et al., 2016), which focuses on the predictions of the underlying complex models to explain individual predictions. LIME relies on the assumption that every complex model can be explained linearly on a local scale, and the following equation should be calibrated to generate an explanation:

$$\xi(y) = arg\min_{\beta} \left\{ \sum_{i=1}^{N} exp(-\frac{D(y,z)^2}{\sigma^2})(f(z_i) - \beta z'_i)^2 + \infty \mathbf{1}_{\{||\beta||_0 > K\}} \right\} \tag{7.11}$$

where $\beta z'_i$ denotes the class of linear models as an explanation family G, the similarity function $exp(-\frac{D(y,z)^2}{\sigma^2})$ denotes an exponential kernel defined on some distance function D as a weight function, $||\beta||_0$ denotes the L_0 norm (i.e. the number of non-zero entries of β) limiting to K parameter which is crucial for the explanation task. One can solve the above equation by selecting K features using a subset selection method, so the coefficients can be estimated through Weighted Least Squares (WLS). In the paper, the authors utilized LASSO regularization for feature selection, the so-called K-LASSO.

In the original paper, the authors described the approach that LIME uses to approximate the solution of the above equation and produce an explanation for instance y as follows:

A. It generates N perturbed samples for each prediction to explain by defining $\{z'_i \in X' | i = 1, \ldots, N\}$ as the set of these observations.

B. LIME would recover the perturbed observations in the original feature space by means of the mapping function defined by $\{z_i \equiv h^y(z'_i) \in X | i = 1, \ldots, N\}$.

C. Then LIME lets the black-box model predict the outcome of every perturbed observation by the set of responses denoted by $\{f(z_i) \in \mathbb{R} | i = 1, \ldots, N\}$. Then the dataset of perturbed sample with their responses can be denoted by $\{(z'_i, f(z_i)) \in X' \times \mathbb{R} | i = 1, \ldots, N\}$. Next, it computes the weight of every perturbed observation by defining $\{(w^y(z_i)) \in \mathbb{R}^+ | i = 1, \ldots, N\}$ as the set of weights.

D. Next, LIME would select K features describing the underlying model outcome from the perturbed dataset. It then fits a weighted linear regression model using the LASSO regularization parameter to a subset of the dataset composed of the K selected features in the previous step.

E. Finally, it extracts the weights from the fitted model and utilizes them as explanations for the black-box model.

In general, LIME has the advantage of training and explaining interpretable model candidates both in research and experience. Because LIME uses LASSO for the local explanation, it enables a human-friendly explanation which makes it more transparent. Also, LIME is one of the few methods that work for tabular data, text and images. The inventors of LIME presented a notation as the fidelity measure, where it provides users with the idea of how well the interpretable model estimates the complex outcomes. This enables users to appreciate the minimum reliability the explanation can afford to interpret the black box predictions in the vicinity of a single observation in the region of interest. LIME is implemented in Python and R and it is easy to use. However, an optimal neighborhood is difficult to define on tabular data, and it appears one of the biggest challenges with LIME. Also, the complexity of the explanation model has to be defined in advance, and data points sampled from a normal distribution can lead to some issues, since the correlation between features could be ignored in a normal distribution setting. Also, simple perturbations are not enough. Ideally, a single point can be perturbed on the variation the model injects in that vicinity. So, one can assume that perturbation is case-specific since variation can be different from one subspace to another, leading to introduce bias into the model explanation. Also, some believe that not all complex models can be explained linearly on a local scale if the region of interest is not small enough. In other words, some models need to be explained in a larger space, so the linearity assumption may not hold for such cases. They believe we can extend this idea by proposing some non-linear interpretable models.

Skater (Choudhary et al., 2018) is the extended version of LIME, where it enables one to take the advantage of using a model-agnostic method for concrete problems. It allows the underlying model to be any form of a complex model, which leads to high popularity in the community. In comparison with other popular MLI toolsets, Skater ought to reveal the learned structures of a complex model on global and local scales. We start with the global interpretation by Skater first in the next section.

Skater focuses on partial dependence and feature importance by introducing a tree family of models to provide one with a range of model-agnostic global interpretation algorithms. Among these, feature importance is one of the most sought after techniques in interpretation in machine learning which we discuss next.

A partial dependence (PD) plot illustrates the relationship between the inputs and output of a model. It shows that relationship by isolating the variable of interest from other effects in the model. PD is able to uncover the type of relationship, which is helpful for model interpretation. If one is interested in the local interpretation, then Individual conditional expectation (ICE) plots are the ones providing the level of individual observations. Both types of plots are considered model-agnostic techniques.

Feature importance is a general concept that represents a level of dependency a predictive model has on a certain feature. Skater computes the feature importance score relying on the idea that how much information one would gain in predictions by perturbing a particular feature. Intuitively, we expect if there is a strong connection between a model's decision criteria and a given feature, then we observe more variability in predictions as a function of perturbing a feature. Skater uses prediction-variance parameter computed by the mean absolute value of changes in prediction, in the event of perturbations in data.

If one is interested in conditioning on a particular feature to measure the level of influence it exerts in the model's prediction, then the partial dependence may be useful. Partial dependence values resemble feature coefficients in a linear regression equation, where one evaluates a conditional effect of a given feature. Thus, the partial dependence plot (PDP) illustrates the conditional effect of a feature on the validated outcome of a trained model. PDP provides graphical information about the trend between the output and a feature.

Local interpretation is fulfilled through two possible options. First, by mimicking the behavior of a black-box model by fitting an interpretable model in the vicinity of a single instance. Second, one may fit a simplified copy of the complex model in an effort to perceive the behavior of a single prediction by approximating a basis function between inputs and outputs. Since Skater can exploit LIME to explain model predictions on matrix data, we can call the related helper function within the package to perform local interpretation. For quantitative variables, Skater generates samples from an $N(0,1)$ and applies the inverse operation of mean and variance of the training data samples. Similarly, categorical variables are evaluated by a binary feature and assigning 1 if the perturbed value is the same as the target point being explained.

Overall, Skater can provide global and local interpretations which makes it desirable to users who are interested in more aspects of the interpretation in machine learning models. However, we were not able to run Skater on our data due to some unfixed bugs in Skater at the time of the experiment.

SHAP (Lundberg & Lee, 2017), which enables game theory to connect to local explanations, where each feature value of the instance is a "player" in a game, and the values are computed as follow:

$$Sh_j(v) = \sum_{S \subseteq \{x_1,\ldots,x_p\} \setminus \{x_j\}} \frac{(p - |S| - 1)!|S|!}{p!} (v(S \cup \{x_j\}) - v(S)) \quad (7.12)$$

where S is a subset of the features used in the model, x is the vector of feature values of the instance to be explained and p is the number of features. Such models generally perturb the input slightly and measure the variations in prediction. If the change in prediction occurred by tweaking the inputs is negligible, then the model would consider it as an unimportant variable. This model is difficult even to approximate for the curvature of response y, and the key limitation is that it is exponentially expensive due to the plethora of evaluations they make to quantify a model's sensitivity.

Another group, which is the easiest option for interpretation, is to employ a subset of methods that are inherently interpretable. Although they are convenient to use, typically they are not complex enough to achieve state-of-the-art performance, so they degrade the accuracy. Linear regression, regression trees and RuleFit algorithms (Friedman, Popescu, et al., 2008) are examples of interpretable models, however, many other transparent algorithms are constantly growing for this purpose.

Previous work has also taken the approach of including and excluding one or more groups of input variables to measure the model's outcome. These approaches are commonly used in computer vision applications, where the deep neural networks take the center stage (Lundberg & Lee, 2017; Simonyan et al., 2013; Springenberg et al., 2014; Zeiler & Fergus, 2014). These methods are somehow similar to gradient-based techniques, in that they analyze the effects of being present and absent parts of the regions to which the model representation is mostly responsive, resulting in highlighting that important region. Most work in this research track are converged into the ability to evaluate the contribution of each input variable to the target of a predictive model. This has encouraged researchers to develop specific interpretability methods that target each neuron in a given layer to the outcome of the model (Leino et al., 2018) and motivated further research to focus on the contribution of each input variable on the activation of a particular hidden unit (neuron conductance) (Dhamdhere et al., 2018a).

Some previous work have used the general concept of global explanation by setting some rules on training points (Plumb et al., 2018; Ribeiro et al., 2018), so the interpretation is run through example-based explanations. This family of explanations is considered, to some extent, as model-agnostic. The example-based models explain an underlying model through instance selection rather than feature selection of a dataset. Each training point is weighted based on the level of impact they have on the prediction, and this relationship is evaluated by an influence function. Some work utilize the idea of "counterfactual explanations" (Van Looveren & Klaise, 2019) and "adversarial examples" (Biggio & Roli, 2018; Su et al., 2019). The example-based models are applicable for image and text explanation, but tabular data remains challenging.

7.4.3 Interpretation by Metamodeling

The *symbolic metamodel* (Alaa & van der Schaar, 2019) is another recent work in machine learning interpretation, trying to decode the underlying black-box model by a metamodel. A symbolic metamodel is able to return a transparent function describing the predictions of the original model. The inputs of the transparent model $\mathcal{G}$ are the trained input variables from the original model $\mathcal{F}$, and the outputs are just some transparent equations. The white (transparent) model may contain different model spaces from polynomial to closed-form expressions, and users are able to choose their desired functions. The paper claimed that this white model $\mathcal{G}$ is achievable under Kolmogorov-Arnold representation theorem (Kolmogorov, 1957) to decompose the metamodel into *univariate* functions. The theorem proves that every multivariate continuous function $g(x)$ can be decomposed into multiple univariate continuous functions as follows:

$$g(x) = g(x_1, \ldots, x_n) = \sum_{i=0}^{r} g_i^{out} \left(\sum_{j=1}^{d} g_{ij}^{in}(x_j) \right) \tag{7.13}$$

Then the metamodel can be represented as:

$$G(x; \theta) = \sum_{i=0}^{r} G_{p,q}^{m,n} \left(\theta_i^{out} | \sum_{j=1}^{d} G_{p,q}^{m,n}(\theta_{ij}^{in} | x_j) \right). \tag{7.14}$$

The challenge is the parameter θ is not known and it should be optimized through:

$$\theta^* = arg \min_{\theta \in \Theta} \ell(f(x), g(x; \theta)). \tag{7.15}$$

The function in equation (11) is not given, so one might think of a general function as the basis function, containing the desired parameters of the metamodel being optimized. The authors used Meijer G-functions for this purpose, namely as:

$$G_{p,q}^{m,n} \left({}^{a_1, \ldots, a_p}_{b_1, \ldots, b_q} \middle| x \right) = \frac{1}{2\pi i} \int_{\mathcal{L}} \frac{\prod_{j=1}^{m} \Gamma(b_j - s) \prod_{j=1}^{n} \Gamma(1 - a_j + s)}{\prod_{j=m+1}^{q} \Gamma(1 - b_j + s) \prod_{j=n+1}^{p} \Gamma(a_j + s)} x^s ds, \tag{7.16}$$

where $\Gamma(.)$ is the Gamma function and $\mathcal{L}$ is the integration path in the complex plane. The main property of the Meijer G-function enables the metamodel to be reduced into some simpler and transparent functions, thus it should be able to interpret the black-box model. Since the Meijer G-function is differentiable, so it can be optimized by any optimizer (e.g. gradient descent) and the convergence rate is fast.

Overall, demystifying the black-box by a symbolic metamodel may have some pros and cons, however, the heavy and complicated math behind the method may discourage users to apply it to their problems.

7.5 Our Contribution

This work, Gaussian Function On Response Surface Estimation or "G-FORSE", differs from previous work in four important ways: 1) in this technique, the relationships between explanatory variables and one or more response variables are explored by studying the curvature of the underlying model via optimizing a metamodel. Spatial covariance functions can guarantee improvement of non-standard metrics of model approximation. We will further discuss this. 2) The nature of the explanation is different. In interpretable models, several transparent algorithms are available to predict outcomes. In model-agnostic methods, practitioners perturb the input variables to measure the change of outcome. In G-FORSE, we approximate the original complex model by sampling points from the underlying model under the Gaussian process assumption. 3) The specification of measuring the level of impact is different. The example-based explanation is described as generating explanations by fitting Random Forrest, yet another black-box model, to identify whether a variable with a zero coefficient contains global effects. Metamodeling is based upon maximizing the likelihood function rather than a randomized process and has a correlation function that a vector of parameters corresponding to each feature seeks to maximize to identify the global explanation of the original model. 4) Metamodeling is an interpolation technique by which the unknown values are estimated through a Gaussian process controlled by a pre-defined correlation function. This property enables G-FORSE to avoid the non-existence of maximum likelihood estimates as a result of the Hauck-Donner effect encountered in classical GLM such as Logistic Regression (Morris, 1990; Venables & Ripley, 2013). This phenomenon occurs when the fitted probabilities are extremely close to zero or one.

7.6 Gaussian Function On Response Surface Estimation; G-FORSE

Our work estimates the black-box information through a Kriging process (Kaymaz, 2005) by defining a model consisting of two parts: linear regression part and non-parametric stochastic part, which can be given as:

$$Y(x) = \mu(x) + Z(x), \tag{7.17}$$

where $\mu(x) = \sum_{i=1}^{m} q_i(x)\beta_i = q^T(x)\beta$. $\beta = [\beta_1, \dots, \beta_m]^T$ is the regression coefficient to be determined, and $q(x) = [q_1(x), \dots, q_m(x)]^T$ is function of vector x which can provide the global approximation. $Z(x)$ is a stationary Gaussian stochastic process with mean 0 and covariance function:

$$\mathbb{C}(x_i, x_j) = Cov(Z(x_i), Z(x_j)) = \sigma^2 \prod_{l=1}^{k} K(h_l; \theta_l), \tag{7.18}$$

where σ^2 is the variance parameter, $h_l = |x_i^{(l)} - x_j^{(l)}|$, $x_i^{(l)}$ and $x_j^{(l)}$ are the lth elements of the ith run x_i and the jth run x_j, k is the number of variables and $K(h_l; \theta_l)$ is a correlation function with a positive parameter θ_l. G-FORSE can be calibrated by choosing an impactful correlation function, and several alternatives such as cubic, exponential and Matérn functions have been studied in (Koehler & Owen, 1996). Among these, the Gaussian model is utilized in this work (Forrester et al., 2008), which is provided in equation 3. We aim to approximate the true underlying model $Y(x)$ by the best linear unbiased estimator (BLUE), which minimizes an objective function $\mathbb{E}\{\hat{Y}(x) - Y(x)\}^2$, under the model in equation 1. The function μ is used to identify the known trends in the equation, so it enables $Z(x)$ to be a stationary process. *Ordinary Kriging* takes μ in equation 1 as constant μ_0, which is widely used in studies (Forrester et al., 2008; P. Wang et al., 2013; Welch et al., 1992).

The G-FORSE framework is most straightforward to apply when the basis function is Gaussian product correlation of the form

$$\psi(\boldsymbol{h}) = exp\left(-\sum_{j=1}^{k} \theta_j h^{p_j}\right) \tag{7.19}$$

This function is very similar to a Gaussian process where the observations have a Gaussian basis function described with two characteristics functions $\mu(x)$ (mean) and $\mathbb{C}(x, x')$ (covariance). We intend to use a vector $\boldsymbol{\theta} = \{\theta_1, \theta_2, \dots, \theta_k\}^T$ in the G-FORSE to control the width of the basis function varying from feature to feature, while the Gaussian basis function has $1/\sigma^2$. Similarly, the Gaussian kernel tends to use a fixed exponent at $p = 2$ to enable smooth function in all dimensions for a point $x^{(i)}$. The Gaussian process is a special case of the metamodeling process when θ_j is fixed and $p_{(1,2,\ldots,k)} = 2$ for all dimensions (isotropic basis function). In the next section, we present a theoretical analysis of G-FORSE process, essentially showing that how smoothness and activeness of $\boldsymbol{p}$ and $\boldsymbol{\theta}$, respectively, affect the underlying correlation. Figure 7.6 shows an intuitive explanation of the G-FORSE process. In practice, we must view our observed responses $\boldsymbol{y} = \{y^{(1)}, y^{(2)}, \ldots, y^{(n)}\}^T$ as if they are sampled from a stochastic process, although they may be deterministic in code. Thus, our observed responses are denoted by $Y = \{Y(x^{(1)}), Y(x^{(2)}), \ldots, Y(x^{(n)})\}$ with a mean of $\mathbf{1}\mu$.

Practically, we assume that $cor[Y(\boldsymbol{x}^{(i)}), Y(\boldsymbol{x}^{(l)})]$ reflects our expectation function (equation 3) and it is smooth and continuous in the defined space. Such assumptions provide some correlations between a set of random variables $\boldsymbol{Y}$ that are relying on parameters θ_j and p and the distance between points $|x_j^{(i)} - x_j^{(l)}|$. The likelihood function, which can be expressed in terms of the sample data, provides us the concentrated log-likelihood function which optimizes the locations of unknown parameters, and consequently, it enables us to determine the rank of importance of variables.

7.7 Theoretical Results

The parameter p controls the smooth correlation with a continuous gradient evaluated on a distance between points (i.e. $|x_j^{(i)} - x_j|$). We expect variables with a high correlation tend to yield $e^{(-|x_j^{(i)} - x_j|^p)} \to 1$ since $|x_j^{(i)} - x_j| \to 0$, so $Y(\boldsymbol{x}_j^{(i)}) = Y(\boldsymbol{x}_j)$, and $e^{(-|x_j^{(i)} - x_j|^p)} \to 0$ implies zero correlation because $|x_j^{(i)} - x_j| \gg 1$. Figure 7.8b shows four different correlations settings for $p = 0.1, 0.5, 1$ and 2. It is shown that the correlation trend is affected by the smoothness parameter p. By decreasing p, the correlation drops significantly as $|x_j^{(i)} - x_j|$ reaches to 1. On the other hand, as $|x_j^{(i)} - x_j|$ increases than 1, $p < 1$ encourages the correlation to decay gradually, while there is a rapid decrease in the correlation effect for $p \geq 1$. This sudden drop encourages a saturation effect, where $p = 0.1$ leads the correlations to break the interpolation between $Y(\boldsymbol{x}_j^{(i)})$ and $Y(\boldsymbol{x}_j)$. Hence, a smooth kernel ($p = 2$) is appropriate for fitting smooth functions, while a non-differentiable kernel ($p = 1$) may be a better choice for fitting nondifferentiable functions (Bindel, 2018). Figure 7.8a presents the relationship between the θ_j and the correlation. As it is shown, a sample point can extend its effect by varying a width parameter θ. A low value of θ_j indicates a high correlation between data points, whereas a high θ_j shows a significant contribution across the $Y(x_j)$'s θ_j. For example, if one examines an underlying model where they measure the income of an individual for varying hairstyle (x_1), office location (x_2) and age (x_3), we would hypothetically expect to see $\theta_1 = 0$, since hairstyle has no impact on income, a slightly higher value of θ_2 since office location would affect the cost of living, and θ_3 would be highest since age is most correlated with income. The parameters θ_j, for $j = 1, \ldots, d$ can be interpreted as increasing θ_i makes the correlations drop to zero between data points. We can then rewrite equation 3 as:

$$\begin{aligned} K(h;\theta) &= cor[Y(\boldsymbol{x}^{(j)}), Y(\boldsymbol{x}^{(l)})] \\ &\approx exp(-\sum_{i=1}^{k} \theta_i |\boldsymbol{x}_i^{(j)} - \boldsymbol{x}_i^{(l)}|^2) \end{aligned}$$

By investigating the parameter of θ for an anisotropic basis function, we can determine the order of importance of variables in a complex model. Generally speaking, a smaller θ means that the learned function varies less in that direction, which encourages that feature to be irrelevant for the learned function (Figure 7.7).

7.8 Experiments

We demonstrate the use cases of G-FORSE through experiments on synthetic and real data. In all experiments, we used `SPOT` Gaussian Process computation library in R) to carry out computations involving G-FORSE.

We pretend the validated prediction of a trained complex model on a dataset is a new target variable, which G-FORSE aims to predict. This enables G-FORSE to provide the global explanation of the complex model after G-FORSE has been enforced to predict the new target (validated prediction) using original inputs. We trained four popular machine learning models on a range of datasets, and we tested G-FORSE on the validated outputs of those algorithms (Tables

7.8 & 7.9). We also tested G-FORSE on simulated data generated from a true GLM model with pre-defined weights and correlations to ensure it performs trustworthily. Among which, the crimes and housing price datasets are regression problems with the continuous response variables, and the remaining ones are the classification problems with the binary response variables. For fitting the G-FORSE model, the values for each variable are normalized to the 0 to 1 range. We only tuned parameter $\hat{\theta}$ since it determines the variable importance of a model, which we desire. While our theoretical framework permits the tune of parameter $\hat{p}$ and one can benefit from optimizing $\hat{p}$ to produce accurate predictions, the parameter $\hat{p}$ was fixed at $p = 2$ value because we have a smooth correlation with a continuous gradient for very close points. We estimate parameter $\hat{\theta}$ by use of differential evolution (DE) and L-BFGS-B optimizers (Byrd et al., 1995; Storn, 1995), and we limit the search region by setting the lower and upper bounds to 10^{-4} and 10^{2}, respectively. The results are provided in Tables 7.8 & 7.9 for comparison. Superficially, G-FORSE with DE optimizer achieved similar results to L-BFGS-B optimizer: Table 7.9 differs from Table 7.8 in that L-BFGS-B optimizer worked better on the Income dataset. However, both tables are showing G-FORSE performed reasonably well in approximating the most algorithms across all the datasets. The results from both tables are in line with the results on probability plots we presented in Figures 7.9 and 7.10, where they are showing that G-FORSE is successfully able to approximate the underlying predictive models.

7.8.1 Hauck-Donner Effect

There is a less-studied case for GLMs with the binary outcome that was specified first by Hauck & Donner (Hauck Jr & Donner, 1977). The Hauck-Donner effect (HDE) occurs when a Wald test statistic is not uniformly increasing as a function of separation between the estimated parameter and the null input. If there are some $\hat{\beta}_i$ which are large, the curvature of the log-likelihood at $\hat{\vec{\beta}}$ can be much less than $\beta_i = 0$, and so the Wald approximation underestimates the change in log-likelihood on setting $\beta_i = 0$. This happens in such a way that as $|\hat{\beta}_i| \to \infty$, the t statistic tends to zero. Thus highly significant coefficients according to the likelihood ratio test may have non-significant t ratios.

This encourages an upward biased p-value and loss of power leads to incorrect variable selection. We intentionally generate synthetic data with HDE, so the capability of accurate estimation by G-FORSE can be evaluated under such adverse effects.

We observe that on the simulated dataset with the Hauck-Donner effect (HDE), the supremacy of G-force over Logistic Regression is particularly pronounced. Figure 7.12a indicates the instability in the estimation of the simulated data in green bars due to HDE, while G-FORSE achieved success in capturing the true effects without signs of performance degradation. This is in keeping with the strong results of Figure 7.13 where the outputs of (a) and (b) from G-FORSE are showing no difficulties in approximating the underlying model, while Logistic Regression tends to be less effective in capturing a model with HDE.

7.9 Results

The prediction of G-FORSE model is obtained by Gaussian process with zero mean and covariance matrix by the affine transformation of correlation matrix Ψ using Cholesky decomposition technique. The performance is measured via the root mean squared error ($RMSE$) and the correlation (r) between predicted and produced observations from the underlying model (Table 7.8 and 7.9). In the G-FORSE prediction process, the errors $\epsilon(x^{(i)})$ are the realization of Gaussian process, and we predict $\hat{y}$ at unknown location x^* by including $\hat{y}$ into the known observations $\tilde{y} = \{y, \hat{y}\}^T$. We treat this as the model parameter which is estimated by the use of MLE. G-FORSE process is parsimonious and appropriate for high-dimensional data while keeping the number of estimated parameters very low. While we make no claim that the G-FORSE model can outperform other existing interpretable methods in explaining and performance, we believe that G-FORSE outperforms the GLMs such as Logistic Regression in interpretation, overcoming the issue of Hauck-Donner effect, and it can be competitive with the methods in the literature and highlight the potential of the G-FORSE model.

7.10 Group Explanation

Of interest to practitioners is to find if global interpretation is appropriate for demystifying the target black-box. Conventionally, MLI techniques deliver variable selection by computing feature importance based on some metric function, so they determine what group of variables are important for prediction. What is less studied in the field of MLI is knowing what *group of samples* is contributing together to the prediction made by the black-box, and G-FORSE undertakes this idea by constructing a *correlation* function during the training phase (Figure 7.11). The correlation function ($\mathbf{\Psi}$) used in G-FORSE provides the benefit to explore correlations between data points given the predictions, which is reflected in matrix $\mathbf{\Psi}$. These correlations are relying on the absolute distance between data points and the parameters p_i and θ_j in the G-FORSE method. Figure 7.11 shows that if instances i and j are both above or below their respective predictions' means, then the plot shows a higher correlation denoted by a darker color. If one is above its prediction's mean and the other is below, we see lighter (yellowish) spots denoting little to no correlation. In other words, the darker areas in the plot indicate an *agreement* on prediction, and the lighter areas mean *disagreement*. In the classification problems such as (e) or (f) in Figure 7.11, when the outcome is binary (0 or 1), the darker areas represent instances that are in accordance with the prediction made by the black-box (e.g. AdaBoost or Neural Network in (e) and (f)). This aspect of interpretation made by G-FORSE provides us with the ability to penetrate inside the black-box and monitor the prediction effect across samples. Another benefit of group explanation is that the global fidelity of a complex model can be visually appreciated by looking at the heatmap plot (Figure 7.11) to find if it can expand its prediction to darker areas. In this case, the user might choose between a global or local explanation depending on how the prediction was made by the underlying model.

7.10.1 G-FORSE Validation

We implement two synthetic experiments with the goal of evaluating the accuracy of model interpretation provided by G-FORSE algorithm. In both experiments, we apply G-FORSE (Section III) on a ground-truth multi-variate function $f(x)$ to fit a metamodel $g(x) \approx f(x)$, and compare the resulting estimated coefficients for $g(x)$ against the true generator function $f(x)$.

In Figure 7.12(a), we compare G-FORSE (blue) and Logistic Regression (yellow) in terms of the coefficients they estimate and their goodness-of-fit with different sample sizes with respect to the true function (Figure 7.13). We consider a true GLM function with five linear terms $X_1, \ldots, X_5$ and one interaction term X_2X_3. As we can see, G-FORSE is generally more accurate than Logistic Regression. Moreover, G-FORSE tends to be more robust than Logistic Regression in the sense that the Hauck-Donner effect would not degrade G-FORSE estimation, but it does so on Logistic Regression. This is vividly visible in Figure 7.12(a) where the yellow bars representing Logistic Regression failed to capture true coefficients (green bars) correctly.

7.11 Advantages and Disadvantages

This new machine learning interpretation technique comes with advantages and disadvantages relative to previous interpretation methods. The disadvantages are primarily that G-FORSE may not be scalable on ultra-high-dimensional data, and data smoothness is necessary to obtain a reliable estimation. Similar to other techniques, multicollinearity can affect explanation negatively. The advantages are that black-box models can be interpreted with fewer parameters, missing data can be interpolated, a wide variety of covariance functions can be plugged into the process, it's applicable on noisy observations, and it can provide uncertainty estimates such as expected improvement. One can also benefit from using G-FORSE on cases with no maximum likelihood estimations as was discussed earlier for Logistic Regression. Table 7.10 summarizes the comparison of G-FORSE method with other interpretable modeling approaches that are actively utilized in machine learning interpretation.

The G-FORSE models may also gain some statistical advantage from the estimate of uncertainty for not being based on pre-assumed models, but rather depending on empirical observations, which makes it superior to linear models or similar interpolation techniques. Another advantage is the tendency of

G-FORSE to be less varied toward specific bias direction, and the estimations are best linear unbiased estimator (BLUE) if the observations are spatially independent. Additionally, G-FORSE can surrogate to interpretable GLMs such as Logistic Regression suffering from HDE.

7.11.1 How G-FORSE Interprets Better?

The actual problem within black-box interpretation is to determine the relationship between samples themselves. A few extra steps are required to reveal the underlying relationships between samples whereas none of the existing interpretable methods provides this relationship. If we are better aware of the connection between the instances and their networks with the outcome, we would have a better interpretation about a black-box that could predict a complex underlying system, as such a model interpreter would align with the actual explanation within the black-box model.

Furthermore, it would have benefited us to appreciate the importance of a metamodel being responsive to nonlinear relationships. LIME (as a popular interpreter) assumes that any nonlinear model can be explained linearly in a local space. Obviously, this assumption is inappropriate if the network of variables is ignored in the explanation task (e.g., a network of variables may have different connections across data points).

Less obvious, the existing interpretable methods sacrifice accuracy for less complexity to ensure that the black-box model would be explained clearly by the interpreter. Because of the incorporation of metamodeling techniques in interpretation problems, we gain more access to powerful and accepted techniques. This results in models with higher accuracy of prediction that can be the same as (or at least comparable to) the underlying complex model while preserving the interpretability. Lastly, the Hauck-Donner Effect (discussed in Section V) as a hidden phenomenon can afflict many types of regression interpretable models, which leads to unreliable explanations. Knowing all of this compelled us to introduce metamodeling as a new gateway to black-box interpretation.

7.11.2 Choice of $\boldsymbol{p}$ and $\boldsymbol{\theta}$

We aim to minimize the generalization error of the model by maximizing the likelihood of $\boldsymbol{y}$ to choose $\boldsymbol{p}$ and $\boldsymbol{\theta}$. The concentrated ln-likelihood function for parameters estimation is:

$$ln(L) = -\frac{n}{2}ln(2\pi) - \frac{n}{2}ln(\hat{\sigma}^2) - \frac{1}{2}ln|\boldsymbol{\Psi}| - \frac{(\boldsymbol{y}-\mathbf{1}\mu)^T\boldsymbol{\Psi}^{-1}(\boldsymbol{y}-\mathbf{1}\mu)}{2\sigma^2} \tag{7.20}$$

where $\boldsymbol{\Psi}$ is an $n \times n$ correlation matrix of all the observed data:

$$\boldsymbol{\Psi} = \begin{bmatrix} cor[Y(\boldsymbol{x}^{(1)}), Y(\boldsymbol{x}^{(1)})] & \dots & cor[Y(\boldsymbol{x}^{(1)}), Y(\boldsymbol{x}^{(n)})] \\ \vdots & \ddots & \vdots \\ cor[Y(\boldsymbol{x}^{(n)}), Y(\boldsymbol{x}^{(1)})] & \dots & cor[Y(\boldsymbol{x}^{(n)}), Y(\boldsymbol{x}^{(n)})] \end{bmatrix} \tag{7.21}$$

We obtain *Maximum Likelihood Estimates* (MLEs) for μ and σ^2 by taking the derivatives of equation 7.18 and setting to zero:

$$\begin{aligned} \hat{\mu} &= \frac{\mathbf{1^T\Psi^{-1}}\boldsymbol{y}}{\mathbf{1^T\Psi^{-1}1}} \\ \hat{\sigma}^2 &= \frac{(\boldsymbol{y}-\mathbf{1}\mu)^T\boldsymbol{\Psi}^{-1}(\boldsymbol{y}-\mathbf{1}\mu)}{n} \end{aligned} \tag{7.22}$$

The probability of the dataset $\{(x^{(1)}, y^{(1)} \pm \epsilon), (x^{(2)}, y^{(2)} \pm \epsilon), \dots, (x^{(n)}, y^{(n)} \pm \epsilon)\}$ given a set of parameters W and the model estimation $\hat{f}(x, W)$ with $i.i.d.$ errors ϵ is computed by:

$$P = \frac{1}{(2\pi\sigma^2)^{n/2}} \prod_{i=1}^{n} \left\{ exp\left[-\frac{1}{2}\left(\frac{y^{(i)} - \hat{f}(x, W)}{\sigma}\right)^2 \right] \epsilon \right\} \tag{7.23}$$

Here we assume that there is no error in Y (due to the deterministic property) and modeling error is not permissible, so our likelihood function is:

$$L(\boldsymbol{Y}^{(1)}, \boldsymbol{Y}^{(2)}, \dots, \boldsymbol{Y}^{(n)}|\mu, \sigma) = \frac{1}{(2\pi\sigma^2)^{n/2}} exp\left[-\frac{\sum(\boldsymbol{Y}^{(i)} - \mu)^2}{2\sigma^2} \right] \tag{7.24}$$

Which can be replaced by the sample data, namely as:

$$\begin{aligned} L =& \frac{1}{(2\pi\sigma^2)^{n/2}|\boldsymbol{\Psi}|^{1/2}} exp\left[-\frac{(\boldsymbol{y}-\mathbf{1}\mu)^T\boldsymbol{\Psi}^{-1}(\boldsymbol{y}-\mathbf{1}\mu)}{2\sigma^2}\right] \Rightarrow \\ ln(L) =& -\frac{n}{2}ln(2\pi) - \frac{n}{2}ln(\sigma^2) - \frac{1}{2}ln|\boldsymbol{\Psi}| - \frac{(\boldsymbol{y}-\mathbf{1}\mu)^T\boldsymbol{\Psi}^{-1}(\boldsymbol{y}-\mathbf{1}\mu)}{2\sigma^2} \end{aligned}$$

We obtain *Maximum Likelihood Estimates* (MLEs) for μ and σ^2 by taking the derivatives of equation 7.20 and setting to zero:

$$\begin{aligned} \hat{\mu} =& \frac{\mathbf{1^T}\boldsymbol{\Psi^{-1}y}}{\mathbf{1^T}\boldsymbol{\Psi}^{-1}\mathbf{1}} \\ \hat{\sigma}^2 =& \frac{(\boldsymbol{y}-\mathbf{1}\mu)^T\boldsymbol{\Psi}^{-1}(\boldsymbol{y}-\mathbf{1}\mu)}{n} \end{aligned} \tag{7.25}$$

Thus, by substituting MLEs back into equation 7.20 and removing the constant terms, concluding the proof.

7.11.3 Theorem

Given correlation parameters, a new prediction of $\hat{\boldsymbol{y}}$ at $\boldsymbol{x}$ is consistent with the observations and estimated correlation parameters if the likelihood of the sample data and the prediction are maximized.

We define $\boldsymbol{\Psi} = (cor[Y(\boldsymbol{x}^{(1)}), Y(\boldsymbol{x})], \ldots, cor[Y(\boldsymbol{x}^{(n)}), Y(\boldsymbol{x})])$ a vector of correlations between the observed data and new prediction to obtain $(\psi^{(1)}, \ldots, \psi^{(n)})$. We define an augmented correlation matrix, namely as:

$$\widetilde{\boldsymbol{\Psi}} = \begin{bmatrix} \boldsymbol{\Psi} & \boldsymbol{\psi} \\ \boldsymbol{\Psi}^T & 1 \end{bmatrix} \tag{7.26}$$

The element one in the matrix represents as the self-correlation where $|x^{(i)} - x^{(i)}| = 0$ or $cor[Y(\boldsymbol{x}^{(i)}), Y(\boldsymbol{x}^{(i)})] = 1$. The natural log-likelihood of the augmented data is:

$$ln(L) = -\frac{n}{2}ln(2\pi) - \frac{n}{2}ln(\hat{\sigma}^2) - \frac{1}{2}ln|\widetilde{\boldsymbol{\Psi}}| - \frac{(\tilde{\boldsymbol{y}}-\mathbf{1}\hat{\mu})^T\widetilde{\boldsymbol{\Psi}}^{-1}(\tilde{\boldsymbol{y}}-\mathbf{1}\hat{\mu})}{2\hat{\sigma}^2} \tag{7.27}$$

Having removed the constant terms from the likelihood function and substituted in the equation for $\tilde{y}$ and $\widetilde{\Psi}$, the ln-likelihood is:

$$ln(L) \approx -\frac{1}{2\hat{\sigma}^2}\begin{bmatrix}\boldsymbol{y}-\mathbf{1}\hat{\mu}\\ \hat{y}-\hat{\mu}\end{bmatrix}^T\begin{bmatrix}\boldsymbol{\Psi} & \boldsymbol{\psi}\\ \boldsymbol{\Psi}^T & 1\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{y}-\mathbf{1}\hat{\mu}\\ \hat{y}-\hat{\mu}\end{bmatrix} \tag{7.28}$$

The inverse of $\widetilde{\Psi}$ obtained by Theil method (See Appendix for details) is substituted into equation 7.27 to construct a simplified ln-likelihood function, and we remove terms without $\hat{y}$ to achieve:

$$ln(L) \approx \left(\frac{-1}{2\hat{\sigma}^2(1-\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})}\right)(\hat{y}-\hat{\mu})^2 + \left(\frac{\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\boldsymbol{y}-\mathbf{1}\hat{\mu})}{\hat{\sigma}^2(1-\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})}\right)(\hat{y}-\hat{\mu}) \tag{7.29}$$

Having taken derivatives with respect to $\hat{y}$ and set to zero, one can obtain MLE for $\hat{y}$ as:

$$\hat{y}(\boldsymbol{x}) = \hat{\mu} + \boldsymbol{\psi}^T\boldsymbol{\Psi}^{-1}(\boldsymbol{y}-\mathbf{1}\hat{\mu}) \tag{7.30}$$

The $\boldsymbol{\psi}$ is the *i*th column of Ψ at the new estimate of $\boldsymbol{x}^{(i)}$, implying that, $\boldsymbol{\psi}\boldsymbol{\Psi}^{-1}$ is the *i*th unit vector of estimate. Thus $\hat{y}(\boldsymbol{x}) = \hat{\mu} + y^{(i)} - \hat{\mu} = y^{(i)}$

7.11.4 Partitioned Inverse

For a non-singular $n \times n$ matrix A, there is a unique $n \times n$ inverse matrix A^{-1} which satisfies $AA^{-1} = A^{-1}A = I$, namely as:

$$A = \begin{bmatrix}P_1 & R_1\\ R_1^T & Q_1\end{bmatrix} \tag{7.31}$$

where P and Q are non-singular submatrices, so one wishes to obtain:

$$A^{-1}A = I = \begin{bmatrix}P_2 & R_2\\ R_2^T & Q_2\end{bmatrix}\begin{bmatrix}P_1 & R_1\\ R_1^T & Q_1\end{bmatrix} = \begin{bmatrix}I & 0\\ 0 & I\end{bmatrix} \tag{7.32}$$

We divide $A^{-1}A$ into the four equations as follow:

$$\begin{aligned} P_2P_1 + R_2R_1^T &= I, \\ P_2R_1 + R_2Q_1 &= 0, \\ R_2^TP_1 + Q_2R_1^T &= 0, \\ R_2^TR_1 + Q_2Q_1 &= I. \end{aligned} \tag{7.33}$$

We obtain equation by substituting equation 7.32 into 7.33 to achieve:

$$Q_2 = (Q_1 - R_1^TP_1^{-1}R_1)^{-1} \tag{7.34}$$

and

$$\begin{aligned} R_2^T &= -(Q_1 - R_1^Tp_1^{-1}R_1)^{-1}R_1^TP_1^{-1}, \\ R_2 &= -P_1^{-1}R_1(Q_1 - R_1^TP_1^{-1}R_1)^{-1}. \end{aligned} \tag{7.35}$$

By substituting equation 7.34 into 7.30, we obtain:

$$P_2 = P_1^{-1} + P_1^{-1}R_1(Q_1 - R_1^TP_1^{-1}R_1)^{-1}R_1^TP_1^{-1}. \tag{7.36}$$

Putting equations 7.34, 7.35, 7.36 and 7.37 together, A^{-1} can be constructed as follows:

$$A^{-1} = \begin{bmatrix} P_1 + P_1^{-1}R_1(Q_1 - R_1^TP_1^{-1}R_1)^{-1}R_1^TP_1^{-1} & -p_1^{-1}R_1(Q_1 - R_1^TP_1^{-1}R_1)^{-1} \\ -(Q_1 - R_1^TP_1^{-1}R_1)^{-1}R_1^TP_1^{-1} & (Q_1 - R_1^TP_1^{-1}R_1)^{-1} \end{bmatrix} \tag{7.37}$$

7.12 Conclusion

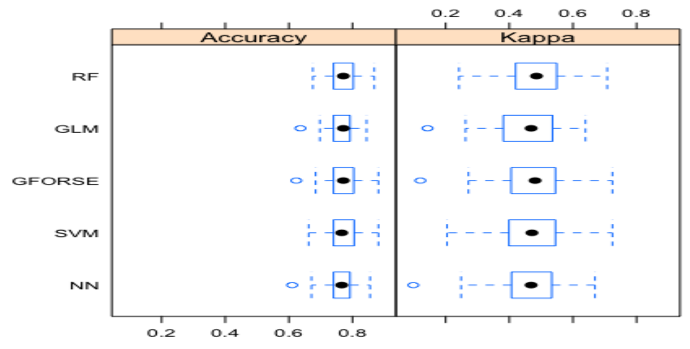
A complicated machine learning model is an abstraction of discovered patterns in massive data; a metamodel is a simplified version, reflecting the properties of that complex model.

The main goal of this work was to propose a new technique based on the metamodeling concept for turning a black-box into a clear-box. Metamodeling is widely used in simulation and engineering fields for covering one aspect of a modeling problem, and this study showed that it can be applied for machine learning interpretation as well.

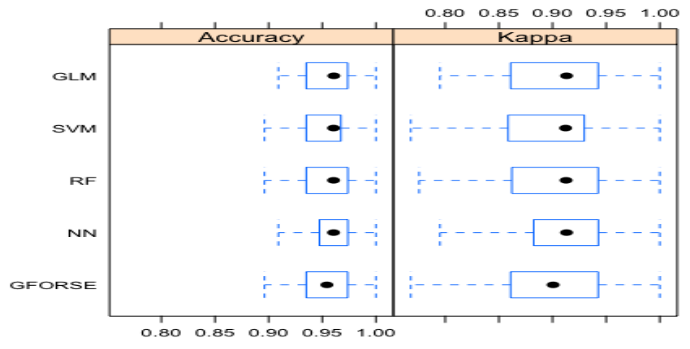
We argued that interpretation in the sample level is vital for an effective explanation, and it provides the user with more insights about the underlying black-box model. We propose G-FORSE, a metamodel inheriting the characteristics of the Kriging process to support global interpretation using activeness parameter, and a network between data points using correlation function, where both aspects can increase the level of transparency for interpretation. We also show that G-FORSE could outperform some types of regression models such as Logistic Regression, where they suffer from the Hauck-Donner effect by running experiments on the synthetic data. Our experiments confirmed that G-FORSE can determine a global view of *any* complex model for a wide range of models, and we prove by several examples that G-FORSE can go beyond the conventional interpretation where global or local aspect was the only concern. G-FORSE provides a global explanation, mimics a complex model, and outperforms certain GLM models. It also acknowledges the idea of using close neighbors for approximation to eliminate the effect of bias in a region of data, whereas in the LIME technique, it explains by simple perturbations which could be a possible source of bias.

Many straightforward extensions are suggested by this study: in the explanations for image classifiers, G-FORSE framework can be extended to *highlight the super-pixels* with the relatedness measurement afforded by variograms (calibrated by lag, sill, range and nugget) towards a specific class, to pictorialize why a certain prediction would happen. The covariance function used in G-FORSE can be replaced by other types of functions, and G-FORSE can be scalable for ultra-high-dimensional data using penalized likelihood function mixed with feature selection techniques (Joseph et al., 2008).

This article has presented the viability of metamodeling in interpreting black-box models, suggesting that these research directions could prove useful.

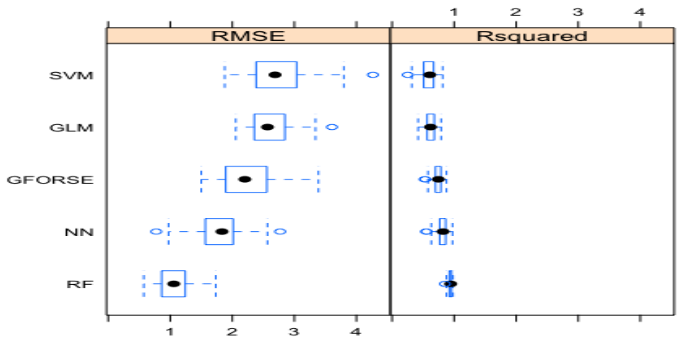


(a) Classification on NN complex model

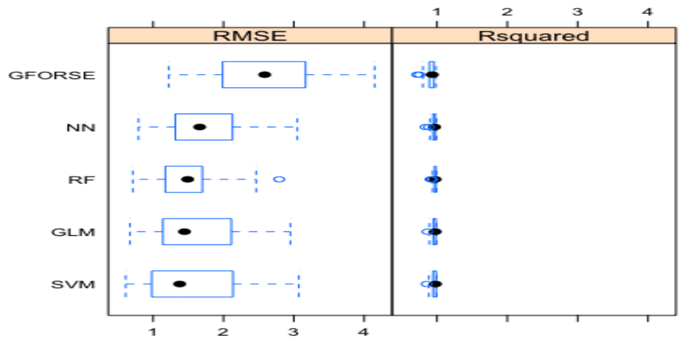


(b) Classification on GB complex model

Figure 7.1: Accuracy and Kappa Box Plot for each meta-classifier



(a) Regression on NN complex model



(b) Regression on GB complex model

Figure 7.2: RMSE and R^2 Box Plot for each meta-classifier

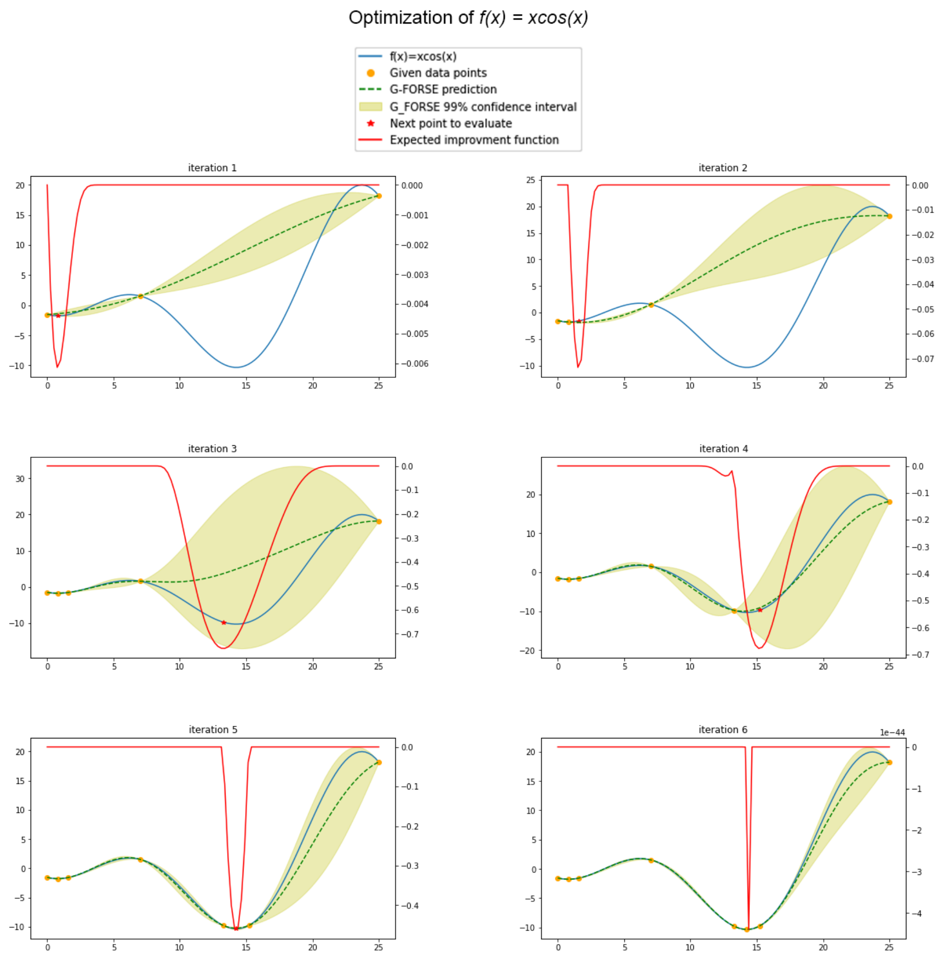


Figure 7.3: G-FORSE model optimization for 1-D function $f(x) = xcos(x)$.

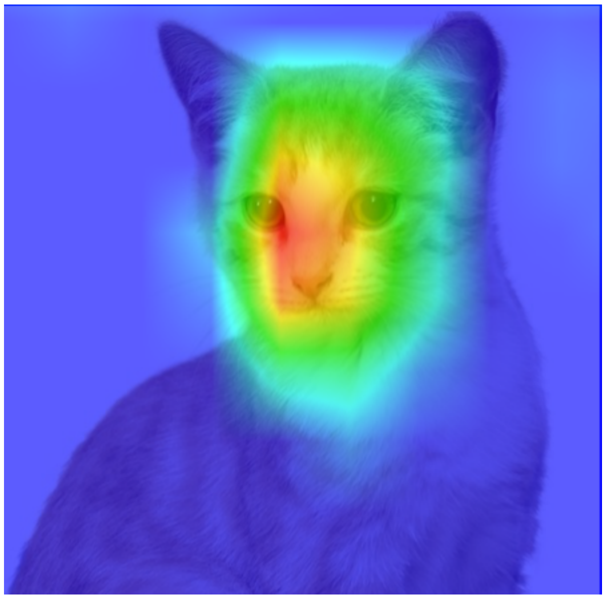


Figure 7.4: Class activation maps (CAM).

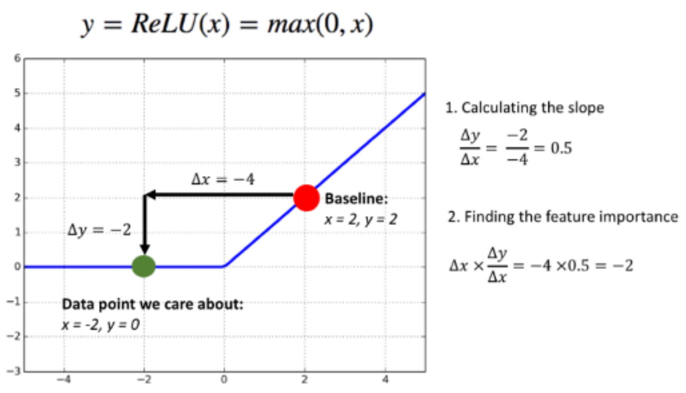

$y = ReLU(x) = max(0, x)$
$\Delta x = -4$
$\Delta y = -2$
Baseline: $x = 2, y = 2$
Data point we care about: $x = -2, y = 0$
1. Calculating the slope
$\frac{\Delta y}{\Delta x} = \frac{-2}{-4} = 0.5$
2. Finding the feature importance
$\Delta x \times \frac{\Delta y}{\Delta x} = -4 \times 0.5 = -2$

Figure 7.5: DeepLIFT: Calculating a multiplier in a simple ReLU activator. Image Credit: HTTPS://TOWARDSDATASCIENCE.COM/INTERPRETABLE-NEURAL-NETWORKS-45AC8AA91411

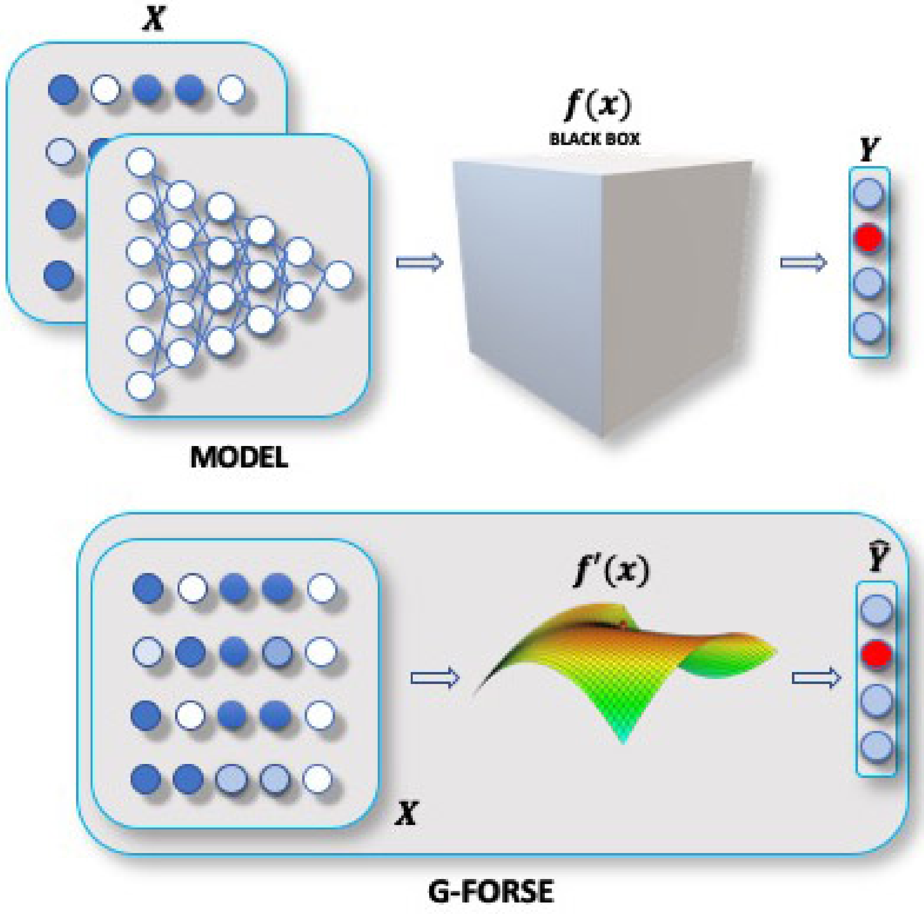


Figure 7.6: G-FORSE process.

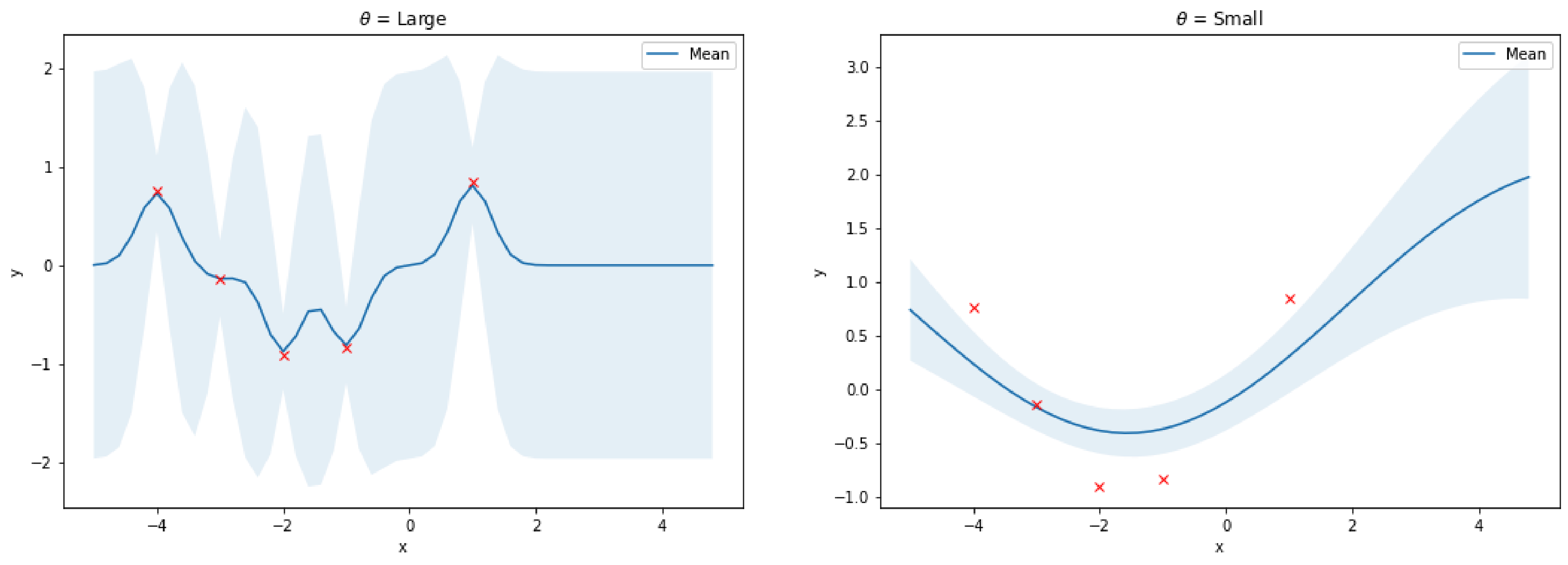
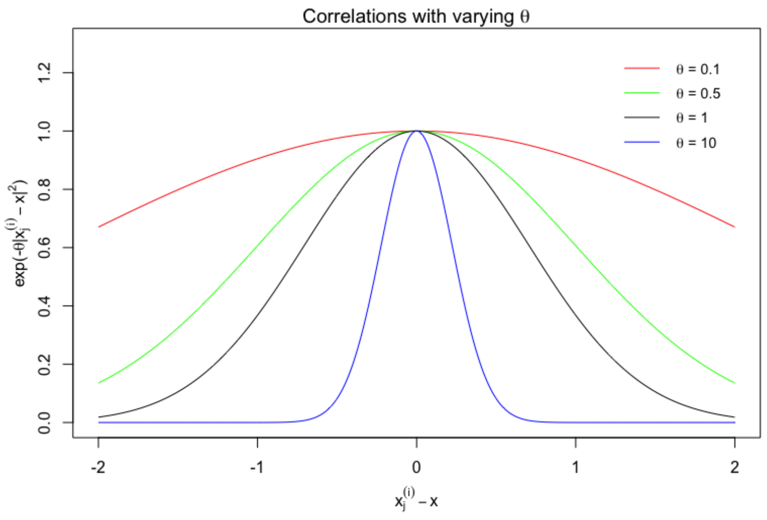
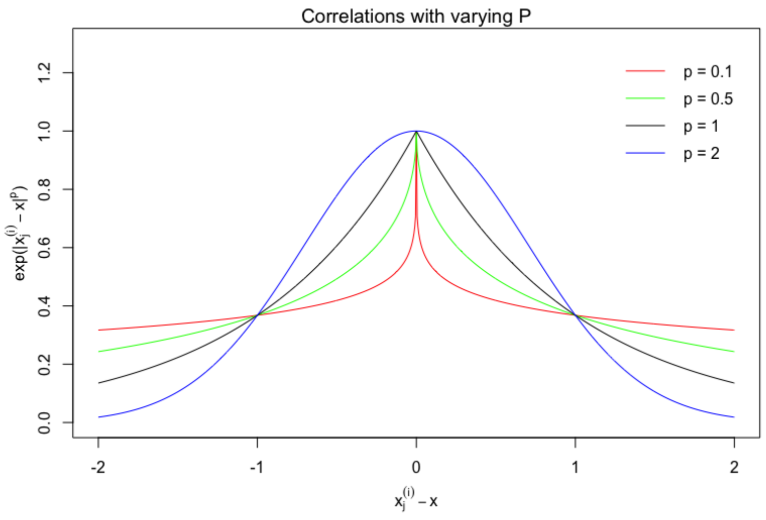


Figure 7.7: θ parameter determines how relevant a variable is for learning a function. Larger θ means more relevant.



(a) Correlations with varying θ.



(b) Correlations with varying p.

		Algorithms - RMSE(r)			
		Neural Network	GBM	AdaBoost	XGBoost
Datasets	Crimes	0.100(0.88)	0.101(0.88)	**0.042(0.95)**	0.113(0.85)
	Housing	**0.022(0.99)**	**1.44(0.99)**	**0.654(0.99)**	**1.48(0.98)**
	Cancer	**0.000(0.99)**	**0.126(0.96)**	**0.015(0.98)**	**0.072(0.98)**
	Diabetes	**0.057(0.96)**	0.361(0.60)	0.001(0.72)	0.320(0.66)
	Income	**0.107(0.94)**	0.154(0.87)	**0.000(0.90)**	0.157(0.86)

Table 7.8: The root mean squared error (RMSE) and correlation (r) obtained via G-FORSE by use of differential evolution optimizer for different complex models on different datasets.

		Algorithms - RMSE(r)			
		Neural Network	GBM	AdaBoost	XGBoost
Datasets	Crimes	0.101(0.88)	0.112(0.86)	**0.045(0.94)**	0.110(0.86)
	Housing	**0.259(0.99)**	**2.26(0.96)**	**2.13(0.96)**	**2.40(0.96)**
	Cancer	**0.145(0.93)**	**0.167(0.93)**	**0.022(0.97)**	**0.115(0.97)**
	Diabetes	**0.059(0.96)**	0.386(0.57)	0.001(0.74)	0.336(0.63)
	Income	**0.094(0.94)**	**0.131(0.91)**	**0.000(0.93)**	**0.144(0.90)**

Table 7.9: The root mean squared error (RMSE) and correlation (r) obtained via G-FORSE by use of L-BFGS-B optimizer for different complex models on different datasets.

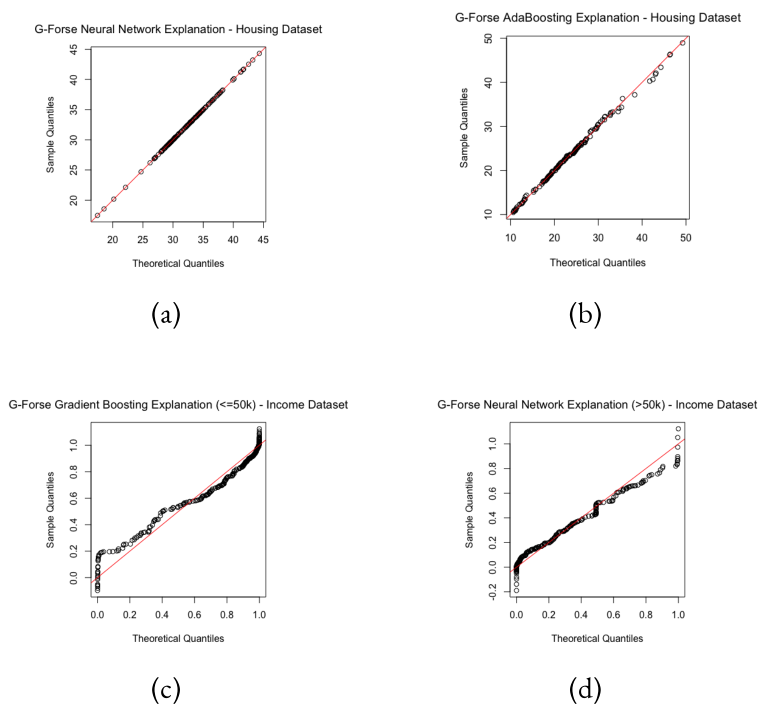


Figure 7.9: Scatter plots of regression outcomes computed by G-FORSE versus outcome values produced by different black-box models across various datasets.

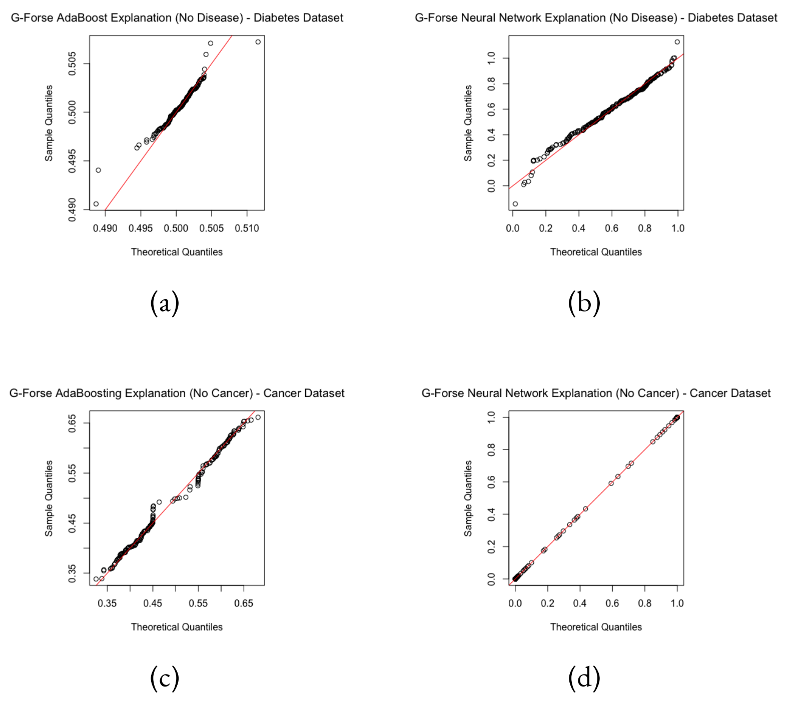
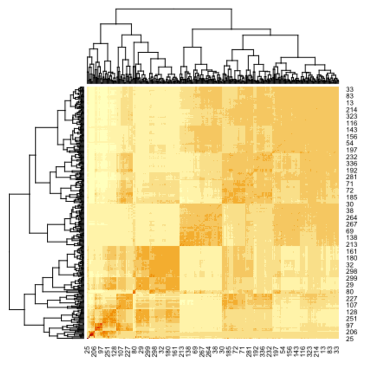
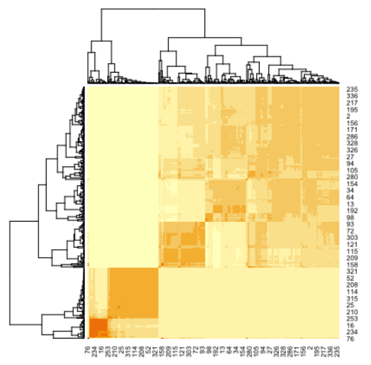
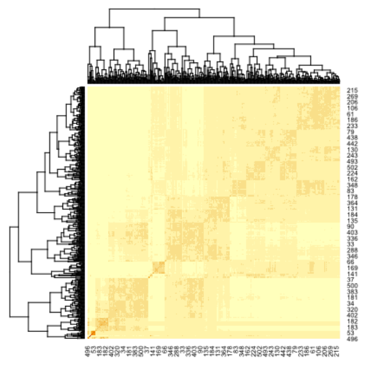
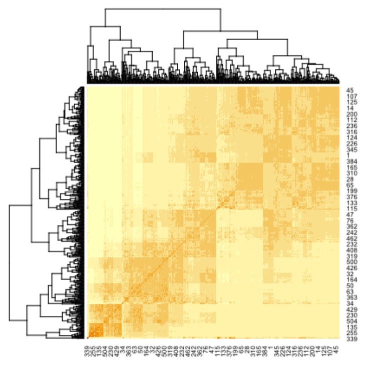


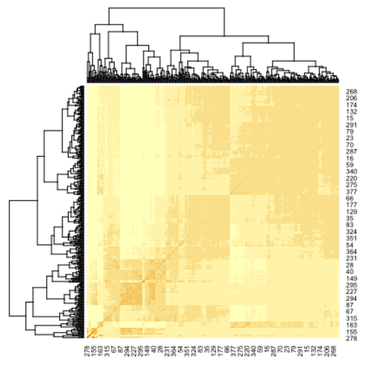
Figure 7.10: Scatter plots of classification outcomes computed by G-FORSE versus outcome values produced by different black-box models across various datasets.

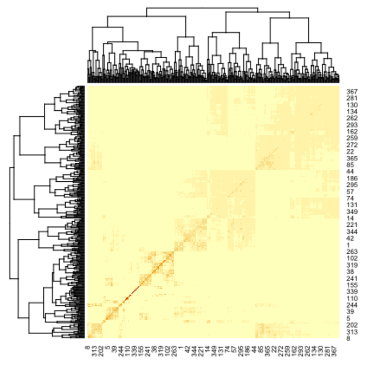

(a) Samples in AdaBoost (Housing data)

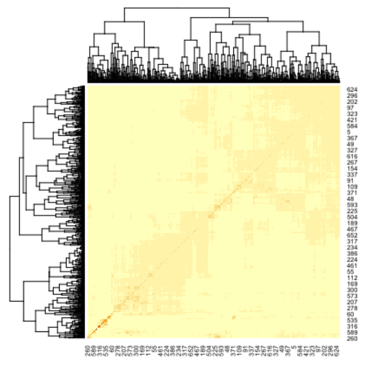

(b) Samples in Neural Network (Housing data)


(c) Samples in Neural Network (Diabetes data)


(d) Samples in AdaBoost (Diabetes data)


(e) Samples in AdaBoost (Cancer data)


(f) Samples in Neural Network (Cancer data)


(g) Samples in Gradient Boosting (Income data)

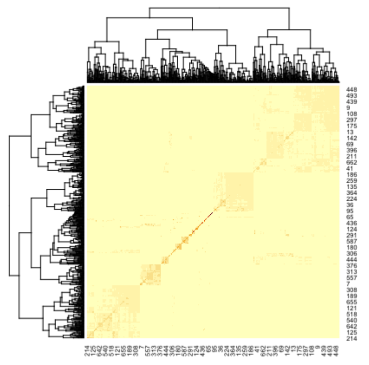
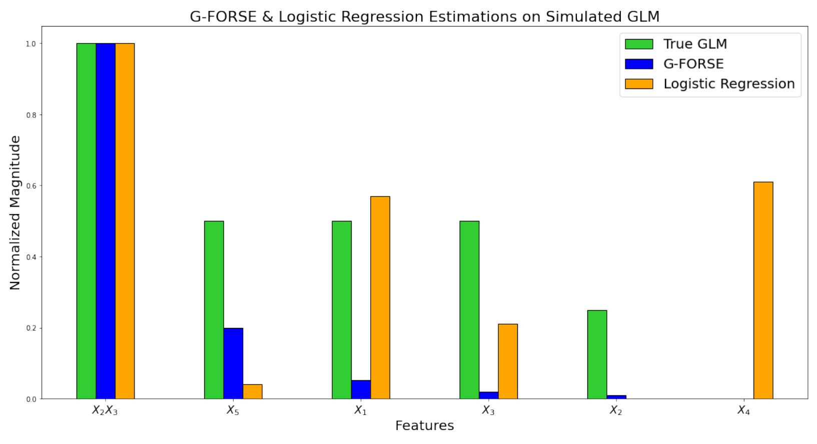
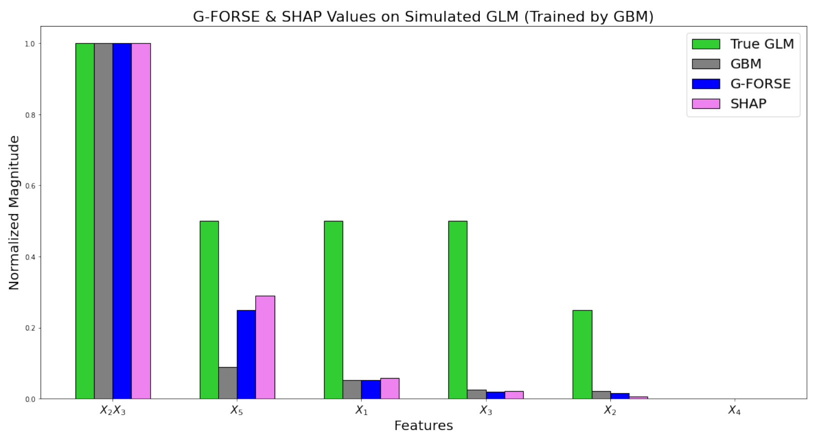

(h) Samples in Neural Network (Income data)

Figure 7.11: Clustering between data points revealed by G-FORSE for different black-box models. These clusters are computed by correlation matrix estimated during G-FORSE optimization. Each machine learning model treats samples differently in predicting the outcome.



(a)



(b)

Figure 7.12: (a) Side-by-side comparison of features' importance evaluated by G-FORSE and Logistic Regression on the synthesized GLM. G-FORSE is superior to Logistic Regression in detecting true signals. (b) G-FORSE and SHAP values on the synthesized GLM trained by GBM. Both plots show the tendency of G-FORSE to be trustworthy to approximate global effects.

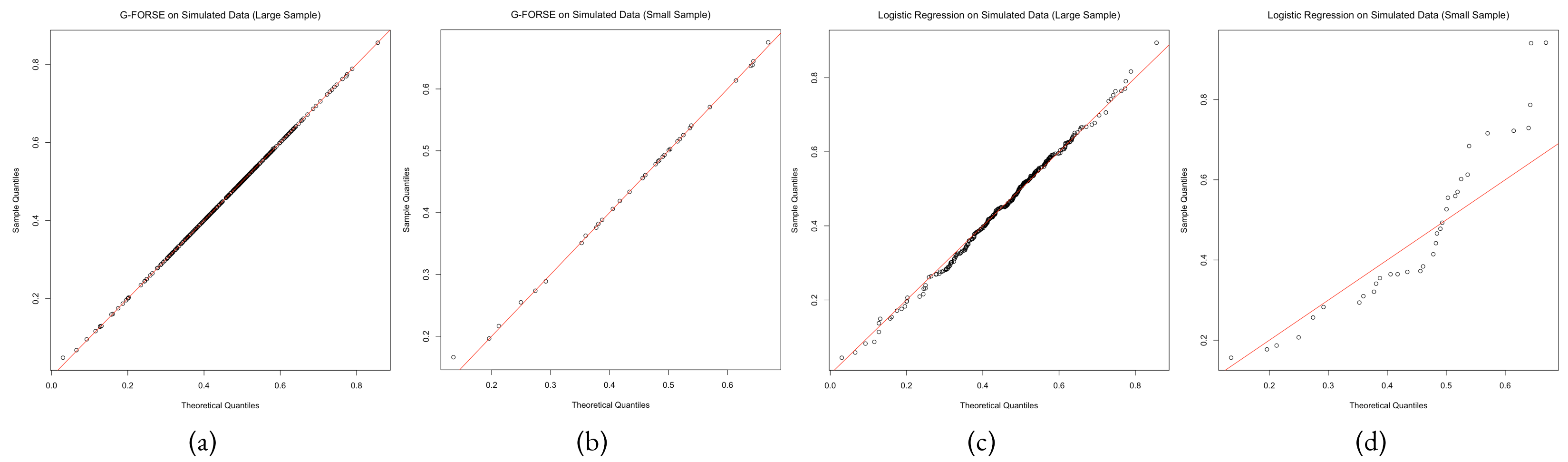


Figure 7.13: Scatter plots of predicted outcomes computed by G-FORSE and Logistic Regression on the simulated data versus the true values. G-FORSE appears to be superior to Logistic Regression in small samples.

	LIME	SHAP	GLM	Decision Trees	CXPlain	G-FORSE
Model passes	Forward	Forward-Backward	Forward	Forward	Forward	Interpolation
Requires baseline	No	Yes	No	No	Yes	No
Computation time	Fast	Slow	Fast	Fast	Fast	Moderate
Model-agnostic	Yes	Yes	Yes	Yes	Yes	Yes
Relationship bw Samples	No	No	No	No	No	Yes
Uncertainty estimate	No	No	No	No	Yes	Yes

Table 7.10: Challenges in interpretable models: a summary of the difficulties encountered by different methods to interpretable machine learning approaches for each of the major operations involving a method.

Chapter 8

Conclusion

Most machine learning models are able to encode input data to a high-dimensional space, to capture more patterns from data. If the training data is not sufficient, the high-dimensional encoding process will prevent the production of optimal answers, and the model degrees of freedom would be negative. On the other hand, recent advances in machine learning for training data in high-dimensional has brought with it many new challenges such as interpretability. This work presented useful solutions to the aforementioned challenges as a result of working with high-dimensional space, and those solutions were applied to high-dimensional applications such as time-series data and video cases. The contributions we have made in each chapter are summarized in the following.

In chapter 2, an extensive literature review was provided, focusing on both traditional methods and machine learning algorithms in time-series forecasting and a chronological overview of the developments in this field.

In chapter 3, we proposed a new technique that generates forecast values in intermediate length time-series analysis, and we applied and evaluated it on COVID-19 data. The technique involves multiple strategies to maximize forecasting accuracy. We first forecasted the series by the SARIMA model and then applied backcasting on the generated forecast values by a neural network model coupled with the wavelet transformation function. This process enables the weak signal to be amplified during the backcasting, leading to improve model accuracy compared to other competitors.

In chapter 3, we argued that a complicated machine learning model is an abstraction of discovered patterns in high-dimensional spaces, leading to obscuring our understanding of relationships between input and output. We then proposed G-FORSE which is based on metamodeling, and it enables us to study the output and input relationships of the underlying black-box model.

In chapter 4, we addressed undersampled problems in video cases that occur as a result of poor quality in frames and deprivation in samples or duration. Since video data is super rich in terms of dimensionality in sample and time spaces, we mentioned that simple deep neural networks such as a convolutional neural network may not be powerful enough. We solved this issue by presenting a transfer learning technique that can benefit from a pre-weighted model without a necessity to train the model from scratch. Not only would this technique save time and computational resources, but also it resolves the issue of parameters deficiency because of data scarcity.

Finally, sophisticated machine learning models tend to be effective in high-dimensional spaces provided that the number of learning parameters would suffice and the relationships between input and output are transparent to humans.

Bibliography

Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, *6*, 52138–52160.

Alaa, A. M., & van der Schaar, M. (2019). Demystifying black-box models with symbolic metamodels. *Advances in Neural Information Processing Systems*.

Aldous, D., & Fill, J. (2014). Reversible markov chains and random walks on graphs, 2002. *Unfinished monograph, recompiled*, *2002*.

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Arunraj, N. S., Ahrens, D., & Fernandes, M. (2016). Application of sarimax model to forecast daily sales in food retail industry. *International Journal of Operations Research and Information Systems (IJORIS)*, *7*(2), 1–21.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Balkin, S. D., & Ord, J. K. (2000). Automatic neural network modeling for univariate time series. *International Journal of Forecasting*.

Barmparis, G., & Tsironis, G. (2020). Estimating the infection horizon of covid-19 in eight countries with a data-driven approach. *Chaos, Solitons & Fractals*, *135*, 109842. https://doi.org/10.1016/j.chaos.2020.109842

Bertasius, G., Shi, J., & Torresani, L. (2015). Deepedge: A multi-scale bifurcated deep network for top-down contour detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4380–4389.

Bertasius, G., Torresani, L., Yu, S. X., & Shi, J. (2017). Convolutional random walk networks for semantic image segmentation. *Proceedings of* *the IEEE Conference on Computer Vision and Pattern Recognition*, 858–866.

Biggio, B., & Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*.

Bindel. (2018). Numerics for data science. *https://www.cs.cornell.edu/ bindel/class/sjtu-summer18/lec/2018-06-27.pdf*.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional time series forecasting with convolutional neural networks. *arXiv preprint*.

Brock, A., Donahue, J., & Simonyan, K. (2018). Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., & Shah, R. (1993). Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, *6*, 737–744.

Bultheel, A. et al. (1995). Learning to swim in a sea of wavelets. *Bulletin of the Belgian Mathematical society-simon stevin*, *2*(1), 1–45.

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*.

Chan, A. B., & Vasconcelos, N. (2005a). Classification and retrieval of traffic video using auto-regressive stochastic processes. *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 771–776.

Chan, A. B., & Vasconcelos, N. (2005b). Probabilistic kernels for the classification of auto-regressive visual processes. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, *1*, 846–851.

Chen, J., Song, L., Wainwright, M. J., & Jordan, M. I. (2018). Learning to explain: An information-theoretic perspective on model interpretation. *arXiv preprint arXiv:1802.07814*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choudhary, P., Kramer, A., & datascience.com team. (2018). *datascienceinc/Skater: Enable Interpretability via Rule Extraction(BRL)* (Version v1.1.0-b1). Zenodo. https://doi.org/10.5281/zenodo.1198885

Commandeur, J. J., & Koopman, S. J. (2007). *An introduction to state space time series analysis*. Oxford University Press.

Data61, C. (2018). Stellargraph machine learning library.

Dhamdhere, K., Sundararajan, M., & Yan, Q. (2018a). How important is a neuron? *CoRR*, *abs/1805.12233*.

Dhamdhere, K., Sundararajan, M., & Yan, Q. (2018b). How important is a neuron? *arXiv preprint arXiv:1805.12233*.

Dua, D., & Graff, C. (2017). UCI machine learning repository. http://archive.ics.uci.edu/ml

Esteban, C., Hyland, S. L., & Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*.

Fan, C., Zhang, Y., Pan, Y., Li, X., Zhang, C., Yuan, R., Wu, D., Wang, W., Pei, J., & Huang, H. (2019). Multi-horizon time series forecasting with temporal attention learning. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

Fan, J., & Lv, J. (2008). Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *70*(5), 849–911.

Fazeli, S., Moatamed, B., & Sarrafzadeh, M. (2020). Statistical analytics and regional representation learning for covid-19 pandemic understanding.

Feng, F., He, X., Wang, X., Luo, C., Liu, Y., & Chua, T.-S. (2019). Temporal relational ranking for stock prediction. *ACM Transactions on Information Systems (TOIS)*, *37*(2), 1–30.

Fernando, Z. T., Singh, J., & Anand, A. (2019). A study on the interpretability of neural retrieval models using deepshap. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*.

Forrester, A., Sobester, A., & Keane, A. (2008). *Engineering design via surrogate modelling: A practical guide*. John Wiley & Sons.

Friedman, J. H., Popescu, B. E. et al. (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*.

Fu, R., Chen, J., Zeng, S., Zhuang, Y., & Sudjianto, A. (2019). Time series simulation by conditional generative adversarial net. *arXiv preprint arXiv:1904.11419*.

Gangopadhyay, T., Tan, S. Y., Huang, G., & Sarkar, S. (2018). Temporal attention and stacked lstms for multivariate time series prediction.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., & Brendel, W. (2018). Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*.

GeoDS. (2021). Geods/covid19usflows. https://github.com/GeoDS/COVID19USFlows

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. *International Conference on Machine Learning*, 1263–1272.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 2672–2680.

Grossmann, A., & Morlet, J. (1984). Decomposition of hardy functions into square integrable wavelets of constant shape. *SIAM journal on mathematical analysis*, *15*(4), 723–736.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in neural information processing systems*.

Harrison Jr, D., & Rubinfeld, D. L. (1978). Hedonic housing prices and the demand for clean air.

Hauck Jr, W. W., & Donner, A. (1977). Wald's test as applied to hypotheses in logit analysis. *Journal of the american statistical association*, *72*(360a), 851–853.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*.

https://www.wsdot.wa.gov. (n.d.).

Huizan, W., Ren, Z., Kefeng, L., Wei, L., Guihua, W., & Ning, L. (2008). Improved kriging interpolation based on support vector machine and its application in oceanic missing data recovery. *2008 International Conference on Computer Science and Software Engineering*, *4*, 726–729.

Ilie, O.-D., Cojocariu, R.-O., Ciobica, A., Timofte, S.-I., Mavroudis, I., & Doroftei, B. (2020). Forecasting the spreading of covid-19 across nine countries from europe, asia, and the american continents using the arima models. *Microorganisms*.

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1125–1134.

Jang, E., Gu, S., & Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Javeri, I. Y., Toutiaee, M., Arpinar, I. B., Miller, T. W., & Miller, J. A. (2021). Improving neural networks for time series forecasting using data augmentation and automl. *arXiv preprint arXiv:2103.01992*.

Jiang, X., Ji, P., & Li, S. (2019). Censnet: Convolution with edge-node switching in graph neural networks. *IJCAI*, 2656–2662.

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, *13*(4), 455–492.

Joseph, V. R., Hung, Y., & Sudjianto, A. (2008). Blind kriging: A new method for developing metamodels. *Journal of mechanical design*.

Kang, Y., Gao, S., Liang, Y., Li, M., Rao, J., & Kruse, J. (2020). Multiscale dynamic human mobility flow dataset in the us during the covid-19 epidemic. *Scientific data*, *7*(1), 1–13.

Kapoor, A., Ben, X., Liu, L., Perozzi, B., Barnes, M., Blais, M., & O'Banion, S. (2020). Examining covid-19 forecasting using spatio-temporal graph neural networks. *arXiv preprint arXiv:2007.03113*.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Kaymaz, I. (2005). Application of kriging method to structural reliability problems. *Structural Safety*.

Kianifar, M. R., & Campean, F. (2020). Performance evaluation of metamodelling methods for engineering problems: Towards a practitioner guide. *Structural and Multidisciplinary Optimization*, *61*(1), 159–186.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kodali, N., Abernethy, J., Hays, J., & Kira, Z. (2017). On convergence and stability of gans. *arXiv preprint arXiv:1705.07215*.

Koehler, J., & Owen, A. (1996). Computer experiments. handbook of statistics. *Elsevier Science*.

Kolmogorov, A. N. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk*, *114*(5), 953–956.

Kumar, N., & Susan, S. (2020). Covid-19 pandemic prediction using time series forecasting models. *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1–7.

Kursuncu, U. (2018). *Modeling the persona in persuasive discourse on social media using context-aware and knowledge-driven learning* (Doctoral dissertation). University of Georgia.

Kursuncu, U., Gaur, M., Castillo, C., Alambo, A., Thirunarayan, K., Shalin, V., Achilov, D., Arpinar, I. B., & Sheth, A. (2019). Modeling islamist extremist communications on social media using contextual dimensions: Religion, ideology, and hate. *Proceedings of the ACM on Human-Computer Interaction*, *3*(CSCW), 1–22.

Kursuncu, U., Gaur, M., & Sheth, A. (2019). Knowledge infused learning (k-il): Towards deep incorporation of knowledge in deep learning. *arXiv preprint arXiv:1912.00512*.

Leino, K., Sen, S., Datta, A., Fredrikson, M., & Li, L. (2018). Influence-directed explanations for deep convolutional networks. *2018 IEEE International Test Conference (ITC)*.

Li, D., Chen, D., Goh, J., & Ng, S.-k. (2018). Anomaly detection with generative adversarial networks for multivariate time series. *arXiv preprint arXiv:1809.04758*.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., & Yan, X. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *arXiv preprint*.

Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2019). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *arXiv preprint*.

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., & Bousquet, O. (2018). Are gans created equal? a large-scale study. *Advances in neural information processing systems*, 700–709.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*.

Makni, B., Abdelaziz, I., & Hendler, J. (2020). Explainable deep rdfs reasoner. *arXiv preprint arXiv:2002.03514*.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020a). The m5 accuracy competition: Results, findings and conclusions. *Int J Forecast*.

Makridakis, S., & Hibon, M. (2000). The m3-competition: Results, conclusions and implications.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020b). The m4 competition: 100,000 time series and 61 forecasting methods. *IJF*.

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint*.

Matsunaga, D., Suzumura, T., & Takahashi, T. (2019). Exploring graph neural networks for stock market predictions with rolling window analysis.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Montero-Manso, P., Athanasopoulos, G., Hyndman, R. J., & Talagala, T. S. (2020). Fforma: Feature-based forecast model averaging. *International Journal of Forecasting*.

Morens, D. M., Taubenberger, J. K., Harvey, H. A., & Memoli, M. J. (2010). The 1918 influenza pandemic: Lessons for 2009 and the future. *Critical care medicine*.

Morris, P. (1990). The statistical analysis of discrete data, by tj santner and de duffy. pp 367. dm88. 1989. isbn 3-540-97018-5 (springer). *The Mathematical Gazette*.

Negahban, S. N., Ravikumar, P., Wainwright, M. J., Yu, B., et al. (2012). A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers. *Statistical science*, *27*(4), 538–557.

Ostapuk, N., Yang, J., & Cudré-Mauroux, P. (2019). Activelink: Deep active learning for link prediction in knowledge graphs. *The World Wide Web Conference*, 1398–1408.

Østergård, T., Jensen, R. L., & Maagaard, S. E. (2018). A comparison of six metamodeling techniques applied to building performance simulations. *Applied Energy*, *211*, 89–103.

Palmonari, M., & Minervini, P. (2020). Knowledge graph embeddings and explainable ai. *Knowledge Graphs for Explainable Artificial Intelligence: Foundations, Applications and Challenges, IOS Press,, Amsterdam*, 49–72.

Peng, H., Bobade, S. U., Cotterell, M. E., & Miller, J. A. (2018). Forecasting traffic flow: Short term, long term, and when it rains. *International Conference on Big Data*.

Peng, H., Klepp, N., Toutiaee, M., Arpinar, I. B., & Miller, J. A. (2019). Knowledge and situation-aware vehicle traffic forecasting. *2019 IEEE International Conference on Big Data (Big Data)*.

Peng, H., & Miller, J. A. (2019). Multi-step short term traffic flow forecasting using temporal and spatial data. *International Conference on Big Data*, 110–124.

Percival, D. B., & Walden, A. T. (2000). *Wavelet methods for time series analysis* (Vol. 4). Cambridge university press.

Peters, J., & Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, *71*(7-9), 1180–1190.

Plumb, G., Molitor, D., & Talwalkar, A. S. (2018). Model agnostic supervised local explanations. *Advances in Neural Information Processing Systems*.

Prasanth, S., Singh, U., Kumar, A., Tikkiwal, V. A., & Chong, P. H. (2020). Forecasting spread of covid-19 using google trends: A hybrid gwo-deep learning approach. *Chaos, Solitons & Fractals*.

Provisional death counts for coronavirus disease 2019 (covid-19). (2021). https://www.cdc.gov/nchs/nvss/vsrr/covid19/index.htm

Przymus, P., Hmamouche, Y., Casali, A., & Lakhal, L. (2017). Improving multivariate time series forecasting with random walks with restarts on causality graphs. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*.

Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., & Cottrell, G. (2017). A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint*.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Rangapuram, S. S., Seeger, M., Gasthaus, J., Stella, L., Wang, Y., & Januschowski, T. (2018). Deep state space models for time series forecasting. *Proceedings of the 32nd international conference on neural information processing systems*.

Ravi, S., & Larochelle, H. (2016). Optimization as a model for few-shot learning.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2018). Anchors: High-precision model-agnostic explanations. *Thirty-Second AAAI Conference on Artificial Intelligence*.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., & Tarantola, S. (2008). *Global sensitivity analysis: The primer*. John Wiley & Sons.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., & Lillicrap, T. (2016). One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, *20*(1), 61–80.

Sheth, A., Gaur, M., Kursuncu, U., & Wickramarachchi, R. (2019). Shades of knowledge-infused learning for enhancing deep learning. *IEEE Internet Computing*, *23*(6), 54–63.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*.

Shrikumar, A., Greenside, P., & Kundaje, A. (2017). Learning important features through propagating activation differences. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Simonyan, K., & Zisserman, A. (2014a). Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 568–576.

Simonyan, K., & Zisserman, A. (2014b). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Smyl, S. (n.d.). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *IJF*.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Srivastava, N., Mansimov, E., & Salakhudinov, R. (2015). Unsupervised learning of video representations using lstms. *International conference on machine learning*, 843–852.

Storn, R. (1995). Differrential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces. *Technical report, International Computer Science Institute*.

Su, J., Vargas, D. V., & Sakurai, K. (2019). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*.

Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv preprint*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Taieb, S. B., Hyndman, R. J. et al. (2012). *Recursive and direct multi-step forecasting: The best of both worlds*. Citeseer.

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*.

Toshev, A., & Szegedy, C. (2014). Deeppose: Human pose estimation via deep neural networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1653–1660.

Toutiaee, M., & Miller, J. A. (2020). Gaussian function on response surface estimation. *2020 IEEE International Conference on Big Data (Big Data)*, 1097–1102.

United states covid-19 cases and deaths by state over time. (n.d.). https://data.cdc.gov/Case-Surveillance/United-States-COVID-19-Cases-and-Deaths-by-State-o/9mfq-cb36

Vaghasia, S. (2018). An approach of traffic flow prediction using arima model with fuzzy wavelet transform.

Van Looveren, A., & Klaise, J. (2019). Interpretable counterfactual explanations guided by prototypes. *arXiv preprint arXiv:1907.02584*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*.

Venables, W. N., & Ripley, B. D. (2013). *Modern applied statistics with s-plus*. Springer Science & Business Media.

Vidakovic, B. (2009). *Statistical modeling by wavelets* (Vol. 503). John Wiley & Sons.

Wang, P., Lu, Z., & Tang, Z. (2013). An application of the kriging method in global sensitivity analysis with parameter uncertainty. *Applied Mathematical Modelling*.

Wang, Y., Smola, A., Maddix, D., Gasthaus, J., Foster, D., & Januschowski, T. (n.d.). Deep factors for forecasting. *International Conference on Machine Learning*.

Welch, W. J., Buck, R. J., Sacks, J., Wynn, H. P., Mitchell, T. J., & Morris, M. D. (1992). Screening, predicting, and computer experiments. *Technometrics*.

Who coronavirus (covid-19) dashboard. (2021). https://covid19.who.int/

Wu, N., Green, B., Ben, X., & O'Banion, S. (2020). Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint*.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1492–1500.

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional gan. *Advances in Neural Information Processing Systems*, 7335–7345.

Yoon, J., Jarrett, D., & van der Schaar, M. (2019). Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, 5508–5518.

Yoon, J., Jordon, J., & van der Schaar, M. (2018). Invase: Instance-wise variable selection using neural networks. *International Conference on Learning Representations*.

Yu, Y., Si, X., Hu, C., & Zhang, J. (n.d.). A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*.

Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., & Toderici, G. (2015). Beyond short snippets: Deep networks for video classification. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. *European conference on computer vision*.

Zhang, C.-H. et al. (2010). Nearly unbiased variable selection under minimax concave penalty. *The Annals of statistics*, *38*(2), 894–942.

Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D. N. (2017). Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *Proceedings of the IEEE International Conference on Computer Vision*.

Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., & Li, H. (2019). T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2921–2929.

Zhou, G.-B., Wu, J., Zhang, C.-L., & Zhou, Z.-H. (2016). Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*.

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *Proceedings of the IEEE international conference on computer vision*, 2223–2232.

Zivot, E., & Wang, J. (2006). Vector autoregressive models for multivariate time series. *Modeling financial time series with S-PLUS®*.