Prediction of COVID-19 Positive Rates in Tokyo Using Neural Network Models

Waseda University School of International Liberal Arts

Yuuki Inada

January 19th, 2022

Abstract

The novel COVID-19 has presented unprecedented impact over the world. Tokyo Japan is no different and the deadly virus has forced administrative management to take immediate measures to contain the virus. Therefore, prediction of COVID-19 patients and the situation in large has been a pressing research interest over the world. The feature in which the number of reported COVID-19 cases have some dependency with earlier observed statistical data, this paper aims to use multiple time series analysis approach to construct an effective predictive model of Tokyo's observed positive rate from PCR testing. This paper will focus on two predictive methods which are Univariate prediction and Multivariate prediction using several conventional models and neural network models. The models used in this paper are: Autoregressive Moving Average model (ARMA), Vector Autoregression Moving Average model(VARMA), Long Short-term Memory(LSTM) model, and Gated Recurrent Unit (GRU) model. These models are known to be robust when constructing predictive models. Using these models and predictive procedures, this paper will predict the observed positive rates of COVID-19 for the latest 40 days for standard prediction, 20 days for short-term prediction and 80 days for long-term prediction. This paper will also discuss the statistical assumptions it needs to pass for a clear prediction which are: stationarity within the input data, and checking for multicollinearity.

The result show that while linear models present some promising predictions, some neural network models were able to predict the model with a very high accuracy rate. This paper also found that the long term prediction of COVID-19 using neural network was difficult. Another difficulty was found during the comparison of Multivariate predictions and Univariate predictions, where Multivariate prediction failed to perform at the standard of accuracy which Univariate prediction was capable of. Extensive research in this paper have found that the foot traffic data, which accounts for people's movement, do not demonstrate a significant factor in the prediction process.

Acknowledgement

I would like to express my deep and sincere gratitude to my supervisor, Professor Satoshi Inaba, for giving me guidance not only through this thesis but also throughout my academic endeavor during my Bachelor's degree. His extensive support went beyond course work and has especially game me the biggest encouragement during my graduate school applications. It was with his warm words and clear cut advice that I was able to pull through in some of the most hectic times in my life. I would also like to thank Professor Ryuichiro Ishikawa for the guidance of my career path. With his warm kindness, he has supported me through the graduation school application process and gave me advice on my career path even though I was not under his direct supervision.

I would also like to thank my classmates and friends who made my 4 years in university so fruitful. It was an honor to interact with such great minds and people with diverse backgrounds and aspirations. Without the experience in SILS, I would not have been able to enjoy learning and formulate motivation of extending my knowledge on the vast horizons of academia.

Above all, I would like to appreciate my family for all the sacrifices for educating and preparing me for my future. Without their unconditional love and support, I would not be here the way I am today. I hope to become myself a person with great love and care to support important people around me.

Contents

List of Figures

List of Tables

1 Introduction

The spread of the novel COVID-19 virus has caused wold-wide turmoil and has been a pressing issue since 2020 when World Health Organization (WHO) declared pandemic. Government officials has been forced upon taking immediate action to contain the highly infectious virus and there is high demand where we project statistics which reflects what the situation will be like in the near future. In reality, the Kanagawa prefecture officials and the Kanagawa University of Human Services has established a project team and released several predictive models which projects the number of people who have severe symptoms (Kanagawa Prefecture Official, 2022). This model has been released in the Kanagawa prefecture's website and is intended for local citizens to have general understanding of what the situation will be like. Thus, having an accurate predicted model is highly desired for both the government officials; having more resources to base their political measures and for the citizens; acquiring prior knowledge of what the situation would be like in the near future. The motivation behind this paper is to reproduce a likewise model for Tokyo prefecture, which predicts the number of COVID-19 positive rates.

Predicting future values using time series modelling is an intensively researched topic amongst academia and business. The most well known predictive model is the Autoregressive Integrated Moving Average (ARIMA) which has been popularly used due to its flexibility in its statistical properties. However, the obstacle in which ARIMA model presents to capture the nonlinear characteristics in data, have characterized the need of an alternative predictive model which captures the volatile nature of time series data (Zhang, 2003). With the rise of Artificial Neural Networks (ANN), the Long Short-Term Memory (LSTM) model have been presenting strong accuracy in its predictive abilities. Firstly introduced by Hochreiter and Schmidhuber (1997) in 1997 as a special case of Recurrent Neural Network(RNN), many researches are comparing the accuracy of predictions with ARIMA models and LSTM models. Some research concluded that LSTM outperformed the conventional ARIMA model of predicting the stock prices by a high margin of 84-87% reduction in error rates (Siami-Namini, 2018). Another breakthrough in predictive models occurred in 2014, when Cho et al. (2014) introduced the Gated Recurrent Unit (GRU) initially to capture the dependencies within sequences in the context of machine translation. Other researches have shown that GRU can be applied to time series data to forecast future values and have concluded GRU to be a superior predictive model over other ANN models (Shen et al., 2018).

This paper aims to utilize several deep learning models as well as conventional time series forecasting models in search for an accurate predictive model for Tokyo's COVID-19 positive rate. Using multiple features which has some correspondence with the value of COVID-19 positive rates, this paper will split the prediction question to two sections: Univariate modelling which only uses the objective value for training and testing, and Multivariate modelling which takes multiple time series features into account for training to produce the predictions. The core deep learning model that will be used in this paper are simple Recurrent Neural Network(RNN), Long Short-term Memory (LSTM), and Gated Recurrent Unit (GRU). For the baseline models, this paper will use a robust linear time series prediction models, which are the Autoregressive Integrated Moving Average (ARIMA) model for univariate modelling and Vector Autoregression (VAR) model for multivariate modelling. Utilizing these models, we compare the results of predictions to suggest the optimal predictive model.

The specific implementation of these models as well as the pre-processing of the time series dataset will be covered in the *Methodology* section, and the performance will be evaluated in the *Results* section. The discussion and topics which needs to be coverred in the future will be mentioned in the *Discussion and Future Works* section.

2 Methodology

This section will mainly cover the basic statistical properties which needs to be addressed in order to handle time series data, as well as the theories behind the models utilized for predictions. Going over the features of data used in this paper in subsection 2.1, 2.2 the features of the dataset and the pre-processing of data which remedies some of the problematic statistical properties would be mentioned. In subsection 2.3 and 2.4, the modelling process behind the conventional model and the deep learning model will be discussed respectively.

2.1 Dataset

This paper uses three datasets which are released from the government. 2 datasets are from the Tokyo Bureau of Social Welfare and Public Health which contains information about the daily observed COVID-19 numbers, hospital vacancy, and PCR results conducted in Tokyo prefecture. The other dataset is from the Cabinet Secretariat of Japan which releases daily numbers of foot traffic data observed in Shinjuku Station. All of the datasets used in this paper are captured in daily spans and the range of the data is fixed from May 2nd, 2020 to December 30th, 2021 for all features (Government, 2022b,a; of Japan, 2022). In this paper, we will concatenate all 3 datasets into 1 master dataset and construct our models.

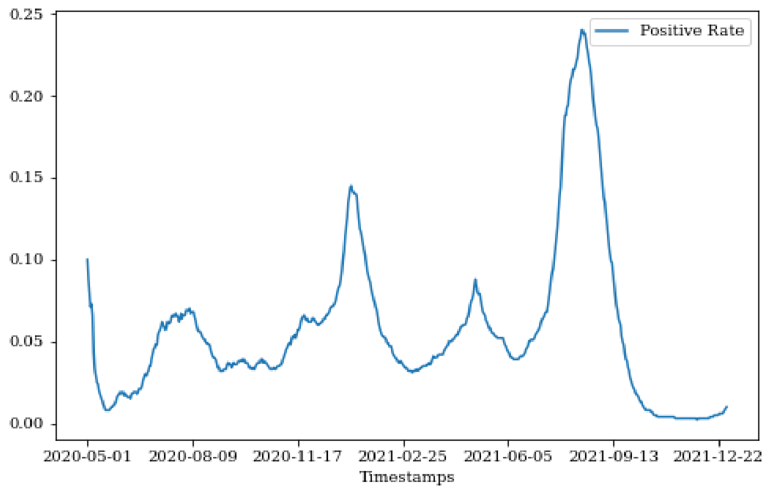


Figure 2.1: This shows an example of a time series line plot. The blue line represents the value of data point at a given timestamp.

The total number of time steps captured in this dataset is **608 rows** by **12 columns**. The features are as follows:

- **Date**: The daily timestamps
- **Hospitalized**: The number of patients being hospitalized on that date
- **Light-Mid_Symptoms**: The number of patients who have light-mid symptoms on that date

- **Severe_Symptoms**: The number of patients who have severe symptoms on that date
- **Dead**: Daily Death tolls
- **Discharged**: The number of patients who are discharged from hospitals on that date
- **PCR_Positive**: The number of people who are tested positive in PCR testing on that date
- **PCR_Negative**: The number of people who are tested negative in PCR testing on that date
- **Tested_MA(7days)**: The moving average of the number of PCR tests done within the last 7days
- **ComparisonPreDay**: The percent change of the population growth rate compared to the former day
- **ComparisonPreDeclare**: The percent change of the population growth rate compared to the population during the 3rd Emergency Declaration issued in January 7th, 2021
- **ComparisonPreSpread**: The percent change of the population growth rate compared to the average population during pre-COVID-19 span (January 18th, 2020 - February 14th, 2020)

2.2 Dataset Pre-processing

2.2.1 Standardization

The aforementioned features vary in terms of numerical characteristics from decimals to percentage points. In this paper, all the features within the dataset is standardized from its individual mean and standard deviation defined as:

$$X_{\text{std}} = \frac{X - \mu}{\sigma}, \tag{2.1}$$

where X are the input data, μ is the mean of the featured data, σ is the standard deviation of the featured data, and X_{std} is the standardized data.

Added to the pre-processing of individual data, there are several statistical properties that each time series data needs to fulfill in order to induce an accurate prediction. Below would explain several statistical features the data needs to fulfill.

2.2.2 Stationarity

Stationarity in time-series data is when the statistical properties, such as the mean, variance, and the covariances between given time periods are constant over time. Assuring stationarity within the dataset precludes the risk of having a spurious forecast result (Granger and Newbold, 1974).

In Figure 2.2, we see that there is a positive trend in our values as time progress. This violates the condition of having a fixed mean over time, hence we could conclude that this process is non-stationary.

To numerically determine if the dataset is in fact stationary or non-stationary, we conduct a unit root test, checking the presence of a unit root within the time series data.

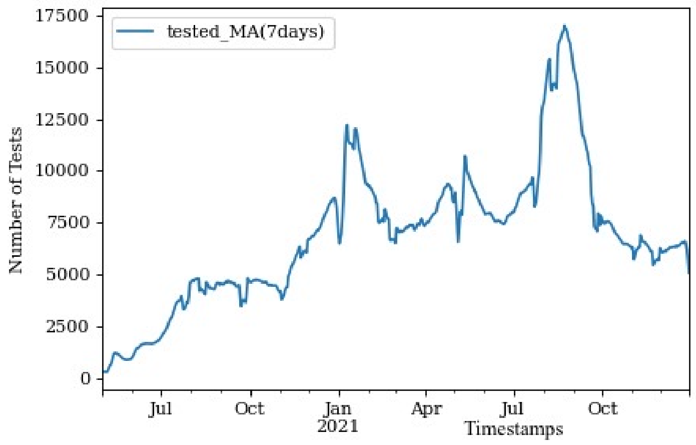


Figure 2.2: The feature **tested_MA(7days)** has an upward incline and the mean at a given time is not constant throughout the timestamps. This statistical feature represents a non-stationary process.

There exists many unit root tests, but this paper would focus on the Augmented-Dickey Fuller (ADF) test (Dickey and Fuller, 1979). Below would be a brief summary cited from the works of Dickey and Fuller (1979); Greene (2003). For explanation of ADF test, we assume an observed time series data of $Y_1, Y_2, \dots, Y_N$ and a differential-form autoregressive equation ΔY_t as:

$$\Delta Y_t = \alpha + \boldsymbol{\beta} t + \gamma Y_{t-1} + \sum_{j=1}^{p-1} (\delta_{\mathrm{j}} \Delta Y_{t-j}) + \varepsilon_t, \tag{2.2}$$

where Δ is the first difference operator, t is the time index, α is the intercept constant often called as drift, β is the coefficient of a time trend, p is the maximum lag order of the autoregressive process, and ε is an independent identically distributes residual term.

The specifications varies different depending on the deterministic elements presented in the time series data; if there is a drift in the data $\alpha \neq 0$, if there is also a linear trend $\beta \neq 0$, and if neither drift nor linear trends and are present, we illustrate by $\alpha = 0, \beta = 0$. We do a hypothesis test if the coefficient γ is equal to 0, which means that the autoregressive process possesses a unit root. Hence, the null hypothesis and the alternative hypothesis is defined as:

$$H_0 : \gamma = 0 \text{ (unit root exist and hence non-stationary).} \tag{2.3}$$

$$H_1 : \gamma < 0 \text{ (no unit root and hence stationary).} \tag{2.4}$$

The test statistic is defined as:

$$ADF_{\mathrm{t}} = \frac{\hat{\gamma} - 1}{\mathrm{SE}(\gamma)}, \tag{2.5}$$

where $\hat{\gamma}$ is the least squares estimate. To test the null hypothesis, we compare this test statistic with the critical value and rejects the null hypothesis when the test statistic is less than the critical value. In this paper, the ADF test is implemented using the ADFULLER module provided by the STATSMODELS library (Seabold and Perktold, 2010b).

2.2.3 First Differencing

When the time series is found to be a non-stationary process, there is a need to convert our process into a stationary process. In this paper, a method called first differencing is applied to remedy the non-stationary process. For a given time series Y_t, the procedure of first difference is defined as:

$$\Delta Y_{\mathrm{t}} = Y_{\mathrm{t}} - Y_{\mathrm{t}-1}, \tag{2.6}$$

where t is the given time period and Δ is the first difference operator.

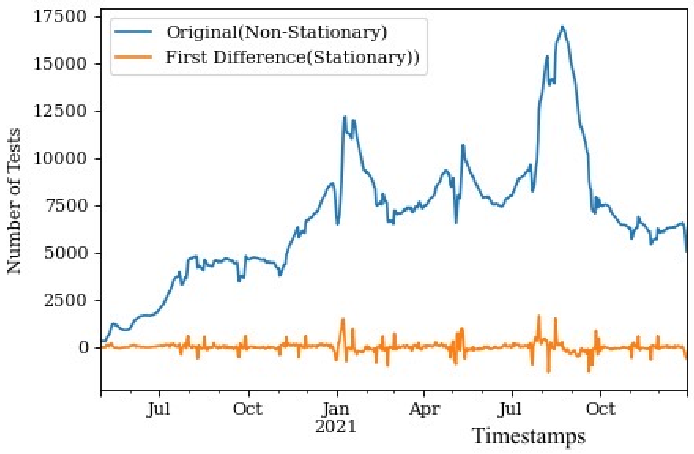


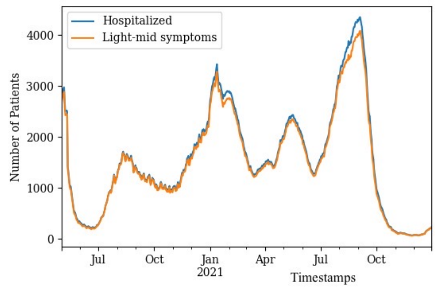
Figure 2.3: The blue line represents the original time series of **Tested_MA(7days)**. The orange line represents the same data feature but after the First Difference is taken. We could see that the modified time series converges close to the axis of 0 and the mean is constant throughout time. Hence, we could now visually indicate that the process has become stationary.

As apparent from Figure 2.3 we could indicate a positive trend in our original data of feature *Tested_MA(7days)*. This indicates that the original data does not fulfill the condition of stationarity from the attribute that the mean alters within different given time. By applying the first difference, the data is now converged close to the axis and do not indicate any drift, hence stationary. Therefore, applying the first difference for the features which is non-stationary in our data establishes a robust foundation for our predictive model.

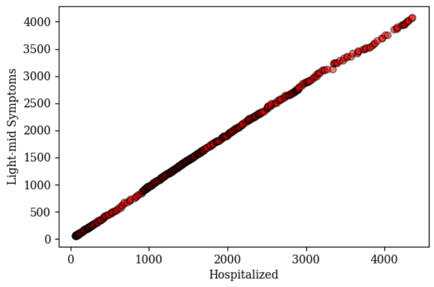
2.2.4 Multicollinearity

Multicollinearity refers to the linear relationship between two or more time series data. More specifically, there is multicollinearity when there is high correlation among 2 or more independent variables. Figure 2.4 shows the multicollinearity of the *Hospitalized* feature and *Light-Mid_Symptom* feature in our data. We could see that the lines are identical and the quantile-quantile plot is linear. Multicollinearity presents detrimental issues in the precision of regression models, whereas the statistical significance of independent variables are undermined due to the partial regression coefficient becoming highly volatile between different samples (Allen, 1997).

To detect multicollinearity within the dataset, the Variance Inflation Factor (VIF) is popularly used defined as:



(a) Line graph of **Hospitalized**(blue) and **Light-Mid_Symptoms**(orange).



(b) Quantile-Quantile plot of **Hospitalized** and **Light-Mid_Symptoms**

Figure 2.4: Figure (a) shows a line plot of two datasets. By plotting them together, the lines indicate an identical trend and is in complete align with each other. From this property, we can indicate multicollinearity within these two data features. Figure (b) is the Q-Q plot for the same datasets. Q-Q plots show whether two given datasets have a similar probability distribution. Using the Q-Q plot we can visually see that the plot of the given dataset is in complete linear relation, which indicates that these two features have a very identical distribution over its respective datasets.

$$VIF_i = \frac{1}{1 - R_i^2}, \tag{2.7}$$

where R_i is the Coefficient of determination for the ith individual variable. A large value in VIF indicates high linear dependency in its variables and the threshold to determine from high to low is generally set at 10 (Alin, 2010). In this paper, the diagnosing of multicollinearity is implemented using the VARIANCE_INFLATION_FACTOR module provided by the STATSMODELS library (Seabold and Perktold, 2010a). Despite multicollinearity causing detrimental effects to the precision of regression models, there are no concrete methods to remedy the statistical feature altogether. In this paper, the explanatory variables which presented high VIF are excluded from the model to avoid jeopardizing the accuracy of predictions.

2.3 Conventional Models

In this paper, several conventional models are used as baseline models to compare the precision of the predictions with the deep learning models.

2.3.1 ARMA model

The *Autoregressive Moving Average (ARMA)* model is a univariate time series model which is widely used for forecasting values. *ARMA* integrates features of *Autoregressive(AR)* model and *Moving Average(MA)* model. When assuming a non-stationary process in the objective model, Autoregressive Integrrated Moving Average (ARIMA) model is used, which takes the differencing process (defined in equation 2.6), into account to convert the process into a stationary time series. In our paper, a preprocessed dataset which already passes the ADF test prior to the modelling process will be used in the modelling process, hence *ARMA* model will be discussed.

AR model assumes that the objective values depend on the values from the past, hence can seek a regressive relationship between past values and future values. MA model on the other hand, focuses on the error term in which the linear regression was unable to incorporate. Finally, by incorporating features of the regressive relationship in the AR model, and the patterns in the error term captured by the MA model we get the ARMA model which is capable to illustrate more complex time series. The AR model, MA model, and ARMA model are defined as:

$$\mathrm{AR}(p) : Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \epsilon_t, \tag{2.8}$$

$$\mathrm{MA}(q) : Y_t = c + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t, \tag{2.9}$$

$$\mathrm{ARMA}(p, q) : Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t, \tag{2.10}$$

where Y is the objective value, t is the given time period, c is the constant accounting for the drift, p and q are the amount of lags used to regress with the value of prediction, ϕ and θ are the corresponding coefficient of the past objective values, and past error terms respectively, and ϵ is the error term. Conventionally, the lag terms in each of these models will be written in parenthesis after the model name such as ARMA(p, q) in (2.8).

In this paper, the ARIMA model will be used as a baseline model for the univariate prediction section. Therefore, if the prediction error in this model is bigger than the neural network model, it suffices to say that the deep learning is ineffective in its prediction.

2.3.2 VARMA model

The **Vector Autoregressive Moving Average (VARMA)** model is a mutivariate time series model which converts the coefficient of ARMA model into a finite vector space corresponding to the number of features included in the model. Therefore, the characteristic of the model incorporating more information within the model is advantageous if the data of our objective value is generated from a complex latent structure.

Below would be a brief summary by Lütkepohl (2006) about the essential theories of VARMA model. Assuming multiple K-dimensional stationary time series $y_1 \dots y_T$, the VARMA model can be expressed in the general form of:

$$\begin{aligned} A_0 y_t = A_1 y_{t-1} + \cdots + A_p y_{t-p} + M_0 u_t + M_1 u_{t-1} + \cdots + M_q u_{t-q}, \\ t = 0, \pm 1, \pm 2, \dots, \end{aligned} \tag{2.11}$$

where $A_0, A_1, \dots, A_p$ are $(K \times K)$ matrices which stores autoregressive coefficients and $M_0, M_1, \dots, M_q$ also being a $(K \times K)$ matrices which stores moving average coefficients. By defining the VAR and MA operators respectively, as $A(L) = A_0 - A_1 L - \cdots - A_p L^p$ and $M(L) = M_0 + M_1 L + \cdots + M_q L^q$, where L is the *backshift operator*, (2.11) can be written in more compact notation as:

$$A(L) y_t = M(L) u_t, \quad t \in \mathbb{Z}, \tag{2.12}$$

where u_t is a white-noise process.

In this paper, VARMA model will be used as a baseline model of the multivariate time series prediction and will be compared with the performance of neural network models. Therefore, if the prediction error in the VARMA model is smaller than the neural network models, it will be concluded that deep learning methods do not suggest a sufficient replacement with the conventional time series forecasting methods.

2.3.3 Model Selection Process

In both **ARMA** and **VARMA** models, we set the order p, q for the autoregressive model and moving average model, respectively. The values are set to determine the amount of past values the model will use to determine future values. There are various ways to determine the amount of order, but this paper will be concerned with two model selection criteria; Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). Below will be a brief summary of both AIC and BIC using the works of Narisetty (2020) for reference.

AIC was developed by Akaike (1974) to provide an adequate hypothesis testing procedure to identify the optimal statistical model. Let us assume a candidate model M_k with dimension k. The AIC is defined as:

$$\mathrm{AIC} = -2\ln L\left(M_k\right) + 2k, \tag{2.13}$$

where $L(M_k)$ is the likelihood corresponding to the model M_k. Looking at the first term $-2\ln L(M_k)$, we could see that this term is the residual sum of squares corresponding to the model for the linear regression model with a Gaussian likelihood which can be shown as:

$$-2\ln L\left(M_k\right) = \sum_{i=1}^{n}\left(y_i - x_i^\top \widehat{\beta}\left(M_k\right)\right)^2, \tag{2.14}$$

where $\widehat{\beta}\left(M_k\right)$ is the least squares estimator for model M_k. This indicates that the first term of (2.13) is the amount of loss of the model M_k with the true observed value, which is preferred to be minimized. The AIC also penalizes the model of having large dimensions by adding the second term of (2.13).
Therefore, the model which minimizes the value of AIC will be considered to be the best model, and we apply this to evaluate the optimal lag length of the considered model.

An alternative information criteria used for model selection is called the BIC. Developed by Schwarz (1978), the BIC has modified the maximum likelihood estimators in the works of Akaike (1974). Let us assume again a candidate model M_k with dimension k. The BIC for this model is defined as:

$$\mathrm{BIC}\left(M_k\right) = -2\ln L\left(M_k\right) + \ln(n)k, \tag{2.15}$$

where n is the sample size. Similar to AIC, the model which minimizes the value BIC will be considered the optimal model within the candidate model. Compared to AIC, the penalty term in BIC, which is represented as $\ln(n)k$, is much harsher. Therefore, when using BIC we get a sparser model compared to AIC.

In this paper, these information criterion will be used to determine the lag length of our conventional models.

2.4 Deep Learning Models

With the expanding availability of huge datasets and high speed computing technology becoming more affordable, Deep learning modelling is becoming a popular method of predicting future value for its high accuracy. Deep Learning methods is accustomed to solve two major domains of issue which are classification and regression. In this paper, we will use multivariate data to predict the value of one objective value, which can be classified as regression problem. Below would go over the theories and basic structure of the models used in this paper.

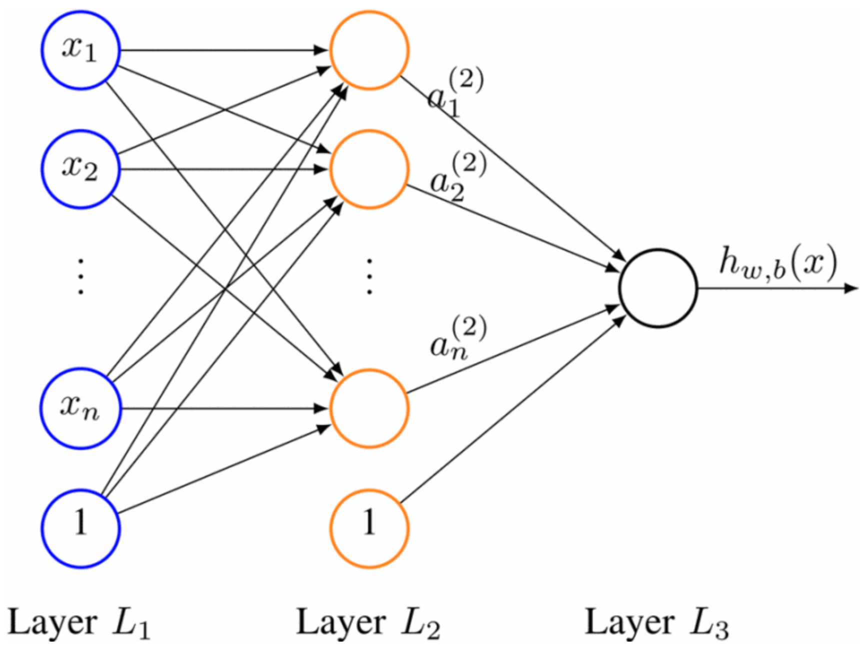


Figure 2.5: Basic Structure of Neural Network(Qiu et al., 2014). The layers indicate a chain of nodes and each nodes are connected with all the nodes in the next layer.

2.4.1 Deep Neural Networks for Regression Problem

Neural Network refers to the structure in which chains of perceptron being connected in multiple layers. We see from Figure 2.5 that the neural network consists of multiple layers and each perceptrons inside the layers are connected with all the perceptrons inside the next layer. The first layer (Layer L_1 in figure 2.5) is called the *input layer* which act as a gate for all input data. The last layer (Layer L_3 in figure 2.5) is called the *output layer* which stores the final outcome of the data which went through the defined network. The layers between *input layer* and *output layer* is called the *hidden layer(s)* (Layer L_2 in figure 2.5) which receives the input data and modifies the value with the designated *activation function* and pass it on to the next layer. By adding more layers into the hidden layer, we get a Deep Neural Network (DNN) which is capable of complex representation.

Whenever the values are passed on to the next layer, the value will be transformed with two sets of parameters which are called *weight* and *bias*. Let us assume that we have a K-layered DNN. The outcome value after the $k-1$th layer is written as $\boldsymbol{Y}^{k-1} = \left\{y_1^{k-1}, y_2^{k-1}, \ldots, y_n^{k-1}\right\}$ where n is the amount of nodes. We get the outcome value of the jth node after the $k-1$th layer a_j^k by calculating:

$$a_j^k = \sum_{i=1}^{n} y_i^{k-1} w_{ij}^k + b_j^k, \tag{2.16}$$

$$y_j^k = h\left(a_j^k\right), \tag{2.17}$$

where w is the weight parameter, and b is the bias parameter. From equation (2.17) we can indicate that the output values y is generated from the activation function which is defined as h. Figure 2.6 illustrates the structure of a node inside the hidden layer.

Activation function h acts as a decision point whether the inputted data should be passed on to the next neuron or not. This paper would introduce one of the fundamental

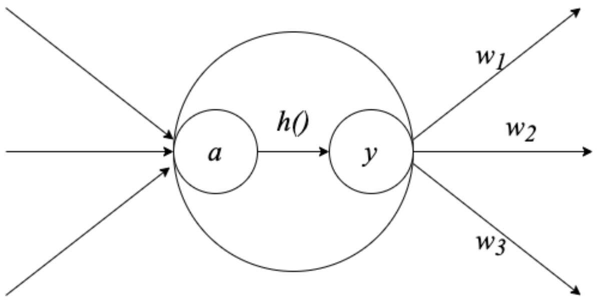


Figure 2.6: Inside Structure of a node in the Hidden Layer. The output value from the previous layer is stored as value a and then modified to value y by the activation function $h()$. y becomes the output for this node and will be passed down to the next layer with the multiplication of their respective weight parameter $\mathbf{w}$

and commonly used activation function called the Sigmoid Function (equation (2.18), Figure 2.7) and the more recently developed activation function called Leaky Rectified Linear Unit (Leaky ReLU) (equation (2.19), Figure 2.8) made by Maas et al. (2013).

Assuming an input data x, the Sigmoid function (2.18) and the Leaky ReLU are defined as:

$$h(x) = \frac{1}{1+e^{-x}}, \tag{2.18}$$

$$h(x) = \begin{cases} x & (\text{ if } x \geq 0) \\ \frac{x}{a} & (\text{ if } x < 0), \end{cases} \tag{2.19}$$

where a is a fixed parameter in the range $(1, +\infty)$. According to Maas et al. (2013), a is preferred to be a large number such as $a = 100$. However, in an empirical study done by Xu et al. (2015), they concluded that $a = 5.5$ performed well than other numbers. Therefore, this paper will adopt $a = 5.5$ for our leaky ReLU parameter.

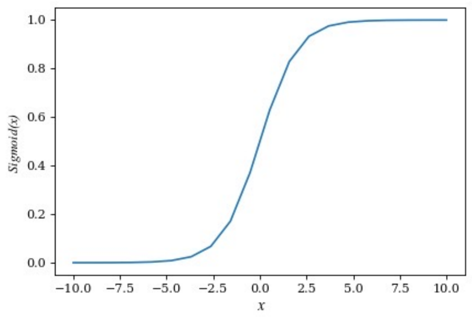


Figure 2.7: A line plot of a Sigmoid function.

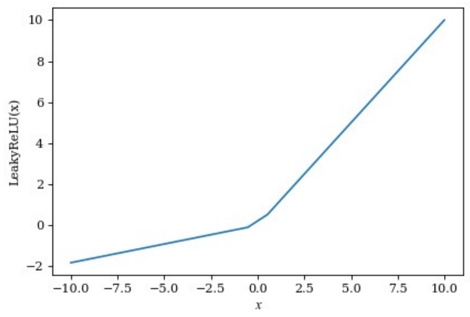


Figure 2.8: A line plot of a Leaky ReLU function.

The activation function in the output layer is different with the function used in the hidden layer. In a regression problem, there should be one final output which reflects all of the information from the outputted data of the previous layer, and to do so we use the Identity Function I defined as:

$$I(x) = x. \tag{2.20}$$

The biggest advantage of using a neural network is that the network is able to optimize its parameters using training data and accurately adapt to the features of the input data.

By setting the loss function E and getting the gradient of this loss function E, we get the indication if the neural network is accurately reflecting the features of the inputted data. For a regression problem we set Sum of Squared Error (SSE) as the loss function which is defined as:

$$\mathrm{E} = \mathrm{SSE} = \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2\,, \tag{2.21}$$

where N is the amount of input data, y as the outputted value from the neural network, and t as the correct data which corresponds to the ith value in the outputted data. Using the loss function E we optimize the weight and bias parameters by the calculation:

$$\hat{w_{ij}} = w_{ij} - \eta\frac{\partial E}{\partial w_{ij}}, \tag{2.22}$$

$$\hat{b_{ij}} = b_{ij} - \eta\frac{\partial E}{\partial b_{ij}}, \tag{2.23}$$

where $\hat{w_{ij}}, \hat{b_{ij}}$ is the updated parameter value of the weight and bias, and η is the learning rate. This modification is aimed to decrease the gradient of the loss function and when the gradient becomes close to 0, we can conclude that the training of neural network is finished. Generally when calculating the gradient of the loss function we use a calculation algorithm called Backpropagation which calculates the error term from the output layer to the previous iteration using the chain rule. Using this algorithm, it is possible to achieve a faster and more efficient way to calculate the gradient.

Another essential concept within DNN is the optimizer. The optimization method shown shown in equations (2.22)(2.23), is called the Stochastic Gradient Descent (SGD), which calculates the gradient from a randomly chosen data. This optimizer however, entails some detrimental issue where the convergence of the gradient is hard to achieve because of the amount of noise the approximation process consist. When the gains are decreasing too slowly, the variance of parameter estimate decreases equally slow. If the gains decrease too quickly on the other hand, the parameter estimate takes a very long time to approach its optimum (Bottou, 2010). Although SGD is a well-known practice with promising results other methods which curves the underlying issue of this method are developed.

One of the promising alternative optimizer is the Adaptive Moment Estimation (Adam) (Kingma and Ba, 2017) which integrated the features of two optimizers; Momentum (Polyak, 1964) and Adaptive Gradient Algorithm (Adagrad) (Duchi et al., 2011). Below would demonstrate the fundamental theory of Adam referring from the works of Ruder (2016). This algorithm consists of two components; the component of gradient by setting the exponential moving average of past gradients m, and the component of learning rate by setting the exponential moving average of past squared gradients v. We compute the two moving averages by:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\,g_t \tag{2.24}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\,g_t^2 \tag{2.25}$$

where m_t and v_t are estimates of the first moment, and the second moment of the gradients respectively. m_t and v_t are initialized in vector 0s and Kingma and Ba (2017) argues that these estimates are biased towards 0. To account for the biases existing in the vectors, Adam optimizer sets the bias correction as:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \tag{2.26}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}. \tag{2.27}$$

This will yield the Adam optimizer defined as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t, \tag{2.28}$$

where η is the learning rate and θ is the parameter in which we are optimizing. Figures 2.9 and 2.10 shows the path of each optimizers. We could indicate that the SDG oscillates between the field of gradient and thus have a redundant path over Adam. Note that this does not however undermine the effectiveness of SGD, but merely stating that SGD takes more iterations to get to the optimum.

There are many other optimization algorithms existing in the field of neural networks. However, this paper will primarily be using Adam for the optimizer hence, will not cover any other optimizers.

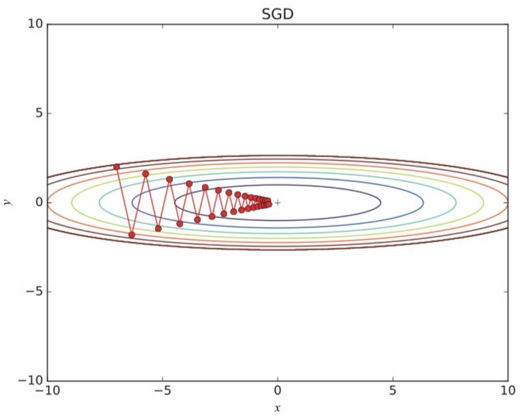


Figure 2.9: The path of SGD optimizer We can see that it has an inefficient route to the optimum. (Saito, 2016)

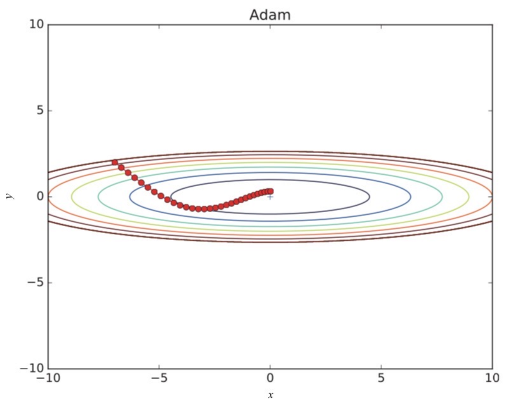


Figure 2.10: The path of Adam optimizer The path efficiently goes to the optimum (Saito, 2016)

This was the basic construct of a neural network used in a regression problem. In a classification problem, despite the structure of the network being the same, the activation function and the loss function used is different. However, this paper will omit the explanation of a classification problem since it is out of the research scope of this paper.

2.4.2 RNN Model

Recurrent Neural Network (RNN) is a type of neural network which is widely used pertaining to research and studies in time series data. The key component of RNNs is that it can store information about the pattern of sequence. Using the output of the previous iteration of the RNN layer as the input for the next layer, the network is able to recognize the features of a sequence of time series data.

Figure 2.11 shows the basic structure of standard recurrent sigma cell. The fundamental difference with a normal DNN is that RNN has a recurrent cell in the hidden unit. The recurrent cell takes the state of the previous iteration as the input which achieves the passing of previous sequential information. Let us assume that in time t we denote x_t as

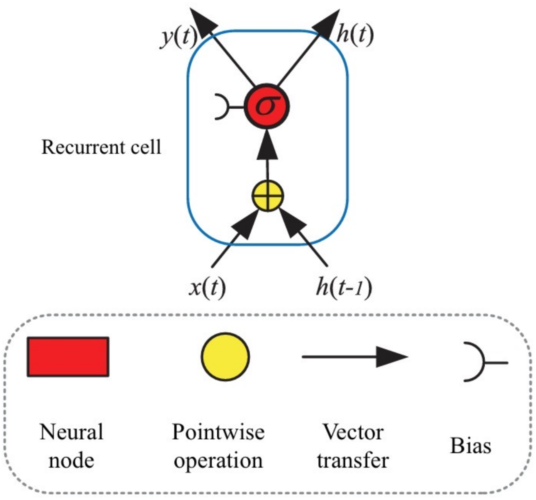


Figure 2.11: The Structure of a RNN node (Yu et al., 2019). We could see that the node consists of multiple operations and has two inputs, which are, an input of the given timestamp, and an input from the hidden layer of the previous iteration.

the input data, h_t as the recurrent information, and y_t as the output data. We can define the output data as:

$$h_t = \sigma\left(W_h h_{t-1} + W_x x_t + b\right), \tag{2.29}$$

$$y_t = h_t, \tag{2.30}$$

where b is the bias, W_h and W_x are the weight of hidden-to-hidden connection, and input-to-hidden connection respectively. σ is the activation function within the hidden layer. The method of optimizing parameters does not differ significantly with DNN. However, the way we compute the gradient of the loss function E differs with DNN since it requires us to expand the computation steps to go back one step at a time to obtain the dependencies among multiple variables and parameters of RNN. Therefore, replacing the normal Backpropagation algorithm utilized in DNNs, we apply Backpropagation Through Time (BPTT) algorithm to calculate the gradients. In this paper however, it would not go over the specific calculation of BPTT since it is out of the scope of this paper.

2.4.3 LSTM Model

RNN performs very strongly in a wide domain of research questions such as speech recognition, natural language processing and various prediction questions in Economics. However, from the research done by Bengio et al. (1994) training RNN presented issues of gradient vanishing and exploding problem, which both causes detrimental effect in the training process of parameters. Furthermore, a more recent research by Karpathy et al. (2015) have found that the RNN is incapable of adequately handling information of the past when the gap of the relevant time series data becomes wide. Addressing these issues, Long Short-Term Memory (LSTM) model was developed by Hochreiter and Schmidhuber (1997) which added the concept of "**gates**" into standard RNN to increase the capacity of its memory. Succeeding the works of Hochreiter and Schmidhuber (1997), Gers and Schmidhuber (2000) modified the LSTM structure by adding a *forget gate* which became the new standard of LSTM.

Figure 2.12 shows the overview of the LSTM cell block. We see that there are several gates inside the cell which are indicated as *Forget gate*, *Input gate*, and *Output gate*. Let

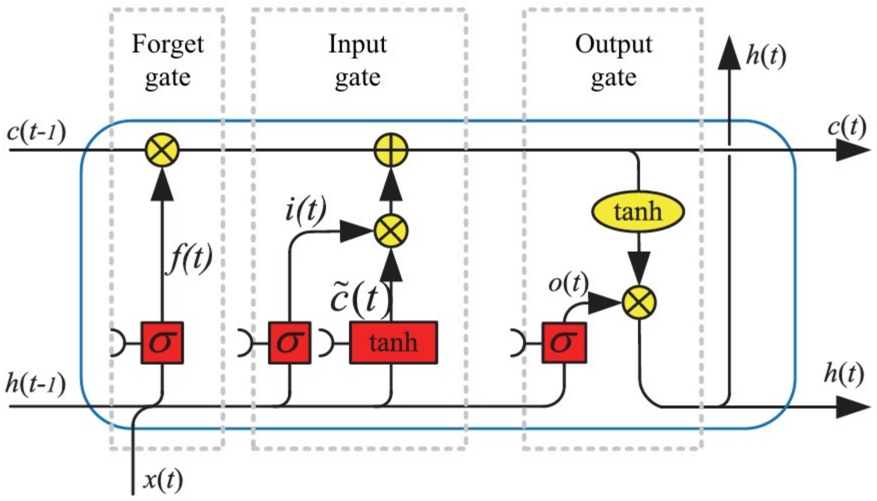


Figure 2.12: The Structure of LSTM node (Yu et al., 2019). The structure is identical with the RNN node but its complexity has increased. The gates within the node is each designed to interpret long sequences which RNN was unable to achieve.

us denote c_t, f_t, i_t, o_t as the cell state, forget gate, input gate, and output gate of LSTM at time t respectively and x_t as the input data. The mathematical expression of Figure 2.12 can be expressed as:

$$f_t = \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f), \tag{2.31}$$
$$i_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i), \tag{2.32}$$
$$\tilde{c}_t = \tanh(W_{\tilde{c}_h}h_{t-1} + W_{\tilde{c}x}x_t + b_{\tilde{c}}), \tag{2.33}$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t, \tag{2.34}$$
$$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o), \tag{2.35}$$
$$h_t = o_t \cdot \tanh(c_t), \tag{2.36}$$

where $W_{\alpha\beta}$ is the weight of the α-to-β connection (e.g. W_{fx} is the weight value of forget gate-to-input data connection), b is the bias term corresponding to the gates, $\tilde{c}_t$ represents the candidate of cell state of the given time stamp which will be used in the computation of cell state c_t in the output gate (shown in equation (2.33)), h_t is the output of this LSTM cell block, σ is the activation function, and the operator '$\cdot$' denotes the pointwise multiplication of two vectors.

The input gate decides the new information stored in the cell state, the forget gate determines how much information from the previous iteration would be passed on to the current cell with values ranging from 1 (passes on all data) to 0 (omits all data from previous cell), and finally the output gate decides what information can be outputted based on the cell state.

2.4.4 GRU Model

Utilizing this LSTM structure, we became capable of handling the long-term dependencies and remedy the issues pertaining to the computation of gradients presented in the standard RNN. However, the addition of parameters attributed to the concept of gates intensified the computational complexity. To modify, as well as simplify the LSTM structure, Cho et al. (2014) developed the Gradient Recurrent Unit (GRU) Model which consisted of only two gates; the Reset gate and the Update gate.

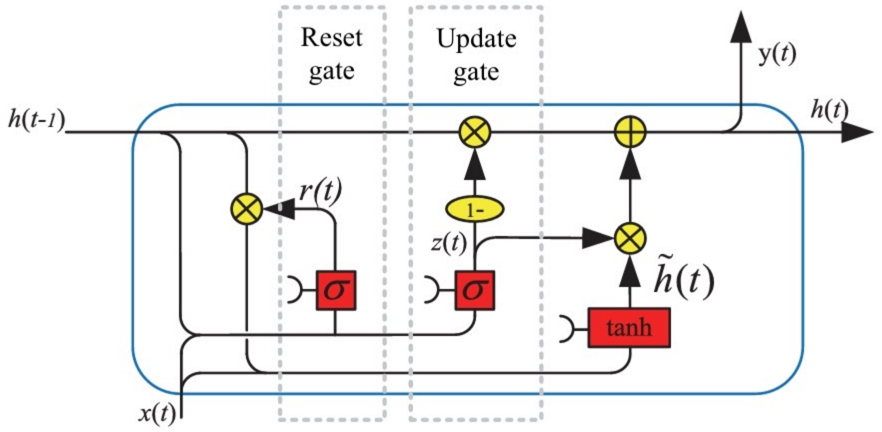


Figure 2.13: The Structure of GRU node (Yu et al., 2019). The structure has decreased its complexity compared to LSTM only having two gates. This accounts for the decrease of parameters used which significantly affects the speed of training.

Figure 2.13 shows the basic construct of the GRU. Compared with Figure 2.12, we can see that the overall cell structure does not have a drastic change but the amount of notations and gates inside the cell have decreased. Let us denote r_t and z_t as the reset gate and the update gate respectively. The mathematical expression of Figure 2.13 can be expressed as:

$$r_t = \sigma\left(W_{rh}h_{t-1} + W_{rx}x_t + b_r\right), \tag{2.37}$$
$$z_t = \sigma\left(W_{zh}h_{t-1} + W_{zx}x_t + b_z\right), \tag{2.38}$$
$$\tilde{h}_t = \tanh\left(W_{\tilde{h}h}\left(r_t \cdot h_{t-1}\right) + W_{\tilde{h}x}x_t + b_z\right), \tag{2.39}$$
$$h_t = (1 - z_t) \cdot h_{t-1} + z_t, \cdot\tilde{h}_t, \tag{2.40}$$

where W and b are the weight and bias parameters respectively which corresponds with each gates, σ is the activation function, and $\tilde{h}_t$ represents the candidate of output of the given time stamp which will be integrated in the computation of output h_t of this GRU cell block which is shown in equation (2.39).

Using less parameters than LSTM, GRU is capable of training its parameters in a rapid nature. However, some studies such as Cahuantzi et al. (2021) have concluded that when the complexity of the input increases, LSTM performed better than GRU.

In this paper, we aim to use these methodologies to compare the results between conventional methods as well as advanced deep learning methods to determine the best performing method for predicting COVID-19 positive rates. For the formulation of prediction models we use the Keras library provided by Chollet et al. (2015).

3 Results

3.1 Data Pre-processing

This section would cover the actual experimentation done in this research. The pre-processing of data, the results for different range of prediction, and the findings from the results will be written accordingly. In this paper, two kinds of predictions were done; Univariate Prediction, and Multivariate Prediction. Univariate Prediction is a prediction procedure which uses only the past data of the objective value for predictions. On the other hand, Multivariate Prediction refers to the prediction procedure using multiple features to predict the objective value. We would like to focus on the accuracy of our models when handling multiple features into the data as well as the efficacy of deep learning models compared to conventional linear methods.

3.1.1 Stationarity

As mentioned in section 2.2.2, stationary diagnostic is imperative to avoid spurious prediction results. Therefore we use the Augmented Dickey-Fuller (ADF) test to determine stationarity in our dataset. This paper would conduct a *Student-t Hypothesis test* shown in (2.3) using the test statistic shown in (2.5) for all data features. We set multiple significance level $(10\%, 5\%, 1\%)$ to see how the extent of stationarity the data features are able to achieve. Below would be a table showing the result of ADF test before pre-processing the data.

Table 3.1: ADF 1st Test

ADF Test results by Features					
Data Feature	Test Statistic	1% Significance	5% Significance	10% Significance	Pass/Reject
Hospitalized	-3.2875	-3.4413	-2.8664	-2.5693	Reject at 1%
Light-Mid_Symptoms	-3.2277	-3.4413	-2.8664	-2.5693	Reject at 1%
Severe_Symptoms	-3.1195	-3.4415	-2.8665	-2.5694	Reject at 1%
Dead	-1.9363	-3.4414	-2.8664	-2.5694	Reject at 10%
Discharged	-3.6668	-3.4414	-2.8664	-2.5694	Pass
PCR_Positive	-3.9643	-3.4415	-2.8665	-2.5694	Pass
PCR_Negative	-2.1540	-3.4415	-2.8665	-2.5694	Reject at 10%
Tested_MA(7days)	-2.3580	-3.4415	-2.8664	-2.5694	Reject at 10%
Positive_Rate	-3.6516	-3.4414	-2.8664	-2.5694	Pass
ComparisonPreDay	-6.7442	-3.4414	-2.8664	-2.5694	Pass
ComparisonPreDeclare	-3.9288	-3.4414	-2.8664	-2.5694	Pass
ComparisonPreSpread	-4.1440	-3.4414	-2.8664	-2.5694	Pass

Looking at the Test Statistic column of Table 3.1, we see the test statistic for the corresponding features and the critical values for all the significance level that we have set. The rejected features are emphasised within the table and we can see features, **Dead**, **PCR_Negative** and **Tested_MA(7days)** demonstrate high non-stationarity being rejected at the siginificance level of 10%, and features, *Hospitalized*, *Light-Mid_Symptoms*, *Severe_Symptoms* demonstrates some non-stationarity which is rejected at the significance level of 1%. These features all requires data pre-processing of taking the first difference which is shown in equation (2.6).

After we modify the dataset by taking the first difference, we determine once again for stationarity using ADF test. Below would be the result of the second ADF test.

Table 3.2: ADF Test after Taking the First Difference for Rejected Features

ADF Test results by Features					
Data Feature	Test Statistic	1% Significance	5% Significance	10% Significance	Pass/Reject
Hospitalized	-5.8545	-3.4414	-2.8664	-2.5694	Pass
Light-Mid_Symptoms	-5.9519	-3.4414	-2.8664	-2.5694	Pass
Severe_Symptoms	-4.3166	-3.4415	-2.8665	-2.5694	Pass
Dead	-8.2268	-3.4414	-2.8664	-2.5694	Pass
PCR_Negative	-7.6164	-3.4415	-2.8665	-2.5694	Pass
Tested_MA(7days)	-4.4329	-3.4415	-2.8664	-2.5694	Pass

As demonstrated in Table 3.2, we can see that all features of data passes the ADF test at all significance level. Therefore, we can confirm that we have a dataset which all features are stationary and precludes the problematic statistical features which affects the prediction results.

3.1.2 Multicollinearity

Multicollinearity is also a statistical property in time series data, which should be avoided in predictive models. As demonstrated in Section 2.2.4, multicollinerarity is a linear relationship within data features in the independent variable which undermines the statistical significance of the independent variable by making the partial regression coefficient highly volatile. We check for multicollinearity using Variance Inflation Factor (VIF) which is computed by equation (2.7). Table 3.3 is the table which shows the value of VIF before any pre-processing procedures.

Table 3.3: VIF values determining Multicollinearity

Data Feature	VIF
Hospitalized	**inf**
Light-Mid_Symptoms	**inf**
Severe_Symptoms	**inf**
Dead	1.0716
Discharged	3.3515
PCR_Positive	4.0029
PCR_Negative	1.8314
Tested_MA(7days)	1.2787
ComparisonPreDay	1.0452
ComparisonPreDeclare	**55.2256**
ComparisonPreSpread	**52.9435**

The threshold of VIF whether the data features suggest multicollinearity is $VIF \geq 10$. Looking at Table 3.3 we can see several features which induces multicollinearity. We first see the first three columns; **Hospitalized**, **Light-Mid_Symptoms**, and **Severe_Symptoms** having **inf** value for its VIF. Looking at Figure 2.4, the features in this example which are **Hospitalized** and **Light-Mid_Symptoms** show high correlation and can estimate that these data demonstrates a linear relationship. Therefore, we exclude the **Hospitalized** column to avoid multicollinearity. The other modification we can suggest is to exclude one of the human foot traffic data, which have an identical data generation process of comparing the traffic data with the past traffic data. Therefore, this paper will exclude **ComparisonPreDeclare** which presents a higher value in VIF than other foot traffic data. Through the modifications of dropping two features within our dataset we get the set of VIF shown in Table 3.4.

Table 3.4: VIF values After Removing Columns of Similar Data Characteristic

Data Feature	*VIF*
Light-Mid_Symptoms	1.1573
Severe_Symptoms	1.3379
Dead	1.0643
Discharged	3.3244
PCR_Positive	3.9300
PCR_Negative	1.1438
Tested_MA(7days)	1.2724
ComparisonPreDay	1.0343
ComparisonPreSpread	1.0407

From the table above, we can see that all values of VIF have dropped to an insignificant level of $VIF \leq 10$. Therefore, by the modifications of dropping identical data features, we can confirm that all independent variables in the dataset does not present multicollinearity and can be used simultaneously in our prediction models. There are other ways to remedy multicollinearity such as combining two similar datasets into a single feature. However, the combining process has to be well planned with extensive knowledge about the data itself and the interdependencies between the combining features. Hence, this paper would take the simple step of excluding features to alleviate multicollinearity.

3.2 Specification of Results

Using the pre-processed dataset, we constructed several models to predict the **Positive Rate** observed in Tokyo. Below would be a description of the used dataset in this subsection.

- Objective Value: Positive_Rate
- Features: Light-Mid_Symptoms, Severe_Symptoms, Dead, Discharged, PCR_Positive, PCR_Negative, Tested_MA(7days), ComparisonPreDay, ComparisonPreSpread (9 Features)
- Total Data Date Range: 2020/05/02-2021/12/30 (length=608)
- Train Data Date Range for 40 days prediction: 2020/05/02-2021/11/20 (length=568)
- Test Data Date Range for 40 days prediction: 2021/11/21-2021/12/30 (length=40)
- Train Data Date Range for 80 days prediction: 2020/05/02-2021/10/11 (length=528)
- Test Data Date Range for 80 days prediction: 2021/10/12-2021/12/30 (length=80)
- Train Data Date Range for 20 days prediction: 2020/05/02-2021/12/10 (length=588)
- Test Data Date Range for 20 days prediction: 2021/12/11-2021/12/30 (length=20)
- Validation split: 10% of training data

This paper would evaluate the model based on the computation of the loss function, Root Mean Squared Error (RMSE) which is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.1}$$

where y_i and $\hat{y}_i$ each denoting the actual value and prediction value respectively. The prediction results from models would be quantitatively evaluated based on the RMSE value of prediction results and qualitatively evaluated using the line plots of the predicted values. Also, in the line plot used below, the test data would be labeled in the color blue, and the predicted value in orange.

For the neural network models, this paper constructed a 5 layer network with unit size of 128, 256, 128, 64 and an output layer. The activation function used in between the layers is LeakyReLU (Maas et al., 2013) with $alpha = 0.1818$ ($a = 5.5$ in 2.19) and a dropout layer of 40% is added in between all hidden layers. The optimizer used in the models were Adam optimizer and each of the models were trained for 500 epochs with a batch size of 32. The input size which stores the amount of past values to be used for prediction is altered within the range of predictions; for 40days prediction, the input size is 14days, for 20days prediction, the input size is 7days, and for 80 days prediction, the input size is 21days. Each taking their respective input size, the neural network will use the input size data to predict one future value.

3.3 40 Days Prediction

40 day prediction is done for standard prediction. By evaluating the performances of this prediction, we are able to understand the models capability of making a general prediction.

3.3.1 Univariate Model (40 Days)

The ARMA model used in this section was ARMA(2,2) which had the lowest AIC (Akaike, 1974) within the tested range of lags ranging from 0 to 7.

Table 3.5: Univariate Prediction Performance of Each Model (Best To Worst) - 40 Days Prediction

Model	RMSE
GRU	**0.000849**
ARMA(2,2)	0.002174
LSTM	0.004307

Table 3.5 shows the result of the predictions. GRU was able to predict with a high accuracy with a RMSE value smaller than the ARMA model by 256%. On the other hand, the LSTM mdoel did not perform well in its predictions with RMSE value bigger than ARMA by 198%.

Figures 3.1, 3.2 shows the line graph of the prediction and the real data. As indicated from the RMSE, the GRU fitted very well with the actual data. Looking at the ARMA model, we could see that the prediction is static and does not accurately illustrate the slight rise of number in the end. LSTM had a spike in the first prediction but after the spike, the shape of the line plot seems to reflect the actual values. Overall, GRU model was able to accurately predict the value in a big margin compared with other models.

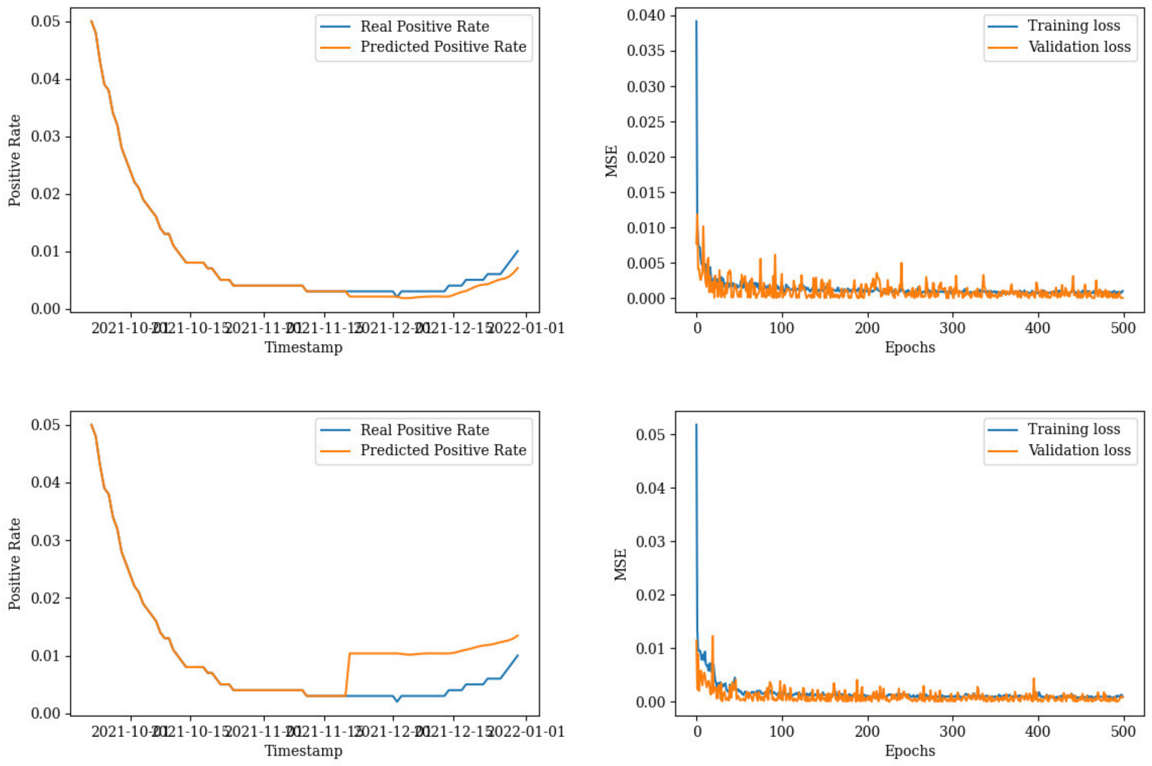


Figure 3.1: The line graph of produced prediction results and process of accuracy performance of univariate prediction of 40 days. Looking at the prediction results of GRU (top-left), we could see that the neural network is able to predict the objective value by a high accuracy. On the other hand, the prediction results of LSTM (bottom-left) shows a spike in its first prediction, but able to track the upward trend in the end. The accuracy performance of both GRU (top-right) and LSTM (bottom-right) shows that the validation loss is decreasing but oscillates in small margins. Furthermore, we can see that the variance of oscillation decreases as the epochs increase.

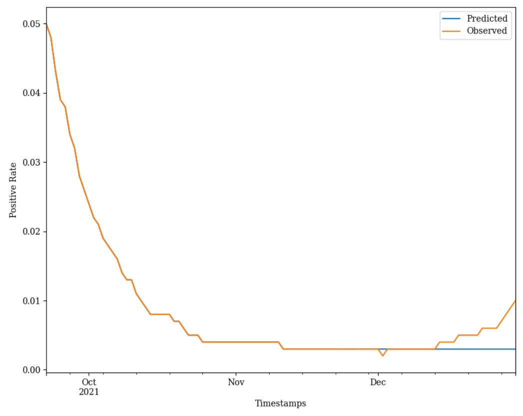


Figure 3.2: The line graph of produced 40 days prediction result from the ARMA(2,2) model. We could see that the prediction is static and is not changing its value at the end.

3.3.2 Multivariate Model (40 Days)

For the conventional model VARMA(2,3) is used. Due to the long computational time the fitting of Auto Regressive model consumes, features used in VARMA needed to be truncated. Therefore in this model, the features Positive_Rate, Discharged, PCR_Positive, and ComparisonPreSpread was used for this model since each of the features showed high correlation between the objective variable. The neural network model each used all the features in the dataset for the predictions.

Table 3.6: Multivaraite Prediction Performance of Each Model (Best To Worst) - 40 Days Prediction

Model	RMSE
GRU	**0.015485**
VARMA(2,3)	0.021116
LSTM	0.030772

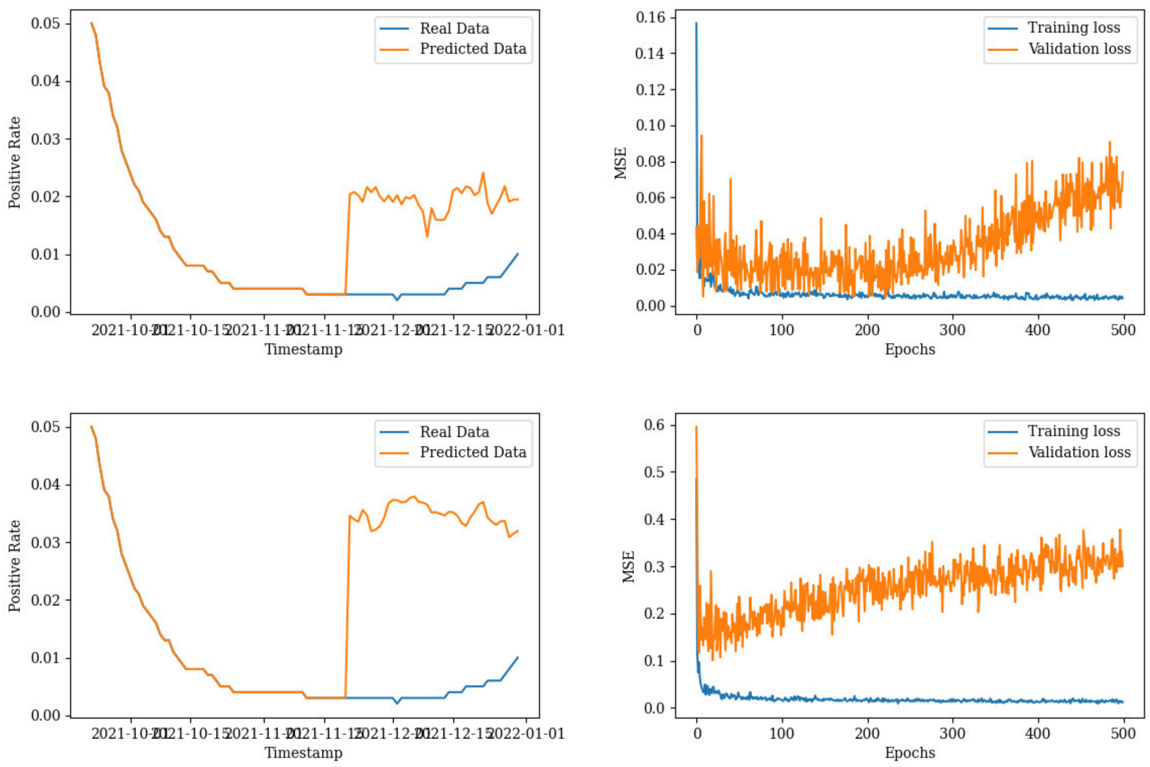


Figure 3.3: The line graph of produced prediction results and process of accuracy performance of multivariate prediction of 40 days. Looking at the prediction results of GRU (top-left) and LSTM (bottom-left) we could see that the model was not able to predict the objective value with high accuracy. They both have a very big spike in its prediction and has a fluctuation which is not observed in real data. The accuracy performance of both GRU (top-right) and LSTM (bottom-right) shows that despite the training loss is able to decrease its value the validation loss is increasing as the epochs increase. This indicates a characteristic of over-fitting and is not versatile in its predictions.

Table 3.6 shows the value of RMSE for the predicted value and the actual value. Looking at the RMSE the multivariate prediction was unsuccessful in making an accurate

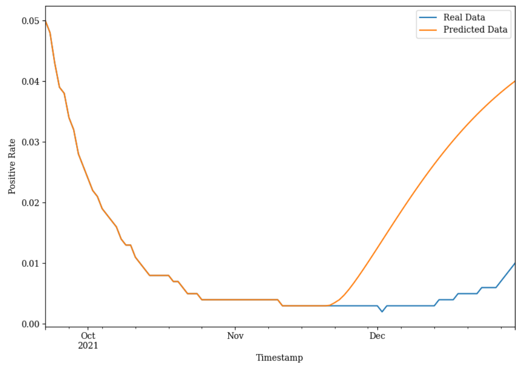


Figure 3.4: The line graph of produced 40 days prediction results from the VARMA(2,3) model. We could see that the model is able to capture the upward trend in the objective value but the margin between the real data is large. We could also see that the predicted value is smoother compared to the neural network prediction results.

prediction when compared with the prediction results of the univariate predictions. Figure 3.3 and 3.4 shows the prediction results. We can see that none of the models showed a very big spike in its prediction values. Judging from the Accuracy Performance in Figure 3.3 the neural network fitted excessively with the training data and did not perform well in the validation dataset. Therefore showing signs of overfitting in our model. Overall, compared to the results of univariate predictions, we can conclude that the increase in features used in the model have worsened the prediction results.

3.4 20 Days Prediction

20 day prediction is done to check the performances for the short term predictions. The performances in this model determines the model's capability of interpreting long sequential patterns in the training data and extract the slight upward trend existing in the end of the data.

3.4.1 Univariate Model (20 Days)

For the conventional model of ARMA for univariate prediction of 20days, ARMA(2,2) model will be used.

Table 3.7: Univariate Prediction Performance of Each Model (Best To Worst) - 20 Days Prediction

Model	RMSE
LSTM	**0.002800**
ARMA(2,2)	0.003043
GRU	0.003290

Table 3.7 shows the performance of the models. For the prediction of 20 days, LSTM performed better than the ARMA model and also the GRU model. However, judging from the small difference in RMSE, we cannot indicate a significant difference in its performances.

Figure 3.5, 3.6 shows the prediction values of the corresponding models. We can see that the neural network both present a spike for the first prediction value but captures

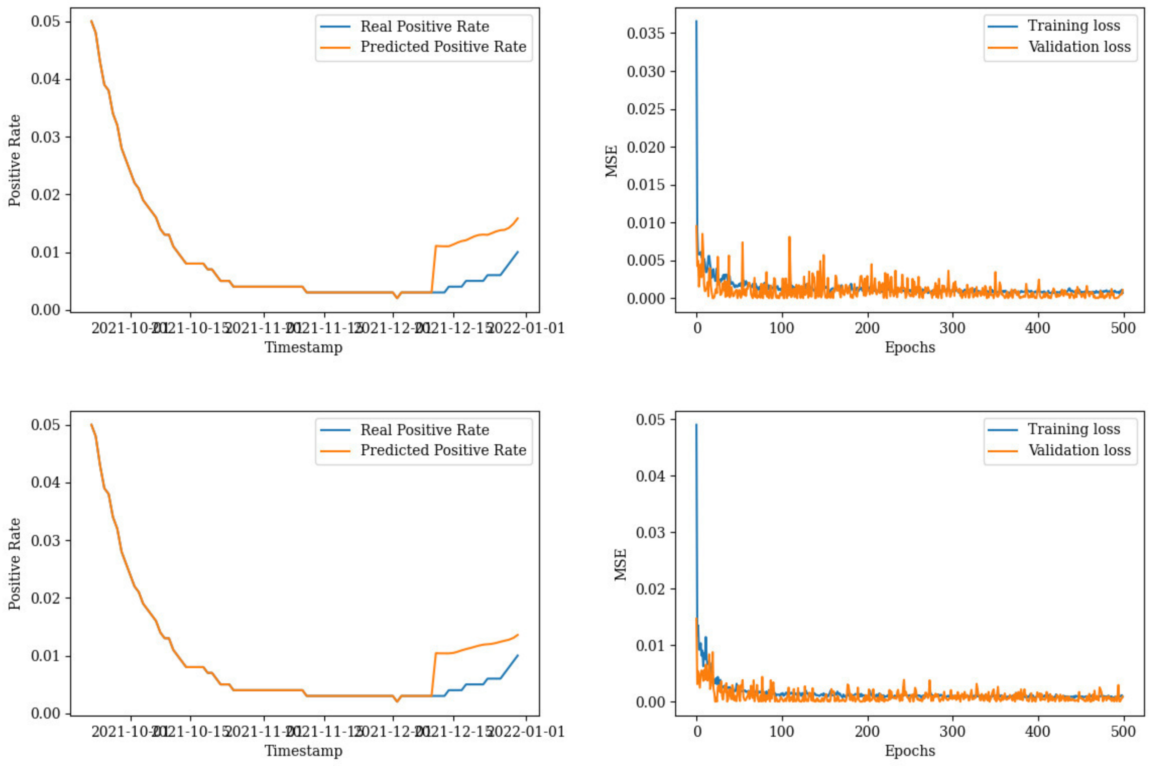


Figure 3.5: The line graph of produced prediction results and process of accuracy performance of the univariate prediction of 20 days. Looking at the prediction results of GRU (top-left) and LSTM (bottom-left) both shows a spike in its first prediction, but able to track the upward trend towards the end. The accuracy performance of both GRU (top-right) and LSTM (bottom-right) shows that the validation loss is decreasing with its training loss. In comparison however, it can be seen that the training accuracy is decreasing more in the LSTM model.

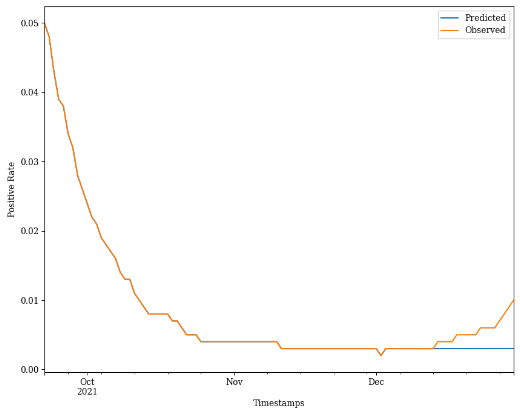


Figure 3.6: The line graph of produced 20 days prediction results of ARMA(2,2) model. We could see that the prediction is static and is not changing its value.

the overall trend of the change of values. We can also see that the neural network model captures the upward trend in the end while the ARMA model does not present any change in its prediction value and is static. Overall, neither of the model does not demonstrate a good fit for a prediction model.

3.4.2 Multivariate Model (20 Days)

Same as the VARMA model for 40days prediction, features had to be reduced to conduct the prediction. Therefore, features Positive_Rate, Discharged, PCR_Positive, and ComparisonPreSpread were used and the appropriate lag was estimated at VARMA(7,2).

Table 3.8: Multivariate Prediction Performance of Each Model (Best To Worst) - 20 Days Prediction

Model	RMSE
VARMA(7,2)	0.005607
LSTM	0.007889
GRU	0.008213

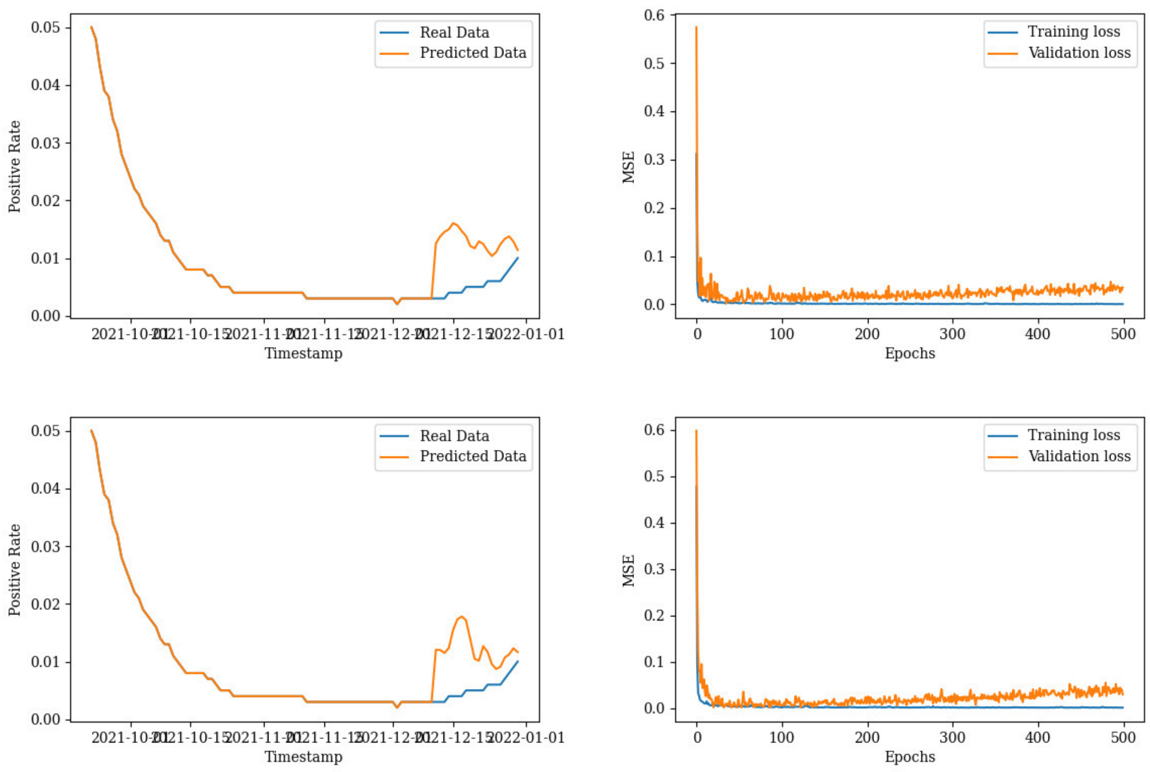


Figure 3.7: The line graph of produced prediction results and process of accuracy performance of multivariate prediction of 20 days. Looking at the prediction results of GRU (top-left) and LSTM (bottom-left) we could see that the model was not able to predict the objective value with high accuracy. They both have a very big spike in its prediction and has a fluctuation which is not observed in real data. The accuracy performance of both GRU (top-right) and LSTM (bottom-right) shows that despite the training loss is able to decrease its value the validation loss has a light increase trend as the epoch increases.

Table 3.8 shows the result of all the model performances. We could see that both neural network model was unsatisfactory of overperforming the VARMA model. However,

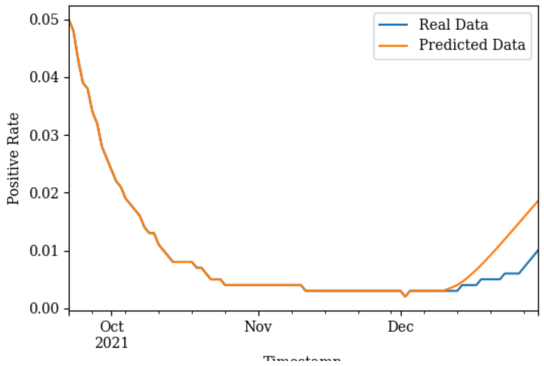


Figure 3.8: The line graph of produced 20 days prediction results of VARMA(7,2) model. We could see that the prediction has a positive trend which the ARMA model was incapable of illustrate. However, we could see that the prediction line goes above the real data.

we can see that the RMSE does not differ significantly between the models, hence an early call to say that the neural network model is ineffective in its prediction. Figures 3.7, 3.8 are the line plots for the predictions. We could first see that both neural network still produce an initial spike that has persisted throughout the other neural network models. The right figures of Accuracy Performance in Figure 3.7 suggest that the data itself can be properly used in the neural network and can very easily train the model with less epochs than the epochs we have set for this test. Another finding is that, the linear model for this 20 days prediction was able to account for the slight upward trend, the 40 day prediction model dismissed. This is arguably because the lag length is set at (7,2) which used the data from 7 days prior for the regression and has picked up on some elements that has shown an upward trend within the dataset. Thus, having the strongest performance out of these models.

Overall, the univariate prediction using LSTM model performed the strongest out of all the models and the increase in features have weakened the performance for our predictions.

3.5 80 Days Prediction

80 days prediction is conducted to evaluate the models' capability of capturing the sequential patterns from a smaller sample size and predict a long sequence. This is generally a difficult task since the model have to fully capture the underlying logic within data features which induces the number of observed positive rates.

3.5.1 Univariate Model (80 Days)

Same as the other univariate linear models, we use ARMA(2,2) for our conventional model and compare the performance for other neural network models.

Table 3.9: Univariate Prediction Performance of Each Model (Best To Worst) - 80 Days Prediction

Model	RMSE
LSTM	**0.001769**
ARMA(2,2)	0.008198
GRU	0.008385

Table 3.9 shows the evaluation for all the predictions conducted in the 3 models. We can see that the LSTM model does significantly better than the other two models. This highlights the LSTM's strength of interpreting the long term dependencies within the dataset.

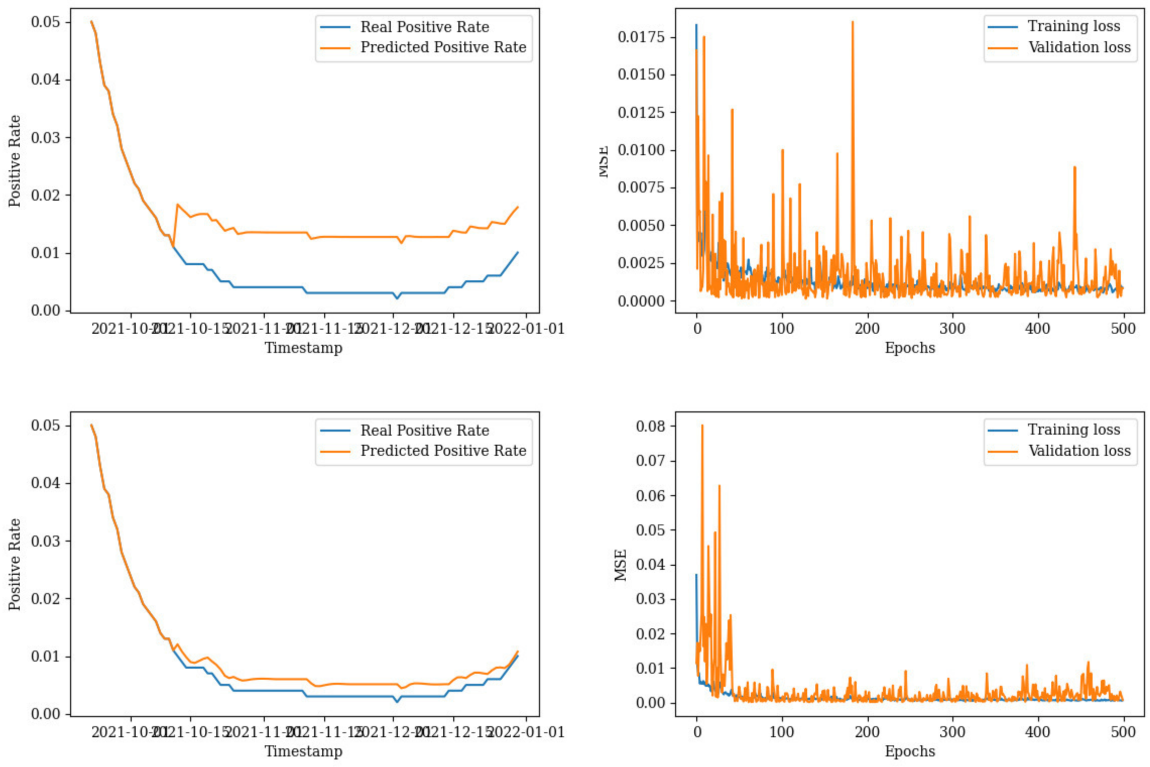


Figure 3.9: The line graph of produced prediction results and process of accuracy performance of the univariate prediction of 80 days. Looking at the prediction results of LSTM (bottom-left), the LSTM model was able to predict in a high accuracy with the same kind of dents. The GRU (top-left) prediction shows a spike in its first prediction, but able to illustrate the dents existing in the real data. From the accuracy performance of GRU (top-right) we could see that the training loss decreases as the epochs increase, and the validation loss is fluctuating within a small region. The accuracy performance of LSTM (bottom-right) shows that both training loss and validation loss is decreasing as the epochs increase. From both accuracy performances we can see that the neural network model is capable of being trained easily.

Figures 3.9, 3.10 shows the prediction done by the models. Looking the Accuracy Performance figure of the bottom-right figure in Figure 3.9, the LSTM as able to predict roughly a similar curve with the same kind of dents at the corresponding time. In the context of dents, the prediction value of GRU (top-left) in Figure 3.9 is also able to illustrate the dents even though the initial spike has lifted the prediction one step higher than the observed positive rates. Hence we can conclude that neural network models are capable of capturing the long term dependencies in time series data and predicting long sequence of future values.

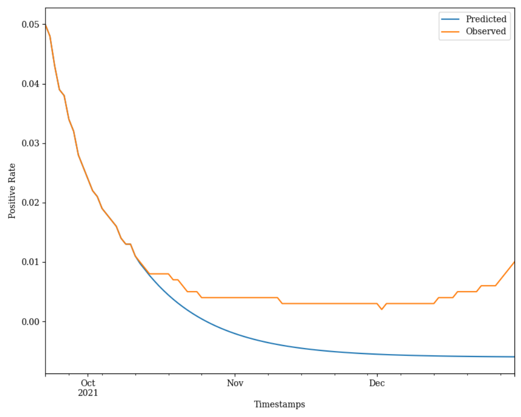


Figure 3.10: The line graph of produced 80 days prediction results of ARMA(2,2) model. The prediction line graph goes below the real data and has a continuing downwards trend.

3.5.2 Multivariate Model (80 Days)

We now conduct the 80 days prediction using multiple features. For the conventional method VARMA we use the lag length (2,2) with the truncated columns shown in the other VARMA models.

Table 3.10: Multivariate Prediction Performance of Each Model (Best To Worst) - 20 Days Prediction

Model	RMSE
VARMA(2,2)	**0.036689**
GRU	0.168341
LSTM	0.179448

Similarly we compile the result of RMSE for all the models and get Table 3.10. We see that the neural network models performs significantly worse compared to the VARMA(2,2) model. The RMSE of GRU and LSTM are 500% more than the conventional model, thus we can conclude that the neural network was ineffective in its prediction using multiple variables. The prediction line figures in the left side of Figure 3.11, 3.12 are the corresponding results for our predictions. We can see that the neural network models both initiates a spike and forms a mountain like shape for its prediction, which was not found in the univariate prediction models. Added to the fact that both neural network models were unable to decrease the error term in the validation data, shown in the accuracy performance in Figure 3.11, it can be concluded that the increase in features created too much noise for the neural network to be trained appropriately.

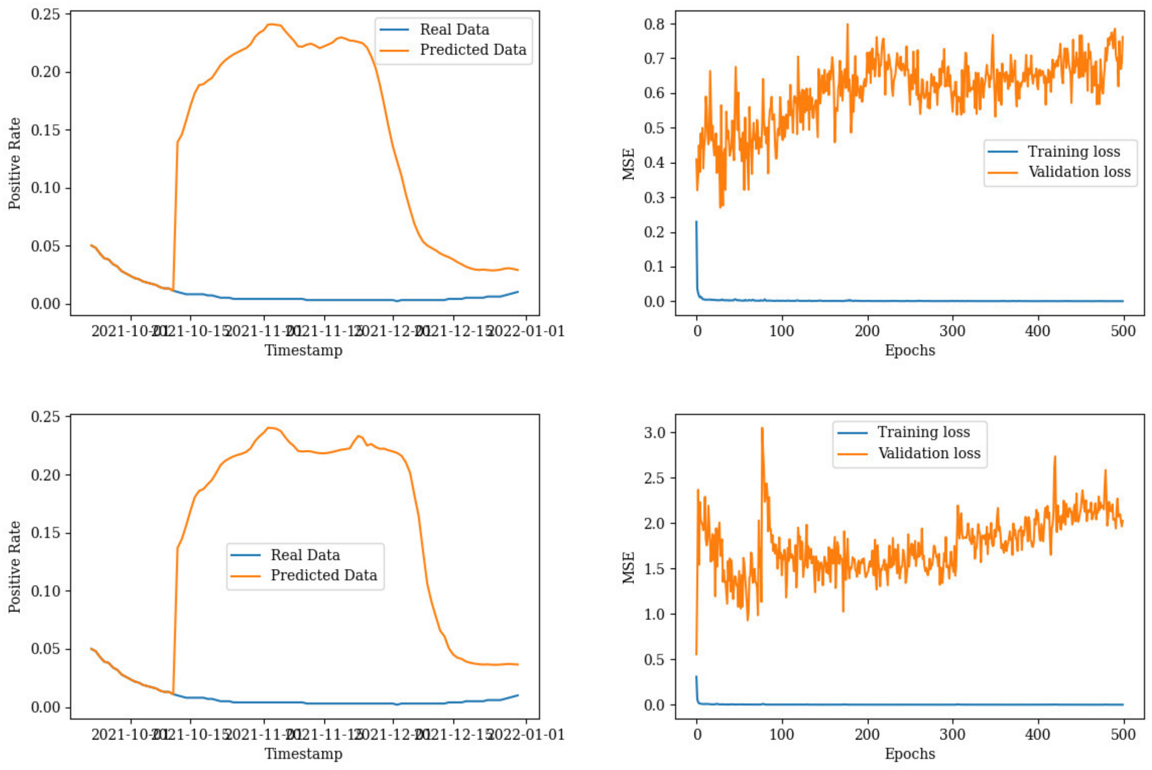


Figure 3.11: The line graph of produced prediction results and process of accuracy performance of multivariate prediction of 80 days. Looking at the prediction results of GRU (top-left) and LSTM (bottom-left) we could see that the model is incapable of predicting the objective values. They both have a mountain like shape in its predictions and does not look feasible. The accuracy performance of both GRU (top-right) and LSTM (bottom-right) shows that the training does not go well in both models and the validation loss is very big in both models.

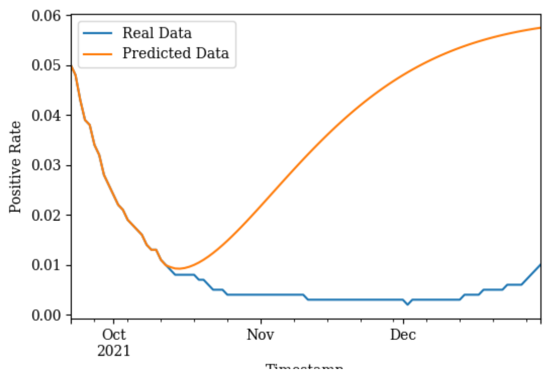


Figure 3.12: The line graph of produced 80 days prediction results of VARMA(2,2) model. The prediction line graph goes above the real data and the trend of value does not reflect the actual data.

3.6 Extensive Experiments

As shown in Sections 3.3, 3.4, 3.5 the case where the increase in features used in the neural network causing a negative affect in its prediction process has been apparent. Especially for the 80 days prediction, the Univariate model presented promising predictions as shown in the prediction line graph of LSTM model in Figure 3.9 while the multivariate model was unable to show the same strong performance when it became a question of multivariate problem. For the pursuit of a better prediction result, this paper will conduct extensive experiments by adjusting the data features.

3.6.1 Dropping Features According to the correlations

For the multivariate predictions using neural network, we used all 9 data features to train the objective value. However, in reality there are some features that does not correlate too much with the objective value. Visualizing the correlation between the objective value and other feature values, we check the heatmap of our dataset.

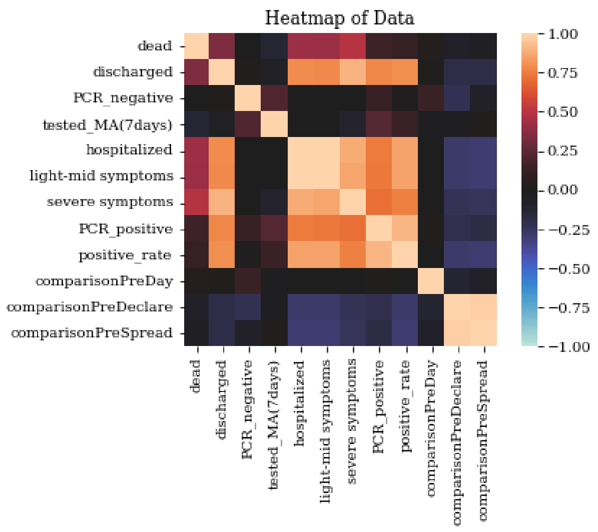


Figure 3.13: Heatmap of the Whole Dataset: The section assosiated with the objective value "Positive_Rates" do not demonstrate strong correlation with most features

From Figure 3.13 we can see that there are only a few columns which shows strong correlation between the objective value. We will now define a new model which only includes the data features of Positive_Rate, Discharged, PCR_Positive, and ComparisonPreSpread, which are the same set of features used in the VARMA models. Naming these models LSTM_Mod, GRU_Mod, we compare the RMSE with the previous results.

Table 3.11: Comparison of Multivariate Prediction Performance

Model	40 Days	20 Days	80 Days
VARMA	0.021116	0.005607	**0.036689**
LSTM	0.030772	0.007889	0.179448
GRU	0.015485	0.008213	0.168341
LSTM_Mod	**0.008431**	**0.005089**	0.175739
GRU_Mod	**0.004324**	**0.003951**	0.170854

As demonstrated in Table 3.11, the modification we made, in which we decreased the number of features, have had a positive affect in terms of decreasing its RMSE for all neural

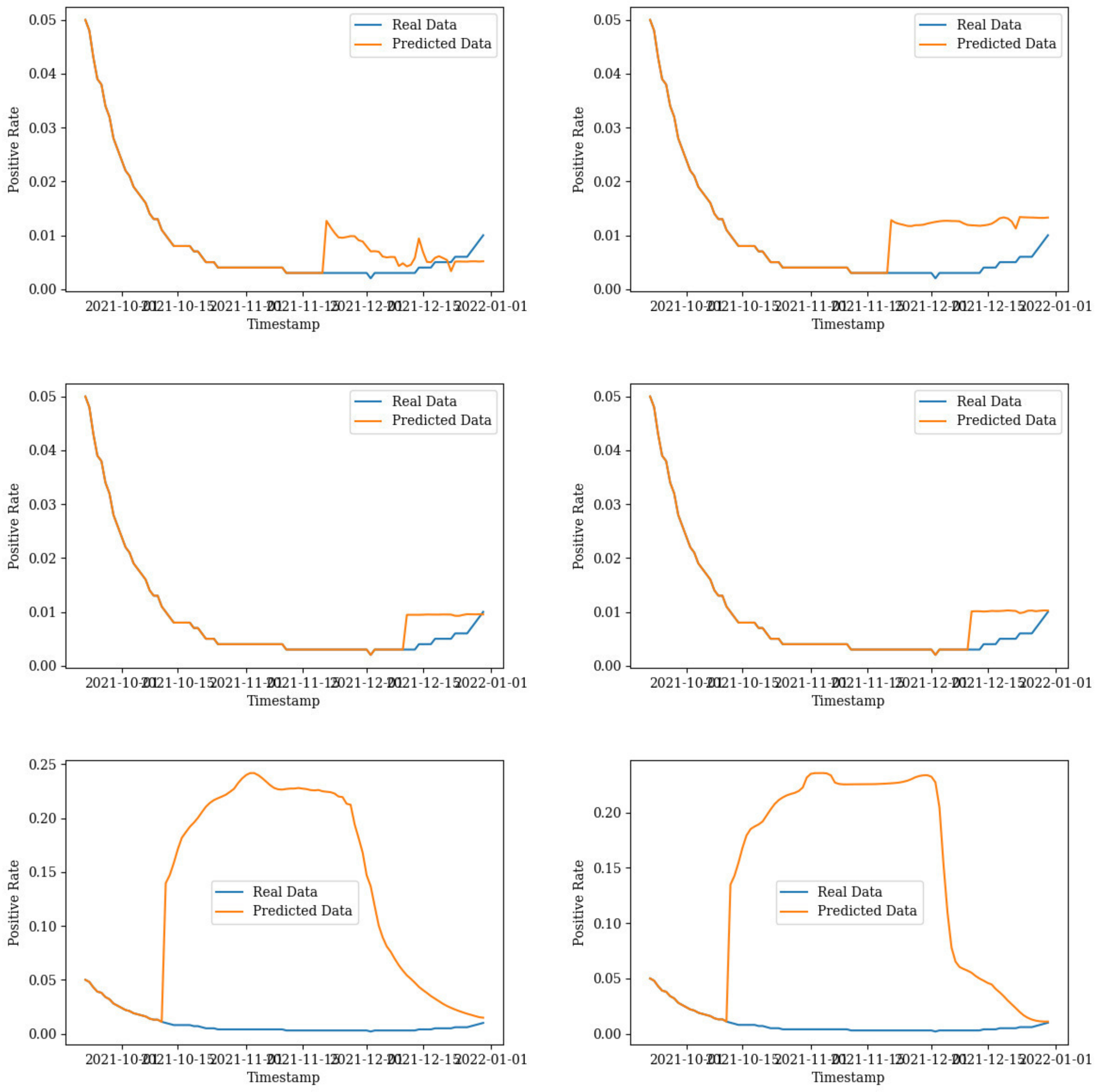


Figure 3.14: The line graph of prediction results produced from the modified neural network models using less features. The left side shows the prediction results from the GRU and the right side the LSTM. The top row are the 40 days prediction and both were not able to remedy the initial spike but have a downward trend which gets closer to the real data than the un-modified models. The middle row are the 20 days prediction which has an initial spike but a static value after the spike. The bottom row shows the results for the 80 days prediction and we can see that the results have a mountain shape graph but the values matches at the end for LSTM (bottom-right).

network models. The less complex dataset improved their performance compared to the result without any modifications and we can conclude that extracting only the correlated features (with the assumption that all data features do not demonstrate multicollinearity) can expect improvement in our model. It still is not to the level where it outperforms the univariate prediction models but has the potential to become so.

3.6.2 Dropping Human Foot Traffic Data

We have intentionally left out **ComparisonPreSpread** feature so that the model incorporates the data about how much people went outside. Since this data feature is the only data which demonstrates the human activity, which is said that it perpetuates the spread of the novel COVID-19 Virus. By comparing the prediction results with the LSTM_Mod and GRU_Mod model, we can determine whether the data feature of ComparisonPreSpread is a significant factor in our model or not. Naming the new model LSTM_Mod_NoHuman and GRU_Mod_NoHuman, this paper would compare the results with the updated model, LSTM_Mod and GRU_Mod made in Section 3.6.1..

Table 3.12: Comparison of Multivariate Prediction Performance Before and After Excluding Human Foot Data

Model	40 Days	20 Days	80 Days
VARMA	0.021116	0.005607	**0.036689**
LSTM_Mod	0.008431	**0.005089**	0.175739
GRU_Mod	**0.004324**	0.003951	0.170854
LSTM_Mod_NoHuman	**0.005089**	0.006547	0.177712
GRU_Mod_NoHuman	0.0051283	**0.003397**	0.169425

Table 3.12 shows the result for our experimentation. Looking at the LSTM model, the NoHuman version has decreased the error term in the 40 days prediction. Also for the GRU model, the 20 days prediction presents a smaller RMSE than the Mod model. Looking at the Figures in general, we could see that the **ComparisonPreSpread** feature held the value to a lower level. As it is not the case where all models performing better without human foot traffic data, and the small difference of RMSE, we cannot conclude that the data of human activity is insignificant. However, it can be seen from the mix of the features used in the model, human activity data is somewhat weak in its deterministic feature for the prediction of **Positive_Rate**.

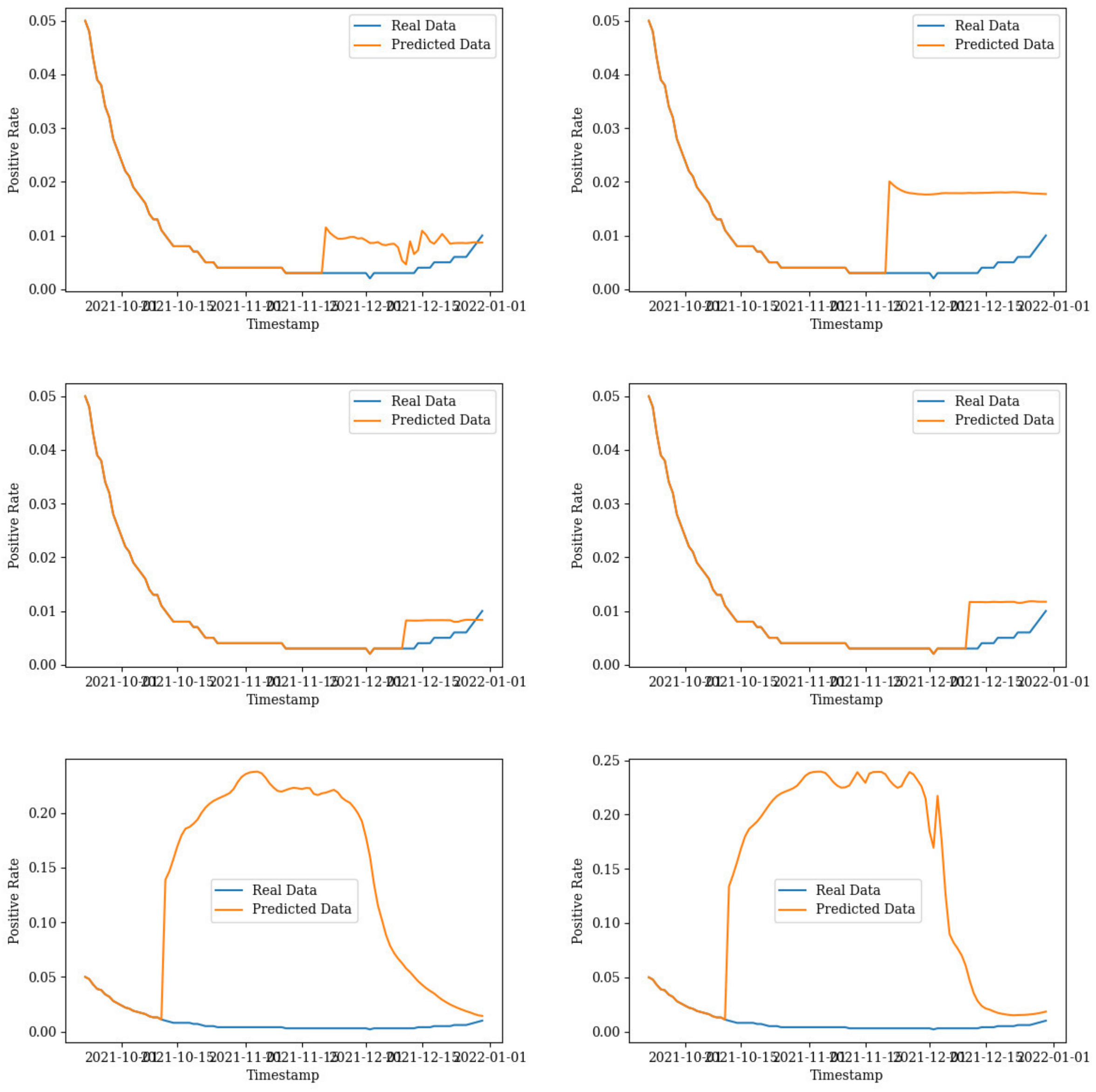


Figure 3.15: The line graph of prediction results produced from the neural network models without human activity features. The left side shows the prediction results from the GRU and the right side the LSTM. The top row are the 40 days prediction and both were not able to remedy the initial spike. The middle row are the 20 days prediction and resembles the shape of the univariate prediction curve. The bottom row shows the results for the 80 days prediction and we can see that the results still have a mountain shape graph.

4 Discussion And Future Works

4.1 Discussion

4.1.1 The "Initial spike" Issue

In the result shown in Section 3, some neural network model was able to produce an "intuitive" prediction model such as the univariate prediction of 80 days using LSTM model in Figure 3.9, and the univariate prediction of 40 days using GRU model in Figure 3.1. However, most of the prediction plots included a spike for the first value of prediction, which looks very odd to say the least. The factors which causes this issue is still a mystery since, the used dataset and the methodology does not differ between other iterations (including epochs and batch number). Therefore, cracking down on the reason behind the calculation process which forms this "mountain" will be a big obstacle to make the result of the prediction more practical.

4.1.2 The Multiple Feature Issue

In this paper, we delved into two prediction processes, which are Univariate prediction and Multivariate Prediction. Comparing the results from different processes, the Univariate prediction almost always had a lower RMSE than the Multivariate prediction. This seems to be intuitively strange, since we expect a better result when we have more information to refer to. There is a possibility where some interdependency should be referred to before rendering all the dataset into a single model. In this paper, the selection and inclusion of features were done solely on the correlation values. A more careful consideration on what features should be included in the model should be done.

4.1.3 Lack of Consideration of Layers

In this paper, LSTM and GRU, which are one of the most complicated neural network models are being used. Therefore, there should be more careful consideration when adding layers and choosing the hyperparamters since it is a very sensitive system. The model was shaped based on separate information found on the internet, which lacks reliability on the model itself. Therefore, a more careful approach of selecting hyperparameters are needed.

4.2 Future Works

COVID-19 prediction is a harsh task to start with. With multiple variables intertwining with each other and some unobservable factors which contributes to the spread of the virus. Therefore, from a data science approach, it is imperative to go deep down on the data generation process to extract and evaluate relevant datasets that effectively tackles the research question at hand. Therefore, more attention is needed on the Public Health terminology and become aware of how the numbers are generated. This point also connects to the discussion point 4.1.2 where the interdependency between multiple variables should be considered when making a predictive model.

This paper was able to construct the advanced deep learning methods using the Keras library (Chollet et al., 2015) which is enriched with handy deep learning methods. However, there are some drawbacks to the blackbox nature of this library, where I cannot review the specific calculation affecting my output. To be specific, my result have produced many "mountain-shaped" results which was counterintuitive, but had no way to figure out how the data was being computed. Therefore, to excel more on the development of accurate prediction models, there is a need for me to start from the base and see how the parameters are being computed.

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.

Alin, A. (2010). Multicollinearity. *WIREs Computational Statistics*, 2(3):370–374.

Allen, M. P. (1997). *The problem of multicollinearity.* Springer US.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bottou, L. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent. In Lechevallier, Y. and Saporta, G., editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg. Physica-Verlag HD.

Cahuantzi, R., Chen, X., and Güttel, S. (2021). A comparison of LSTM and GRU networks for learning symbolic sequences. *CoRR*, abs/2107.02248.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches.

Chollet, F. et al. (2015). Keras. https://keras.io.

Dickey, D. and Fuller, W. (1979). Distribution of the estimators for autoregressive time series with a unit root. *JASA. Journal of the American Statistical Association*, 74.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

Gers, F. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194 vol.3.

Government, T. M. (2022a). Positive rate and number of people tested for novel coronavirus infection in tokyo - tokyo metropolitan government open data catalog site.

Government, T. M. (2022b). Situation regarding coronavirus infected patients in tokyo - tokyo metropolitan government open data catalog site.

Granger, C. and Newbold, P. (1974). Spurious regressions in econometrics. *North-Holland Publishing Company*, 2(Journal of Econometrics):111–120.

Greene, W. H. (2003). *Econometric analysis / William H. Greene.* Prentice Hall, Upper Saddle River, N.J., 5th ed. edition.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Kanagawa Prefecture Official, K. U. o. H. S. (2022). Simulation of the number of severe cases using a new covid-19 prediction model.

Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Lütkepohl, H. (2006). Chapter 6 forecasting with varma models. In Elliott, G., Granger, C., and Timmermann, A., editors, *Handbook of Economic Forecasting*, volume 1 of *Handbook of Economic Forecasting*, pages 287–325. Elsevier.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Narisetty, N. N. (2020). Chapter 4 - bayesian model selection for high-dimensional data. In Srinivasa Rao, A. S. and Rao, C., editors, *Principles and Methods for Data Science*, volume 43 of *Handbook of Statistics*, pages 207–248. Elsevier.

of Japan, C. S. (2022). Response to the novel coronavirus infection (covid-19) office of infectious disease control and prevention.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17.

Qiu, X., Zhang, L., Ren, Y., Suganthan, P. N., and Amaratunga, G. (2014). Ensemble deep learning for regression and time series forecasting. In *2014 IEEE Symposium on Computational Intelligence in Ensemble Learning (CIEL)*, pages 1–6.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

Saito, K. (2016). *Deep Learning from Scratch: Theories and Practice of Deep Learning using Python*. O'Reilly Japan.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464.

Seabold, S. and Perktold, J. (2010a). statsmodels.stats.outliers_influence — statsmodels.

Seabold, S. and Perktold, J. (2010b). statsmodels.tsa.stattools.adfuller — statsmodels.

Shen, G., Tan, Q., Zhang, H., Zeng, P., and Xu, J. (2018). Deep learning with gated recurrent unit networks for financial sequence predictions. *Procedia Computer Science*, 131:895–903. Recent Advancement in Information and Communication Technology:.

Siami-Namini, Neda Tavakoli, A. S. N. (2018). A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401.

Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853.

Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270.

Zhang, G. (2003). Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175.

Appendices

A Data Pre-processing

A.1 Stationarity

```python
# Import modules
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import adfuller

# Load csv
df = pd.read_csv('df_master.csv')

# Define ADF test for multiple significance levels
def ad_test(df, col):
  name = str(col)
  dataset = df[col]
  result = True
  dftest = adfuller(dataset, autolag = 'AIC')
  sig_level = ['10%', '5%', '1%']
  for sig in sig_level:
    if dftest[0] > dftest[4][sig]:
      reject = sig
      result = False
      break
    else:
      pass
  if result == True:
    result_text = 'Pass'
  else:
    result_text = 'Reject at ' + sig
  return name + '&' + "{:.4f}".format(dftest[0]) + '&' + "{:.4f}".format(dftest[4]['1%'])
        + '&' + "{:.4f}".format(dftest[4]['5%']) + '&' + "{:.4f}".format(dftest[4]['10%'])
        + '&' + result_text + "\\" +"\n"

# Run the function and store the results in a txt file
file = open("ADF_tabular.txt","w")
for col in df_norm.columns:
  file.writelines(ad_test(df_norm, col))
file.close()

# Take the First Difference for non-stationary processes
## Take the features of non-stationary process
non_stationary_feat = ['hospitalized', 'light-mid_symptoms', 'severe_symptoms', 'dead',
                        'PCR_negative', 'tested_MA(7days)']
df_copy = df[non_stationary_feat]

## Take the first difference
df_diff = df_copy.diff()
df_diff = df_diff.dropna()
```

```

## Redo the ADF test and store the results to a separate file
file2 = open("ADF_tabular2.txt","w")
for col in df_diff.columns:
  file2.writelines(ad_test(df_diff, col))
file2.close()

# Make a single dataframe with all stationary features
stat_cols = []
for col in df.columns:
  if col not in non_stationary_feat:
    stat_cols.append(col)
df_stat = df[stat_cols]
newdata = pd.merge(df_stat, df_diff, on = 'date')
newdata = newdata.reindex(columns = df.columns)
```

A.2 Multicollinearity

```
# Import modules
from statsmodels.stats.outliers_influence import variance_inflation_factor as VIF

# Prepare dataframe
newdf = newdata.copy()
## Drop the objective function
newdf = newdata.drop(columns = {'positive_rate'})

# Define the VIF generator from a given dataframe
def VIF_operator(df):
  X_variables = df.copy()
  vif_data = pd.DataFrame()
  vif_data["feature"] = df.columns
  vif_data["VIF"] = [VIF(X_variables.values, i) for i in range(len(X_variables.columns))]
  return vif_data

# Run the function for dataframe
vif1 = VIF_operator(newdf)

# Store the results in txt file
file2 = open('VIF_result1.txt', 'w')
for i in range(11):
  text = str(vif1.values[i][0]) + ' & ' + str(vif1.values[i][1]) + " \\" + "\\" + '\n'
  file2.write(text)
file2.close()

# From the results drop identical features
newdata1 = newdf.copy()
newdata1 = newdata1.drop(columns = ['hospitalized', 'comparisonPreDeclare'])
newdata1.index = pd.to_datetime(newdata1.index)

# Recalculate VIF
vif2 = VIF_operator(newdata1)
```

```python

#Store the results in txt file
file3 = open('VIF_result2.txt', 'w')
for i in range(9):
  text = str(vif2.values[i][0]) + ' & ' + str(vif2.values[i][1]) + " \\" + "\\" + '\n'
  file3.write(text)
file3.close()

# Store the Pre-processed data set to a csv file
## Get the dataframe which includes the objective function
positive = df.loc[df['positive_rate', 'comparisonPreDeclare']]
df_master_final = pd.merge(positive, newdata1, on = 'date')
## Drop unnecessary column
df_master_final = df_master_final.drop(columns={'comparisonPreDeclare'})
## store the result to "df_master_final.csv"
df_master_final.to_csv('df_master_final.csv')
```

B Conventional Model

B.1 ARMA model

```python
# Import necessary modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pmdarima import auto_arima
import warnings
from statsmodels.tsa.arima.model import ARIMA
from copy import deepcopy as dc
from sklearn.metrics import mean_squared_error
from math import sqrt
plt.rcParams["font.family"] = "serif"
% matplotlib inline
warnings.filterwarnings('ignore')

# Read data
df = pd.read_csv('df_master_final.csv', index_col='date')
df.index = pd.to_datetime(df.index)

# Calculate the lags using 'auto_arima' module
## Optimal lags will be calculated which minimizes the AIC
stepwise_fit = auto_arima(df['Positive\_Rate'], trace=True, suppress_warnings = True)
stepwise_fit.summary()

# 40 days prediction
## Split dataset
train40 = df.iloc[:-40]
test40 = df.iloc[-40:]
model = ARIMA(train40['Positive\_Rate'], order = (2, 1, 2))
model = model.fit()
```

```python

## Make a prediction and store to a dataframe
start = len(train40)
end  = len(df)
pred = model.predict(start=start, end=end-1, typ='levels')
pred.index = df.index[start:end+1]
df_pred = pd.DataFrame(pred)
df_pred = df_pred.rename(columns={'predicted_mean': 'Positive\_Rate'})

# Make a line graph of the prediction
## Prepare the dataframes
df_plot = df[df.columns[0]]
df_pred_prev = pd.DataFrame(df_plot[-100:-40])
df_plot = pd.DataFrame(df_plot[-100:])
df_pred_plot = pd.concat([df_pred_prev, df_pred])

# Caclate the RMSE
rmse = sqrt(mean_squared_error(df_pred, test40['Positive\_Rate']))

## Generate graph
ax = df_plot.plot(figsize=(10,8), legend=True)
df_pred_plot.plot(ax=ax, figsize=(10,8), legend=True)
plt.xlabel('Timestamps')
plt.ylabel('Positive Rate')
plt.legend(['Predicted','Observed'])
### save figure
plt.savefig('ARIMA(2,1,2)_40Days.png', dpi=100)
```

B.2 VARMA model

```python
# Import modules
from statsmodels.tsa.statespace.varmax import VARMAX
from pmdarima import auto_arima
from sklearn import metrics
from timeit import default_timer as timer
import warnings
warnings.filterwarnings("ignore")

# Prepare the used dataframe
df1 = pd.read_csv('df_master_final.csv', index_col='date')
df1.index = pd.to_datetime(df1.index)
data = df1.copy()
## Drop some columns to truncate the calculation time necessary for VARMA modelling
data = data.drop(columns={'Dead', 'PCR_negative', 'ComparisonPreDay', 'Tested\_MA(7days)',
                          'Light-Mid\_Symptoms', 'Severe\_Symptoms'})]

# Define a function which shows the evaluation metric
def timeseries_evaluation_metrics_func(y_true, y_pred):
    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    print('Evaluation metric results:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(y_true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error(y_true, y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',end='\n\n')

# Define a function which takes the inverse of a given column in a dataframe
def inverse_diff(actual_df, pred_df):
    df_res = pred_df.copy()
    columns = actual_df.columns
    for col in columns:
        df_res[str(col)+'_1st_inv_diff'] = actual_df[col].iloc[-1] + df_res[str(col)].cumsum()
    return df_res

# 40Days Prediction
X = data.copy()
train, test = X[0:-40], X[-40:]
cols = list(train.columns)

## Get the lag length by calculating the AIC for all the used features
pq = []
for name, column in train[['Positive\_Rate', 'Discharged',
                           'PCR\_Positive','ComparisonPreSpread']].iteritems():
    print(f'Searching order of p and q for : {name}')
    stepwise_model = auto_arima(train[name],start_p=1, start_q=1,max_p=10, max_q=10, seasonal=False,
        trace=True,error_action='ignore',suppress_warnings=True, stepwise=True,maxiter=1000)
    parameter = stepwise_model.get_params().get('order')
    print(f'optimal order for:{name} is: {parameter} \n\n')
    pq.append(stepwise_model.get_params().get('order'))

## VARMA modelling for the calculated lags and show the RMSE for the predicted results
df_results_moni = pd.DataFrame()
for i in pq:
    if i[0]== 0 and i[2] ==0:
        pass
    else:
        print(f' Running for {i}')
        model = VARMAX(train[['Positive\_Rate', 'Discharged', 'PCR\_Positive',
                              'ComparisonPreSpread']], order=(i[0],i[2])).fit(disp=False)
        result = model.forecast(steps = 40)
        inv_res = inverse_diff(data[['Positive\_Rate', 'Discharged', 'PCR\_Positive',
                                     'ComparisonPreSpread']] , result)
        rmse = np.sqrt(metrics.mean_squared_error(test['Positive\_Rate'], inv_res['Positive\_Rate']))
        df_results_moni = df_results_moni.append({'p': i[0], 'q': i[2],
                                                  'RMSE Positive Rate':rmse}, ignore_index=True)

## Sort the models in descending order according to the RMSE
df_results_moni.sort_values(by = ['RMSE Positive Rate'])

## We use the VARMA model which gives the least RMSE value
model = VARMAX(train[['Positive\_Rate', 'Discharged', 'PCR\_Positive',
                        'ComparisonPreSpread']], order=(2,3)).fit( disp=False)
result = model.forecast(steps = 40)
res = inverse_diff(data[['Positive\_Rate', 'Discharged', 'PCR\_Positive','ComparisonPreSpread']],result)
df_pred = pd.DataFrame(res['Positive\_Rate'])
original = pd.DataFrame(df1['Positive\_Rate'][-100:])
## Get the RMSE of the predicted values
rmse = sqrt(mean_squared_error(df_pred, original[-40:]))
## Prepare the dataframe to plot
original_pred = pd.DataFrame(df1['Positive\_Rate'][-100:-40])
df_pred_plot = pd.concat([original_pred, df_pred])

## plot the prediction data
ax = original.plot(legend=True)
df_pred_plot.plot(ax = ax, legend=True)
plt.legend(['Real Data', 'Predicted Data'])
plt.xlabel('Timestamp')
plt.ylabel('Positive Rate')
### save the figure
plt.savefig('VARMA(2,3)_40Days.png', dpi=100)
```

C Neural Networks

C.1 Univariate Prediction

```
# Importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, LeakyReLU
from tensorflow.keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error

# Preparing data
df = pd.read_csv('df_master_final.csv', index_col='date')
df.index = pd.to_datetime(df.index)

# Some functions to help out with
def plot_predictions(test, predicted, name):
    figure_name = str(name)+'.png'
    plt.plot(test,label='Real Positive Rate')
    plt.plot(predicted,label='Predicted Positive Rate')
    plt.xlabel('Timestamp')
    plt.ylabel('Positive Rate')
    plt.legend()
    plt.savefig(figure_name, dpi=100)

def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))

# LSTM 40 Days

## Preparing data
### Splitting data
training_set = np.array(df['Positive\_Rate'][:-40])
training_set = training_set.reshape(-1, 1)
test_set = np.array(df['Positive\_Rate'][-40:])
test_set = test_set.reshape(-1, 1)
limit = len(training_set)
### Scaling the training set
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
### We create a data structure with 15 timesteps and 1 output
X_train = []
y_train = []
memory_days = 15
for i in range(memory_days, limit):
    X_train.append(training_set_scaled[i-10:i,0])
    y_train.append(training_set_scaled[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
### Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))

## The LSTM architecture
regressor40 = Sequential()
### First LSTM layer with Dropout regularisation
regressor40.add(LSTM(units=128, return_sequences=True, input_shape=(X_train.shape[1],1)))
regressor40.add(LeakyReLU(alpha=0.181818))
regressor40.add(Dropout(0.4))
### Second LSTM layer
regressor40.add(LSTM(units=256, return_sequences=True))
regressor40.add(LeakyReLU(alpha=0.181818))
regressor40.add(Dropout(0.4))
### Third LSTM layer
regressor40.add(LSTM(units=128, return_sequences=True))
regressor40.add(LeakyReLU(alpha=0.181818))
regressor40.add(Dropout(0.4))
### Fourth LSTM layer
regressor40.add(LSTM(units=64))
regressor40.add(LeakyReLU(alpha=0.181818))
regressor40.add(Dropout(0.4))
### The output layer
regressor40.add(Dense(units=1))
### Compiling the RNN
regressor40.compile(optimizer='adam',loss='mean_squared_error')
### Fitting to the training set
history40 = regressor40.fit(X_train,y_train,epochs=500,batch_size=32, validation_split=0.1)

## Plot the accuracy preformance results
```

```
plt.plot(history40.history['loss'], label='Training loss')
plt.plot(history40.history['val_loss'], label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.savefig('LSTM_Uni_40Days_Training.png', dpi=100)

## Prepare prediction
dataset_total = pd.concat((df[:-40]['Positive\_Rate'],df[-40:]['Positive\_Rate']),axis=0)
inputs = dataset_total[len(dataset_total)- len(test_set) - memory_days:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(memory_days,40+memory_days):
    X_test.append(inputs[i-10:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1], 1))

## Store the prediction results to a dataframe
predicted_positive_rate = regressor40.predict(X_test)
predicted_positive_rate = sc.inverse_transform(predicted_positive_rate)
pred = predicted_positive_rate.copy()
pred.reshape(40)
date = pd.date_range('2021-11-21', '2021-12-30')
df_pred = pd.DataFrame(pred)
df_pred.index = date
df_pred.index.name = 'date'
df_pred = df_pred.rename(columns={'[0]':'Predicted_Positive\_Rate'})

## Evaluating our LSTM model
test_test = pd.DataFrame(df['Positive\_Rate'][-100:])
train_test = df['Positive\_Rate'][:-40]
pred_test1 = pd.concat([train_test, df_pred])
pred_test = pred_test1[-100:]
test_array = np.array(test_test)
pred_array = np.array(pred_test)
return_rmse(test_array, pred_array)

## Visualizing the results for LSTM
plot_predictions(test_test, pred_test, 'LSTM_Uni_40Days')

# GRU 40 Days
## The GRU architecture
regressorG40 = Sequential()

### First GRU layer with Dropout regularisation
regressorG40.add(GRU(units=128, return_sequences=True, input_shape=(X_train.shape[1],1)))
regressorG40.add(LeakyReLU(alpha=0.181818))
regressorG40.add(Dropout(0.4))

### Second GRU layer
regressorG40.add(GRU(units=256, return_sequences=True))
```

```python
regressorG40.add(LeakyReLU(alpha=0.181818))
regressorG40.add(Dropout(0.4))

### Third GRU layer
regressorG40.add(GRU(units=128, return_sequences=True))
regressorG40.add(LeakyReLU(alpha=0.181818))
regressorG40.add(Dropout(0.4))

### Fourth GRU layer
regressorG40.add(GRU(units=64))
regressorG40.add(LeakyReLU(alpha=0.181818))
regressorG40.add(Dropout(0.4))

### The output layer
regressorG40.add(Dense(units=1))

### Compiling the RNN
regressorG40.compile(optimizer='adam',loss='mean_squared_error')

### Fitting to the training set
historyG40 = regressorG40.fit(X_train,y_train,epochs=500,batch_size=32, validation_split=0.1)

## Plot the accuracy preformance results
plt.plot(historyG40.history['loss'], label='Training loss')
plt.plot(historyG40.history['val_loss'], label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.savefig('GRU_Uni_40Days_Training.png', dpi=100)

## Predicting values
predicted_positive_rateG = regressorG40.predict(X_test)
predicted_positive_rateG = sc.inverse_transform(predicted_positive_rateG)
predG = predicted_positive_rateG.copy()
predG.reshape(40)
predG = pd.date_range('2021-11-21', '2021-12-30')
df_predG = pd.DataFrame(predG)
df_predG.index = date
df_predG.index.name = 'date'
df_predG = df_predG.rename(columns={'[0]':'Predicted_Positive\_Rate'})

## Evaluating our GRU model
test_test = pd.DataFrame(df['Positive\_Rate'][-100:])
train_test = df['Positive\_Rate'][:-40]
pred_test1 = pd.concat([train_test, df_predG])
pred_test = pred_test1[-100:]
test_array = np.array(test_test)
pred_array = np.array(pred_test)
return_rmse(test_array, pred_array)

## Visualizing the results for GRU
plot_predictions(test_test, pred_test, 'GRU_Uni_40Days')
```

C.2 Multivariate Prediction

```
# Import modules
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense, Dropout, LeakyReLU
from sklearn.preprocessing import StandardScaler
import seaborn as sns
plt.rcParams["font.family"] = "serif"
% matplotlib inline

# Prepare dataframes
df1 = pd.read_csv(path +'df_master_final.csv', index_col='date')
df1.index = pd.to_datetime(df1.index)
df = df1.copy()
## Drop if modification are made
#df = df.drop(columns={'Dead', 'PCR_negative', 'ComparisonPreDay', 'Tested\_MA(7days)',
                      #'Light-Mid\_Symptoms', 'ComparisonPreSpread', 'Severe\_Symptoms'})

# 40 Days Prediction LSTM (Multivariate)
## Split data
df_training = df[:-40].copy()
df_master = df.copy()

## Separate dates for future plotting
train_dates = pd.DataFrame(df_training.index)
predict_period_dates = pd.DataFrame(df_master[-40:].index)

## Variables for training
cols = list(df_master)[0:10]

## Plot the used data features
df_for_training = df_training[cols].astype(float)
df_for_plot=df_for_training.tail(100)
df_for_plot.plot.line()

## normalize the dataset
scaler = StandardScaler()
scaler = scaler.fit(df_for_training)
df_for_training_scaled = scaler.transform(df_for_training)

## Prepare training data
trainX = []
trainY = []
```

```python
### Number of days we look into the future and the past days we use to predict the future
n_future = 1
n_past = 14
### Reformat input data into a shape
for i in range(n_past, len(df_for_training_scaled) - n_future +1):
    trainX.append(df_for_training_scaled[i - n_past:i, 0:df_for_training.shape[1]])
    trainY.append(df_for_training_scaled[i + n_future - 1:i + n_future, 0])
trainX, trainY = np.array(trainX), np.array(trainY)

## define the LSTM model
model40 = Sequential()

### First Layer
model40.add(LSTM(128, input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model40.add(LeakyReLU(alpha=0.181818))
model40.add(Dropout(0.4))

### Second Layer
model40.add(LSTM(512, return_sequences=True))
model40.add(LeakyReLU(alpha=0.181818))
model40.add(Dropout(0.4))

### Third Layer
model40.add(LSTM(256, return_sequences=True))
model40.add(LeakyReLU(alpha=0.181818))
model40.add(Dropout(0.4))

### Fourth Layer
model40.add(LSTM(64, return_sequences=False))
model40.add(LeakyReLU(alpha=0.181818))
model40.add(Dropout(0.4))

### Output Layer
model40.add(Dense(trainY.shape[1]))

### Compile
model40.compile(optimizer='adam', loss='mse')

### Fit the model
history40 = model40.fit(trainX, trainY, epochs=500, batch_size=32, validation_split=0.1, verbose=1)

## Plot the accuracy preformance results
plt.plot(history40.history['loss'], label='Training loss')
plt.plot(history40.history['val_loss'], label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.savefig('LSTM_Mult_40Days_Training.png', dpi=100)

## Make prediction
n_past = 40
n_days_for_prediction=40
```

```
prediction = model40.predict(trainX[-n_days_for_prediction:])
prediction_copies = np.repeat(prediction, df_for_training.shape[1], axis=-1)
y_pred_future = scaler.inverse_transform(prediction_copies)[:,0]
### Convert the data into a dataframe
df_forecast = pd.DataFrame({'date':predict_period_dates.values.T[0], 'Positive\_Rate':y_pred_future})
df_forecast['date']=pd.to_datetime(df_forecast['date'])
df_forecast.index = df_forecast['date']
df_forecast = df_forecast.drop(columns={'date'})
original = pd.DataFrame(df_master['Positive\_Rate'][-100:])
original_pred = pd.DataFrame(df_master['Positive\_Rate'][-100:-40])
df_pred_plot = pd.concat([original_pred, df_forecast])
### Evaluate our LSTM model
rmse = sqrt(mean_squared_error(df_forecast, original[-40:]))

## Plot the prediction result
sns.lineplot(data=original, x=original.index, y=original['Positive\_Rate'])
sns.lineplot(data=df_pred_plot, x=df_pred_plot.index, y=df_pred_plot['Positive\_Rate'])
plt.xlabel('Timestamp')
plt.ylabel('Positive Rate')
plt.legend(['Real Data', 'Predicted Data'])
### Save figure
plt.savefig('LSTM_Mult_40Days.png', dpi=100)

# 40 Days Prediction GRU (Multivariate)
## define the GRU model
modelG40 = Sequential()

### First Layer
modelG40.add(GRU(128, input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
modelG40.add(LeakyReLU(alpha=0.181818))
modelG40.add(Dropout(0.4))

### Second Layer
modelG40.add(GRU(512, return_sequences=True))
modelG40.add(LeakyReLU(alpha=0.181818))
modelG40.add(Dropout(0.4))

### Third Layer
modelG40.add(GRU(256, return_sequences=True))
modelG40.add(LeakyReLU(alpha=0.181818))
modelG40.add(Dropout(0.4))

### Fourth Layer
modelG40.add(GRU(64, return_sequences=False))
modelG40.add(LeakyReLU(alpha=0.181818))
modelG40.add(Dropout(0.4))

### Output Layer
modelG40.add(Dense(trainY.shape[1]))

### Compile
modelG40.compile(optimizer='adam', loss='mse')
```

```

### Fit the model
historyG40 = modelG40.fit(trainX, trainY, epochs=500, batch_size=32, validation_split=0.1, verbose=1)

## Plot the accuracy preformance results
plt.plot(historyG40.history['loss'], label='Training loss')
plt.plot(historyG40.history['val_loss'], label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.savefig('GRU_Mult_40Days_Training.png', dpi=100)

## Make prediction
n_past = 40
n_days_for_prediction=40
predictionG = modelG40.predict(trainX[-n_days_for_prediction:])
prediction_copies = np.repeat(predictionG, df_for_training.shape[1], axis=-1)
y_pred_future = scaler.inverse_transform(prediction_copies)[:,0]
### Convert the data into a dataframe
df_forecast = pd.DataFrame({'date':predict_period_dates.values.T[0], 'Positive\_Rate':y_pred_future})
df_forecast['date']=pd.to_datetime(df_forecast['date'])
df_forecast.index = df_forecast['date']
df_forecast = df_forecast.drop(columns={'date'})
original = pd.DataFrame(df_master['Positive\_Rate'][-100:])
original_pred = pd.DataFrame(df_master['Positive\_Rate'][-100:-40])
df_pred_plot = pd.concat([original_pred, df_forecast])
### Evaluate our LSTM model
rmse = sqrt(mean_squared_error(df_forecast, original[-40:]))

## Plot the prediction result
sns.lineplot(data=original, x=original.index, y=original['Positive\_Rate'])
sns.lineplot(data=df_pred_plot, x=df_pred_plot.index, y=df_pred_plot['Positive\_Rate'])
plt.xlabel('Timestamp')
plt.ylabel('Positive Rate')
plt.legend(['Real Data', 'Predicted Data'])
### Save figure
plt.savefig('GRU_Mult_40Days.png', dpi=100)
```