The UWI, St.Augustine Campus

BSc Computer Science (Special) Final Year Project

Minter

Student Names

Jada Gooding: 816001706
Saleem Ali: 816007643
Kyron Saunders: 816010263
Simon Ramdath: 816009535

Supervised by

Mr. Inzamam Rahaman
Department of Computing and Information Technology

16 May 2020

Declaration

We hereby certify that the material, which we now submit for assessment on the programme of study leading to the award of BSc Computer Science (Special), is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Student Names
Jada Gooding,
Saleem Ali,
Kyron Saunders,
Simon Ramdath,
16 May 2020

List of Tables

Listings

Contents

Abstract

In the past decade, traditional forecasting methods have played a key role for researchers studying time series analysis. Recently, as advances in machine learning have gained widespread attention, researchers have brought new techniques to the table. In this paper, we present a comprehensive comparative study of the Seasonal Auto-regressive Integrated Moving Average (SARIMA) traditional model and Three Deep Learning architectures, namely, Back Propagation Neural Networks (BPNN), The Long Shot Term Memory model and, Neural Ordinary Differential Equations (ODE). The model's performance were evaluated based on three metrics: Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Square Error (RMSE). The results showed that the SARIMA model obtained the smallest error rate throughout the experimental process, followed by LSTM, BPNN, and Neural ODE, respectively. This gives us valuable insight into discovering which model is the best at forecasting the seasonal trend of time series data.

1 Introduction

1.1 Problem Statement

The main research question of this paper is to investigate and compare the effectiveness of the time series models to make seasonal predictions on real data, thus identify the most suitable models for analyzing time series data. Specifically, we are interested in evaluating the difference in performance between Seasonal Auto-regressive Integrated Moving Average (SARIMA) and the artificial neural networks, namely, Neural Ordinary Differential Equations (Neural ODE), Back-Propagation Neural Network (BPNN), and Long Short-Term Memory (LSTM) as we are of the assumption that Neural ODE will be more effective is conducting the time series forecast.

1.2 Project Background

A time series is a set of numerical measurements, taken at equal intervals over a designated time. Time series analysis deals with analyzing data from a time series (known as time series data) to give graphical and numerical output which is then used to obtain useful statistics about the data; including its trend, irregularities, cycles and variation [1] An extension to this analysis is known as time series forecasting, which is the use of a model to predict future values based on previously observed values [2]. Time series data can fluctuate at specific, regular intervals; this fluctuation is known as the seasonality of the data.

Unfortunately, various properties of time series data make them inherently challenging to analyze. Firstly, the data are highly dynamic. It is often difficult to tease out the structure that is embedded in time series data. Secondly, time series data can be nonlinear and contain highly complex autocorrelation structure. Data points across different periods of time can be correlated with each other and a linear approximation sometimes fails to model the seasonal structure in the data. Therefore, identifying a model that is broadly applicable has been difficult [3].

In this study, we tried to alleviate the main challenges faced by researchers studying time series by answering which model is the most applicable. To test the hypo-

thesis, each model will be constructed and used to forecast the time series datasets and the accuracy of the prediction will be compared against the raw series using the performance results of the RMSE, MAE and MAPE of each model. Lastly, a comparative analysis will be conducted with a conclusion being made on the accuracy of each respective model.

2 Literature Review

Time series analysis plays a key role in the decision-making process of business stakeholders and government policy makers. The need to predict probable future occurrences becomes even more evident when making decisions that may have a long-lasting effect. There exists a property of time-series data refers to as seasonality. It is a regular and predictable change that repeats in the data over a one-year period. Thus, businesses and government agencies have become more cognizant of the benefits involved in predicting this property. Therefore, it is necessary that the methods used to discern such predictions are efficient. This paper seeks to determine the most efficient approach by comparing a traditional forecasting method against artificial neural networks (ANN) methods.

The traditional forecasting method chosen for this experiment is the SARIMA model. The SARMA model is an extension the of the popular ARIMA model, which is one of the most popular time-series forecasting models. SARIMA contains additional components to factor in seasonality within a dataset. The basis by which the model functions is that it considers the time-series to be linear. Linearity refers to the property of data to exhibit the characteristics of a straight line. Additionally, stationary of the data is also a factor for traditional methods to consider. Stationarity refers to aspects of data such as mean and variance, remaining constant over time. In real-life scenarios however, time-series data usually do not display these characteristics. It is often non-linear and non-stationary. Consequently, as mentioned above SARIMA operates on the basis that the time-series is linear and as such its practical use is very much limited [4]. SARIMA does consider that the data may exhibit non-stationary characteristics and handles it by seasonal differencing of an appropriate order is used. Another issue was raised by Adhikari [4], in which he states that there is an increase in the complexity of the model that arises when the seasonal patterns of the time-series becomes too dominant. Due to the aforementioned reasons, it would appear that there exist some limitations to the traditional methods that would need to be considered prior to practical use.

Three ANN methods were chosen for this experiment, they are the LSTM, BPNN

and neural ODE models. Three ANN methods were chosen for this experiment, they are the LSTM, BPNN and neural ODE models. The LSTM model comprises several units made up of a cell state, an input, output and a forget gate. The cell state acts a form of memory whilst the three gates regulate the flow of information into and out of the cell, forming a feed-back loop that helps adjust the cell state [5]. The BPNN model is separated into three layers, the input layer, hidden layer, and output layer. It works by repeatedly adjusting the weights of the connections between layers in the network to reduce the difference between the actual output layer and the desired output [6]. The neural ODE model removes the need to specify the number of layers in a neural network and instead it uses the derivative of the hidden state as its parameter. Thus, the network can be represented as a continuous function, i.e. with a continuous depth. The model takes the desired accuracy and it trains itself within that margin of error [7], [8]. There has been much apply in the literature regarding ANN. This is mainly due to their nonlinear, nonparametric, data driven and self-adaptive nature [4]. Firstly, their inherent nonlinear nature would theoretically make it a more practical solution when it comes to handling time-series data, which as mentioned above is usually nonlinear. Secondly, being nonparametric allows them to handle situations where the input data corrupted or unclear. Finally, their data-driven and self-adaptive nature means that the desired model is formed based on the features presented from the data [9]. There are a lot of benefit to using ANN, however, there still exist some challenges such as there is no clear way to choose the best ANN model [5].

To summarize, time-series analysis is an important aspect in the decision-making process of businesses and government agencies. The occurrence of seasonality within the data provokes an even greater need for the predictions to be accurate. Thus, methods used to make such predictions must be effective. Hence, this experiment seeks to determine which method, traditional or ANN is the most efficient regarding predicting seasonality. The literature shows a clear favorite, but it is yet to be decided as both approaches has it strengths and weakness.

3 Methodology

3.1 Model Overview

The SARIMA model, which combines seasonal differencing with an ARIMA model, is used when the time series data exhibits periodic characteristics. SARIMA models have been widely used in forecasting areas when data exhibit a seasonal trend, this model has been demonstrated as an effective linear model that can grasp the linear trend of the series.

Models based on artificial neural networks can effectively extract nonlinear relationships in the data. They have been widely used in time series predictions because of their characteristics of robustness, fault tolerance, and adaptive learning ability [7], they have no need to assume the underlining distribution for the data collected [7]. Unlike the SARIMA model of linearity, neural network models have nonlinear functions that constitute the linkage between the value at time t and its previous value at p time points. Neural networks have successfully shown their usefulness in several types of classification and nonlinear regression problems. In the succeeding section we would look at the models used in this present study.

3.1.1 Seasonal Autoregressive Integrated Moving Average (SARIMA)

As mentioned above the SARIMA models is an extension of the ARIMA model, with the addition of seasonal differencing components. "It adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series, as well as an additional parameter for the period of the seasonality."[9]. The smaller the output error (cost function) the better fit of the model. If the error does not meet the accuracy requirements set previously, the network weights and bias will be adjusted along the opposite direction of the network until the required minimum network error is eventually achieved [7]. The parameters for the SARIMA models is as follows:

- p: Trend autoregression order.
- d: Trend difference order.
- q: Trend moving average order.
- P: Seasonal autoregressive order.
- D: Seasonal difference order.
- Q: Seasonal moving average order.
- m: The seasonal length in the data

The AR component (p,P) refers to how many series from the past are going to be used to forecast periods ahead. The I component (d,D) refers to how many differencing orders are needed to make the series stationary by removing any trends. The MA component (q,Q) refers to the number of errors from the previous period used to inform future predictions.

3.1.2 Back-Propagation Neural Network (BPNN)

The big picture in BPNN is how we go from having some data, throwing it into some algorithm and hoping for the best. But what happens inside that algorithm? In the case of modeling time series data, the historical data are sent into the input layer, which sends the data ping-ponging forward, through the different connections, from one neuron to another in the network [10] and the corresponding predicted data is generated from the output layer after the network is adequately trained.

The network "learns" the information contained in the frequency of time series by adjusting the weights and biases between layers. The structure and interconnections change with the variation of the time series data in a typical data-driven and adaptive learning process [11]. Artificial neural networks can only be viewed in terms of the input, output, and transfer characteristics as the neural network acts like a black box [10].

Ultimately, BPNN is a type of feed forward artificial neural networks, after each forward pass through a network, back-propagation optimizes the cost function-

called a backward pass while adjusting the interconnections for each layer, so that we get the desired result. Back-propagation aims to minimize the cost function by adjusting network's weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters [10]. And so, back-propagation is not just a fast algorithm for learning. It gives us detailed insights into how changing the weights and biases changes the overall behavior of the network [10]. All the connection weights are initialized randomly, and then modified according to the results of the BP training process.

3.1.3 Long Short-Term Memory (LSTM)

Long Short-Term Memory networks are a special kind of RNN, capable of learning long-term dependencies and are designed to avoid the long-term dependency problem. Recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure [12].

The main aspect of LSTMs is the cell state which transfers relative information down the sequence of chains. Information runs down this sequence of chains with only some minor linear interactions or no changes. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a point-wise multiplication operation [13] [12].

Inputs gates pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. The hidden state and current input are passed into the tanh function to squish values between -1 and 1 to regulate the network. The tanh and sigmoid output are multiplied. The sigmoid output decides which information is important to keep from the tanh output [13].

The forget gate decides what information to keep. The closer to 0 means to forget

closer to 1 means keep the information.

The cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. The output from the input gate is used to do a pointwise addition which updates the cell state to new values that the neural network finds relevant. This is the new cell state [13].

The output gate decides what the next hidden state should be. The previous hidden state is passed into the current input into a sigmoid function and pass the newly modified cell state to the tanh function. The tanh output is multiplied with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step [13].

3.1.4 Neural Ordinary Differential Equations

Neural Ordinary Differential Equations combines Neural Networks and Ordinary Differential Equations to which generalize standard layer to layer propagation to continuous depth [14].

Neural networks are typically organized in layers. Layers are made up of several interconnected nodes which contain an activation function. Patterns are presented to the network via the input layer, which communicates to one or more hidden layers where the actual processing is done via a system of weighted connections. The hidden layers then link to an output layer where the result is the output [15].

Ordinary differential equations (ODE) are equations that involve some ordinary derivatives of a function. i.e. Equations that relate a function which has one variable for its derivatives. Partial differential equations involve multiple variables [16] [17].

Example of a simple ODE:

$$\frac{dy}{dx}(t) = \cos t$$
$$x(t) = \sin t + C$$

Neural Ordinary Differential Equations parameterize the derivative of the hidden

state using a neural network. The output of the network is computed using a black-box differential equation solver. These continuous-depth models have constant memory cost, adapt their evaluation strategy to each input, and can explicitly trade numerical precision for speed [7].

3.2 Datasets Employed

The data acquired for the present study consist of monthly univariate time series datasets. These several datasets contain two (2) columns representing date of the event and frequency recorded, respectively. The dataset was cleaned to allow for consistent data types, column names, and date formats between datasets. This enable us to easily switch between datasets when testing by just replacing the file name. More details about each dataset is listed below.

Description	Duration	Unit	Source
Beer production in Australia: contains information giving Monthly beer production figures in Australia	JAN 1956 – JUN 2010	megaliters	The Australian Government open data initiative.
International Airline Passenger: contains information on monthly International Airline Passengers	JAN 1949 – DEC 1960	Thousands of passengers	the Time Series Data Library
US Liquor Sales: contains information on the monthly sales of the Liquor industry in the US	JAN 1980 – DEC 2007	Millions of dollars	The U.S Census Bureau.
CO2 Mauna Loa: contains information on the trends in Atmospheric Carbon Dioxide Levels taken by the Mauna Loa Observatory	JAN 1959 – DEC 1997	parts per million (ppm)	The US Government's Earth System Research Laboratory, Global Monitoring Division.
Critical Radio Frequencies: contains information on the radio frequencies in Washington D.C	MAY 1934 – APR 1954	highest radio frequency	The Time Series Data Library.
Electricity Production in Australia: contains information on the electricity production in Australia	JAN 1956 – AUG 1995	million kilowatt hours	The Time Series Data Library.

Table 1: Datasets Employed

3.3 Model Selection Criterion and Evaluation Index

Different parameters or network structures can be determined for individual datasets in the SARIMA model and deep learning architectures. For the SARIMA

model, the AIC and BIC, act as the criteria for selecting the best model. However, in neural networks, no criteria for model selection like AIC and BIC exist. Usually, the modeling sample is divided into two parts: the training set, which is used for training, and the test set, which is used to test the efficiency of the built structure. The selection of a best structure for each model is based on the minimization of the bias between the values obtained from the training and validation samples and their corresponding observed values from the raw data.

Furthermore, the contrast between the observed value of the raw series and the predicted values obtained by the four methods will be compared to determine the efficiency of the models' forecasting performance. The mean absolute error (MAE), mean absolute percentage error (MAPE), and the root mean square error (RMSE) were selected as the measures of evaluation because as empirical methods they are widely used in combining and selecting forecasts for measuring bias and accuracy of models [11]. The measures of evaluation were calculated using the equations below P is the predicted value at time t, Z is the observed value at time t, and T is the number of predictions.

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|Pt - Zt|$$

$$MAPE = \frac{1}{T}\sum_{t=1}^{T}\left|\frac{(Pt-Zt)}{Zt}\right|$$

$$RMSE = \frac{1}{T}\sum_{t=1}^{T}(Pt - Zt)^2$$

4 Results

4.1 Development of the SARIMA

The SARIMA model was created using the autoarima function was utilized to obtain the seasonal parameters and the AIC and BIC results. This assisted us in determining an appropriate model when assessing the best fitting characteristics per dataset. After, the SARIMA model was defined by calling the SARIMAX function which was fitted with the training data observations and the seasonal parameters obtained from the autoarima. Lastly, the model was fitted and trained, and the predicted results was obtained.

4.2 Development of the Neural Networks

The NN developed for the LSTM and BPNN models utilized the TensorFlow keras library. In addition to, a timeseries generator object was created to foster code reuse. When constructing the models, we utilized the 'relu' and 'sigmoid' activations functions, as well as the Adams and SDG optimizers for LSTM AND BPNN, respectively. Lastly the model created was fitted and trained with the time series generator object along with 20 epochs cycles.

Different activations functions and optimizers, and the number of neurons in hidden layers will affect computation efficiency of the Neural networks. To date, no standard rules for selecting the number of hidden neurons and layers exist. The specific network structure is generally fixed by trial and error. When developing our model's different parameter settings were tested to find tune the models. Therefore, the network structure and corresponding parameters was selected using the trial-and-error method for each presenting dataset. Once the structure is determined, it is used to forecast the test set.

The Neural ODE model was obtained from [7]. The model's parameters were also tuned by trial and error, such as, the activation method, and batch size to name a few. We also updated the code to ensure the data type was changed to a float Tensor, as one of the hard-coded elements required it to be float. This was facilitated, as during the experiments we saw the code automatically changed the

data type for one or two datasets but failed in automating the rest.

4.3 Performance Measurement Results

Dataset 1: Beer production in Australia			
Model	RMSE	MAE	MAPE
SARIMA	8.491	7.369	5.127
LSTM	10.134	8.325	5.604
BPNN	12.131	9.436	6.042
Neural ODE	12.504	10.300	7.306

Table 2: Dataset 1: Beer production in Australia

Dataset 2: International Airline Passengers			
Model	RMSE	MAE	MAPE
SARIMA	18.257	15.281	3.213
LSTM	30.208	26.102	5.376
BPNN	160.567	148.264	29.976
Neural ODE	206.715	162.231	30.977

Table 3: Dataset 2: International Airline Passengers

Dataset 3: CO2 Mauna Loa Levels			
Model	RMSE	MAE	MAPE
SARIMA	0.459	0.387	0.106
LSTM	1.209	1.065	0.293
BPNN	3.928	3.744	1.027
Neural ODE	5.463	5.154	3.911

Table 4: Dataset 3: CO2 Mauna Loa Levels

Dataset 4: US Liquor Sales			
Model	RMSE	MAE	MAPE
SARIMA	29.865	24.811	1.384
LSTM	139.811	120.941	7.387
BPNN	166.507	123.467	6.502
Neural ODE	287.933	234.603	13.404

Table 5: Dataset 4: US Liquor Sales

Dataset 5: Electricity Production in Australia			
Model	RMSE	MAE	MAPE
SARIMA	283.586	239.601	1.698
LSTM	484.780	385.827	2.729
BPNN	763.064	606.962	4.255
Neural ODE	1572.927	1443.146	11.392

Table 6: Dataset 5: Electricity Production in Australia

Dataset 6 :Radio Frequency			
Model	RMSE	MAE	MAPE
SARIMA	0.786	0.745	13.311
LSTM	0.301	0.252	4.848
BPNN	2.296	2.248	42.886
Neural ODE	OOT	OOT	OOT

Table 7: Dataset 6 :Radio Frequency

OOT: Out of Time – Due to the interest in time, we took the decision to record the findings as inconclusive. We tried to acquire the results, however, the datasets size prevented us from running the experiment in a timely manner. In addition to, as the results were highly favour towards the SARIMA model we did not see the urge to find an alternate method as the results would not have been significant enough to change the conclusion of the study.

4.4 Comparison of forecasting Performance

Section 4.3 displayed the performance evaluation results of the four methods, as well as the plots of the predicted model against the raw series observations.

The table indicated that the predicted values by most of the selected models reasonably matched the test data set. The MAE, MAPE and RMSE measures were the lowest for SARIMA throughout the study, with one occurrence of LSTM leading among the four methods. The Neural ODE model had the largest MAE, MAPE, and RMSE among the four methods. We found that , the traditional SARIMA model produced a more accurate forecast than the neural network-based models in forecasting, according to all three measures. The performances of the neural networks were also different, with the LSTM outperforming the others in the study. The overall performance of the four models ranked in descending order are as follows: Neural ODE, BPNN, LSTM, and SARIMA.

We do note, that the Neural Network models did better at predicting single instances of the test data, however, SARIMA managed the best overall as the distance from the raw series to the predicted value was the least for all instances compared to the Neural Networks. One reason for inferior performance by the Neural Networks is simply the model tends to underestimate at the peaks and overestimate at the troughs, preventing it from overfitting to the test set. This is advantageous when past observations are not good indicators of future observations, but in the datasets chosen, where trends are predictable, this leads to a worse forecast [18].

5 Discussion

The effectiveness of statistical models in forecasting future trends has been proved useful over the past decade. Moreover, as there have been numerous time series models proposed for prediction, the issue of which model will be the "best" at forecasting seasonality was quite minimal when it came to determining trends in a data set.

In the present study, we address this problem by providing a comprehensive comparative analysis of the performance of a classical forecasting model and three deep learning architectures in predicting seasonal trends. This was facilitated by using several uni-variate time series data sets. (Refer to section 3.2) At the beginning of the study, we believed that the Neural Networks would outperform the SARIMA model, with the Neural ODE leading the way. This assumption was formulated based on recently studies published in time series analysis, where Deep Learning models outperformed traditional methods. Although based on our findings, we saw that this was not the case.

Surprisingly, we saw a complete reverse of our initial assumption based on the ranked performance in section 4.4. Nowadays, in most forecasting applications, we do not see traditional methods faring well against artificial Neural Networks. The SARIMA model outperforming three neural networks was quite a shock, that the researcher in the study assumed something went wrong and reran and retuned the experiments to ensure the figures recorded was not taken at face value. But essential, the SARIMA model did well at grasping the linear trend of almost all data sets. This indicates to us that; statistically complex methods do not necessarily provide more accurate forecasts than simpler ones.

Given the sophistication of the Neural models, it may seem surprising how ineffective they are at making accurate forecasts. On the other hand, the amount of information we have at hand to make these forecasts is minimal [18]. Furthermore, comparing the two types of models used, NN are undoubtedly more complicated and difficult to train and yet it did not surpass the performance of a simple SARIMA model for the majority of the series. Neural Networks may not be up for the task of analyzing simple time series data like those considered in this paper.

It has been the central focus of researchers hoping to solve highly complex tasks, such as generation of text and handwriting. We believe that for simple settings, traditional methods such as SARIMA that make reasonable assumptions about the underlying structure tend to be more effective.

In summary, classical traditional time-series forecasting models such as SARIMA differ from artificial neural networks time series models in both theoretical and practical aspects. Comparative studies of different forecasting techniques on several data sets can facilitate the selection of the best time series model for when forecasting the future behavior of a data set; and based on our findings the SARIMA model is adequately prepared for this.

6 Conclusions

In recent times, the ability to accurately predict future outcome has become more necessary in the decision-making process of companies and government agencies. This increased re-license begets the need for the most accurate and efficient prediction method. In this paper, we show comparisons between traditional (SARIMA) and artificial neural networks approaches when applied to monthly uni-variate time series data sets. The results indicated that the traditional approach used had the lowest error rate than the artificial neural networks employed. It should be noted that the SARIMA models required a long time to determine the most appropriate parameters, therefore, the time it takes to produce results from the model takes much longer than the others. Additionally, attention should be drawn to the LSTM model, which ranked second in terms of accuracy. However, the speed by which this done is approximately ten times quicker than the SARIMA model. Given these detailed comparisons, companies and government agencies can now make a more informed decision when choosing a method to project their data.

7 Future Work

After learning about Time series analysis, we would like to look at evaluating other types of data sets, such as those with daily frequencies or even multivariate data series. Secondly, we also wish to improve on our existing models. In addition to, possibly including other Time series forecasting methods such as, fbprophet or Radial Basis Function Neural Network (RBFNN) to name a few.

8 Appendix

GitHub Repository for code implementation: https://github.com/brynjolf23/minter

Neural Network (NN) - A neural network simply consists of neurons (also called nodes). These neurons are split between the input, hidden and output layer. These nodes are connected in some way. Then each neuron holds a number, and each connection holds a weight.

8.1 SARIMA Model

```
#Loading libraries and packages

import numpy as np
import pandas as pd
import os
import seaborn as sns

from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose

#from pmdarima import auto_arima

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from statsmodels.tools.eval_measures import rmse

import warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
%matplotlib inline

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

df = pd.read_csv('Monthly_electricity-production.csv')
```

```
29
30 df.head()
31
32 df.info()
33 df.Month = pd.to_datetime(df.Month)
34
35 df = df.set_index("Month")
36 df.head()
37
38 df.index.freq = 'MS'
39
40 plt.figure(figsize=(18,9))
41 plt.plot(df.index, df["Frequency"], linestyle="-")
42 plt.xlabel=('Dates')
43 plt.ylabel=('Total Production')
44 plt.show()
45
46 a = seasonal_decompose(df["Frequency"], model = "add")
47 a.plot()
48
49 #We run auto_arima() function to get best p,d,q,P,D,Q values
50 #Then, we use sarimax to account for seasonality and then
      forecasting
51 #building the model
52
53 from pmdarima.arima import auto_arima
54
55 model = auto_arima(df['Frequency'],trace=True, error_action='
      ignore', suppress_warnings=True, seasonal=True, m=12,max_p=7,
      max_d=5,max_q=7, max_P=4, max_D=4,max_Q=4).summary()
56 #model.summary()
57
58 model
59
60 #Let's split the data into train and test set
61 train_data = df[:len(df)-12]
62 test_data = df[len(df)-12:]
63
64 #plotting the data
```

```
ax = train_data.plot(figsize=(15,10))
test_data.plot(ax=ax,figsize=(15,10))

#Building the SARIMAX Model
sarima_model = SARIMAX(train_data['Frequency'], order = (0,1,1),
    seasonal_order = (1,0,[1,2,3],12))
sarima_model = sarima_model.fit()
sarima_model.summary()

#As the model was built, fitting and trained, now we can predict
    the results
sarima_pred = sarima_model.predict(start = len(train_data), end =
    len(df)-1, typ="levels").rename("SARIMA_Predictions")
#Prints the results
sarima_pred

test_data['SARIMA_Predictions'] = sarima_pred

#Ploting the predicted data aginst the original dataset for the
    test data
test_data['Frequency'].plot(figsize = (16,5), legend=True)
sarima_pred.plot(legend = True,linestyle="--")

#Errors which would be used to esitmate a model accuarcy
sarima_rmse_error = rmse(test_data['Frequency'],test_data['
    SARIMA_Predictions'])

sarima_mse_error = sarima_rmse_error**2

sarima_mae_error = mean_absolute_error(test_data['Frequency'],
    test_data['SARIMA_Predictions'])
sarima_mape_error = mean_absolute_percentage_error(test_data['
    Frequency'],test_data['SARIMA_Predictions'])

mean_value = df['Frequency'].mean()

print(f'MSE Error: {sarima_mse_error}\nRMSE Error: {
    sarima_rmse_error}\nMAE: {sarima_mae_error}\nMAPE: {
```

```
   sarima_mape_error}\nMean: {mean_value}')
```

Listing 1: SARIMA Model

8.2 LSTM Model

```
import numpy as np
import pandas as pd
import os
import seaborn as sns

from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose

#from pmdarima import auto_arima

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

from statsmodels.tools.eval_measures import rmse

import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
%matplotlib inline

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

df = pd.read_csv('Monthly_US Liquor Sales.csv')

df.Month = pd.to_datetime(df.Month)

df = df.set_index("Month")
df.head()

df.index.freq = 'MS'

```

```
plt.figure(figsize=(18,9))
plt.plot(df.index, df["Frequency"], linestyle="-")
plt.xlabel=('Dates')
plt.ylabel=('Total Production')
plt.show()

#Let's split the data into train and test set
train_data = df[:len(df)-12]
test_data = df[len(df)-12:]

#plotting the data
ax = train_data.plot(figsize=(15,10))
test_data.plot(ax=ax,figsize=(15,10))

from __future__ import absolute_import, division, print_function,
    unicode_literals
import tensorflow as tf

#First we'll scale our train and test data with MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

scaler.fit(train_data)
scaled_train_data = scaler.transform(train_data)
scaled_test_data = scaler.transform(test_data)

n_input = 12
n_features= 1
generator = tf.keras.preprocessing.sequence.TimeseriesGenerator(
    scaled_train_data, scaled_train_data, length=n_input,
    batch_size=1)

lstm_model = tf.keras.Sequential()
lstm_model.add(tf.keras.layers.LSTM(200, activation='relu',
    input_shape=(n_input, n_features)))
lstm_model.add(tf.keras.layers.Dense(1))
lstm_model.compile(optimizer='adam', loss='mse')

lstm_model.summary()
```

```
70
71 lstm_model.fit_generator(generator,epochs=20)
72
73 losses_lstm = lstm_model.history.history['loss']
74 plt.figure(figsize=(12,4))
75 plt.xticks(np.arange(0,21,1))
76 plt.plot(range(len(losses_lstm)),losses_lstm)
77
78 lstm_predictions_scaled = list()
79
80 batch = scaled_train_data[-n_input:]
81 current_batch = batch.reshape((1, n_input, n_features))
82
83 for i in range(len(test_data)):
84     lstm_pred = lstm_model.predict(current_batch)[0]
85     lstm_predictions_scaled.append(lstm_pred)
86     current_batch = np.append(current_batch[:,1:,:],[[lstm_pred]],
       axis=1)
87
88 lstm_predictions_scaled
89
90 #As you know we scaled our data that's why we have to inverse it
       to see true predictions.
91 lstm_predictions = scaler.inverse_transform(
       lstm_predictions_scaled)
92
93 lstm_predictions
94
95 test_data['LSTM_Predictions'] = lstm_predictions
96
97 test_data
98
99 test_data['Frequency'].plot(figsize = (16,5), legend=True)
100 test_data['LSTM_Predictions'].plot(legend = True,linestyle="--")
101
102 lstm_rmse_error = rmse(test_data['Frequency'], test_data["
       LSTM_Predictions"])
103
104 lstm_mse_error = lstm_rmse_error**2
```

```
105
106 lstm_mae_error = mean_absolute_error(test_data['Frequency'],
    test_data["LSTM_Predictions"])
107 lstm_mape_error = mean_absolute_percentage_error(test_data['
    Frequency'],test_data["LSTM_Predictions"])
108
109 mean_value = df['Frequency'].mean()
110
111 print(f'MSE Error: {lstm_mse_error}\nRMSE Error: {lstm_rmse_error
    }\nMAE: {lstm_mae_error}\nMAPE: {lstm_mape_error}\nMean: {
    mean_value}')
```

Listing 2: LSTM Model

8.3 BPNN Model

```
1  import numpy as np
2  import pandas as pd
3  import os
4  import seaborn as sns
5
6  from statsmodels.tsa.statespace.sarimax import SARIMAX
7  from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
8  from statsmodels.tsa.seasonal import seasonal_decompose
9
10 #from pmdarima import auto_arima
11
12 from sklearn.metrics import mean_squared_error
13 from sklearn.metrics import mean_absolute_error
14
15 from statsmodels.tools.eval_measures import rmse
16
17 import warnings
18 warnings.filterwarnings("ignore")
19 import matplotlib.pyplot as plt
20 get_ipython().run_line_magic('matplotlib', 'inline')
21
22 import random
23 random.seed(0)
24
```

```
25
26 def mean_absolute_percentage_error(y_true, y_pred):
27     y_true, y_pred = np.array(y_true), np.array(y_pred)
28     return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
29
30 df = pd.read_csv('Monthly_electricity-production.csv')
31
32 df.info()
33
34 df.Month = pd.to_datetime(df.Month)
35
36 df = df.set_index("Month")
37 df.head()
38 df.index.freq = 'MS'
39
40 plt.figure(figsize=(18,9))
41 plt.plot(df.index, df["Frequency"], linestyle="-")
42 plt.xlabel=('Dates')
43 plt.ylabel=('Total Production')
44 plt.show()
45
46 train_data = df[:len(df)-12]
47 test_data = df[len(df)-12:]
48
49 ax = train_data.plot(figsize=(15,10))
50 test_data.plot(ax=ax,figsize=(15,10))
51
52 from __future__ import absolute_import, division, print_function,
      unicode_literals
53 import tensorflow as tf
54
55 #First we'll scale our train and test data with MinMaxScaler
56 from sklearn.preprocessing import MinMaxScaler
57 scaler = MinMaxScaler()
58
59 scaler.fit(train_data)
60 scaled_train_data = scaler.transform(train_data)
61 scaled_test_data = scaler.transform(test_data)
62
```

```
n_input = 12
n_features= 1
generator = tf.keras.preprocessing.sequence.TimeseriesGenerator(scaled_train_data, scaled_train_data, length=n_input, batch_size=1)

bp_model = tf.keras.Sequential()

bp_model.add(tf.keras.layers.Dense(20,activation='relu', input_shape=(n_input, n_features)))
bp_model.add(tf.keras.layers.Dense(10, activation='relu'))
bp_model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

bp_model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
bp_model.summary()

bp_model.fit_generator(generator,epochs=20)

losses_bp = bp_model.history.history['loss']
plt.figure(figsize=(12,4))
plt.xticks(np.arange(0,21,1))
plt.plot(range(len(losses_bp)),losses_bp)

bp_predictions_scaled = list()

batch = scaled_train_data[-n_input:]
current_batch = batch.reshape((1, n_input, n_features))

for i in range(len(test_data)):
    bp_pred = bp_model.predict(current_batch)[0][i]
    bp_predictions_scaled.append(bp_pred)

bp_predictions_scaled

bp_predictions = scaler.inverse_transform(bp_predictions_scaled)

bp_predictions

```

```
test_data['BP_Predictions'] = bp_predictions

test_data

test_data['Frequency'].plot(figsize = (16,5), legend=True)
test_data['BP_Predictions'].plot(legend = True,linestyle="--")

test_data

bp_rmse_error = rmse(test_data['Frequency'], test_data["
    BP_Predictions"])

bp_mse_error = bp_rmse_error**2

bp_mae_error = mean_absolute_error(test_data['Frequency'],
    test_data["BP_Predictions"])
bp_mape_error = mean_absolute_percentage_error(test_data['
    Frequency'],test_data["BP_Predictions"])

mean_value = df['Frequency'].mean()

print(f'MSE Error: {bp_mse_error}\nRMSE Error: {bp_rmse_error}\
    nMAE: {bp_mae_error}\nMAPE: {bp_mape_error}\nMean: {mean_value
    }')
```

Listing 3: BPNN Model

8.4 Neural ODE Model

```
import argparse
import time
import numpy as np

import torch
import torch.nn as nn
import torch.optim as optim

import pandas as pd

import os
```

```
12
13 from sklearn.metrics import mean_squared_error
14 from sklearn.metrics import mean_absolute_error
15 from statsmodels.tools.eval_measures import rmse
16
17 import warnings
18 warnings.filterwarnings("ignore")
19
20 import matplotlib.pyplot as plt
21
22
23
24 def mean_absolute_percentage_error(y_true, y_pred):
25     y_true, y_pred = np.array(y_true), np.array(y_pred)
26     return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
27
28 df = pd.read_csv('Monthly_passengers.csv')
29 rows, columns = df.shape
30
31 df.Month = pd.to_datetime(df.Month)
32
33 df = df.set_index("Month")
34 df.head()
35 df.index.freq = 'MS'
36
37
38 train_data = df[:len(df)-12]
39 test_data = df[len(df)-12:]
40
41
42 parser = argparse.ArgumentParser('ODE demo')
43 parser.add_argument('--method', type=str, default='adams')
44 parser.add_argument('--data_size', type=int, default=rows)
45 parser.add_argument('--batch_time', type=int, default=10)
46 parser.add_argument('--batch_size', type=int, default=12)
47 parser.add_argument('--niters', type=int, default=rows)
48 parser.add_argument('--test_freq', type=int, default=1)
49 parser.add_argument('--viz', action='store_false')
50 parser.add_argument('--gpu', type=float, default=0)
```

```
parser.add_argument('--adjoint', action='store_true')
args = parser.parse_args()

if args.adjoint:
    from torchdiffeq import odeint_adjoint as odeint
else:
    from torchdiffeq import odeint

device = torch.device('cuda:' + str(args.gpu) if torch.cuda.
    is_available() else 'cpu')

true_y0 = torch.tensor([df['Frequency'][0]])
true_y0 = true_y0.type(torch.FloatTensor)

t = torch.linspace(0., rows, rows)

true_y_list = df['Frequency'].values

tensor_arr = []
for i in true_y_list:
    tensor_arr.append([i])

true_y = torch.tensor(tensor_arr)


def get_batch():
    s = torch.from_numpy(np.random.choice(np.arange(args.data_size
    - args.batch_time, dtype=np.int64), args.batch_size, replace=
    False))
    batch_y0 = true_y[s]  # (M, D)
    batch_y0 = batch_y0.type(torch.FloatTensor)
    batch_t = t[:args.batch_time]  # (T)
    batch_y = torch.stack([true_y[s + i] for i in range(args.
    batch_time)], dim=0)  # (T, M, D)
    return batch_y0, batch_t, batch_y


# the ode function
class ODEFunc(nn.Module):
```

```
86
87      def __init__(self):
88          super(ODEFunc, self).__init__()
89
90          self.net = nn.Sequential(
91              nn.Linear(1, 50),  # 2 changed to 1
92              nn.Tanh(),
93              nn.Linear(50, 1),
94          )
95
96          for m in self.net.modules():
97              if isinstance(m, nn.Linear):
98                  nn.init.normal_(m.weight, mean=0, std=0.1)
99                  nn.init.constant_(m.bias, val=0)
100
101     def forward(self, t, y):
102         return self.net(y**3)
103
104
105 class RunningAverageMeter(object):
106     """Computes and stores the average and current value"""
107
108     def __init__(self, momentum=0.99):
109         self.momentum = momentum
110         self.reset()
111
112     def reset(self):
113         self.val = None
114         self.avg = 0
115
116     def update(self, val):
117         if self.val is None:
118             self.avg = val
119         else:
120             self.avg = self.avg * self.momentum + val * (1 - self.
    momentum)
121         self.val = val
122
123
```

```
124 if __name__ == '__main__':
125
126     func = ODEFunc()
127     optimizer = optim.RMSprop(func.parameters(), lr=1e-3)
128     end = time.time()
129
130     time_meter = RunningAverageMeter(0.97)
131     loss_meter = RunningAverageMeter(0.97)
132
133     neural_ode_loss = []
134
135     for itr in range(1, args.niters + 1):
136         optimizer.zero_grad()
137         batch_y0, batch_t, batch_y = get_batch()
138         pred_y = odeint(func, batch_y0, batch_t)
139         loss = torch.mean(torch.abs(pred_y - batch_y))
140         neural_ode_loss.append(loss.item())
141         loss.backward()
142         optimizer.step()
143
144         time_meter.update(time.time() - end)
145         loss_meter.update(loss.item())
146
147         if itr % args.test_freq == 0:
148             with torch.no_grad():
149                 pred_y = odeint(func, true_y0, t)
150                 loss = torch.mean(torch.abs(pred_y - true_y))
151
152         end = time.time()
153
154 print(pred_y)
155
156 ode_predictions = pred_y.data.numpy()
157 print(ode_predictions)
158
159
160 test_data['NODE_Predictions'] = ode_predictions
161 print(test_data)
162
```

```
163
164 #--------------ERROR
165 node_rmse_error = rmse(test_data['Frequency'], test_data["
    NODE_Predictions"])
166 node_mse_error = node_rmse_error**2
167 node_mae_error = mean_absolute_error(test_data['Frequency'],
    test_data["NODE_Predictions"])
168 node_mape_error = mean_absolute_percentage_error(test_data['
    Frequency'],test_data["NODE_Predictions"])
169
170
171 print(f'MSE Error: {node_mse_error}\nRMSE Error: {node_rmse_error
    }\nMAE: {node_mae_error}\nMAPE: {node_mape_error}')
```

Listing 4: Neural Ode

[19]

References

[1] Margaret Rouse. 2018. What is time series forecasting? - definition from whatis.com. (2018). https://whatis.techtarget.com/definition/time-series-forecasting.

[2] Chandu Siva. 2018. Time series forecasting - dzone ai. (2018). https://dzone.com/articles/time-series-forecasting.

[3] Martin Längkvist, Lars Karlsson and Amy Loutfi. 2014. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 11–24.

[4] Ratnadip Adhikari and R. K. Agrawal. 2013. An introductory study on time series modeling and forecasting. *CoRR*, abs/1302.6613. arXiv: 1302.6613. http://arxiv.org/abs/1302.6613.

[5] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9, 8, (November 1997), 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. https://doi.org/10.1162/neco.1997.9.8.1735.

[6] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 1986. Learning representations by back-propagating errors., 323, 6088, (October 1986), 533–536. DOI: 10.1038/323533a0.

[7] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt and David Duvenaud. 2018. Neural ordinary differential equations. *CoRR*, abs/1806.07366. arXiv: 1806.07366. http://arxiv.org/abs/1806.07366.

[8] Siraj Raval. [n. d.] Neural differential equations - youtube. (). https://www.youtube.com/watch?v=AD3K8j12EIE.

[9] Jason Brownlee. 2019. A gentle introduction to sarima for time series forecasting in python. (2019). https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/.

[10] Casper Hansen. 2020. Neural networks: feedforward and backpropagation explained. (2020). https://mlfromscratch.com/neural-networks-explained/.

[11] Xingyu Zhang, Yuanyuan Liu, Min Yang, Tao Zhang, Alistair A. Young and Xiaosong Li. 2013. Comparative study of four time series methods in forecasting typhoid fever incidence in china. *PLoS ONE*, 8.

[12] [n. d.] Understanding lstm networks. (). https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[13] Michael Phi. 2020. Illustrated guide to lstms and grus: a step by step explanation. (2020). https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[14] [n. d.] Neural odes. (). https://www.depthfirstlearning.com/2019/NeuralODEs.

[15] [n. d.] (). https://users.cs.duke.edu/~brd/Teaching/Previous/AI/Lectures/NN/neural.html.

[16] The Editors of Encyclopaedia Britannica. 2016. Ordinary differential equation. (2016). https://www.britannica.com/science/ordinary-differential-equation.

[17] [n. d.] Math insight. (). https://mathinsight.org/ordinary_differential_equation_introduction.

[18] Jae Hyuk Han. [n. d.] Comparing models for time series analysis. (). https://repository.upenn.edu/wharton_research_scholars/162/.

[19] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt and David Duvenaud. 2018. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*.