UNIVERSITY OF MORATUWA

Predictive Model for Gap Reduction Between Web Analytics and Business Strategy

by

P.H.A. Nissanka (158230N)

A thesis submitted in partial fulfillment of the requirements for the Degree of MSc in Computer Science specializing in Cloud Computing

in the
Faculty of Engineering
Department of Computer Science and Engineering

28th February 2019

Declaration

I declare that this is my own work and contains no material that has been published previously in whole or in part for the fulfillment of any degree program. All the referenced materials have been acknowledged in text.

Student:

P.H.A. Nissanka (158230N)

Supervisor:

Dr. Shantha Fernando

Abstract

Digital marketing and web analytics are two distinct areas that have captured the attention of many industrial firms. There are a lot of tools developed and a lot of studies carried out in each area separately. But still, a firms ability to harness web analytics to optimize digital marketing elements is limited. This work focuses on evaluating previous work in each of these areas and combine them to build a model that would define the relationship between digital marketing and web analytics. Data captured through each area is expected to be analyzed in the form of a time series forecasting problem. Time series forecasting is a very popular area that captured a lot of firms attention in recent years. This is due to the fact that most real-world problems are linked to a temporal component, and thus can be considered as a time series. Furthermore, this work utilizes cloud services for building and running the learning models.

Acknowledgements

I would like to express my sincere thank to my parents and my beloved wife for the continuous support given to manage my studies. And I would like to thank Dr. Shantha Fernando for the guidance and support given throughout the research to make it a success. Furthermore, I would like to thank Mr.Ashan Fernando for the encouragement and knowledge shared to continue this research until the end. Finally, my sincere appreciation goes to my colleagues at ShoutOUT Labs for the support given to manage my research work with office work.

Contents

List of Figures

List of Tables

Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
ARMA	**A**uto **R**egressive **M**oving **A**verage
ARIMA	**A**uto **R**egressive **I**ntegrated **M**oving **A**verage
AWS	**A**mazon **W**eb **S**ervices
DeepAR	**Deep A**uto **R**egressive
GCP	**G**oogle **C**loud **P**latform
GUI	**G**raphical **U**ser **I**nterface
IaaS	**I**nfrastructure **a**s **a** **S**ervice
K-NN	**K**-**N**earest **N**eighbour
LSTM	**L**ong **S**hort-**T**erm **M**emory
MAD	**M**ean **A**bsolute **D**eviation
MLMVN	**M**ulti **L**ayer **M**ulti **V**alued **N**eurons
MSE	**M**ean **S**quared **E**rror
PaaS	**P**latform **a**s **a** **S**ervice
QRF	**Q**uantile **R**andom **F**orest
RNN	**R**ecurrent **N**eural **N**etwork
S3	**S**imple **S**torage **S**ervice
SDK	**S**oftware **D**evelopment **K**it
TS	**T**ime **S**eries

Chapter 1

Introduction

Time series forecasting is an important area of machine learning. It is important because there are a lot of forecasting problems that involve a time component. Depending on the problem nature there are different types of approaches to follow. These vary from classical statistical approaches to neural network-based approaches. Web analytics is a good measurement to observe the performance of anything running on the web. It generates a lot of stationary data, that is time-dependent data which is often only used for observation purposes. Digital marketing is an area which can make an impact on data generated through web analytics. So, there is a relation between digital marketing and web analytics. However, there is no proper model that identifies this relation and optimize the usage of digital marketing activities to effectively improve the results from web analytics. There are a lot of studies carried out in the domain of time series forecasting, and it is proven that the selection of the approach has to be made based on the nature of the problem domain. This work focuses on evaluating different approaches for time series forecasting in the domain of web analytics using digital marketing as a feature to the data set.

1.1 Background

Web analytics generate time series data and the measured values can be considered as target values in a forecasting problem. Digital marketing activities also have a time component which in most cases define the time that particular activity was performed.

Here the marketing activities can be quantified and measured with respect to time and thus it produces a time series dataset, but this time series has no meaning if analyzed in isolation since its a human activity that is performed manually. However, when the digital marketing events are combined with web analytics data, the forecasting problem becomes feature rich and a model can be implemented to find the relationship between web analytics and digital marketing and that model can be used to optimize the usage of marketing activities to gain better results from web analytics.

Web analytics and digital marketing data can be combined to generate a single time series dataset. This is a multivariate time series forecasting problem. There are several works that have addressed these types of forecasting problems and among them, neural network-based approaches very popularly since they produce more reliable and accurate models. This work evaluates some of these approaches to find the best-suited approach for the problem domain.

1.2 Problem statement

The role that digital marketing plays in a firms marketing strategy has been expanded and the investments in digital marketing activities have increased[11]. Gartner reports[13] state that industrial firms allocate approximately one quarter (26%) of the total marketing budget for digital marketing activities. Web analytics data is often used by these firms to measure the online customers responses and behaviors to digital marketing stimuli[11][3]. These analytic observations can be used to optimize digital marketing elements. But still, the firm's ability to harness web analytics to improve the performance of marketing strategies remains limited. So, on average web analytics is utilized on an ad-hoc basis and the data captured are not used for strategic purposes[11]. There are a lot of tools available for both digital marketing and web analytics, but the challenge in most cases is that these are not linked and often operates separately.

Researches on digital marketing performance measurement with web analytics are theoretically underdeveloped[11]. Even though there are several other previous works done by researchers on these two areas separately, there is no proper method that would measure and quantify the outcome of web analytics as a result of digital marketing activities.

Thus there is no single model that would define the relationship between the web analytics and business strategy. So based on the literature this work focuses on building a single model that would define the relationship between digital marketing activities and web analytics to support making proper decisions on business strategy.

1.3 Research questions

In order to find a solution for the above-specified problem, in this research we raise the following questions:

1. What is the most suitable model to reduce the gap between business strategy and web analytics?
2. What is the best-suited model and the machine learning algorithm for reducing the gap between business strategy and web analytics?

1.4 Objectives and output

The main objective of this research is to come up with a strategy to reduce the gap between business strategy and web analytics. With this regard, this work tries to achieve the following objectives.

- Analyze web analytics mechanisms, tools, and technologies
  The main objective here is to identify the data sources which generate the data related to the problem domain as stated in section 1.2
- Map business strategies to a machine-readable format to analyze
  Digital marketing activities need to be converted to a time series dataset in order to be analyzed and modeled as a forecasting problem.
- Analyze and evaluate learning algorithms and models for predicting analytics based on the multiple features (business features)
  The datasets collected through web analytics and digital marketing are combined resulting in a time series dataset with multiple variables. Based on the problem

domain and nature of the dataset, a proper learning approach need to be found to build the most suitable forecasting model.

- Find the most suitable approach to reduce the gap between business strategy and web analytics using the identified model.
  The model generated above then needs to be used to derive an approach for making digital marketing decisions which optimizes what the generated through web analytics.

By achieving the above objectives this work focus on building a machine learning model which can be used to reduce the gap between business strategy and web analytics.

1.5 Research strategy

In order to achieve the objectives and generate the expected output, this work was executed according to the following steps.

1. Literature survey on web analytics methods, tools, and technologies
   This work mainly built on top of the data generated through web analytics. So, it is important to identify the sources from which these data can be collected and effectively use them to extract the relevant datasets.

2. Literature survey on learning models for time series forecasting
   The dataset collected in step 1 needs to be used to build a model that defines the relationships between web analytics and digital marketing activities. Thus, it is required to find a proper learning approach to derive the best-suited model

3. Transforming the data to the format required by the selected learning models
   The dataset collected in step is expected to have unfiltered raw data, and the learning models cant be applied to unfiltered data. Thus, it is required to filter and normalize the data as required by the learning model selected in step 2.

4. Test and evaluate the performance and accuracy of the models after running on the dataset.
   Once a model is trained using the dataset generated from step 3, it is important

to test it against similar datasets to evaluate the performance and accuracy of the forecasted values. This is a costly process and needs to be carried out in a specialized environment.

1.6 Thesis outline

The rest of the chapters of this document has been structured in the following manner. Chapter 2 contains a literature survey on various web analytics methods, tools, and technologies. And various machine learning algorithms and models for time series prediction. Chapter 3 elaborates the methodology followed to set up the development environment, transform the dataset to the format required, how the models were trained and tuned for the context using the transformed data. Then Chapter 4 elaborates the detailed steps carried out in order to build the solution and Chapter 5 elaborates a performance evaluation done on the built model. Finally, it describes how the performance and accuracy were measured.

Chapter 2

Literature Survey

Web analytics is an important area of interest for anything running on the web. It provides detailed reports on how the user behavior patterns. Digital marketing is also an important area which is used to enhance and increase what is measured through web analytics for business purposes. This chapter provides a survey of the previous works in these areas.

2.1 Web analytics tools

Azim, M. and Hasan, N. in their study Web Analytics Tools[3] have done a survey among Indian library professionals to analyze the usage of web analytics tools in library websites. The main objective of their study was to investigate web analytics tools used by Indian libraries and analyze their website metrics. Azim et al define web metrics/web analytics as a measurement of web site metrics or web data for the purpose of understanding the usage pattern of a website. Through a comprehensive survey, Azim et al have observed that Google Analytics is the most popular and widely used tool for measuring site analytics. However, its a time taking process and requires more people to analyze the large data for further improvement and enhancement of the site. Web analytics data can be used to understand online customer behavior, to measure online customer responses to digital marketing stimuli[15]. Thus, the digital marketing elements can be optimized to improve the customer behavior that benefits the business[15]. However, still, the firms ability to harness web analytics to improve marketing performance remains limited[15].

So still on average, the web analytics is utilized on an ad-hoc basis and the metrics are not used for strategic purposes. Predicting website visiting patterns have a significant value for every site owner to target their business for the right customers at right time[18]. However, most of the available web analytics tools dont support forecasting features, and forecasting includes only in specialized tools like Business Intelligence (BI) tools which are expensive and not affordable to small and medium-sized business owners[18].

2.2 Time series forecasting

A time series (TS) is a sequence of historical events of an observable variable at equal time intervals[4]. Prediction of future behavior using these historical events is called forecasting[18], and the forecasting knowledge helps in many ways in various domains such as controlling load balance, predicting future marketing campaigns, allocating and deallocating resources and caching, prefetching web pages for improve performance[18]. In a TS, ordering of the data points is important and thus the standard propositional learning algorithms cant process them directly unlike in other forecasting problems where the data points are independent[18]. One approach to overcome this challenge is to remove the temporal ordering from the TS and apply the so-called algorithms[18]. And another approach is to use algorithms that preserve the temporal order of the TS. Gianluca B. et al in their study, Machine Learning Strategies for Time Series Forecasting[4] have reviewed machine learning techniques that are being used for time series forecasting. They state that linear statistical methods such as ARIMA models have influenced the forecasting domain for a long time. However, since linear models were not adapted to many real-world applications, non-linear models such as the bilinear model, threshold autoregressive model, and autoregressive conditional heteroscedastic (ARCH) model were proposed. But these models are still at their early stages. However, machine learning models were introduced during the last two decades and captured a lot of attention compared to statistical models. Gianluca B. et al state that it was found that the Artificial Neural Networks (ANNs) outperforms the classical statistical models such as linear regression. Later the models such as decision trees, support vector machines, and nearest neighbor regression were introduced.

2.3 Components of a time series

Time-series can be explained as an aggregation or combination of four primary components[5]. They are Trend, Seasonality, Level (Cyclical) and Noise[14]. Here the trend component is the long-term tendency to rise or fall (upward or downward), seasonality component is the periodic fluctuation which forms a pattern from one season to the other, level or the cyclical component is the average of the time series and the noise component is the random variation in the series. Trend, seasonality and level components have consistency or recurrence and can be described or modeled and thus considered as systematic components. Noise component cant be modeled directly and thus considered as a non-systematic component[5]. The composition of these components can be denoted as an additive model or a multiplicative model. The addictive model assumes the four components are independent and the multiplicative model assumes that the four components are dependent.

Additive model

$$Ut = Tt + St + Lt + Nt \tag{2.1}$$

Multiplicative model

$$Ut = Tt * St * Lt * Nt \tag{2.2}$$

Ut - Time Series, Tt - Trend, St - Seasonality, Lt - Level, Nt - Noise

An additive model is linear where the changes over time are consistently variated by the same amount. Here the trend is a straight line and seasonality has the same frequency and amplitude. And a multiplicative model is nonlinear where the changes increase or decrease over time. In this case, the trend is a curved line and seasonality have increasing or decreasing frequency and/or amplitude over time. Figure 2.1 below shows a sample decomposition of an additive time series.

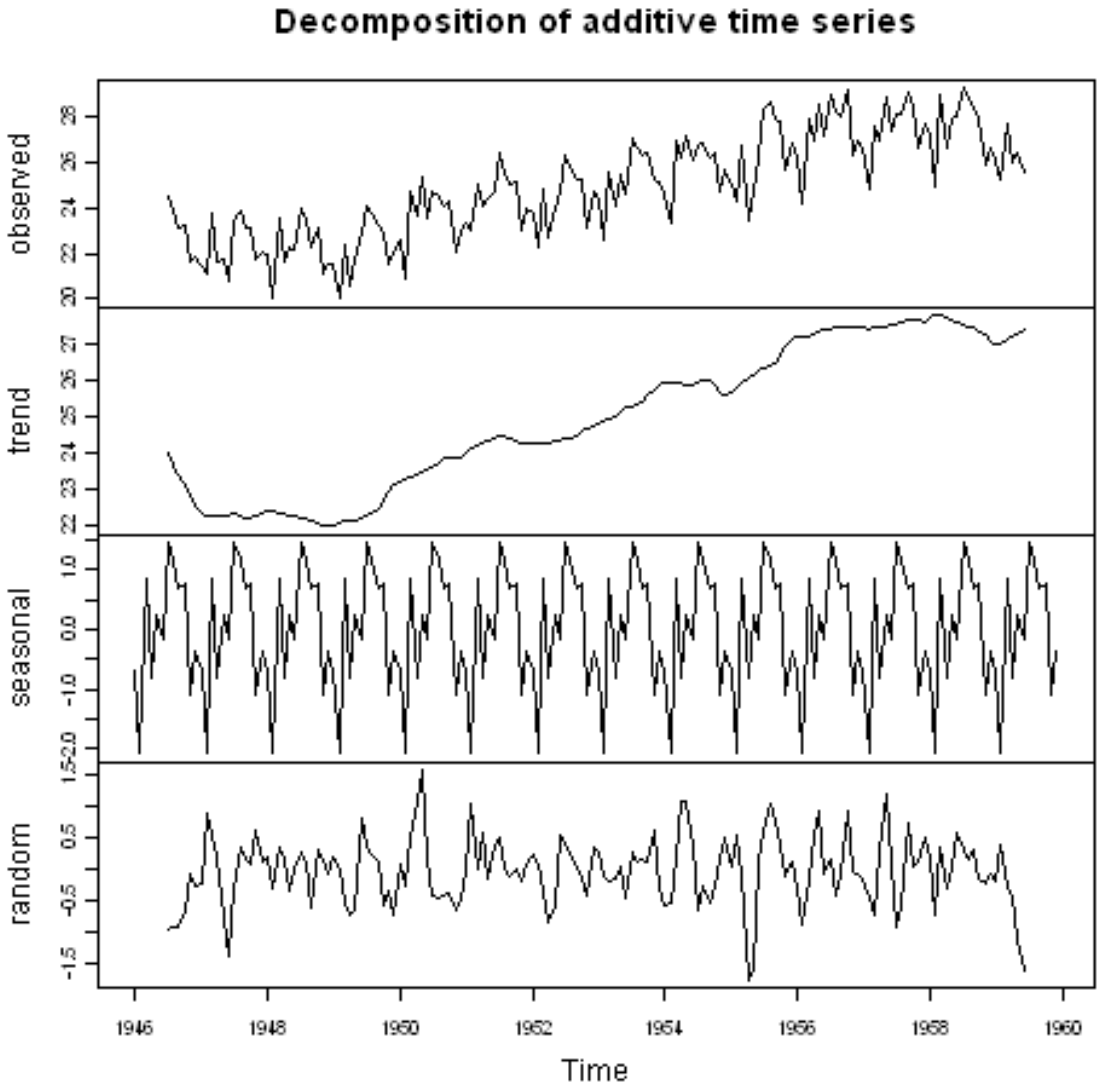

FIGURE 2.1: Decomposed time series

2.4 Feature engineering

Feature engineering is the re-framing of time series data as a supervised learning dataset to be used with learning algorithms[6]. There is no concept of input and output features in a time series, so we have to choose the variable to be predicted and use feature engineering to construct all of the inputs that can be used to make the predictions of future time series steps of the selected variable[6].

Most of the time datasets contain irrelevant and/or redundant features. So pre-processing techniques like Feature Subset Selection (FSS) need to be used to identify a subset of original input features [24]. Feature Extraction (FE) on the other hand is the process of deriving new features by linearly or non-linearly mapping the original input features

into more effective ones[24]. Yoon H. et al in their work propose a new feature selection technique for multivariate time series datasets where spatial features are much larger than temporal observations.

2.5 Forecasting methodologies

Forecasting methodologies can be grouped into three main categories as classical and statistical, machine learning based and as a hybrid.

2.5.1 Classical, statistical methodologies

Forecasting domain has been influenced by linear statistical models for a long time. These models have been adapted to many real-world applications as well[4]. Mentioned below are few such popular models that are widely used and attracted the attention of many other types of research works.

- Autoregressive integrated moving average (ARIMA)
- Bayesian inferences
- K-Nearest Neighbor (K-NN) predictors
- Exponential smoothing methods (Holt-winters, Theta)

Classical statistical methodologies are often used for building linear forecasting models. The predictions of the future values of these models are constrained to be linear functions of past observations[25]. Autoregressive Integrated Moving Average is one such linear forecasting model which gained popularity due to its statistical properties and the usage of Box-Jenkins methodology during the model building process[25]. ARIMA models can be used to build various exponential smoothing models[25]. However, due to the pre-assumed linear correlation structure among the time series values, ARIMA models are not capable of capturing nonlinear patterns. ARIMA is a generalization of the simpler autoregressive moving average (ARMA) models, with an addition of the notion of integrated. ARIMA modeling is a stochastic process coupling autoregressive component

(AR) to a moving average (MA) component[21]. Its a class of model that captures a suite of different standard temporal structures in time series data.

Bayesian inference is also a classical technique which uses evidence or observations to update or newly infer the probability that a hypothesis may be true [21]. Bayes is a useful theorem to estimate the probability that the time series is in the state yk at time t[21].

The K-Nearest Neighbors algorithm (K-NN) is a method for classifying objects based on the closest training examples in the feature space. K-NN is a type of instance-based learning or lazy learning where the function is only approximated locally, and all computation is deferred until classification. It can also be used for regression[21].

2.5.2 Machine learning methodologies

The real-world time series are often complex and nonlinear and thus make it hard to be modeled using linear methodologies as mentioned in section 2.4.1. As a solution to this limitation, machine learning models were introduced. These models are also called black-box or data-driven models[4]. Mentioned below are a few popular examples of machines learning models.

- Quantile random forest (QRF)
- Support vector regression (SVR)[22]
- Recurrent neural networks (RNN)
  - LSTMs
  - DeepAR
  - GRUs

An Artificial Neural Network (ANN) is a network of computing units linked by directed connections[12]. These connections normally have weights that correspond to how strong two units are linked. Computation performed by a unit is separated into two stages as aggregation and activation. Applying the aggregation function commonly corresponds to calculating the sum of the inputs received by the unit through all its incoming connections and applying the activation function corresponds to calculating the final result

from the aggregated result. ANNs offer an attractive alternative by providing nonlinear parametric models compared to conventional algorithms based linear models[21]. Christophe P. et al in their work state that Multi-Layer Perceptron (MLP) network has been the most used ANN architecture in the renewable energy domain[21]. They have used an ad hoc time series pre-processing step before using neural networks. MLP or in other words feedforward neural networks is an extension of the linear regression models[9].

Classical time-series models are popular for univariate forecasting[17]. When there are exogenous variables, machine learning approaches like Quantile Random Forest are used[17]. Even though QRFs are accurate, they are not flexible enough and does not scale when there are a high retraining frequency[17]. To effectively deal with exogenous variables Opitz T. et al have proposed a combination of univariate modeling and a machine learning model to handle residuals[19]. However, this approach requires manual feature extraction and is hard to tune[17].

Deep learning is a part of the broader family of the machine learning methods. And recently due to the groundbreaking successes in other areas, deep learning techniques were applied to time series forecasting[9]. Recurrent Neural Networks (RNNs) is a deep learning technique which gained a lot of attention and popularity due to its ability to handle sequence dependence. The Long Short-Term Memory network (LSTM) is a type of RNN which gained popularity due to its ability to end-to-end modeling, ease of incorporating exogenous variables and automatic feature extraction[2]. Nikolay L. et al have proposed a new LSTM based architecture and train a single model using heterogeneous time-series since the vanilla LSTM models performance is poor[17]. Neural networks are sensitive to unscaled data, so it is important to normalize before feeding into a model[17].

Valentin et al[10] proposes a forecasting method named DeepAR which is based on autoregressive recurrent neural networks. It tailors a similar LSTM based neural network architecture to the probabilistic forecasting problem. Compared to the classical approaches and other global method, DeepAR models learn seasonal behavior and dependencies and gives covariates between time series. DeepAR requires a minimal feature engineering to capture complex, group-dependent behavior[10]. It makes probabilistic forecasts in the form of Monte Carlo samples that can be used to compute consistent

quantile estimates for all sub-ranges in the prediction horizon[10]. DeepAR also able to provide forecasts for an item with little or no history at all[10].

Multivariate time series data sometimes contain missing values, and this is referred to as informative missingness[8]. These missing values and patterns might give rich information about target labels in a supervised learning task [8]. Zhengping C. et al in his work propose a novel deep learning model namely Gated Recurrent Unit (GRU) which is a state-of-the-art RNN. GRU takes two representations of missing patterns and effectively incorporate them into a deep model architecture so that it captures long-term temporal dependencies and utilizes the missing patterns to achieve better prediction results[8].

For complex real-world problems that require long term forecasting, Igor A. et al propose a multilayer neural network with multi-valued neurons (MLMVN)[1]. They state that machine learning-based prediction models are especially good for time series governed by nonlinear dynamics. Here the main distinction from the classical feed-forward neural network is that MLMVN consists of multi-valued neurons with complex-valued weights[1].

2.5.3 Hybrid methodologies

It is difficult to know the exact characteristics of a real-world forecasting problem. So, its not wise to apply a statistical method or a machine learning method blindly. However, it is reasonable to consider a time series is composed of a linear autocorrelation structure and a nonlinear component[25]. Zhang and Kashei have proposed hybrid methodologies[25][16] which combines ARIMA and ANNs for better performance with modeling both linear and nonlinear forecasting problems. Combining different models can increase the chance to capture different patterns in data improve performance. Deep generative models are examples for hybrid models which combine the strengths of both RNNs and classical probabilistic graphical models[9].

2.6 Evaluation methodologies

Evaluation methodologies for time series can be grouped into three main areas as performance evaluation, model evaluation, and uncertainty estimation.

2.6.1 Performance evaluation

There are different approaches to measure the performance of a model. Mentioned below are a few methods can be used. Measure forecast performance

- The Mean Forecast Error (MFE)
- The Mean Squared Error (MSE)
- The Root Mean Squared Error (RMSE)
- The Normalized Mean Squared Error (NMSE)
- The Coefficient of Determination (R2)
- Theils U

Once the model is built the next obvious requirement is to measure the accuracy and performance. There are several methods to measure the forecasting accuracy. Among these methods, the popular ones are the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). Mentioned below is how the MSE and RMSE are calculated[7].

Mean Square Error (MSE)

$$MSE = mean(forecast\_error^2) \tag{2.3}$$

Where;

$$forecast\_error = expected\_value - predicted\_value \tag{2.4}$$

Root Mean Squared Error (RMSE)

This is calculated by taking the square root of the mean squared error

$$RMSE = \sqrt{MSE} \tag{2.5}$$

2.6.2 Model evaluation

Evaluating time series models is called backtesting or hindcasting. The methods used here need to be aligned with the temporal components of the time series. The time dimension of the observations means that we need to split the data up respecting the temporal order in which the values were observed. Mentioned below are three different methods that can be used to backtest machine learning models on time series problems.

1. Train-test split that respect temporal order of the observations
2. Multiple train-test split that respect temporal order of the observations
3. Walk-forward validation

In train-test split, an arbitrary split point is selected from the ordered dataset and all data up to the split point is used for training and data after the split point is used for testing. In the second method, multiple split points are selected and use different dataset combinations for training and testing. In the third method, training is continuously running as the new data points are collected. Selection of the training dataset window can be done in one of the following ways[20],

1. Sliding window
2. Expanding window

In the sliding window approach, we decide a window size and move it along the temporal component as the new data points are collected. So, the training dataset is selected from the recent dataset. In the expanding window, we use all the data from the start until the recent observation. Figure 2.2 below shows a graphical view of the window along with the selection of training and test datasets.

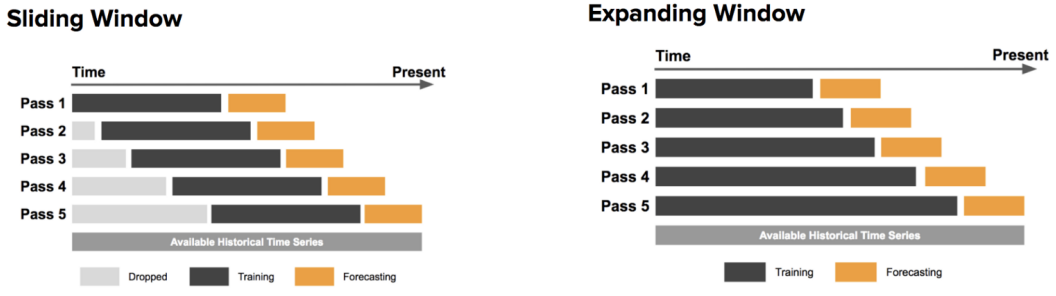

FIGURE 2.2: Window selection in walk forward validation

2.6.3 Uncertainty estimation

Time series problems which are probabilistic in nature require robust uncertainty estimation when using neural network-based approaches[17]. Combination of Bootstrap and Bayesian approaches can be used to produce a simple, robust and tight uncertainty bound with good coverage and provable convergence properties[17].

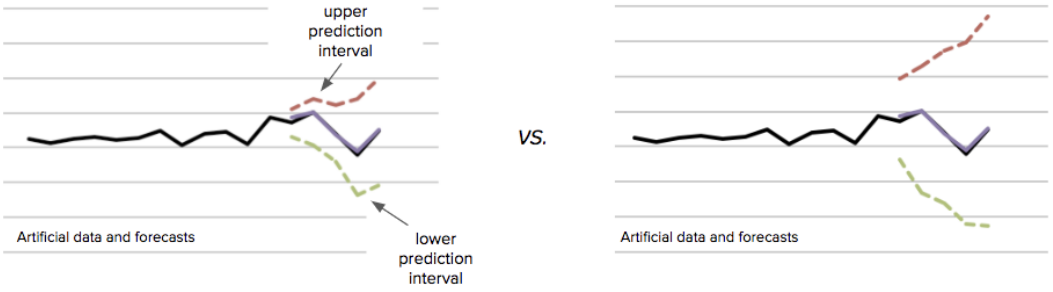

FIGURE 2.3: Uncertainty estimation

2.7 Summary

Time series forecasting has been an area of interest to many firms and researches. It has been applied in various business domains. Since most of the real-world forecasting problems are multivariate and dynamic in nature, machine learning approaches are most popular compared to classical statistical methods. Machine learning models are mainly based on artificial neural networks (ANN) and among them, the recurrent neural networks which are a type of ANN is very popular due to their ability to handle sequence dependence. Both LSTM and DeepAR are RNN based learning models. However, DeepAR can be identified as an improved version of LSTM. And compared to LSTM, DeepAR is capable of training a single model jointly over multiple target time series. It makes probabilistic forecasts in the form of Monte Carlo sample and requires minimal feature engineering to capture complex group dependent behavior. DeepAR models learn seasonal behaviors and able to provide forecasts for items with no history at all. Considering all these facts, DeepAR is the most suitable method for the problem domain been addressed in this work.

Chapter 3

Methodology

Based on the literature revision in Chapter 2, DeepAR forecasting algorithm was selected to train a model that would define the relationship between digital marketing activities and web analytics. For the purpose of training the model, a publicly available Google analytics dataset which contained both digital marketing related data and web analytics data was used. This dataset can be considered as a time series dataset since the data points contain a temporal component.

Amazon Web Services (AWS) which is an Infrastructure as a Service (IaaS) provides a Platform as a Service (PaaS) named Sagemaker for building, training and running machine learning models in the cloud. This platform provides on-demand compute resources which is a challenge in certain cases when running compute-intensive training jobs in local environments. The trained models can be hosted within Sagemaker itself to be used for making predictions. Furthermore, most importantly Sagemaker provides a built-in implementation of the DeepAR forecasting algorithm which can be used out of the box for training models.

Python was selected as the programming language and Jupyter notebook was selected as the development environment. Sagemaker provides Jupyter hosted Jupyter notebook instances as a service. However, since the cost of running the hosted Jupyter notebook is high, a local installation of the notebook was used. So both cloud-hosted tools and locally installed tools were used for the implementation. Figure 3.1 below shows a high-level architecture of the components/tools that were used and how they were connected.

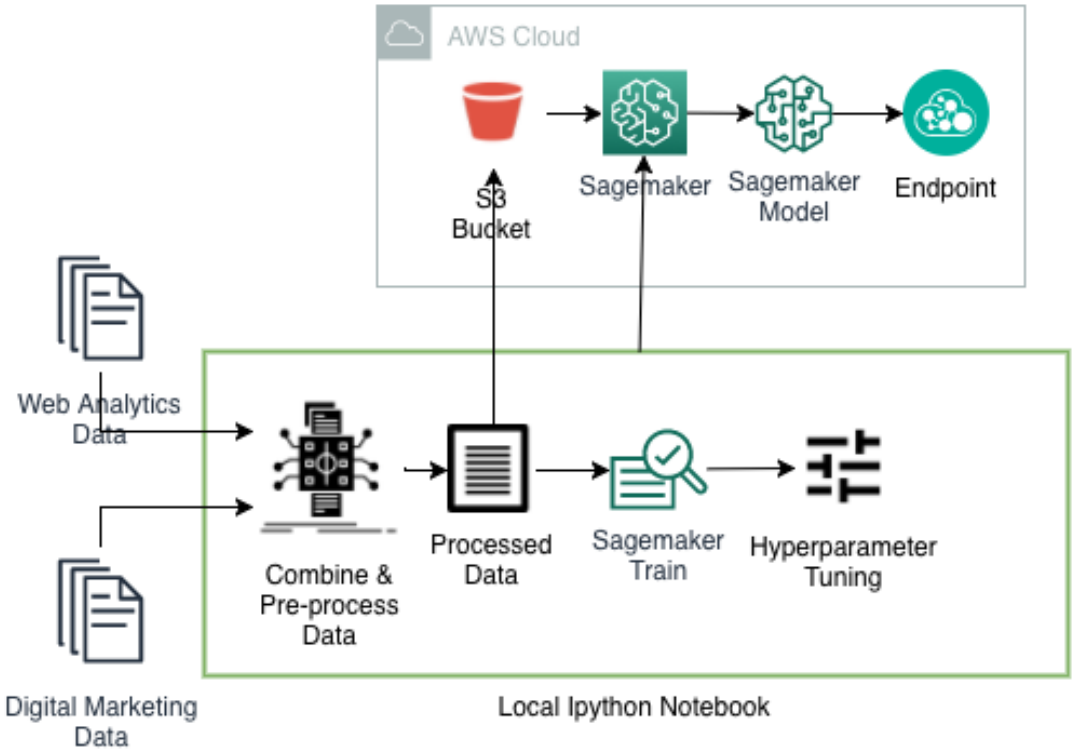

FIGURE 3.1: High level architecture

3.1 Setting up the environment

The environment that was used for this work was a hybrid environment. Both the local environment and cloud environment was used in combination to observe the results faster and deploy the trained model to be used in production systems. Local environment was used to analyze the data set, prepare the data set and initiate processing in the cloud. And the cloud environment was used to train the model and deploy the trained model for making predictions in production systems.

3.1.1 Setting up the local environment

Python was selected as the programming language for the implementation and Jupyter notebook was selected as the development environment for python. Jupyter is a web-based application which allows to create and share live code, equations, visualizations, and narrative text. Anaconda navigator was used to installing Jupyter notebook since

it provides a desktop graphical user interface (GUI) for managing tools like Jupyter and manage the required dependencies easily in an isolated environment.

3.1.2 Setting up the cloud environment

Amazon web services (AWS) was selected as the cloud infrastructure provider due to the comprehensive platform as a service (PaaS) tools it offers especially for machine learning and the attractive pricing model for on-demand resources. Sagemaker is a PaaS platform in AWS which offers a fully-managed environment for building, training and deploy machine learning models at any scale. In this work, it was used mainly for training and deploying the model.

3.2 Finding the dataset

The problem domain considered in this work is an online business which has a website to offer products/services for customers. Here the page visits of the company website is a time series and the business strategies that carried out to increase the page visits can be considered as the features (variables) of the time series. Google analytics dataset is a public data set published by Google which contains the analytics data collected from Google merchandise store which is an e-commerce platform that sells Google-branded merchandise. This dataset contains information about the traffic source, type (organic, paid or search) and the behavior of the users on the site. Dataset has been published as a Google big query table which continuously gets updated with new data. For the scope of this work, a subset of data was downloaded into CSV files using Google big query driver package for python and pandas python package. For the rest of the process, these downloaded CSV files were read again using the Pandas package.

3.3 Analyze and prepare the dataset

3.3.1 Filter the dataset

Filtering process was to remove the unwanted data attributes from the dataset and shape the data into the format required for the rest of the operations. Python pandas

library was used for this purpose. Table 3.1 below shows a sample of the dataset after the filtering and transformation process.

TABLE 3.1: Transformed dataset

2017-06-27 07:00:00	72	8
2017-06-27 08:00:00	95	13
2017-06-27 09:00:00	86	5
2017-06-27 10:00:00	96	0
2017-06-27 11:00:00	76	2
2017-06-27 12:00:00	127	5
2017-06-27 13:00:00	104	5
2017-06-27 14:00:00	135	8
2017-06-27 15:00:00	166	7
2017-06-27 16:00:00	145	9

3.3.2 Visualize the dataset

The transformed dataset was then visualized using the Matplotlib python package. Following figure 3.2 shows the hourly campaign count variation over time.

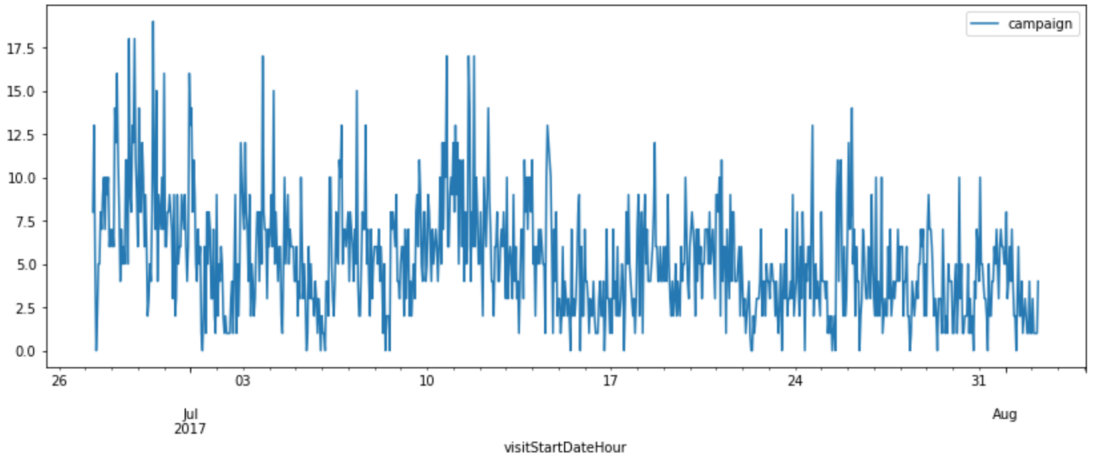

FIGURE 3.2: Campaign count variation over time

Figure 3.3 below shows the hourly web site visits variation over time.

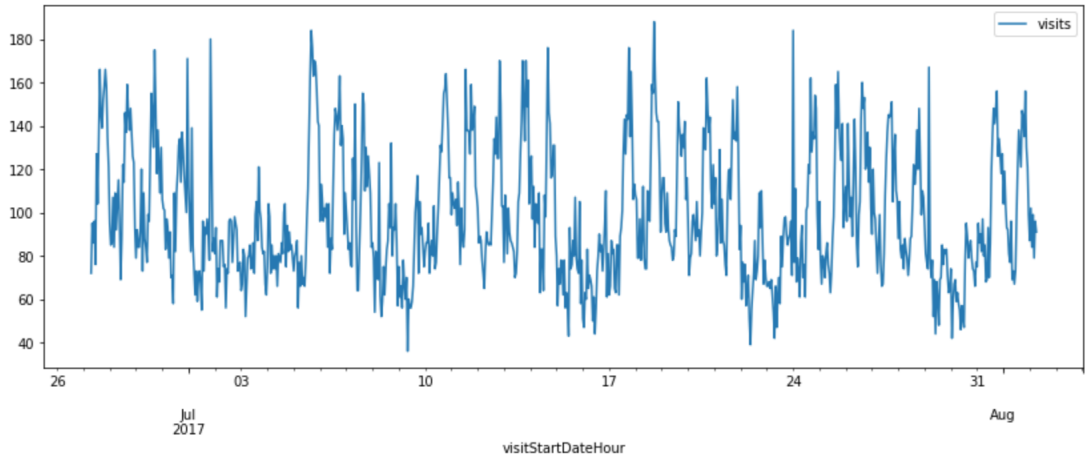

FIGURE 3.3: Website visits variation over time

3.3.3 Transform the dataset

DeepAR algorithm provided within AWS Sagemaker requires the input to be in JSON lines format as mentioned in Figure 3.4 below[23], for the training a model.

```
{
    "start": "2009-11-01 00:00:00",
    "target": [4.3, "NaN", 5.1, ...],
    "cat": [0, 1],
    "dynamic_feat": [[1.1, 1.2, 0.5, ...]]
}
{
    "start":
    "2012-01-30 00:00:00",
    "target": [1.0, -5.0, ...],
    "cat": [2, 3],
    "dynamic_feat": [[1.1, 2.05, ...]]
}
{
    "start": "1999-01-30 00:00:00",
    "target": [2.0, 1.0],
    "cat": [1, 4],
    "dynamic_feat": [[1.3, 0.4]]
}
```

FIGURE 3.4: DeepAR input format

The filtered dataset was transformed into the above format using Numpy and Pandas python libraries. During this process, web site visits property was selected as the target value and campaigns count was selected as a dynamic feature. Since the whole dataset was from the same category, the property cat in the above input structure was not used for training the model.

3.4 Training the learning models

The transformed dataset was then separated into two sub-datasets as training and test. Figure 3.5 below show the separation of the dataset in time series plots.

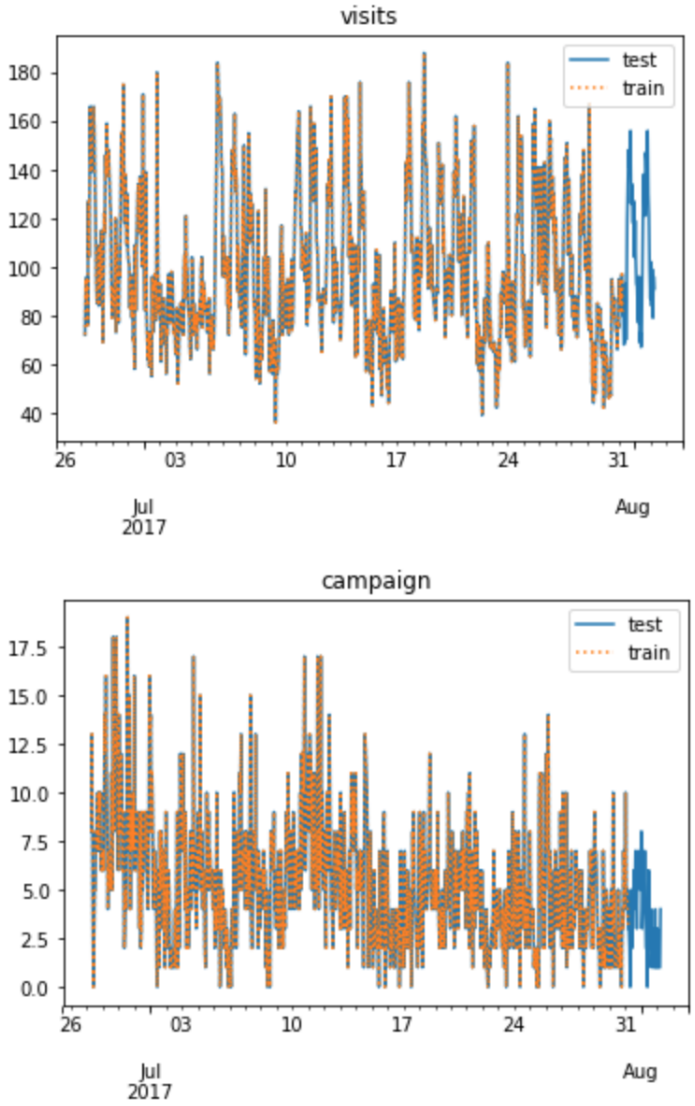

FIGURE 3.5: Dataset separation as training and test

In this work number of web site visits property has been selected as the target value to be predicted and campaigns count has been selected as a dynamic feature. AWS Sagemaker service provides pre-built images for time series forecasting using DeepAR, and furthermore, it offers on-demand compute resources for faster training of the models. As an added advantage Sagemaker facilitate to easily expose an endpoint for production access using the trained models. So, its a convenient environment for training and running machine learning models in production. Sagemaker uses Amazon Simple Storage Service (S3) for reading input dataset and writing back the output and results generated during the training process. So the transformed dataset was uploaded to an S3 bucket from the local environment via the Jupyter notebook. Next using the Sagemaker python Software Development Kit (SDK) via the local Jupyter notebook. Then a Sagemaker estimator was initiated. DeepAR pre-built image support certain hyperparameters to be configured to adjust the training process according to the use case. Table 3.2 shows the tunable hyperparameters supported by DeepAR sorted according to the most to least impact.

TABLE 3.2: DeepAR Tunable Hyperparameters

Parameter Name	Parameter Type	Recommended Ranges
mini_batch_size	IntegerParameterRanges	MinValue: 32, MaxValue: 1028
epochs	IntegerParameterRanges	MinValue: 1, MaxValue: 1000
context_length	IntegerParameterRanges	MinValue: 1, MaxValue: 200
num_cells	IntegerParameterRanges	MinValue: 30, MaxValue: 200
num_layers	IntegerParameterRanges	MinValue: 1, MaxValue: 8
dropout_rate	ContinuousParameterRange	MinValue: 0.00, MaxValue: 0.2
embedding_dimension	IntegerParameterRanges	MinValue: 1, MaxValue: 50
learning_rate	ContinuousParameterRange	MinValue: 1e-5, MaxValue: 1e-1

According to the use case, an initial set of hyper-parameters were selected and the model training process was launched providing the path to the training and test set in S3.

DeepAR algorithm produce three types of metrics which are calculated during the training process as a measure of accuracy and model performance. Table 3.3 shows these three metrics along with a description of what each metrics denotes.

TABLE 3.3: Metrics Computed by DeepAR Algorithm

Metric Name	Description	Optimization Direction
`test:RMSE`	The root mean square error between the forecast and the actual target computed on the test set.	Minimize
`test:mean_wQuantileLoss`	The average overall quantile losses computed on the test set. To control which quantiles are used, set the `test_quantiles` hyperparameter.	Minimize
`train:final_loss`	The training negative log-likelihood loss averaged over the last training epoch for the model.	Minimize

3.5 Hyperparameter tuning

In order to tune the model for best accuracy and performance Sagemaker provides a feature called Automatic Model Tuning or in other words hyperparameter tuning where the optimum values for the hyperparameters are selected by running multiple training jobs using a range of hyperparameter values validating against a selected metric (objective metric) from the three metrics mentioned in the table 3.3 above. For the purpose of this work, Root Mean Squared Error (test: RMSE) between the forecast and the actual target computed was selected as the objective metric and its optimization direction Minimize was configured as the objective type. Figure 3.6 below shows the hyperparameter ranges used for tuning as recommended for DeepAR algorithm.

```
'num_cells': IntegerParameter(min_value=30,max_value=200),
'epochs':IntegerParameter(min_value=1,max_value=1000),
'mini_batch_size':IntegerParameter(min_value=32,max_value=1028),
'context_length':IntegerParameter(min_value=1,max_value=200)
```

FIGURE 3.6: Selected hyperparameters

Automatic tuning was launched with a total of 30 jobs, running 5 jobs parallelly at a time. After running the tuning process, the following hyperparameters (Table 3.4) were reported as the most suitable combination for the best results.

TABLE 3.4: Best Training Job Hyperparameters

Name	Type	Value
_tuning_objective_metric	FreeText	test:RMSE
context_length	Integer	140
dropout_rate	FreeText	0.05
early_stopping_patience	Freetext	10
epochs	Integer	932
learning_rate	Freetext	0.001
likelihood	FreeText	gaussian
mini_batch_size	Integer	64
num_cells	Integer	181
num_layers	FreeText	3
prediction_length	FreeText	48
sagemaker_estimator_class_name	FreeText	"Estimator"
sagemaker_estimator_module	FreeText	"sagemaker.estimator"
time_freq	FreeText	H

3.6 Testing trained model

Sagemaker has a feature for creating endpoints using the trained models to use in a production environment. So, the tuned model mentioned above was used to create a predictor endpoint to test the model using datasets.

Figure 3.7 below shows the results received from the prediction.

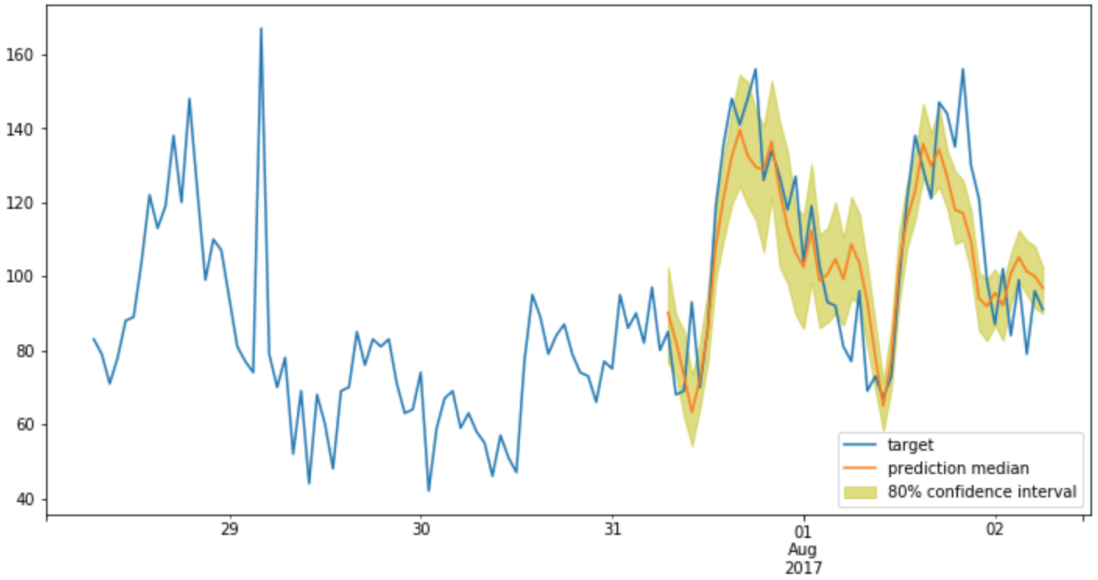

FIGURE 3.7: Prediction Results

Figure 3.7 above shows the target value and the prediction median along with a confidence interval colored in yellow color.

Chapter 4

Implementation

The high level architecture showed in figure 3.1 above can be further elaborated by breaking in to the key activities. And the figure 4.1 below shows the activities sequence that was performed to train the model.

The development environment used for the implementation was Jupyter notebooks running on a local computer. And as mentioned in Chapter 3, the programming language was python. Jupyter python environment comes with several pre-packaged libraries and modules to be used. However, in this case, there were certain libraries missing and they were installed separately. They were mainly to interact with AWS Sagemaker environment, Google cloud environment, and scientific computations. The implementation of the solution shown in the high-level architecture (figure 3.1) was broken into three main components. The first component was the data loader, the second component was the model builder and the third component was the evaluator. More details about each of these components are elaborated below.

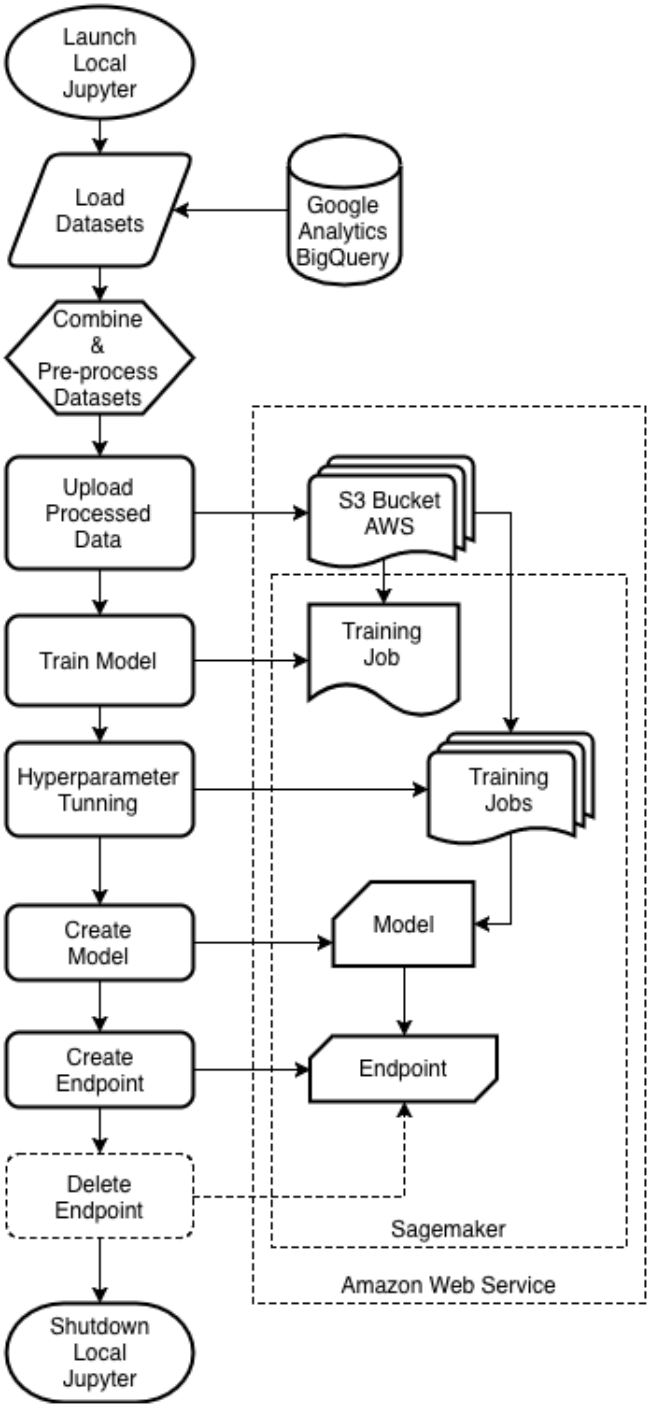

FIGURE 4.1: Activities diagram of the process

4.1 Data loader

The datasets used had been stored in google big query which is a serverless data warehousing service offered by Google Cloud Platform (GCP). So the data loader is mainly responsible for connecting to google big query, downloading the required datasets and

storing them locally as files. The pseudocode in figure 4.2 below briefly explains the implementation of the data loader.

```
# Establish Big Query Helper Object for data scanning
google_analytics = bq_helper.BigQueryHelper(active_project="bigquery-public-data",
                                            dataset_name="google_analytics_sample")

# Create list of tables to later assist with queries
tablelist = google_analytics.list_tables()

# Export tables to files
for tablename in tablelist:
    query_oneTable = """
    #standardSQL
    SELECT *
    FROM
        # enclose table names with WILDCARDS in backticks `` , not quotes ''
        `bigquery-public-data.google_analytics_sample."""" + tablename+"""`
    """
    google_analytics.estimate_query_size(query_oneTable)
    oneTable = google_analytics.query_to_pandas_safe(query_oneTable, max_gb_scanned=.5)
    file_name = 'datasets-20190114/' + tablename + '.csv'
    oneTable.to_csv(file_name, encoding='utf-8', index=False)
```

FIGURE 4.2: Data loader pseudocode

Refer Appendix A for the complete code of the pseudocode give in figure 4.2.

4.2 Model builder

The model builder was mainly responsible for reading the files stored by data loader, pre-process and filter removing the unwanted data attributes and upload them to an AWS S3 bucket. The most importantly connect with AWS Sagemaker environment and initiate the training process in the AWS cloud. Here the training process reads the input dataset from the AWS S3 bucket. Once the model is trained, the hyperparameter tuning job is initiated to determine the best hyperparameter combination for the best accuracy of the model. Finally, a predictor endpoint is created using the tuned model for making predictions with new datasets. The pseudocode in figure 4.3 below shows a summarized view of the implementation.

```
# Preparing data
googledata['visitStartDateHour'] = googledata['visitStartTime'].apply(convert_to_date_hour)
googledata['hits'] = googledata['totals'].apply(map_hits)
googledata['visits'] = googledata['totals'].apply(map_visits)
googledata['pageviews'] = googledata['totals'].apply(map_pageviews)
googledata['campaign'] = googledata['trafficSource'].apply(map_campaign)

mapped_googledata = googledata.loc[:,['visitStartDateHour','visits','campaign']]
grouped_googledata = mapped_googledata.groupby(['visitStartDateHour']).agg({'visits':'sum','campaign':'sum'})
grouped_googledata = grouped_googledata.sort_values(by='visitStartDateHour')

# Train model

estimator = sagemaker.estimator.Estimator(
    sagemaker_session=sagemaker_session,
    image_name=image_name,
    role=role,
    train_instance_count=1,
    train_instance_type='ml.c4.2xlarge', # $0.279/hour
    base_job_name='research-ga-deepar',
    output_path="s3://" + s3_output_path
)

data_channels = {
    "train": "s3://{}/train/".format(s3_data_path),
    "test": "s3://{}/test/".format(s3_data_path)
}

estimator.fit(inputs=data_channels)

# Hyperparameter tuning

# Configure HyperparameterTuner
my_tuner = HyperparameterTuner(estimator=estimator,  # previously-configured Estimator object
                               objective_metric_name='test:RMSE',
                               objective_type='Minimize',
                               hyperparameter_ranges={
                                   'num_cells': IntegerParameter(min_value=30,max_value=200),
                                   'epochs':IntegerParameter(min_value=1,max_value=1000),
                                   'mini_batch_size':IntegerParameter(min_value=32,max_value=1028),
                                   'context_length':IntegerParameter(min_value=1,max_value=200)

                               },
                               max_jobs=30,
                               max_parallel_jobs=5)

data_channels = {
    "train": "s3://{}/train/".format(s3_data_path),
    "test": "s3://{}/test/".format(s3_data_path)
}

# Start hyperparameter tuning job
my_tuner.fit(inputs=data_channels)
```

FIGURE 4.3: Model builder pseudocode

Refer Appendix B for the complete code of the pseudocode given in figure 4.3.

4.3 Model evaluator

The model evaluator was an additional component implemented which is mainly for evaluating the created model by making predictions on different datasets which were not

used for the training process. Endpoint created by the model builder is used here for making the predictions. Evaluator uses a sliding window method for running predictions on the new datasets. The pseudocode in figure 4.4 below shows a summary view of the implementation.

```
# Create instance of the predictor
endpoint_name = 'forecasting-deepar-190113-1413-023-1e96cb06'
predictor = DeepARPredictor(
    endpoint=endpoint_name,
    sagemaker_session=sagemaker_session,
    content_type="application/json"
)
predictor.set_prediction_parameters(freq, prediction_length)

# Create sliding windows from the dataset
Input = collections.namedtuple('Input', ['ts', 'dyf'])

def moving_window(x, y, length, pred_length):
    return [Input(ts=x[i: i + length], dyf=y[i: i + length + pred_length]) for i in
            range(0, (len(x) + 1) - (length + pred_length), length)]

window_size = context_length + prediction_length
slices = moving_window(time_series, dynamic_feat_series, window_size, prediction_length)

ts_array = []
dyf_array = []

for k in range(len(slices)):
    ts_array.append(slices[k].ts)
    dyf_array.append(slices[k].dyf)

# Make predictions

list_of_df = predictor.predict(ts_array[:],dyf_array[:])
```

FIGURE 4.4: Evaluator pseudocode

Refer Appendix C for the complete code of the pseudocode given in figure 4.4.

Chapter 5

Performance evaluation

Performance of a machine learning model defines how accurately it can forecast and how fast it can execute the model on datasets and produce results. For the purpose of this work, the evaluation methodologies described under section 2.6 were used. The trained model was used to create an endpoint in AWS Sagemaker environment for making predictions for new datasets. And in order to effectively evaluate the performance of the trained models, multiple datasets were used.

5.1 Workload

Here new datasets of similar nature were received for a different date range from the data source where the initial dataset was received. Then the same data preprocessing process was applied to transform the data into the format required by the model used. Figure 5.1 and 5.2 below shows variation of visits and campaign property of the preprocessed dataset with time.

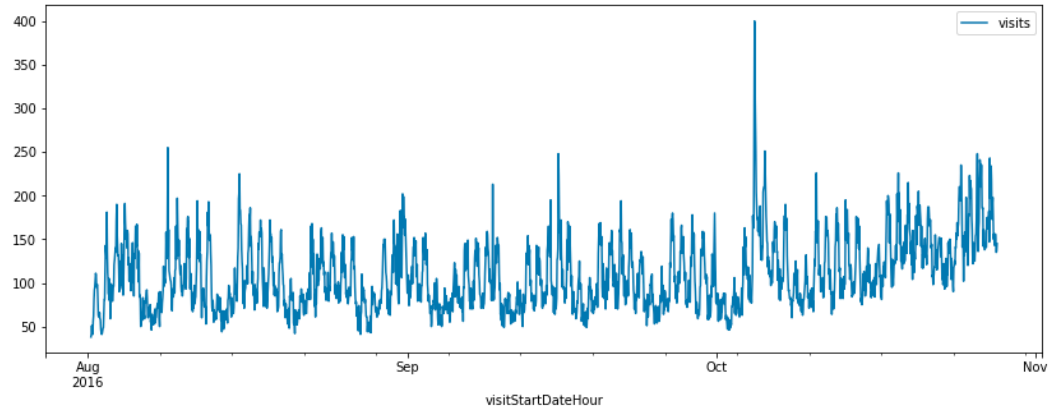

FIGURE 5.1: Visits variation of the evaluation dataset

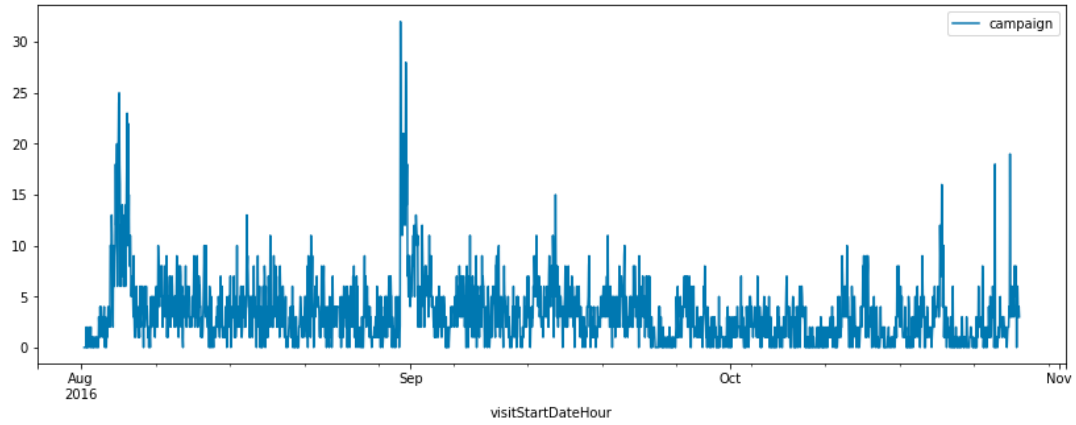

FIGURE 5.2: Campaigns variation of the evaluation dataset

Model evaluation was done using the walk-forward validation with a sliding window as described in section 2.5.2. Here the window size was determined using the context length and prediction length, and the equation below shows how it was calculated.

$$windowsize = contextlength + predictionlength \tag{5.1}$$

Where context length = Number of data points the model should take into account when making the prediction prediction length = Number of values the model should predict

Total of 17 sliding windows was used and Figure 5.3 and 5.4 show how these windows were selected from the original dataset. Different colors represent the windows.

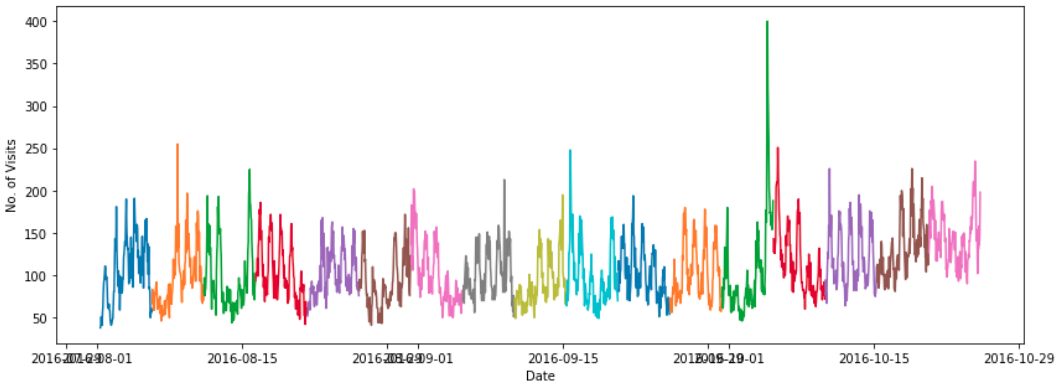

FIGURE 5.3: Selection of visit dataset windows

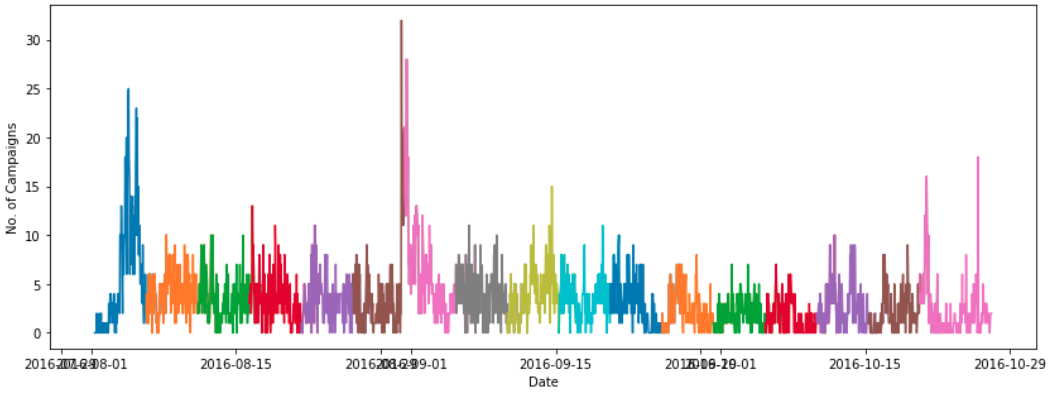

FIGURE 5.4: Selection of campaign dataset windows

5.2 Experimental setup

Amazon Sagemaker was used for evaluating the performance of the model generated. Sagemaker platform provides a feature for running the models generated in production as an endpoint which can be accessed remotely to generate predictions. So the model generated using the optimum hyperparameters that were found from the hyperparameter tuning process, was used to create an endpoint. Next, the sliding windows extracted from the dataset as described in section 4.1 were used to generate predictions using the endpoint.

5.3 Accuracy

Prediction results received from executing each of the sliding window datasets were then aggregated back to a series and plotted with a confidence interval to observe the accuracy of each prediction. Figure 5.5 below shows the aggregated result set visualization and Figure 5.6 below show a zoomed in taken from visualizing three result sets.

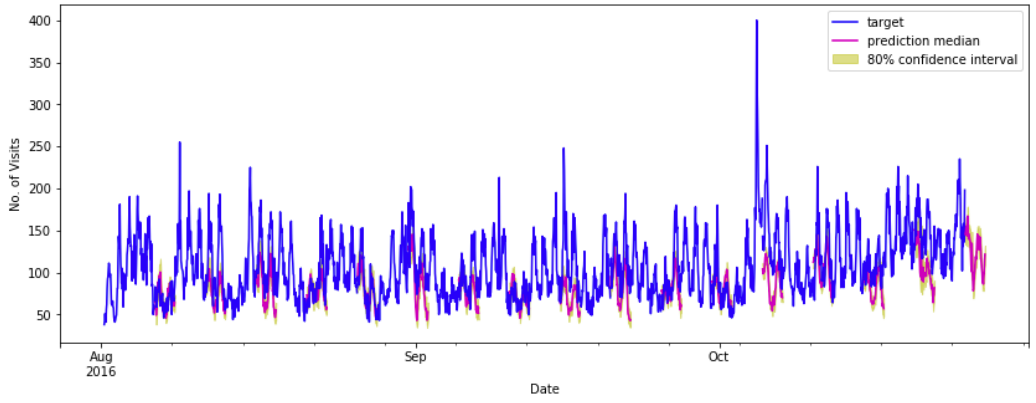

FIGURE 5.5: Predictions from all sliding datasets

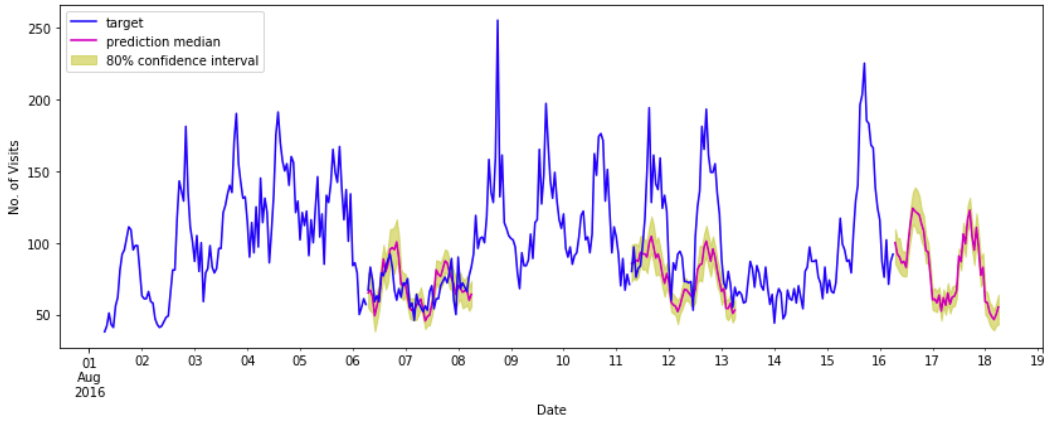

FIGURE 5.6: Predictions from three sliding datasets

5.4 Summary

Based on the observations in Figure 5.5 and 5.6. We can state that the trained model performs well at forecasting future data points. However here we have to make assumptions on the future values of the dynamic features in order to make predictions for the target variable.

Chapter 6

Conclusions

6.1 Summary

Time series forecasting is a vastly improving area of interest in many domains. Many studies have been carried out for different types of use cases. This work focused on inlining web analytics with business strategy to determine the impact of business strategy on web analytics and reduce the gap between these two components for better results.

Multiple time series forecasting approaches were analyzed and evaluated for the use case mentioned above and finally, for the best results, DeepAR algorithm was selected as the most suitable approach and AWS Sagemaker provided a built-in implementation for DeepAR was used for training the model.

Using the Sagemakers auto tuning feature, the best hyperparameter combination for the use case was determined. And the model was trained and tuned with those hyperparameters and deployed in Sagemaker itself to test and make predictions using different datasets.

The dataset extracted from Google analytics contained both web analytics data and business strategy related data. Here the web analytics data were considered as the target variable and business strategy related data were considered as the dynamic features. So the model trained using this dataset, was able to define a relationship between web analytics and business strategy. It was able to make predictions of future data points

with a satisfactory level of accuracy. So this model can be used to predict the future web analytics observations based on the future business strategy related activities.

According to the results received, its clear that there is a clear impact of business decisions on the web site metrics, and we can model this relationship using machine learning. These models can be used to properly plan for the necessary business decisions for the best results from the web metrics, reducing the gap between business strategy and web analytics. On an additional note, AWS Sagemaker provides a great platform for training and running machine learning models. It provides on-demand compute resources as required for the use case we have. Its an ideal platform for cloud applications to run machine learning models on the cloud.

6.2 Research limitations

This work uses DeepAR, a supervised learning algorithm which uses recurrent neural networks for forecasting scalar time-series. This approach gives better results with forecasting the time series once the dynamic features are provided. So currently we have to provide different combinations of dynamic features to find out the forecasting result which adds the most business value.

6.3 Future work

To improve this work further, its planned to overcome the above-mentioned limitation using reinforcement learning approaches where the dynamic features can also be predicted for the forecasting with the most business value.

References

[1] I. Aizenberg, L. Sheremetov, L. Villa-Vargas, and J. Martinez-Muñoz. Multilayer neural network with multi-valued neurons in time series forecasting of oil production. *Neurocomputing*, 175:980–989, 2016.

[2] M. Assaad, R. Boné, and H. Cardot. A new boosting algorithm for improved time-series forecasting with recurrent neural networks. *Information Fusion*, 9(1):41–55, 2008.

[3] M. Azim and N. Hasan. Web analytics tools.

[4] G. Bontempi, S. B. Taieb, and Y.-A. Le Borgne. Machine learning strategies for time series forecasting. In *European business intelligence summer school*, pages 62–77. Springer, Berlin, Heidelberg, 2012.

[5] J. Brownlee. What is time series forecasting?, 2016 (accessed January 2, 2019).

[6] J. Brownlee. Basic feature engineering with time series data in python, 2016 (accessed March 16, 2019).

[7] J. Brownlee. Time series forecasting performance measures with python, 2017 (accessed July 5, 2018).

[8] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018.

[9] C. Faloutsos, J. Gasthaus, T. Januschowski, and Y. Wang. Forecasting big time series: old and new. *Proceedings of the VLDB Endowment*, 11(12):2102–2105, 2018.

[10] V. Flunkert, D. Salinas, and J. Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *arXiv preprint arXiv:1704.04110*, 2017.

[11] M. Franco and M. Bourne. Are strategic performance measurement systems really effective: a closer look at the evidence. In *Proceedings of the EurOMA Conference*, volume 2, pages 163–74. INSEAD Paris, France, 2004.

[12] J. C. B. Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*, 2017.

[13] Gartner. Key findings from u.s. digital marketing spending survey, 2013. http://www.gartner.com/technology/research/digital-marketing/digital-marketing-spend-report.jsp.

[14] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[15] J. Järvinen and H. Karjaluoto. The use of web analytics for digital marketing performance measurement. *Industrial Marketing Management*, 50:117–127, 2015.

[16] M. Khashei and M. Bijari. A novel hybridization of artificial neural networks and arima models for time series forecasting. *Applied Soft Computing*, 11(2):2664–2675, 2011.

[17] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl. Time-series extreme event forecasting with neural networks at uber. In *International Conference on Machine Learning*, number 34, pages 1–5, 2017.

[18] C. Napagoda. Web site visit forecasting using data mining techniques. *international journal of scientific & technology research*, 2(12):170–174, 2013.

[19] T. Opitz. Modeling asymptotically independent spatial extremes based on laplace random fields. *Spatial Statistics*, 16:1–18, 2016.

[20] A. Osipenko. Backtesting time series modelsweekend of a data scientist, 2018 (accessed March 16, 2019).

[21] C. Paoli, C. Voyant, M. Muselli, and M.-L. Nivet. Forecasting of preprocessed daily solar radiation time series using neural networks. *Solar Energy*, 84(12):2146–2160, 2010.

[22] N. I. Sapankevych and R. Sankar. Time series prediction using support vector machines: a survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, 2009.

[23] A. W. Services. Deepar forecasting algorithm - amazon sagemaker, 2018 (accessed January 5, 2019). https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html.

[24] H. Yoon and C. Shahabi. Feature subset selection on multivariate time series with extremely large spatial features. In *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*, pages 337–342. IEEE, 2006.

[25] G. P. Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

Appendix A

Data loader source

```
# Importing modules

import bq_helper
import pandas as pd
import os

# Establish Big Query Helper Object for data scanning
google_analytics = bq_helper.BigQueryHelper(active_project="bigquery-public-data",
                                    dataset_name="google_analytics_sample")

# Create list of tables to later assist with queries
tablelist = google_analytics.list_tables()

# Export tables to files
for tablename in tablelist:
    query_oneTable = """
    #standardSQL
    SELECT *
    FROM
        # enclose table names with WILDCARDS in backticks `` , not quotes ''
        `bigquery-public-data.google_analytics_sample.""" + tablename+"""`
    """
    google_analytics.estimate_query_size(query_oneTable)
    oneTable = google_analytics.query_to_pandas_safe(query_oneTable, max_gb_scanned=.5)
    file_name = 'datasets-20190114/' + tablename + '.csv';
    oneTable.to_csv(file_name, encoding='utf-8', index=False)
```

Appendix B

Model builder source

```
# Importing modules

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import ast
import glob
import boto3
import s3fs
import json
import sagemaker
from sagemaker import get_execution_role
from sagemaker.tuner import HyperparameterTuner, ContinuousParameter, IntegerParameter
from sagemaker.amazon.amazon_estimator import get_image_uri

bucket = 'sagemaker-datasets'
prefix = 'research/google-analytics'

boto3_session = boto3.session.Session(region_name='us-east-1',profile_name='academic')
sagemaker_session = sagemaker.Session(boto3_session)
role = 'arn:aws:iam::206166443071:role/service-role/AmazonSageMaker-ExecutionRole-20171210T15338

s3_data_path = "{}/{}/data".format(bucket, prefix)
s3_output_path = "{}/{}/output".format(bucket, prefix)

image_name = get_image_uri(boto3_session.region_name, 'forecasting-deepar')

# Loading data

path = './datasets-20181230'
allFiles = glob.glob(path + "/*.csv")
googledata = pd.DataFrame()
```

```
list_ = []
for file_ in allFiles:
    df = pd.read_csv(file_,parse_dates=True)
    list_.append(df)
googledata = pd.concat(list_,sort=False)

# Preparing data
googledata['visitStartDateHour'] = googledata['visitStartTime'].apply(convert_to_date_hour)
googledata['hits'] = googledata['totals'].apply(map_hits)
googledata['visits'] = googledata['totals'].apply(map_visits)
googledata['pageviews'] = googledata['totals'].apply(map_pageviews)
googledata['campaign'] = googledata['trafficSource'].apply(map_campaign)

mapped_googledata = googledata.loc[:,['visitStartDateHour','visits','campaign']]
grouped_googledata = mapped_googledata.groupby(['visitStartDateHour']).agg({'visits':'sum','camp
grouped_googledata = grouped_googledata.sort_values(by='visitStartDateHour')

# Train model

estimator = sagemaker.estimator.Estimator(
    sagemaker_session=sagemaker_session,
    image_name=image_name,
    role=role,
    train_instance_count=1,
    train_instance_type='ml.c4.2xlarge', # $0.279/hour
    base_job_name='research-ga-deepar',
    output_path="s3://" + s3_output_path
)

data_channels = {
    "train": "s3://{}/train/".format(s3_data_path),
    "test": "s3://{}/test/".format(s3_data_path)
}

estimator.fit(inputs=data_channels)

# Hyperparameter tuning

# Configure HyperparameterTuner
my_tuner = HyperparameterTuner(estimator=estimator,  # previously-configured Estimator object
                               objective_metric_name='test:RMSE',
                               objective_type='Minimize',
                               hyperparameter_ranges={
                                   'num_cells': IntegerParameter(min_value=30,max_value=200),
                                   'epochs':IntegerParameter(min_value=1,max_value=1000),
                                   'mini_batch_size':IntegerParameter(min_value=32,max_value=102
                                   'context_length':IntegerParameter(min_value=1,max_value=200)
```

```
                                },
                                max_jobs=30,
                                max_parallel_jobs=5)

data_channels = {
    "train": "s3://{}/train/".format(s3_data_path),
    "test": "s3://{}/test/".format(s3_data_path)
}

# Start hyperparameter tuning job
my_tuner.fit(inputs=data_channels)


# Create predictor endpoint

my_predictor = my_tuner.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')
```

Appendix C

Model evaluator source

```
# Importing modules

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import ast
import glob
import boto3
import s3fs
import json
import sagemaker
import collections
from collections import OrderedDict
from sagemaker import get_execution_role
from sagemaker.tuner import HyperparameterTuner, ContinuousParameter, IntegerParameter
from sagemaker.amazon.amazon_estimator import get_image_uri

bucket = 'sagemaker-datasets'
prefix = 'research/google-analytics-evaluation'

boto3_session = boto3.session.Session(region_name='us-east-1',profile_name='academic')
sagemaker_session = sagemaker.Session(boto3_session)
role = 'arn:aws:iam::206166443071:role/service-role/AmazonSageMaker-ExecutionRole-20171210T15338

s3_data_path = "{}/{}/data".format(bucket, prefix)
s3_output_path = "{}/{}/output".format(bucket, prefix)

image_name = get_image_uri(boto3_session.region_name, 'forecasting-deepar')

# Data mapping functions

def convert_to_date_hour(data):
```

```
    converted_time = pd.to_datetime(data,unit='s')
    return converted_time.replace(minute=0,second=0)

def map_hits(data):
    return ast.literal_eval(data).get('hits')

def map_visits(data):
    return ast.literal_eval(data).get('visits')

def map_pageviews(data):
    return ast.literal_eval(data).get('pageviews')

def map_campaign(data):
    campaign = ast.literal_eval(data).get('campaign')
    if campaign == '(not set)':
        return 0
    else:
        return 1

path = './datasets-20190114'
allFiles = glob.glob(path + "/*.csv")
googledata = pd.DataFrame()
list_ = []
for file_ in allFiles:
    df = pd.read_csv(file_,parse_dates=True)
    list_.append(df)
googledata = pd.concat(list_,sort=False)

googledata['visitStartDateHour'] = googledata['visitStartTime'].apply(convert_to_date_hour)
googledata['hits'] = googledata['totals'].apply(map_hits)
googledata['visits'] = googledata['totals'].apply(map_visits)
googledata['pageviews'] = googledata['totals'].apply(map_pageviews)
googledata['campaign'] = googledata['trafficSource'].apply(map_campaign)

mapped_googledata = googledata.loc[:,['visitStartDateHour','visits','campaign']]

grouped_googledata = mapped_googledata.groupby(['visitStartDateHour']).agg({'visits':'sum','camp
grouped_googledata = grouped_googledata.sort_values(by='visitStartDateHour')

start_time = grouped_googledata.first_valid_index()
end_time = grouped_googledata.last_valid_index()
context_length = 72 # Length of context to look for when making predictions
data_length = len(grouped_googledata)
freq = 'H' # Granularity
prediction_length = 48 # Predict next 48 points

index = pd.DatetimeIndex(start=start_time, freq=freq, periods=data_length)
grouped_googledata_indexed = grouped_googledata.reindex(index,fill_value=0)
```

```
data = np.array(grouped_googledata_indexed['visits'])
dynamic_feat = np.array(grouped_googledata_indexed['campaign'])
time_series = pd.Series(data=data,index=index)
dynamic_feat_series = pd.Series(data=dynamic_feat,index=index)

time_series_training = time_series[:-prediction_length]
dynamic_feat_series_training = dynamic_feat_series[:-prediction_length]

def series_to_obj(ts,dyf, cat=None):
    obj = {"start": str(ts.index[0]), "target": list(ts),"dynamic_feat":[list(dyf)]}
    if cat is not None:
        obj["cat"] = cat
    return obj

def series_to_jsonline(ts, cat=None):
    return json.dumps(series_to_obj(ts, cat))

class DeepARPredictor(sagemaker.predictor.RealTimePredictor):

    def set_prediction_parameters(self, freq, prediction_length):
        """Set the time frequency and prediction length parameters. This method **must** be call
        before being able to use `predict`.

        Parameters:
        freq -- string indicating the time frequency
        prediction_length -- integer, number of predicted time points

        Return value: none.
        """
        self.freq = freq
        self.prediction_length = prediction_length

    def predict(self, ts,dyf, cat=None, encoding="utf-8", num_samples=100, quantiles=["0.1", "0.
        """Requests the prediction of for the time series listed in `ts`, each with the (optiona
        corresponding category listed in `cat`.

        Parameters:
        ts -- list of `pandas.Series` objects, the time series to predict
        cat -- list of integers (default: None)
        encoding -- string, encoding to use for the request (default: "utf-8")
        num_samples -- integer, number of samples to compute at prediction time (default: 100)
        quantiles -- list of strings specifying the quantiles to compute (default: ["0.1", "0.5"

        Return value: list of `pandas.DataFrame` objects, each containing the predictions
        """
        prediction_times = [x.index[-1]+1 for x in ts]
        req = self.__encode_request(ts,dyf, cat, encoding, num_samples, quantiles)
        res = super(DeepARPredictor, self).predict(req)
```

```
        return self.__decode_response(res, prediction_times, encoding)

    def __encode_request(self, ts,dyf, cat, encoding, num_samples, quantiles):
        instances = [series_to_obj(ts[k],dyf[k], cat[k] if cat else None) for k in range(len(ts)
        configuration = {"num_samples": num_samples, "output_types": ["quantiles"], "quantiles":
        http_request_data = {"instances": instances, "configuration": configuration}
        return json.dumps(http_request_data).encode(encoding)

    def __decode_response(self, response, prediction_times, encoding):
        response_data = json.loads(response.decode(encoding))
        list_of_df = []
        for k in range(len(prediction_times)):
            prediction_index = pd.DatetimeIndex(start=prediction_times[k], freq=self.freq, perio
            list_of_df.append(pd.DataFrame(data=response_data['predictions'][k]['quantiles'], in
        return list_of_df

# Create instance of the predictor

endpoint_name = 'forecasting-deepar-190113-1413-023-1e96cb06'
predictor = DeepARPredictor(
    endpoint=endpoint_name,
    sagemaker_session=sagemaker_session,
    content_type="application/json"
)
predictor.set_prediction_parameters(freq, prediction_length)

# Create sliding windows from the dataset

Input = collections.namedtuple('Input', ['ts', 'dyf'])
def moving_window(x,y, length,pred_length):
    return [Input(ts=x[i: i + length],dyf=y[i: i + length+pred_length]) for i in range(0, (len(x

window_size = context_length+prediction_length
slices = moving_window(time_series,dynamic_feat_series,window_size,prediction_length)

ts_array = []
dyf_array = []
for k in range(len(slices)):
    ts_array.append(slices[k].ts)
    dyf_array.append(slices[k].dyf)

# Make predictions

list_of_df = predictor.predict(ts_array[:],dyf_array[:])#predictor.predict([time_series_training
```