UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADA EN INGENIERÍA INFORMÁTICA, MENCIÓN SISTEMAS DE LA INFORMACIÓN.

APLICACIÓN DE DEEP LEARNING A LA PREDICCIÓN DE TURISMO EN LA COSTA DEL SOL

DEEP LEARNING APPLIED TO TOURISM FORECAST IN "COSTA DEL SOL"

Realizado por
Hind Labzioui

Tutorizado por
Leonardo Franco

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2020

Fecha defensa: JULIO DE 2020

Resumen

En los últimos años Deep Learning se ha convertido en uno de los tópicos más importantes a nivel mundial en las áreas de Inteligencia Artificial y sus aplicaciones. Muchas empresas lo ven como una herramienta potencial para mejorar su productividad, predecir eventos futuros y prever posibles ganancias y pérdidas. Deep Learning ha tenido mucho éxito en tareas de reconocimiento de imágenes, textos y series temporales, en algunos casos con un rendimiento superior al nivel humano, por lo que se ha convertido en la tecnología dominante dentro de la inteligencia artificial. Por otro lado, la información sobre las búsquedas en Internet ya es accesible desde varias fuentes, por ejemplo a través de las tendencias de Google (Google Trends), y se ha demostrado que estos datos tienen capacidades predictivas.

El objetivo principal de este proyecto es aplicar modelos de Deep Learning para pronosticar el volumen de turistas en La Costa del Sol en una época determinada del año, utilizando datos de agencias oficiales y también de las tendencias de Google. Los datos se analizaron como una serie temporal utilizando modelos univariados y multivariados utilizando los modelos de memoria a corto plazo (LSTM), un tipo de redes neuronales recurrentes que han tenido mucho éxito con los datos de series temporales. Los modelos multivariados, que contienen la información de varias búsquedas de términos en las tendencias de Google superaron a los modelos univariados. Este resultado esperado confirma el potencial de las tendencias de Google para anticipar el comportamiento de las personas, que combinado con el poder de los modelos de Deep Learning constituye un enfoque muy interesante para el pronóstico del turismo, y puede ayudar a optimizar los recursos en este importante mercado mundial, y en particular para la región de la Costa del Sol.

Palabras clave:
Inteligencial artificial
Deep Learning
LTSM
Predicción del turismo
Google Trends

Abstract

In the last few years, Deep Learning has become one of the most worldwide influencing subjects in Computer Science and Artificial Intelligence in particular. Many companies see it as a potential tool to enhance their productivity as they will take advantage of its ability to predict future events and foresee eventual gains and losses. Deep Learning has been very successful in recognizing images, text, and temporal series, in some cases with a performance superior to the human level, so it has become the dominant technology within artificial intelligence. On the other hand, access to information about what people search on the internet became available from several sources, like Google Trends, and these data have been shown to have predictive capabilities.

The main purpose of this project is to apply Deep Learning models to forecast the volume of tourists in La Costa del Sol in a given time of the year, using data from official agencies and also from Google trends. The data were analyzed as a temporal series using univariate and multivariate models using the Long Short-Term Memory (LSTM) models, a kind of recurrent neural networks that have been very successful with temporal data. Multivariate models, containing the information from several term searches on Google trends plus the official information about the observed number of tourists, outperformed the univariate models. This expected result confirms the potential of Google Trends to anticipate people's behavior, that combined with the power of Deep Learning models constitutes a very interesting approach for tourism forecast that can help to optimize resources in this important market worldwide but in particular for the Costa del Sol areas.

Keywords:
Artificial intelligence
Deep Learning
LSTM
Tourism Forecast
Google Trends

Index

1

1. Introduction

Tourism is an energetic, vibrant, and evolution oriented industry. 1.5 billion international tourist arrivals were recorded in 2019, globally. And since the modern traveler is now more caring about their needs and their trip details, there is no more effective tool than the internet to obtain the information and reservations needed in its most specific details. Further, Machine Learning based algorithms permits, if good quality data are fed into them, to obtain accurate predictions about the novel of future input data. The idea behind this work is to use both internet search rates and Deep Learning (a branch of Artificial Intelligence to be introduced later) to forecast the number of tourist arrivals to a given city or region for a certain time range.

1.1 Motivation and goals

Tourism is considered one of the most important activities for the Spanish economy, mostly for the southern coast of the country. And one of the most touristic regions of Spain is La Costa del Sol, which is known to be a perfect destination for tourists in search of sun and warm temperatures. However, if companies find themselves in the impossibility of predicting the number of incoming tourists, many problems may arise: the presented services might be either too poor or too excessive for the volume of visitors. In both situations, the companies in charge could suffer from damage. On the other hand, with the fast increase in the amount of data of the big corporations, Deep Learning algorithms proved themselves stronger than any other ones in identifying patterns (Figure 1.1), as well as increasing their chance to recognize profitable opportunities and avoid unknown risks.

The purpose of this work is to forecast the volume of tourists in a given region on a specific date, while studying the accuracy and effectiveness of using Deep learning and Google Trends data to achieve it. As a case (and local) example, we have chosen La Costa del Sol, one of the most well-known sunny regions in the world.

Having been able to obtain data from official sources (Instituto Nacional de Estadística, INE) about the number of night stays registered by tourism in hotels and private accommodations, the following goals were defined at the beginning of this study:

1) To analyze how good are the predictions using the LSTM univariate models on the nights stays data treated as a temporal series.

2) To analyze the added predictive value of Google trend data when using multivariate predictive models.

3) To analyze whether state-of-the-art LSTM models are significantly better performant than standard models (ARIMA, etc.) on a relatively small to medium size data set (monthly data for the last ten years were collected).

Figure 1.1 shows the schematic behavior of Deep Learning models in comparison to standard algorithms as a function of the amount of data available, a thing that we are going to explore in relation to the third goal.

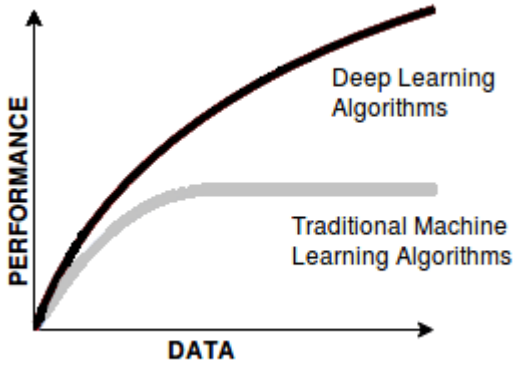

Figure 1.1 The performance of Deep Learning algorithms compared to the traditional ones in the face of the increasing amount of data.

1.3 Report structure

This work is divided into six main parts: Introduction (1), State of the Art (2), Methodology and Software Tools (3), Phases of Development (4), Results (5) and Conclusions (6)

The present chapter (*Introduction*) essentially states the objectives of the work. The second section (*State of the Art*)

includes historical background and the current state of Artificial Intelligence, Deep Learning, and LSTM.
The third section (*Methodology and software tools*) is essentially technical, aiming to present and explain the methodology and technologies that will be used, in addition to the reasons why they have been chosen over the other alternatives.
The fourth section (*Phases of development*) is the core of the study, where the development stages are clearly explained, and relevant code lines clarified. Section five contains the results obtained and comparison studies, to finalize this work extracting the conclusion in the last section, discussing also possible extensions of this work.

2

2. State of the art

2.1 Artificial intelligence

"I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think".".

Alan Turing

With this sentence from his paper "Computing Machinery and Intelligence", as well as its subsequent "Turing Test", Alan Turing, the man ahead of his time changed history, establishing the vision and the goal of artificial intelligence. In the same paper, Alan Turing defined intelligence as "a departure from the completely disciplined behaviour involved in computation, but a rather slight one, which does not give rise to random behaviour, or to pointless repetitive loops."

Parting from this definition, Artificial Intelligence (AI) is the simulation of human intelligence actions by machines. The idea behind it is to make it possible for them to reason, discover meaning, and more importantly, learn from past experiences by adjusting algorithms, and therefore perform human-like tasks. Note that one of the most important characteristics of artificial intelligence is its capability to take actions that have the best chance of achieving a specific goal. Because while the large amounts of daily created data would bury a human researcher, computers can be

trained to process them, recognize patterns, and accomplish tasks commonly associated with intelligent beings.

Today, many products and services rely on AI. Some of the most common include text analytics, Speech Recognition, robotic vacuum, and spam filters used to analyze emails and detect which ones might be spam.

2.1.1 Machine Learning and Neural Network

while Artificial intelligence is the concept of creating intelligent machines capable of mimicking human rational thinking and behavior, Machine Learning is a subset or an application of AI that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

Artificial Neural Networks are computing systems created to simulate the neural network that makes up the human brain. These systems learn to perform tasks by considering previous data, generally with no specific task rules. Neural Networks are made of several nodes organized in three classes of layers: An Input layer, an output layer, and might have one or more hidden layers situated between the two of them. In a neural network, any given node is connected with every node from the next layer, and each connection has its specific weight. The weight between a defined two nodes represents the strength of the connection between them. For example, if weight is near zero, it means changing this input will not change the output. If the weight is negative we must conclude that if this input is decreased, the output will be increased instead. Another important element of Neural Networks is the bias vector, which is an extra set of weights with no input needed, it stores the value 1 and has its own connection weight. This guarantees that the neuron gets activated even when all the inputs have value 0.

Artificial Neural networks are one of the key tools used in machine learning to find patterns, mostly in the case of complex information or when the volume of data is too big for a human being to extract. In figure 2.1 we can see the architecture of a neural network.

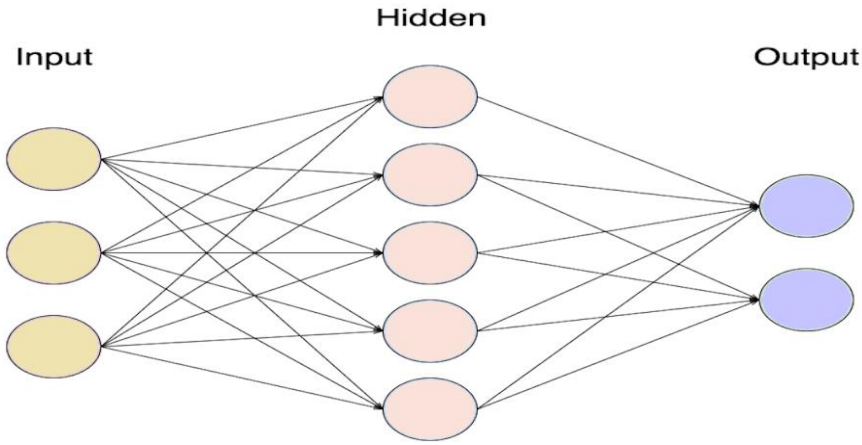

Figure 2.1 The architecture of an artificial neural network with one hidden layer.

2.1.2 Deep Learning models and LSTM

As we can see in Figure 2.2, Deep learning is a subfield of machine learning that consists of programming and teaching computers to learn by example and experience, imitating the same method used by humans naturally. As opposed to the rest of the echnologies, Deep Learning uses a big number of hidden layers to extract features from data, analyze patterns, and infer the expected result.

In classic Machine Learning, the data scientist will make a choice for himself and extract the data that will influence the prediction: variables or characteristics. However, In Deep Learning, often, it is not even possible to extract features because we process unstructured data, images, sound, text, etc. The good news is that there is no need for Feature Extraction in DL since the algorithm will be trained to take out the influencing elements itself.

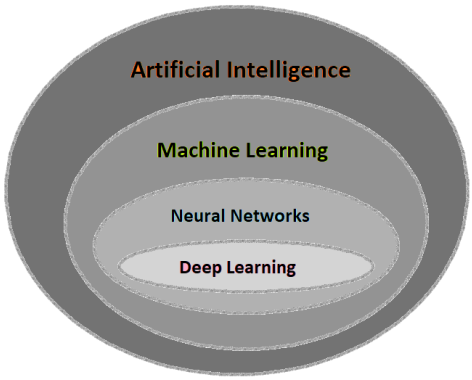

Figure 2.2 Relationships between Artificial Intelligence, Machine Learning, Neural Networks, and Deep Learning.

To talk about the origin of Deep Learning we must go back to 1943 with the appearance of the "formal neuron" model, which was a representation of the functioning of the human brain. Years later, In 1957, the "perceptron" (a single layer neural network) was invented to be considered the first artificial neural network.

It is important to mention that in 1950, the "Turing test" was born to mark a turning point in the history of artificial intelligence, however, its evolution experienced a serious slowdown, until the 1980s, when research on deep learning shows significant progress. New concepts are developed, such as the multilayer perceptron or convolutional neural networks. This is thanks in particular to the work of a French researcher named Yann LeCun. However, these new neural networks with several layers faced two serious obstacles, the fact that they required huge computing power to be efficient, and the need to access a large amount of data to "train" these algorithms. Because of these two reasons, the scientific community turned their back on deep learning in the 1990s.

Since 2010, Stanford University started organizing the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) to confront IT research teams with the best algorithms around the world on an image recognition contest. During 2012's edition, the winning program smashed the records relying for the first time on deep learning. Ever since that year, every competitor started to use deep learning in the contest.

Figure 2.3 shows the results of ILSVRC during the first 4 years, and we can see clearly the efficiency of deep learning algorithms.

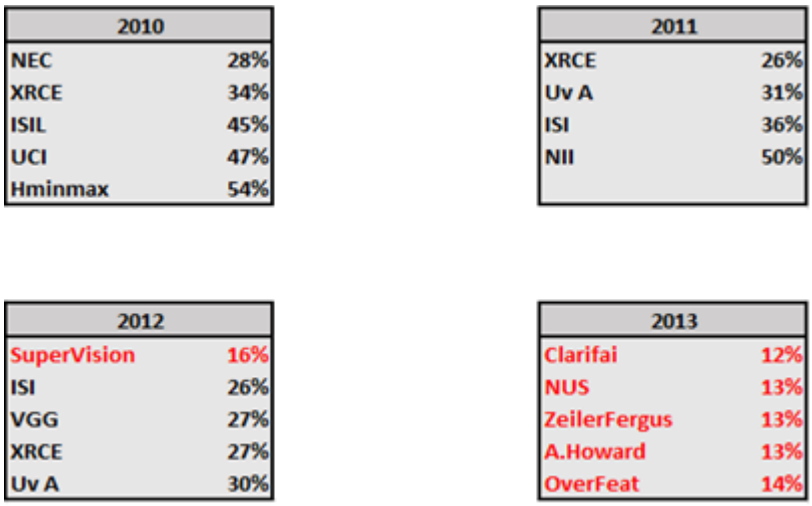

Figure 2.3 Error rates of ImageNet Large Scale Visual Recognition Challenge in 2010, 2011, 2012, and 2013 editions (Deep Learning algorithms are in red color).

Nowadays, it is well-known that many of the applications and devices we use everyday take advantage of Artificial Intelligence to serve our needs in the most personalized way.

Some clear examples are Netflix, Siri, Alexa, and even Spotify.

The problem we are treating in this work is a time series forecast problem, and when it comes to this kind of problem, deep learning takes a slightly different approach.

First, we must define the time series to put our problem in its context.

A **time series** is a sequence of data points listed in time order, at equally spaced points In time. The purpose of the time series analysis is to identify the nature of the observation sequence phenomenon and consequently predict time series variables. Time series problems are considered one of the hardest ones to solve in data science. And although with the latest revolutions that have been happening in data science there are many models to forecast this type of data, we can affirm that the type of neural network called LSTM is the one leading the race in this field.

Recurrent neural networks (RNNs) are a class of neural networks suited to processing time-series and other sequential data, due to the feedback loop of connections they include as a difference to feedforward architecture. Due to this kind of connections, the network can maintain a sort of state, allowing to perform sequence-prediction tasks.

LSTM stands for Long-Short Term Memory. It is a unique type of Recurrent Neural Network capable of learning long-term dependencies and selectively remembering patterns for long durations of time. This characteristic is very useful for the types of prediction that involves retaining information over long time periods of time, which Is quite a limitation for the traditional RNNs. Unlike classic neural networks, An LSTM network contains different memory blocks named cells (Figure 2.4). There are two states that are being transmitted to the next cell; the cell state and the hidden state.

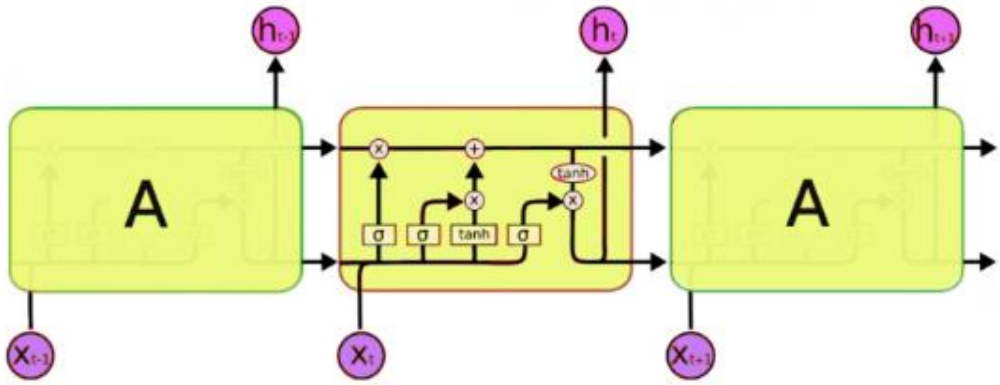

Figure 2.4 LSTM cell architecture

The memory blocks are responsible for remembering data, and in order to manipulate these blocks, the network possesses three mechanisms named gates.

A forget gate (Figure 2.5)has the function of eliminating the information either no longer needed or less important from the cell state. This action Is completed via multiplication of a filter with the intention of optimizing the performance of the LSTM network

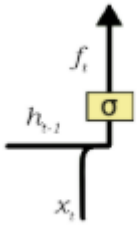

Figure 2.5 Internal architecture of a Forget Gate

The second kind of gates is **The input gate** (Figure 2.6).

The input gate Takes care of adding of information to the cell state. This addition of information proceeds basically in three steps:

1. Using a Sigmoid function to regulating what values need to be added to the cell state.

2. Using the tanh function to create a vector containing all possible values that can be added to the cell state.

3.Multiplying the regulatory filter to the created vector,then adding it to the cell state by the use of addition operation.

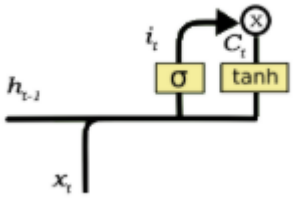

Figure 2.5 Internal architecture of an Input Gate

When this process is over, we guarantee that only that information is added to the cell state.

The last gate is the **Output gate** (Figure 2.6).

The functioning of an **output gate** is a three steps process as well:

1. Applying tanh function to the cell state, scaling the values to [-1,1] interval, and then Creating a vector.

2. Using a sigmoid function to make a filter and regulate the values that need to be output from the vector created above.

3. Sending the regulatory filter value out as output and to the hidden layer after multiplying it by the vector already created.

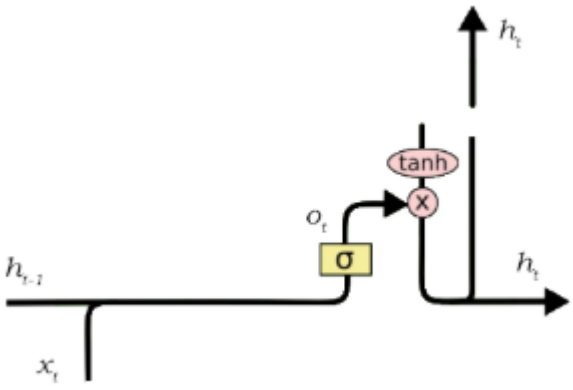

Figure 2.5 Internal architecture of an Output Gate

Furthermore, LSTM's use of the gated cells to store information outside the regular flow of the RNN allows the network to manipulate the information in several ways, including reading from a cell after storing information in it. The cells are separately able to make decisions concerning the information and execute these decisions by either opening or closing the gates. The ability to remember information for

a long period of time gives LSTM advantage over traditional RNNs in these types of tasks.

2.2 Applications of artificial intelligence In tourism

Currently, it is difficult to mention one industry that has not been impacted by AI. From health to finance to agriculture, most of them have been profoundly transformed by the phenomena of Artificial Intelligence. One more industry taking advantage of the benefits of AI is the tourism industry. There are many applications of Artificial Intelligence to help flourish tourism. From the traveler perspective, when a trip is mentioned, the first taken step is looking for information using a recommendation application. About 99% of successful products have a recommendation solution integrated. There are many Platforms to assist a person in each step of the trip suggesting solutions. For example, when looking for a flight, a person gets recommendations about renting a car, reserving a hotel, and sometimes even about restaurants and activities locations.

From the tourism industry perspective, Hotels and resorts count on offering exceptional customer service to shape their name. AI applications are known to achieve tasks that have always required human cognitive abilities, at any time of the day, in much less time than human beings, and theoretically, with fewer errors. All these qualities have made the use of AI tools in the tourism industry convenient to save companies a lot of time and money. As an example of these applications, we can mention the use of chatbots to provide the clients with instant automatic answers to their questions, or an even more futuristic approach like the one taken by Hilton deploying the robot concierge named "Connie". Also, many applications are being used for data and sentiment analysis in order to better know the client and consequently give a better service.

But to be back to an application even more similar to our work, let's mention that AlexSoft developed a self-learning algorithm, able to predict the future movements of prices based on a number of features like trends, demand growth and airlines offers. And these applications are just the tip of the iceberg, because with every passing day, Artificial intelligence in the tourism scene becomes even more present.

3

3. Methodology and Softwate tools

3.1 Methodology

In this chapter, the research methodology used in this work will be discussed, together with a description of the technologies needed to develop it.

This work aims to gain more understanding about the use of Deep Learning time series algorithms. Our method is testing the efficiency of Deep Learning's time series models when applied to trends, with the purpose of making a one-Step forecast of tourism in the region La Costa del Sol.

Furthermore, we are interested in corroborating whether better results are obtained using trends rather than just a single column in a univariate model.

The duration of the collected data for this work runs from January 2010 to the same month of 2019, which is the period of 10 years of monthly data. The information we used is quantitative data, the first fifty-five features are percentages, being 0 the sign that the data has not been searched enough in this date compared to the rest of the timeline, and 100 Indicating the period with the most interest In the keyword. The last feature is also numerical, it

designates the number of tourists in La Costa del Sol in the corresponding month. The study data have been collected from two sources:

-The first one is Google Trends. The information obtained is a set of trends related to tourism in La Costa del Sol.

-And the second source is the National Institute of Statistics known as INE, which provided us with the number of tourists.

The searched keywords contained the names of cities and villages of La Costa del Sol combined with either touristic services or attractive features about the region. For instance, the word beach was combined with Málaga, Fuengirola, and the rest of La Costa del Sol, also the words hotel, weather, hostel, flight Málaga and trip were explored. We used some known travel applications too, such as booking. Some of the keywords were also searched in Spanish, as we have to consider the local tourism as well, and the results are definitely not the same. We noticed as we were using Google Trends that the search amounts kept increasing with the growth of the use rate of the internet all around the world.

After trying to adapt the problem to classic Deep Learning models, it turned out to be a bad approach to the problem. So, driven by the Interest in predicting future information, the model chosen for this work ended up being LSTM, as it is considered one of the most effective models for time series. Moreover, LSTM models can be used in both its multivariate and univariate versions, in both cases being trained and supervised.

3.2 Software tools

As deep learning became an interesting field for the advance of technology, countless software libraries became available to make use of it.

For this work, we chose to use Keras in Its version 2.2.4 on top of Tensorflow 2.1.0, as it makes it almost intuitive to build networks and evaluate them, thanks to the simplicity of Python language and the effectiveness of Keras functions and graphic tools. When it comes to virtual environments, each person has their own preference. In this work, we selected Anaconda 3, as it is considered a good option for deep learning beginners that brings many tools with just one install. The package manager is called Conda and it allows us to install the libraries we consider essential for our work.

The Notebook we will be using to write and execute our code Is Jupyter. Thanks to its simplicity, Jupyter allows us to write our code in different blocks so we can execute different parts in the order we decide, in addition to commenting it in a rather practical way.

Before starting to present the algorithms and results obtained, here follows some technical information about the software tools used.

3.2.1 Tensorflow

As deep learning became an interesting field for the advance of technology, countless software libraries became available to make use of it. Tensorflow is one of the most used open-source software libraries. TensorFlow was developed by the Google Brain team for internal Google use, to be later released as open source code under the Apache License 2.0 on November 9, 2015. Its flexible architecture allows for an easy deployment of models across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and other devices. TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.

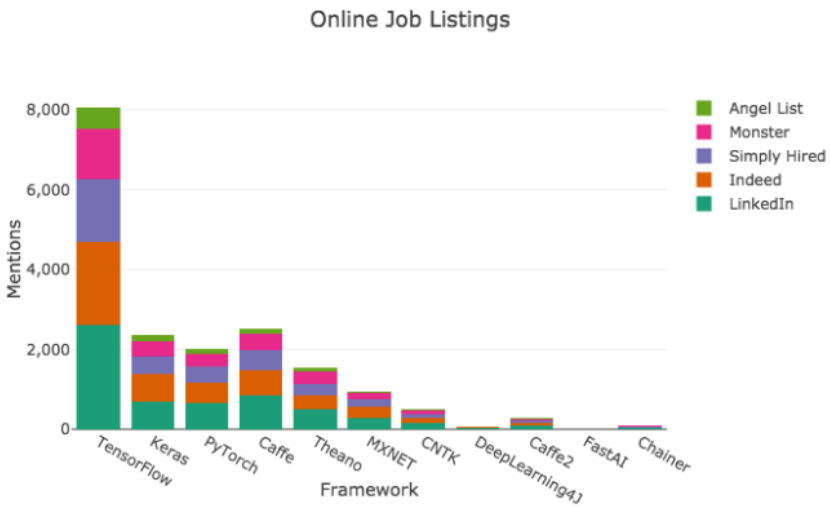

Figure 3.1 Online Job Listings of Tensorflow compared to Its competitors.

Backed by Google, Tensorflow has also the greatest number of developers use and Google searches compared to its competitors (Figure 3.3), in addition to having the most books on Amazon as we can see in Figure 3.2.

Figure 3.2 The number of Tensorflow's books on amazon compares to competitors.

3.2.2 Keras, Anaconda, Jupyter and Python

Although there are some advantages to using the low-level TensorFlow Core API like in the case of debugging, this library requires working with computational operations, graphs, and tensors, which can be challenging when we have no experience working with it. Therefore, it is very beneficial to use a high-level API to facilitate learning and building models. Written in Python, **Keras** is a high-level neural network APIs adopted by the Tensorflow project. It was designed following the best practices for reducing the cognitive load to be easily used to work with Python, in order to combine separate modules and create new models.

Backed by Google, Microsoft, Amazon, Apple, Nvidia, Uber, and others, Keras offers integration with many back-end engines like TensorFlow, Theano, CNTK, PlaidML, and MXNet, as well as strong support for multiple GPUs. Keras makes it also possible to mix the high-level and low-level TensorFlow APIs as needed. In Figure 3.2 we can see clearly that both Tensorflow and Keras are far more popular among programmers than the competitors.

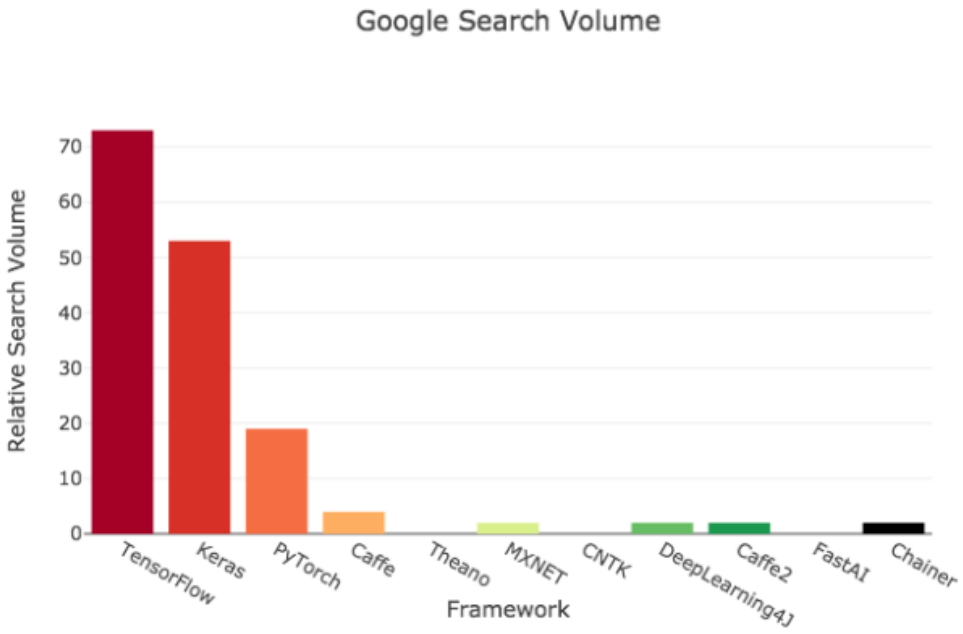

Figure 3.3 The trends of Tensorflow and Keras in against their competitors

To make it easy for people interested in machine learning and data science, **Anaconda** allows most of the packages needed to be installed either from the Anaconda repository, Anaconda Cloud, or using the "conda install" command. Anaconda is a free and open-source distribution of Python and R. It comes with the Python interpreter as well as 1,500 packages related to machine learning and data science, in addition to the Conda package and virtual environment manager.

To write a Python code, an easy option is using Jupyter Notebook. **Jupyter Notebook** is an open-source web application developed by Jupyter to create and share documents with live code, equations, visualizations, and narrative text. Jupyter includes data cleaning and transformation, numerical simulation, statistical modeling, data visualization, and machine learning.

Python is an interpreted high-level programming language with dynamic semantics. Python's cost maintenance is reduced due to its simplicity and easy to learn syntax, also, Python supports modules and packages, which encourages program modularity and code reuse. The edit-test-debug cycle of Python is incredibly fast because there Is no compilation step. It is furthermore easy to debug since a bug or bad input does not cause a segmentation fault, but Instead, it raises an exception when the program catches it, and when not, the interpreter prints a stack trace. Moreover, Python supports

object-oriented language, and consequently, the concept of classes and objects come into existence. IT is also considered a portable language since it can run equally on Windows, Linux, Unix, and Macintosh, etc. We chose to program in Python 3.7 due to its developer-friendly high-level property, which makes it one of the most expressive and more understandable languages. For a novice, it is an easy approach to use an interpreted language, since the interpreter will execute the code one line at a time, which makes debugging easy and fit for beginners. Moreover, Python is an extensible language allowing to develop graphical user interfaces (GUIs).

However, the hardware was a limiting circumstance to accomplish this work, since to execute the code we used a Windows 10 Operating Sytem of 64 bits installed In a laptop device with the following characteristics an Intel Core i3-3110M CPU of 2.40 GHz and 2 cores, and a RAM of 4Go DDR3. These low characteristics made the execution time quite long for such a code; about 1 minute 20 seconds to execute the Multivariate LSTM model with the window of size 3, two minutes to execute the regular Multivariant, and 2 minutes 15 seconds to execute the Univariant model.

4

4. Development process

The different development phases are described in this chapter, starting with the data collection process and its description to follow with technical details of the most relevant parts of the code in order to implement the LSTM models in its univariate and multivariate cases.

4.1 Data Collection

As mentioned in earlier parts of this work, two different sources of data were used in this work: the number of nights stays and the number of travelers provided by the INE and data from Google trends on selected chosen keywords. Data were collected monthly for the ten years range between January 1st, 2010 till December 31st, 2019.

4.1.1 Tourist hotel and apartments occupation and travelers data

The National Institute of Statistics (INE) was contacted about the possibility of obtaining data about the number of tourists arriving in La Costa Del Sol in the last ten years. They provided us monthly data based on the hotels and touristic appartments occupations (total number of nights stays) and the number of travelers. We decided to work with a variable that was the sum of the total number of nights stays by individual travelers and the number of registered travelers in one month. These data constitute a unidimensional temporal series, and it is the relevant variable in this study to be estimated from all the implemented models.

4.1.2 Google trends data

Google Trends is a tool from Google Labs allowing to know the frequency with which a term was looked for in the Google search engine. Google Trends can be accessed from the link https://trends.google.es and allows the user to enter a keyword (or a set of keywords) to see how its search volume has varied over time, refining the search also in terms of locations. The user can also change the time frame, region, category, and type of search (web, news, shopping, or YouTube). A useful feature of the application is that while searching for the trends of the chosen words, Google Trends also suggests tendencies of topics related to the user's keyword. This option has been very useful to acquire more parameters for this work and has made it easier to have as many potential inputs as possible, to the point that it has become necessary to stop, as more than 50 trends have been downloaded.

4.1.3 Final data sets.

Once all the information was received and downloaded, two datasets were created:

1) A temporal series comprising 120 values of nights stays of tourists in hotels and apartments for the last 10 years from 01/01/2010 to 31/12/2019. This data was used to train the univariate models.

2) A set of data ordered as a matrix of 120 x 56 that was stored in .csv format. The first 55 columns correspond to data downloaded through Google trends while the last column is the temporal series from 1), which can be considered the relevant variable to be predicted.

After the two data sets were created, a check was carried to make sure there were no missing values or outliers, and that they are suitable for the algorithms. A further inspection check was done to see whether a priori there was predictive information in the Google trends data using correlation analysis. The Pearson correlation coefficient was measured between the sum of all Google trends values and the variable of interest (number of tourist pernoctations), to obtain a value of 0.78 (high correlation), indicating that there was indeed relevant information in the data and thus that good results can be expected.

4.2 Structure of the code

Most of the steps in the execution of the code were executed in Jupyter, for that reason, an environment was created in Anaconda, to then opening the Jupyter Notebook.

Now, let's describe the characteristic of the model we have used.

LSTM models allow the network to use multiple timesteps windows to forecast the output. In other words, instead of receiving one input, the output in the time (t+1) will be predicted based on the Inputs of (t-2), (t-1) and (t). This is called a window.

To evaluate the efficiency of the window method, the model has a window size of three, meaning that our output will use three previous timesteps to be forecasted. The task is to use a fixed number of months currently available in the time series to predict the future point. Unlike the classical Deep Learning networks, LSTM layers have memory blocks instead of neurons, which proved to be efficient even with a small number of layers. To demonstrate it, as stated above, we used only one LSTM layer with 100 units and 70 training epochs.

Now that the idea of our model is already set, we start putting together the code lines.

A set of libraries is needed in order to import, cleaning, transforming and analyze the data. **Pandas** was the tool used. In order to make all the numerical operations including creating an array out of our data and calculating the error, package **NumPy** was chosen. The math library is necessary for all the mathematical operations, and **Matplotlib** is the library responsible fore plotting data, especially the component Pyplot. **Scikit-learn** (also known as **sklearn**) is one of the most useful libraries for machine learning in Python, as it contains efficient tools including statistical modeling like classification, regression, clustering and dimensionality reduction. From this library mainly used the MinMaxScaler to scale data and the mean_squared_error functions to calculate the error.

From Keras we need to import the Sequential model, the Dense layer, and LSTM model.

The Sequential model is imported for being appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. The most basic neural network architecture in deep learning is the Dense neural network, which contains fully-connected layers. In a dense layer, all the neurons are connected to all the neurons in the next layer. The following screen capture shows the code used to

import the previously mentioned libraries.

```
import pandas as pd
from numpy import concatenate
from math import sqrt
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

The previously prepared data file needs to be loaded and processed. Therefore, the read_data function is to be executed specifying that the first row is the header and the file has no index column. Since our data has the same interval of time between the rows, the column 'Month' is no longer necessary for this model.

```
# load dataset and drop the month column
from datetime import datetime
dataset = read_csv('Dataset-Multi.csv', header=0,  index_col=False)
dataset = dataset.drop('Month', 1)
values = dataset.values
```

If we consult the first five rows of our dataframe using the head() function we will obtain this table.

Entrée [6]: `dataset.head()`

Out[6]:

	beach_costa_del_sol	beaches_costa_del_sol	beaches_malaga	beach_malaga	beach_manilva	benahavis	benalmadena	best_beaches_costa_del_sol	best_be
0	69	46	30	34	41	38	44	37	
1	56	56	36	44	28	57	49	52	
2	80	53	46	56	51	55	55	35	
3	65	96	26	46	27	50	58	73	
4	78	52	67	53	35	34	62	36	

5 rows × 56 columns

Figure 4.1: The dataset table after dropping the 'Month' column.

We convert next the data to float format to have better mathematical manipulation within the code.

```
# make sure all our data is float to be scaled and processed
values = values.astype('float32')
```

It is necessary now to build a function to convert the series (dataset values) into supervised learning. Hence, each feature will appear in the table in three timesteps.

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, -i)) for j in range(n_vars)]
    agg = concat(cols, axis=1)
    agg.columns = names
    # The rows with NaN values must be dropped
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

LSTM models perform better with normalized data, therefore, we scale the data using the function MinMaxScaler with the range 0-1.

As mentioned in the tools chapter, one of the components of a common LSTM unit is the cell, which has the advantage of remembering values over arbitrary time intervals. The number of months indicates the window of samples that the model will memorize to predict the next time step. At this point, we reframe the data using the previously built function series_to_supervised. As arguments the function received the scaled data.

```
# Indicate the number of lag months and features
n_months = 3 #only if we want the input to take info about 3 months earlier, otherwise this variable is not specified
n_features =56

# convert scaled data/series to supervised learning
reframed = series_to_supervised(scaled, n_months, 1)
```

After reframing the data, the number of columns we take as input for the previous 3 months is "the number of features * the number of months", and as output, we take n_of_tourists column.

This means that each input feature from the dataset is now multiplicated by 3, one for each time step (t-3, t-2, t-1). Let's specify which data rows form our input data, and which row is the output.

```
# split our data into input and output subsets
values = reframed.values
n_obs = n_months * n_features
In= values[:, :n_obs]
out=values[:, -1]
```

A neural network needs to receive at least two sets of data, the training subset, and the test subset.

As the names indicate, the neural network makes use of the training data to learn from the features, before we test the accuracy of its predictions comparing the test output data to the predicted one. To split the reframed values we define a variable indicating the number of rows for the training data. We decided to use 70 rows as training data because we deemed necessary to have enough rows for that matter, and still keep enough to test the network.

```
# split both X and y sets into train and test subsets
values = reframed.values
n_train_months = 70
train_X = In[:n_train_months] #the first 70 raws, all the columns
train_y =out[:n_train_months]
test_X= In[n_train_months:]
test_y= out[n_train_months:]
```

Next, it is important to reshape X into the 3D format, as It is the appropriate one for an LSTMs network to receive.

The three parameters of the 3D format are: Samples, timesteps and features.

```
# reshape input into 3D data : samples, timesteps and features
train_X = train_X.reshape((train_X.shape[0], n_months, n_features))
test_X = test_X.reshape((test_X.shape[0], n_months, n_features))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(70, 3, 56) (70,) (47, 3, 56) (47,)
```

The moment has come to build our network:

As mentioned earlier, our model has an LSTM layer of 100 units, and one neuron in the dense output layer to predict the number of tourists.

The chosen loss function is the Mean Absolute Error (MAE), and the optimizer chosen is the Adam function known to be the most effective one. **ADAM** is an adaptive learning rate algorithm, specially designed for Deep Learning networks training. The model was set for 70 training epochs with a batch size of 1.

MAE is a loss function calculated as the sum of absolute differences between the expected value and the predicted variables.

```
# build the network
model = Sequential()
model.add(LSTM(100, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
```

And with the last step done, our model now is completely built.

5

5. Results

This section describe the results obtained from using the main model: an LSTM recurrent neural network using a window length of 3 and trained supervisely with multivariate data. A comparison was made with two other models: Multivariate LSTM with window size equals to 1, and Univariate LSTM model. The Root Mean Square Error (RMSE) was the performance measure used to compare the results, and also the approximate training times of the models are reported.

5.1 Training, test and results

Once our model is designed and our data is scaled and well processed, the network was trained using the fit function while checking the error.

```
# fit network
history = model.fit(train_X, train_y, epochs=70, batch_size=1, verbose=2, shuffle=False)
```

The figure 5.1 shows the loss values of the training process first steps.

```
(70, 3, 56) (70,) (47, 3, 56) (47,)
Epoch 1/70
 - 1s - loss: 0.1838
Epoch 2/70
 - 0s - loss: 0.1261
Epoch 3/70
 - 0s - loss: 0.1322
Epoch 4/70
 - 0s - loss: 0.1138
Epoch 5/70
 - 0s - loss: 0.0964
Epoch 6/70
 - 0s - loss: 0.0748
Epoch 7/70
 - 0s - loss: 0.0739
Epoch 8/70
 - 0s - loss: 0.0713
Epoch 9/70
 - 0s - loss: 0.0660
Epoch 10/70
```

Figure 5.1: The loss value of the first 9 of the total of 70 epochs.

After this training step, the number of tourists can be finally predicted using the test data "test_X" and then compare them to the expected values test_y.

```
# make a prediction using the test_X subset
pred = model.predict(test_X)
```

An inverse transformation is then implemented in order to plot the results:

```
#We invert the scale of the predicted data to have the corresponding values
test_X = test_X.reshape((test_X.shape[0], n_months*n_features))
inv_pred = concatenate((test_X[:, -(n_features-1):], pred ), axis=1)
inv_pred = scaler.inverse_transform(inv_pred)
inv_pred =inv_pred[:,-1]
inv_pred

#We invert the scale of test_y data to have the corresponding values
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_X[:, -(n_features-1):],test_y), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,-1:]
inv_y
```

With the data back to normal scale, the line plots can be printed in order to compare the expected values with the predicted by the model as shown in Figure 5.2.

To have a numerical comparison of the results, we also calculate the RMSE value with and without scaling.

```
# we show the line plot of the real observed vs predicted values
pyplot.plot(inv_pred) #blue line for the predicted values
pyplot.plot(inv_y) #orange line for the expected values
pyplot.show()

#Calculate the RMSE
norm_rmse = sqrt(mean_squared_error(test_y, pred))
print('Test normalized RMSE: %.3f' % norm_rmse)

#Calculate the RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_pred))
print('Test RMSE: %.3f' % rmse)
```

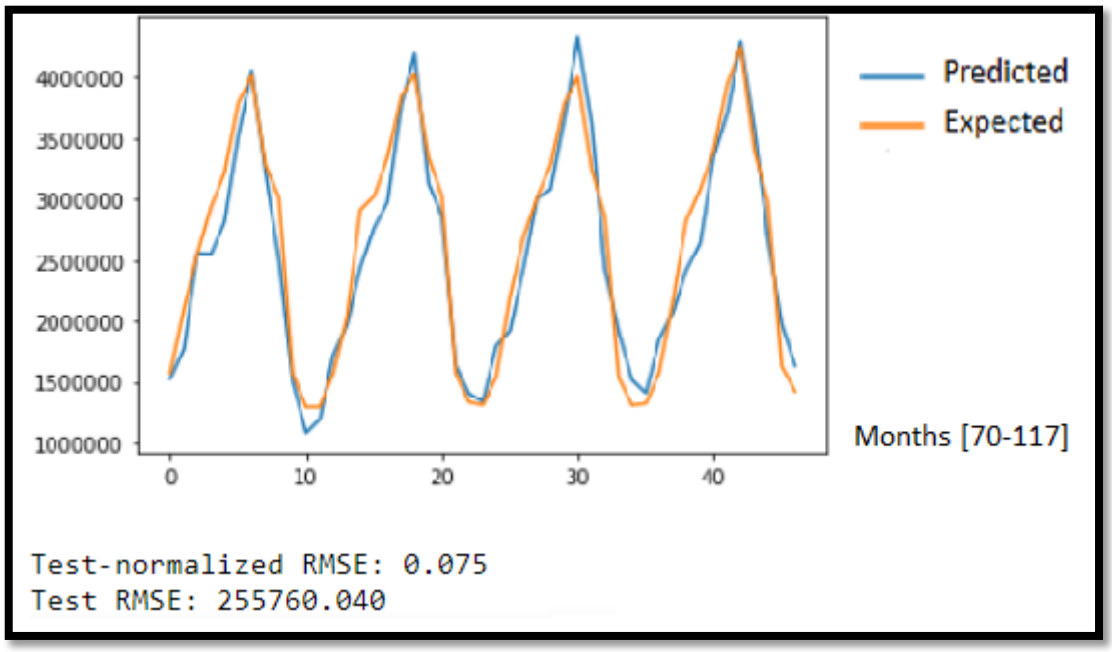

Figure 5.2: Plot of predicted and expected number of tourists in a monthly scale. The Test RMSE is indicated below.

A graphical inspection of the curve showed in Figure 5.2 suggests that the predictions follows the real values quite close but to have an idea of the quality of the results further comparisons are needed. The model used at this stage is the one that utilizes the whole data set available, combining Google trends data and the variable of interest.

5.2 Training the model without time window

This section compares two LSTM networks, designed under the same conditions with the exception of the time window value. The aim of this comparison is to study whether the time window increases the performance of the neural network. In the new model, the network is treating the input data one at a time, which means that the time window has been removed, and replaced by the value 1.

In this section, in order to avoid diving into code details, and since the same code has been used, only the parts of codelines that have been altered are shown.

Instead of indicating the number of lag and features, only the number of features will be specified:

```
# Indicating the number of features
n_features =56
```

The timestep dimension of the reframed data is set in 1, and the numbers of input values are not multiplicated by three, but rather conserve the same number.

```
# convert data/series to supervised learning
reframed = series_to_supervised(scaled, 1, 1)
print(reframed.shape)

# split our data into inputs and output
values = reframed.values
In= values[:, :n_features] #
out=values[:, -1]
```

As we execute this version of our code, the output plot shows the accuracy of the model in Figure 5.3, however, the comparison results have revealed the interesting although expected finding shown in figure 5.4; the 3 size window model has clearly outperformed the last developped model.

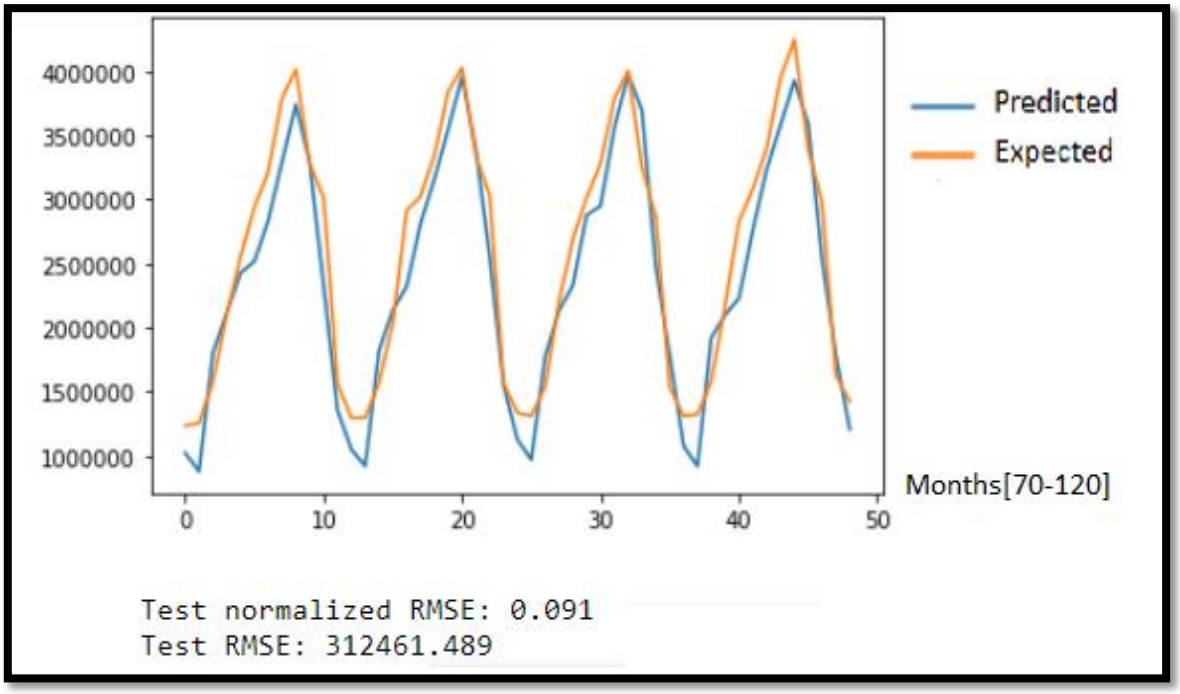

Figure 5.3: Plot of the expected number of tourists In a monthly scale, and pthe redicted one by the regular Multivariate LSTM model. The Test RMSE is indicated below.

```
1  #Calculate the regular RMSE
2  norm_rmse = sqrt(mean_squared_error(test_y2, pred2))
3  print('Test RMSE One Input: %.3f' % norm_rmse)
4
```

```
Test RMSE One Input: 0.091
```

```
1  #Calculate the 3 winow size RMSE
2  norm_rmse = sqrt(mean_squared_error(test_y, pred))
3  print('Test RMSE One Input: %.3f' % norm_rmse)
4
```

```
Test RMSE window size = 3: 0.075
```

Figure 5.4: The screen shot of RMSEs corresponding first to the 3 window size model, then to the 1 size window model.

As expected the use of the LSTM model with a memory of the three last steps before the recent to be predicted outperformed the model with no time window. It is a reasonable result that confirms that the LSTM design is useful for this kind of problems.

5.3 Comparison with Univariate LSTM

It has just been shown in the previous subsection that the Multivariate LSTM model proved its efficiency in forecasting the number of tourists based on internet searches. To a certain extent, this result was to be expected, since it is a well-known fact that Deep Learning Algorithms are more efficient with larger amounts of data, and it has been previously verified (using a correlation analysis) that there were relevant information in the Google trend data.

To verify this assumption a Univariate LSTM model was executed only with the interest variable to be predicted. Univariate LSTM models are used to model univariate time series forecasting problems, these contain a single sequence of observations where the model is forced to learn from the past ones to predict the next value in the series. The data used for this model is gathered in a csv file of two columns, one containing the month of the year, and the second indicating the number of tourists in La Costa del Sol In the specified month.

To be able to fairly compare this model with a multivariate one, both need to have the same parameters, therefore we will set the next values for both:

Number of timesteps: 1.
Number of units: 100.
Number of epochs: 70.

To execute this model, the set of operations for loading the libraries was again used:

```
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from datetime import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy
```

It is important to turn our series into supervised learning which Is the only format an LSTM model accepts. The function

data_to_supervised is responsible for changing our one column data into two, where the second is shifted so that the input is one time ahead of the output. To understand the functioning of this function, let's have a look at its implementation.

```
# frame a sequence as a supervised learning problem
def data_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df
```

The figure 5.5 explains fully how the function data_to_supervised shifts the new column and concatenates it with the Input.

0.0	841633
841633.0	1078277
1078277.0	1447047
1447047.0	1868211
1868211.0	2041989
...	...

Figure 5.5: The supervised learning of our data 'df' after applying the data_to_supervised function.

To scale the train and test subsets we created a function called scale. This function uses MinMaxScaler from the module sklearn.preprocessing to normalize the subsets of our data. After predicting the data using our neural network, it will be necessary for the model to return the forecasted data non scaled. The function scale_inv is generated then to Invert the scaled data once it is no more needed. The function fit_mdl will be used to split the train data input and output sets, then design and train our neural network. And finally, forecast_mdl as its name indicates will be invoked to predict the output of each input of our test data. After defining the functions, we can finally load the corresponding data file. Only this time we will try another approach using the data column as index instead of dropping it.

```
# load dataset
df = read_csv('Dataset-Uni.csv', header=0, parse_dates=[0], index_col=0, date_parser=parser)
raw_values = df.values.astype('float32')

# transform data to be supervised learning
supervised = data_to_supervised(df, 1)
supervised_values = supervised.values
```

As we did in the last model, we transform our data to supervised learning before spliting It into train and test subsets, then scale it using the scale function, and scale test_y also to use as we need t calculate RMSE.

```
# transform data to be supervised learning
supervised = data_to_supervised(df, 1)
supervised_values = supervised.values

# split data into train and test subsets
train, test = supervised_values[0:-50], supervised_values[-50:]

#scale test_y to use for rmse
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(raw_values[-50:])
test_y_sc= scaler.transform(raw_values[-50:])

# scale our test and train data, and obtain the scaler
scaler, train_scaled, test_scaled = scale(train, test)
test_scaled
```

In order to have some similarity we chose to use the same number of rows as the last model.At this point of the code we consider that our data is satisfactorily processed, scaled, reshaped and ready to be used. But first, we need to build the model. As stated above, the function fit_mdl is respnsible for separating the Input and output subgroups of the train set before creating the model, and then training it.

```
def fit_mdl(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
        model.reset_states()
    return model
```

Fit_mdl receives four arguments, the train set, the number of training epochs, the batch size and the number of neurons we decide for our model to have. we decided our model has 100 neurons, trains over 70 epochs, the batch size is one and our network receives one month input. The forecast_mdl function will be used In a for loop for each row of the scaled test, then stored In an array called predictions.

```
# fit the model
lstm_model = fit_mdl(train_scaled, 1, 70, 100) #fit_mdl(train_scaled, batch size=1,nb_epochs=70, neurons=100)

# forecast the training dataset to build up state for forecasting
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1) #train_reshaped(train samples, 1 step, 1 feature)

# walk-forward validation on the test data
predictions =[]
predictions_sc=[] #scaled predictions
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1] # X, y) test row i, all columns except
    yhat = forecast_mdl(lstm_model, 1, X)
    predictions_sc.append(yhat)
    # invert scaling
    yhat = scale_inv(scaler, X, yhat)
    # store forecast
    predictions.append(yhat)
    expected = raw_values[len(train+1) + i ]
    print('Month= %d, Predicted= %f, Expected= %f' % (i+1, yhat, expected))
```

The predictions_sc stores the scaled predictions In order to use them to calculate the RMSE.

After runing the last code, we obtain the following results for the whole 50 forecasts:

```
Month= 1, Predicted= 2450843.166443, Expected= 1390855.000000
Month= 2, Predicted= 1635738.218514, Expected= 1233759.000000
Month= 3, Predicted= 1351167.096924, Expected= 1253704.000000
Month= 4, Predicted= 1426935.096075, Expected= 1575504.000000
Month= 5, Predicted= 1738005.266923, Expected= 2101778.000000
Month= 6, Predicted= 2158914.277609, Expected= 2553528.000000
Month= 7, Predicted= 2499301.178655, Expected= 2939623.000000
Month= 8, Predicted= 2688652.114945, Expected= 3214757.000000
Month= 9, Predicted= 2615971.611201, Expected= 3792285.000000
Month= 10, Predicted= 2384561.895485, Expected= 4014084.000000
Month= 11, Predicted= 1913442.091464, Expected= 3277004.000000
Month= 12, Predicted= 1390375.793813, Expected= 3017944.000000
Month= 13, Predicted= 1350896.449677, Expected= 1566418.000000
Month= 14, Predicted= 1059449.527221, Expected= 1294626.000000
Month= 15, Predicted= 1239284.700272, Expected= 1295910.000000
```

Figure 5.6: The predicted and the expected values of each month.

The predicted and expected results are shown in Figure 5.7, as well as the RMSE obtained for each model.

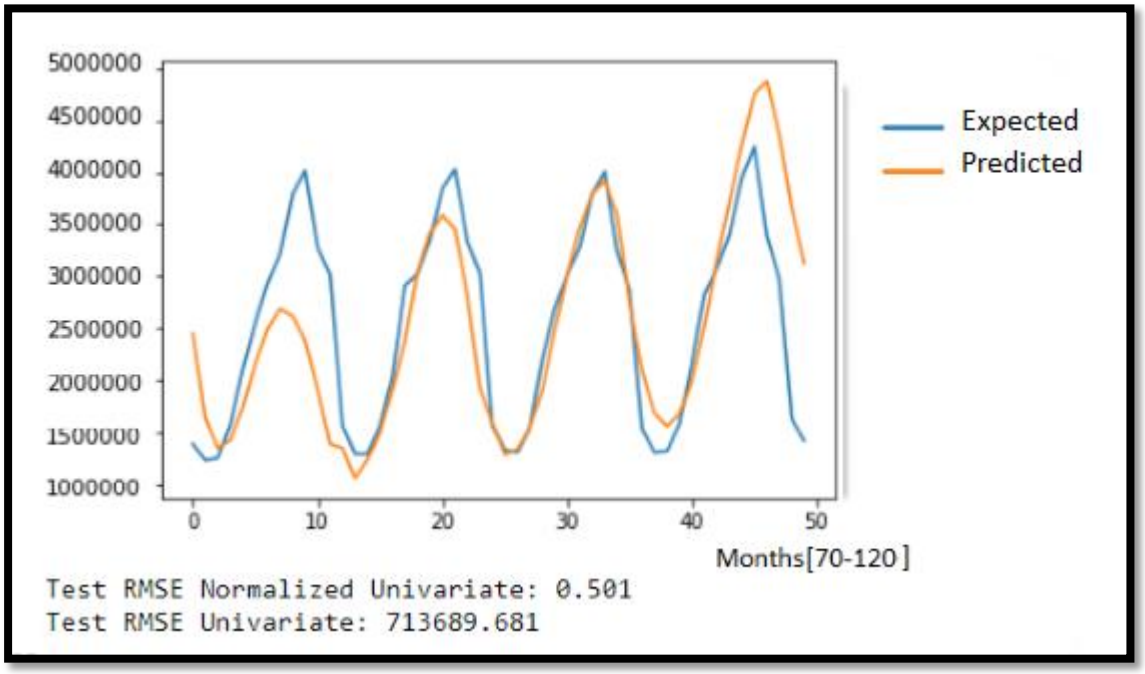

Figure 5.7: Plot of expected and predicted number of tourists by the univariate model in a monthly scale.

To compare this model with Multivariate LSTM network we just have to execute the same code previously implemented as it has the same univariate network parameters (cells, epochs and timesteps). Note that everytime we delete the results and train the network over again, the forecasted values change and thus the RMSE value.

The figure 5.8 shows how data vary from an execuction to another, this time, the RMSE value is .

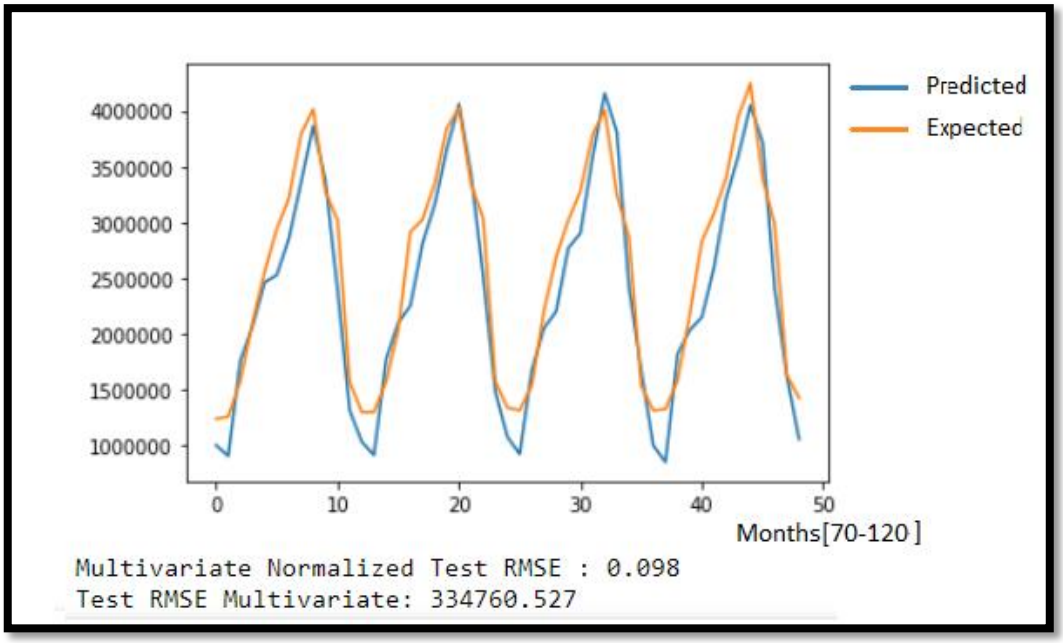

Figure 5.8: Plot of expected and predicted number of tourists by the multivariate model in a monthly scale.

We now can finally make the comparison between the two models to determine whether the Multivariate model has better performance.

```
#Calculate the RMSE
norm_rmse = sqrt(mean_squared_error(test_y, pred))
print('Multivariate Test RMSE : %.3f' % norm_rmse)
# report performance by calculating normalized RMSE
rmse_norm = sqrt(mean_squared_error(test_y_sc, predictions_sc))
print('Univariate Test RMSE: %.3f' % rmse_norm)
```

```
Multivariate Test RMSE : 0.098
Univariate Test RMSE: 0.501
```

Figure 5.9: The screen shot of RMSEs corresponding first to the Multivariate model, then to the univariate one.

The RMSE values obtained clearly indicates that the Multivariate LSTM outperformed the Univariate model and then confirming our expectations.

To have a graphic view of the matter, we made the plots of the expected values, the ones predicted by the multivariate LSTM model, and finally, the values forecasted by the univariate model. The figure 5.10 gives a clear vision of the performance of each model.

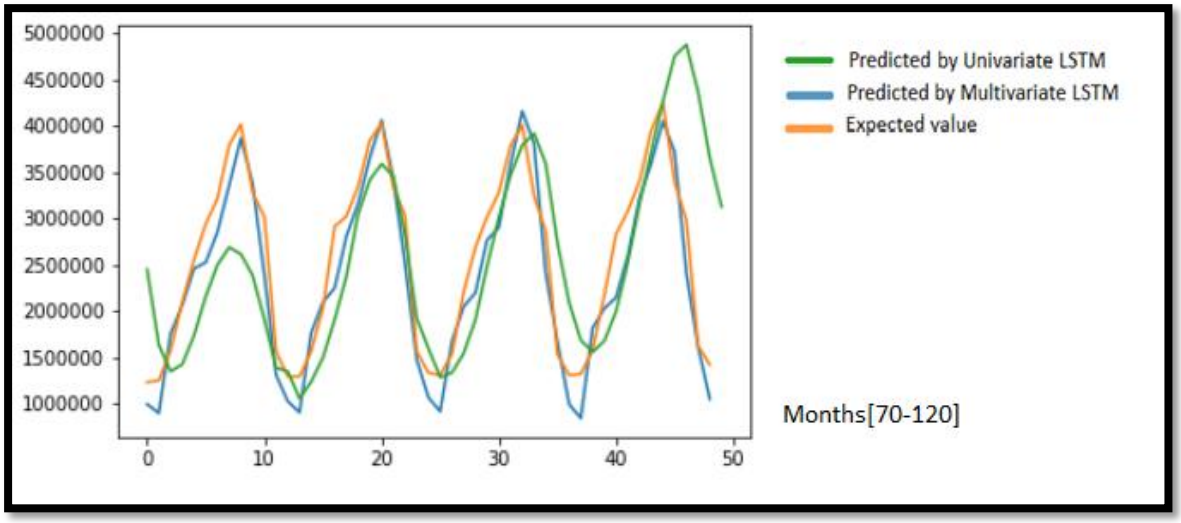

Figure 5.10: Plot of the expected values, the predicted number of tourists by the multivariate, and the predictions of the univariate model in a monthly scale.

The experiment matched fully our expectations, proving that although the univariate model used practical data, the multivariate one outperformed it making use of only search rates.

6

6.a Conclusion and future studies

The main objective of this work was to analyze the potential of Deep Learning based algorithms for the prediction of the volume of tourists arriving in La Costa del Sol for a given time, so after obtaining the results in the previous section is now time for stating the conclusions at a personal and at the scientific level.

First of all, I found this study very interesting and motivating from the very beginning as I wanted to learn from first hands about using artificial intelligence techniques. For Deep Learning is a vast and innovative area, several parts of the work were rather a challenge, however I have learnt much and feel now much more confortable working with it, and hopes to have the opportunity to apply this technology in a real problem in the near future.

From the scientific objectives set at the beginning o fthe project related to the task of predicting number of visitors to the Costa del Sol for a given date, we applied mainly two models considering the data as a temporal series: a univariate LSTM model and a multivariate LSTM model using extra information collected from Google trends. The conclusion is that the multivariate model is much better (lower prediction error) and thus the hypothesis that Google trend data contains useful information can considered as valid.

Because ot time restrictions, it was not possible to compare the performance of lower complexity models (like ARIMA/SARIMA ones), and this is left for the future, as this will tell us whether a very complex model like the recurrent LSTM networks are necessary for these data or not.

Regarding extensions and possible real applications of the present work, an interesting step forward would be to contact the Faculty of Tourism of Malaga University to discuss with them about this problem, gather more information and try to apply artificial intelligence for predicting tourism behavior in the Costa del Sol.

6

6.b Conclusión y líneas futuras

El objetivo principal de este trabajo fue el de analizar el potencial de los algoritmos basados en Deep Learning para la predicción del volumen de turistas que llegan a La Costa del Sol durante un tiempo determinado, por lo que después de obtener los resultados en la sección anterior, ahora es el momento de indicar las conclusiones a nivel personal y científico.

En primer lugar, este estudio me pareció muy interesante y motivador desde el principio, ya que quería aprender de primera mano sobre el uso de técnicas de inteligencia artificial. Deep Learning es un área vasta e innovadora, varias partes del trabajo fueron un verdadero desafío. Sin embargo he aprendido mucho y ahora me siento mucho más cómoda trabajando con estas tecnologías, y espero tener la oportunidad de aplicarlas en problemas reales en un futuro cercano.

A partir de los objetivos científicos establecidos al principio del proyecto relacionados con la tarea de predecir el número de visitantes a la Costa del Sol para una fecha determinada, aplicamos principalmente dos modelos que consideran los datos como una serie temporal: un modelo LSTM univariante y un modelo LSTM multivariante que utiliza información adicional recopilada de las tendencias de Google. La conclusión es que el modelo multivariante es significativament mejor (menor error de predicción) y, por lo

tanto, la hipótesis de que los datos de tendencias de Google contienen información útil puede considerarse como válida. Debido a restricciones de tiempo, no fue posible comparar el rendimiento de modelos de menor complejidad (como los ARIMA / SARIMA), y esto se deja para el futuro, ya que esto nos dirá si es realmente necesario utilizer un modelo tan complejo como las redes LSTM a estos datos.

Con respecto a las extensiones y posibles aplicaciones reales del presente trabajo, un paso interesante sería contactar a la Facultad de Turismo de la Universidad de Málaga para discutir con ellos sobre este problema, reunir más información e intentar aplicar métodos de inteligencia artificial para predecir el comportamiento del turismo en la Costa del sol.

Bibliography

Books and articles

Claveria, O., Monte, E., Torra, S.(2016) Tourism Demand Forecasting with Neural Network Models: Different Ways of Treating Information. International Journal of tourism Research, 17, pp.492-500.

Geron, A. (2019) Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT. In Press

Gulli, A., and Pal, S. (2017). Deep Learning with Keras. Packt.

Haykin S. (1999)Neural Networks. Prentice Hall.

Law, R, and Li, G.(2019). Tourism demand forecasting: A deep learning approach. Annals of Tourism Research, Elsevier, 75, pp. 410-423.

LeCun, Y., Bengio, Y., & Hinton, G. E. (2015). Deep learning. Nature, 521 ,pp. 436-444.

Li, F. And Cao, H. (2018). Prediction for Tourism Flow based on LSTM Neural Network. Procedia Computer Science, 129, pp. 277-283.

Raschka, S. and Mirjalili, V. (2019) Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition (English Edition). Packt.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85{117. doi:10.1016/j.neunet.2014.09.003.

Sencheong, K, and Turner, L.W. (2005). Neural network forecasting of tourism demand . Tourism Economics, 11 , pp. 301-328.

Sun, S., Weia, Y., Tsuic, K-L. and Wang, S.(2019) Forecasting tourist arrivals with machine learning and internet search. Tourism Management, 70, pp. 1-10.

Witt, S.F., and Witt, C.A. (1995). Forecasting tourism demand: A review of empirical research. International Journal of Forecasting, 11, pp. 447-475.

Web sites

[ARIMA] Comparing ARIMA Model and LSTM RNN Model in Time-Series. https://analyticsindiamag.com/comparing-arima-model-and-lstm-rnn-model-in-time-series-forecasting/ Last accessed 23/06/2020.

[Keras] https://en.wikipedia.org/wiki/Keras . Last accessed 23/06/2020.

[Machinelearningmastery.com] www.machinelearningmastery.com Last accessed 23/06/2020.

[Modelos Multivariantes] https://unipython.com/modelos-mlp-multivariantes-1-2/ . Last accessed 23/06/2020.

[Time series forecasting] https://medium.com/analytics-vidhya/time-series-forecasting-arima-vs-lstm-vs-prophet-62241c203a3b. Last accessed 23/06/2020.

[Towardsdatascience.com] www.towardsdatascience.com. Last accessed 23/06/2020.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga