UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Bachelor as Ingeniería Informática Bilingüe

BACHELOR THESIS

Developing a web application to interact with predictive algorithms about the stock market.

Autor: Juan Llamazares Ruiz
Tutor: Francisco Saiz López

Junio 2023

Juan Llamazares Ruiz
Developing a web application to interact with predictive algorithms about the stock market.

Juan Llamazares Ruiz

C\ Francisco Tomás y Valiente No 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Agradecimientos

En primer lugar, quiero expresar mi gratitud a mi tutor Francisco Saiz López por haber confiado en mí al aceptar la propuesta de llevar a cabo este proyecto. Ha sido un proceso largo y en mi caso lento. Estoy muy agradecido por tu amabilidad y apoyo siempre que lo he necesitado, junto con la paciencia que ha tenido conmigo.

Además, me gustaría dar las gracias a mi familia y amigos, ellos han vivivo conmigo estos últimos años, que han tenido momentos buenos y malos y que los he disfrutado mucho. Por otro lado, me gustaría dar las gracias a todas las personas que me han ayudado con este proyecto, ya sea por consejos, aportaciones o por pequeños empujes que he han ayudado a finalizar el trabajo de fin de carrera.

Abstract

The interest of modern technology has made it easier to stay informed about the stock market and make investments accordingly. With growing interest among individuals and investors to actively participate in the stock market, having access to tools that help them make better decisions has become increasingly desirable. To study in more detail the stock market is a complex and dynamic asset and can be difficult for these individuals to keep up with the latest trends, company results and world news. By developing a web application that can interact with predictive algorithms, investors can access real-time data and historical trends to make informed investment decisions.

The main purpose of this Bachelor Thesis will be to demonstrate explore the possibility of obtaining a predictive algorithm on the stock market that can be interacted with through a developed web application tool to help individuals and investors to optimize their time and effort to decide the best financial asset. We will consider the utilization of Recurrent Neuronal Networks (RNNs) to support a more accurate stock prediction, and the development of a forecasting model that could be easily implemented into the web application. Finally, we will illustrate the user the strategies to measure the accuracy of the algorithm and the performance of the developed web application.

Furthermore, we go into detail the many technical aspects of web application development such as user interface design and implementation, use of web services, front-end and back-end development.

Keywords

Web application, predictive algorithms, stock market, stock prediction, machine learning, neuronal networks, LSTM, GRU, data analysis, market trends, historical data.

Resumen

El interés de la tecnología moderna ha hecho más fácil mantenerse informado sobre el mercado de valores y realizar inversiones en consecuencia. Con el creciente interés de particulares e inversores por participar activamente en el mercado bursátil, cada vez es más deseable tener acceso a herramientas que les ayuden a tomar mejores decisiones. El mercado bursátil es un activo complejo y dinámico y puede resultar difícil para estos particulares mantenerse al día de las últimas tendencias, los resultados de las empresas y las noticias mundiales. Mediante el desarrollo de una aplicación web capaz de interactuar con algoritmos predictivos, los inversores pueden acceder a datos en tiempo real y a tendencias históricas para tomar decisiones de inversión con conocimiento de causa.

El objetivo principal del Trabajo de Fin de Grado será demostrar explorar la posibilidad de obtener un algoritmo predictivo sobre el mercado de valores con el que se pueda interactuar a través de una herramienta de aplicación web desarrollada para ayudar a particulares e inversores a optimizar su tiempo y esfuerzo para decidir el mejor activo financiero. Consideraremos la utilización de Redes Neuronales Recurrentes (RNNs) para apoyar una predicción bursátil más precisa, y el desarrollo de un modelo de predicción que pueda ser fácilmente implementado en la aplicación web. Por último, ilustraremos al usuario las estrategias para medir la precisión del algoritmo y el rendimiento de la aplicación web desarrollada.

Además, entraremos en detalle en los numerosos aspectos técnicos del desarrollo de aplicaciones web, como el diseño y la implementación de la interfaz de usuario, el uso de servicios web y el desarrollo front-end y back-end.

Palabras clave

Aplicación web, algoritmos predictivos, bolsa, predicción bursátil, aprendizaje automático, redes neuronales, LSTM, GRU, análisis de datos, tendencias del mercado, datos históricos.

Table of contents

List of figures

List of tables

List of codes

1 Introduction

In recent years, machine learning has improved to the point that it is a very powerful tool to analyse and improve the performance and predictions about future events. It has been used by many institutions and individuals as it has plenty of use cases. For financial purposes it has heavily increased in the past years, therefore it will help investigate about the use cases of machine learning in the stock market.

The stock market is a huge financial market where publicly traded companies' stocks (shares) are bought and sold. It is one of the most important sources of capital for companies, and it allows many individual investors and institutions to buy and sell ownership stakes in publicly traded companies.

One of the main advantages of using machine learning for stock market predictions is its ability to process large amounts of data and identify patterns that may not be visible for the human eye. Another advantage of using machine learning for stock market forecasting, is its expertise in building models that can detect associations between data points. By revealing the links and relationships between data that might be difficult for humans to recognize or track, the models can provide more accurate predictions. This type of forecasting allows investors to make better-informed of the stock market.

People invest in this kind of asset as it helps them earn profits and diversify their investment collection. It also offers a special chance for investors to benefit from potentially profitable investment options. Also, people choose to invest in this kind of asset considering fundamental analysis, which involves financial and economic information.

1.1 Motivation

Stock market is a laborious asset that requires time and a deep understanding of the financial markets. More and more people are entering this sector and because of the vast amount of data available its learning curve is increasing constantly. A web-based interactive tool could help investors and non-experience individuals to learn about the effectiveness of prediction models and make more informed decisions. Additionally, the interactive web-based tool will provide an intuitive and user-friendly platform for users to interact with the predictions making it accessible for a large range of users.

1.2 Objectives

The goal of this work is to develop a web-based application tool to provide an interactive platform that allows users to learn about stock market predictions and make informed decisions using different machine learning prediction algorithms. Since knowledge has no boundaries, the primary goals focus around connecting each aspect together and that works in a coherent structure. The main objectives will now be described:

- Obtain the minimum possible error compared to the actual closing price; it will be visualized through the web-based tool. Historical data will be used for these predictions.
- Provide different techniques to predict the stock market to improve the user decision-making with visual representations.
- Facilitate data analysis. The tool will allow users to filter, search and analyse different prediction models and compare their effectiveness based on various parameters. It will help users to understand the performance of different models and identify patterns and trends in the data that may not be immediately apparent to the human eye.
- The tool will be up to date as it uses data from financial APIs, as well as providing accurate up-to-date predictions.

2

State of the art

Forecasting the stock market is the act of trying to determine the future value of a company's shares or other financial instruments traded on an exchange. Most studies that focus on creating prediction methods for stock market asset values use machine learning techniques. As we all know, machine learning necessitates a level of technical expertise that not everyone possesses. There is an entry barrier for those who are more focused on specialized financial aspects. Then, they must learn about it and spend time learning to understand these models. While these studies on developing a stock prediction model go in great depth into the technical side, they are most times coded in notebook interfaces. These studies are highly beneficial, and there is a limited number of web applications that connect the financial and the technical aspect. Thus, there exists an area for in-depth exploration, which is one of the factors contributing to the composition of this project.

Starting with the core of the project, most machine learning techniques imply the use of times series forecasting, like Long Short-term Memory (LSTM) networks. They have been widely used in recent years for stock market predictions. LSTM networks are a recurrent neural network, a type of artificial neural networks in which the connections between nodes can form a loop. It can learn and remember past patterns in the data, making them very useful for predicting future trends.

Apart from LSTM networks, other studies have also used other machine learning techniques such as Random Forest and Gated Recurrent Unit (GRU). During this project, we will focus on the use of the gated recurrent unit and long short-term memory recurrent networks.

In the following section, we will see the explanation about each of the technologies involved in the development of the project. All the stages in creating a proper web application are interlinked, and the project would lose its meaning if any of them were missing.

2.1. Machine Learning

The technology used for predictions is called “Machine Learning” is a division of artificial intelligence and computer science that uses data and algorithms to generate an imitation to an analysis improving its accuracy. This technology is used in many sectors such as finance, marketing, health, and transportation.

In the context of machine learning, we refer to recurrent neural networks. They can anticipate future values by analysing previously observed values. This forecasting method is commonly applied to non-stationary data, a term used to describe data with statistical characteristics, like mean and standard deviation, it changes over time instead of remaining constant. Apart from time-series data, these models facilitate the use for more complex sequence processing tasks like natural language processing.

2.1.1 Recurrent neuronal networks

Recurrent neural networks (RNNs) represent a type of neural network architecture capable of handling sequential data through the maintenance of an internal memory. They are very effective in scenarios where data sequence and historical context play an important role.

We will now go into detail with the recurrent neuronal networks used on the project, LSTM and GRU, two powerful neuronal network models capable of learning complex long-term dependencies between input and output sequences and make predictions based on past data.

2.1.2 Long Short-Term Memory neuronal network

Long Short-term Memory (LSTM) networks is a type of neural network (RNN) used for a variety of tasks, including stock market predictions, machine translation, speech recognition, and more. It is particularly effective for tasks that involve long-term dependencies, where the past inputs have a significant impact on future outputs,

The LSTM architecture consists of memory cells and gates that regulate the flow of the information through the network. The memory cells are used to store information over time, allowing the network to selectively remember and forget information as needed. The gates are made up of sigmoid and tanh activation functions, were they control the flow of the information into and out of the memory cells. A visual example of the LSTM architecture is shown in **Figure 2.1**.

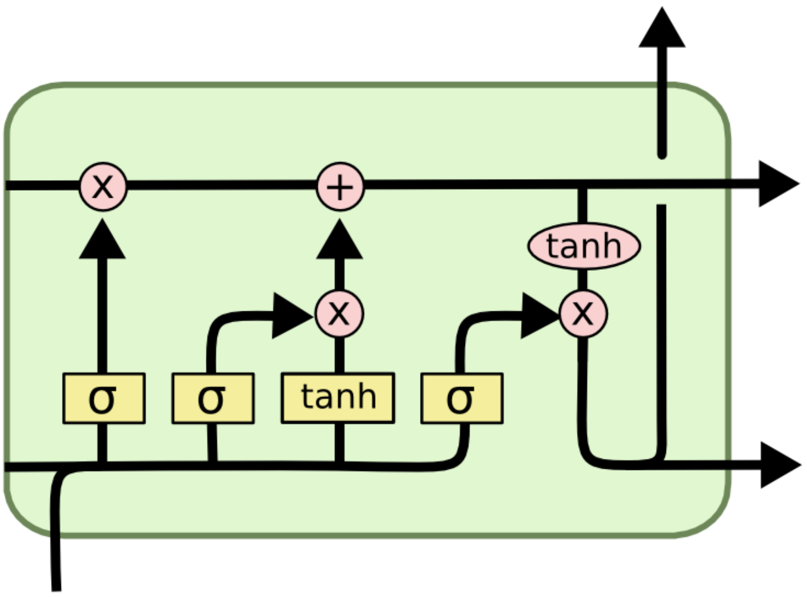


Figure 2.1: LSTM architecture as illustrated in [1]

2.1.3 Gated Recurrent Unit (GRU)

A Gated Recurrent Unit (GRU) is a type of recurrent neural network architecture for sequential data processing. The have been designed to handle sequential data, such a speech signals, time series data and text.

The architecture of the GRU neuronal network consists of an input layer, a hidden layer, and an output layer. The hidden layer contains a set of GRU units, each of which maintains a hidden state vector that is updated based on the input data and the previous hidden state.

The GRU network is computationally efficient and can be trained using standard backpropagation techniques. A visual example of the GRU architecture is shown in **Figure 2.2**.

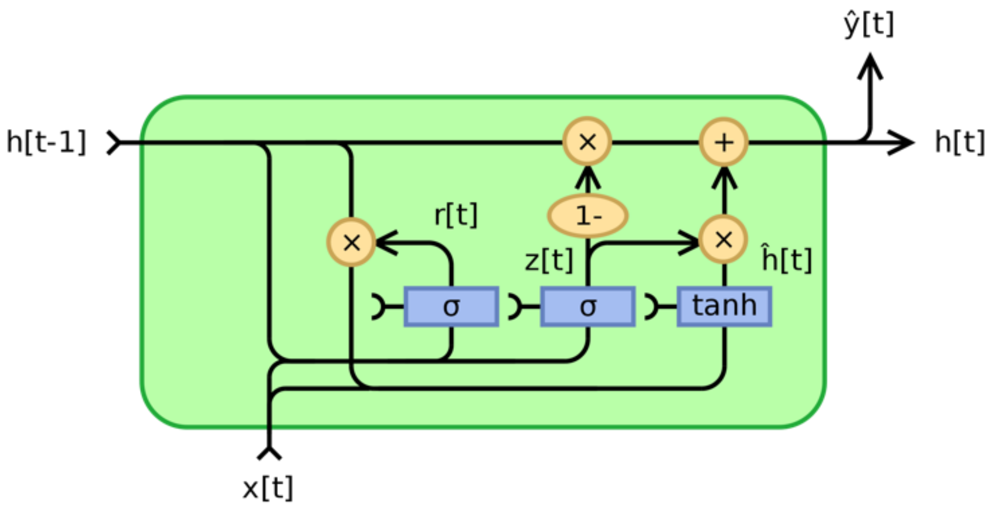


Figure 2.2: GRU architecture as illustrated in [2]

2.1.4 Long Short-Term Memory vs Gated Recurrent Units

The LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) networks are both types of recurrent neural networks that have been widely used for sequential data processing. While both networks are designed to handle long-term dependencies in sequential data, there are some key differences between the two.

One of the main differences between these two neuronal networks is the number of control gates to control the flow of the information. While GRU uses two gates (an update gate and a reset gate), the LSTM networks uses three gates (an input gate, an output gate and a forget gate). This last gate allows to selectively discard information that is no longer relevant, which is particularly useful for tasks such as language modelling and performance.

LSTM is more powerful and flexible than GRU and it is more accurate in larger datasets, but it is also more complicated and susceptible to over-fitting. In fact, GRU uses less training parameters and therefore uses less memory and executes faster than LSTM.

2.2. Python 3

The project will be entirely developed using Python [3], which is a popular, high-level programming language used in program modern web applications. It is the most recent version of the Python programming language and has the strongest emphasis on code readability. Python3 offers a comprehensive and extensive library of pre-built modules and packages and allows developers to create highly functional and dynamic applications. It is an easy to learn language and can be used for both small- and large-scale projects.

To simplify the development of the frontend and backend, there are architecture frameworks widely used to increase efficiency. Furthermore, we will discuss the primary libraries responsible for improving the project's efficiency and functionality.

2.2.1 Framework Django

The framework used to connect the backend with the frontend is Django [4], it is one of the most used frameworks in Python 3 [3]. This framework provides a platform for robust web applications with rapid development and scalability. Using Django's features, developers can make the process of creating the project easier by concentrating on the logic instead of dealing with basic implementation details. This results in quicker progress, improved code, and greater productivity.

The iteration from the user with the components are shown in **Figure 2.3**, it shows the details of main components of the framework, which are the model, view, and template. Details of each component will now be described.

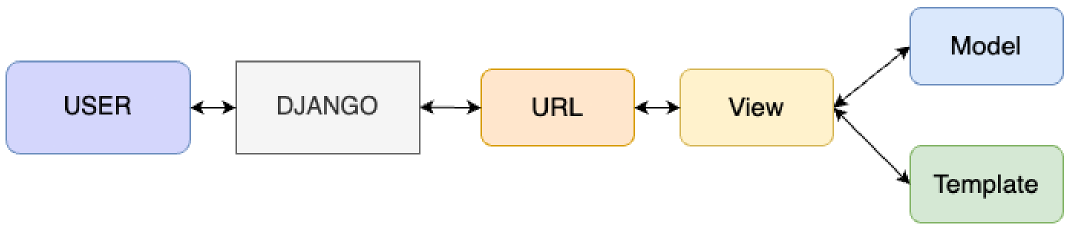


Figure 2.3: Django framework architecture

- The **model component:** responsible for maintaining the data for an application. This includes defining the structure of the data, the relationships and performing operations on data.

- The **view component**: responsible for processing user requests and returning responses. Views can contain both the logic and templates needed to generate a response for a request.

- The **template component**: responsible for defining the visual structure and appearance of a web application. This includes the use of HTML, defining CSS styles and using JavaScript.

Apart from the Model-View-Controller (MVC) architecture, Django offers other features like database abstraction providing an Object-Relational Mapping (ORM) layer that abstracts the database interactions, which allows developers to use Python code instead of SQL queries.

2.2.2 Python libraries

As part of the development, different Python libraries have been used to accomplish the scope of the project. Each library provides a contribution to the overall project, the combination of all these libraries has facilitated the completion of the backend design. These stack libraries are very popular among developers working in the deep learning field, so you might already be familiar with some of them from your machine learning projects. Some of the most important libraries are:

- **Tensorflow:** is an open-source library developed by Google for machine learning and artificial intelligence. It is used to train and deploy machine learning models.
- **Keras:** is an open-source software library that provides a Python interface of artificial neural networks.
- **Scikit-learn:** an open-source Python library for machine learning. It provides implementations of many machine learning algorithms and convenient APIs for efficient model training and prediction.
- **Numpy:** is a Python library that supports a large collection of high-level mathematical functions to operate multi-dimensional arrays and matrices.
- **Pandas:** is a robust Python library designed for managing and analysing data. It offers user-friendly data structures and functions for managing structured data, executing

operations such as filtering, organizing, and consolidating, and allows integration with other libraries for visualization and machine learning purposes.

- **Matplot:** Matplot is a plotting library for Python. It provides many customizable plotting options, allowing to visualize data in 2D and 3D. It is well used with other Python libraries such as Numpy and Pandas.
- **CSV File reading and writing:** Python library used to read and write the financial data used in the model prediction.

2.3. Frontend libraries

As previously mentioned, Django [4] acts as the connector between the back and front-end sides. In the context of frontend development, it is essential to discuss about new technologies and widely used libraries that significantly contribute to develop fast web applications.

Starting with **JavaScript** [5]. It has grown exponentially in recent years due to its capability and relative ease of use. Together with HTML and CSS [6], JavaScript is now essential to create interactive, dynamic websites. More and more applications are being developed with it.

Within JavaScript we discover new relevant libraries such as **ChartJs** [7]**,** which is an open-source tool for creating charts on web pages. It allows to customize and modify various types of charts, including bar charts, pie charts and scatter plots, and it makes it the best choice for visually presenting the results. In this project it will be used to display the results of the stock prediction algorithms.

Furthermore, the utilization of **Bootstrap** [8], an open-source CSS framework, ensures the development of responsive and mobile-first websites. Compatible with all major web browsers, Bootstrap incorporates HTML and CSS-based design templates and provides numerous helpful components and conventions that can help speed up the frontend development process.

3

Analysis and design

This chapter will describe the design realised and the procedures for achieving the project goals. it will be divided in two main layers, the backend, responsible for processing and managing data, logic, and communication between the server and client-side of the web application, and the frontend which is responsible for displaying and interacting with the user interface of the web application, including layout, design, and user experience. In **Figure 3.1** it is described the distributed system architecture.

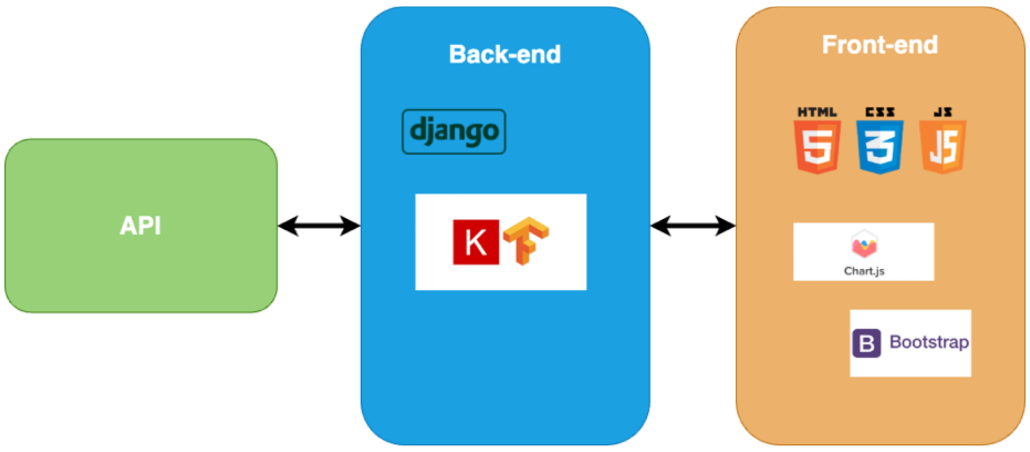


Figure 3.1: Distributed system architecture

Each part of the distributed system is essential for the proper functioning of the application. The idea to separate every part in different layers provides some benefits like:

- Modularity and scalability, it allows independent development and scalability of each component.
- Flexibility and reusability, by separating the backend from the frontend you are allowed to use different technologies and even programming languages for each layer. Additionally, the backend services can be reused across multiple frontend applications.

3.1 Backend development

The backend or “server-side” is the portion of the website that you don’t see. In a distributed system it refers to the components that make up the system’s infrastructure. It also communicates with the frontend, sending and receiving information. We will break down comprehensively the design of the back end of our web application.

3.1.1 Datasets

The data supplied for the evolution of the project as well as the data used to train the models to predict the future values of the company stock comes from an API [9]. In this case we will use the free tier version of the Alphavantage API [10].

As we have just mentioned, the information will be provided by Alphavantage API [10]. It provides financial market data from traditional asset classes such as stocks, ETFs, or mutual funds. It further offers real-time stock data, charts, and analytics tools. It is designed as a RESTful interface allowing users to easily access data from API endpoints. From its many features, we will use the free version tier for the evolution of the project.

The API endpoint used to get historic closing prices of each stock is called *TIME_SERIES_DAILY_AJUSTED* and provides daily open/high/low/close/volume rate values, end-of-day adjusted closing prices, and records of any past stock splits or dividend payments for a chosen global equity, with data spanning over 20 years. The API endpoint call is described in **Code 3.1**. It has few parameters such as symbol, API key and the output size.

```
url = 'https://www.alphavantage.co/query'
params = {"function": "TIME_SERIES_DAILY_ADJUSTED", "symbol": symbol, "apikey": API_KEY, "outputsize": "full"}
try:
    r = requests.get(url, params=params)
except requests.exceptions.HTTPError as err:
    raise SystemExit(err)
```

Code 3.1: *Alphavantage* API endpoint

The response is in JSON format, and contains different metadata including the symbol, the date and the close price used to train the neuronal networks.

```
{
    "Meta Data": {
        "1. Information": "Daily Time Series with Splits and Dividend Events",
        "2. Symbol": "AAPL",
        "3. Last Refreshed": "2023-05-02",
        "4. Output Size": "Full size",
        "5. Time Zone": "US/Eastern",
    },
    "Time Series (Daily)": {
        "2023-05-02": {
            "1. open": "181.03",
            "2. high": "181.78",
            "3. low": "179.26",
            "4. close": "180.95",
            "5. adjusted close": "180.95",
            "6. volume": "61996913",
            "7. dividend amount": "0.0000",
            "8. split coefficient": "1.0",
        },
    },
}
```

Code 3.2: API JSON example response

3.1.2 Neuronal networks

As previously highlighted, the implementation of neuronal networks represents a fundamental aspect of backend development. There are numerous approaches to predict the value of an asset. In this case, deep algorithms are used for the estimation. Inside of deep learning we find a variety of recurrent neural networks that are algorithms used for learning long term dependencies. Two of the most used recurrent neural networks are the Long Short-Term Memory Network (LSTM) and the Gated Recurrent Unit (GRU). The design of layers in these two architectures may have minor differences, but they play a critical role in the overall structure.

The Long Short-Term Memory Network (LSTM) architecture consists of multiple layers, including two LSTM layers with 50 units each, followed by a dense layer containing 1 unit.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 1, 50)             10400

 lstm_1 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 30,651
Trainable params: 30,651
Non-trainable params: 0
```

Code 3.3: LSTM model layers

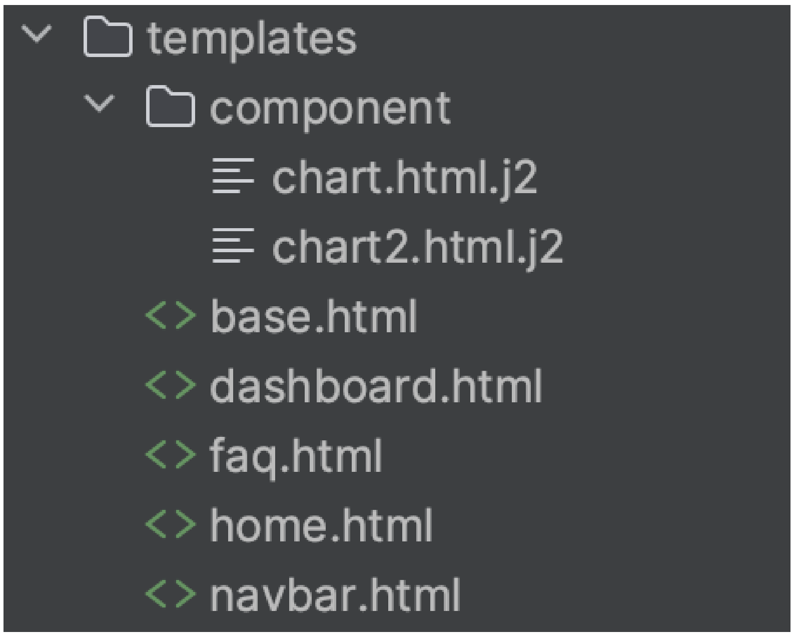
The next neural network utilized is the Gated Recurrent Unit (GRU), which includes two GRU layers. In these layers, the first two layers contains 50 units, followed by the last layer of type dense and with single value.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 1, 50)             7950

 gru_1 (GRU)                 (None, 50)                15300

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 23,301
Trainable params: 23,301
Non-trainable params: 0
_________________________________________________________________

```

Code 3.4: GRU model layers

3.2 Frontend development

Another main part and the most visual one is the front-end. All the views, components and functions used will be described. In **Figure 3.2** you can see the front-end structure divided in different templates.



```
templates
    component
        chart.html.j2
        chart2.html.j2
    base.html
    dashboard.html
    faq.html
    home.html
    navbar.html
```

Figure 3.2: Frontend template structure

3.3 Methodology

The project has been developed following a lean process. A lean process refers to adopting an approach that focuses on efficiency and flexibility by minimizing the time used for each task. Each part of the project has been first created with a simple functional version. Starting from a first web version that worked with all the components of the project, connected with the financial API, and then focusing on improving each part of the system. Following the lean process also helps to finish the scope of the project with a very simple version of it.

4

Implementation

Through this section, the development of the project will be described. The implementation focuses primarily on satisfying the main objectives previously described, as it entails the practical integration of the technologies discussed during the *State of the art* section and the *Analysis and design* section. It is made of two main parts, the backend development and frontend development.

4.1 Backend development

The core of the project and the not visual part is the backend, in this project is divided into two main parts. First one focuses on the neuronal network model which is the most important part since it is used to make the stock predictions. The second main part is the development of bridge between the machine learning functions and the Django framework.

4.1.1 Developing advance neuronal networks

One of the easiest methods to develop neuronal networks is with the use of two famous libraries. Tensorflow and Keras, both open-sourced and designed for fast experimentation with deep neuronal networks.

To begin with the implementation, we will use the use methods from Keras (**Code 6.1**). The process of creating the neuronal network is very systematic for most neural networks and the structure is used in many other projects [11]. We will go into detail for each step in the following paragraphs.

Begin the process **importing the essential data**, in our case from the API earlier described and the use of reading and math libraries, such as pandas and numpy. The use the of the csv library is to read in the data from a CSV file. If the data is not available locally, make use of a financial API to acquire the entire database of stock information. The only metadata necessary for time series prediction is the date and the closing price and date of the corresponding asset.

Once the relevant data has been collected, the next step is **data normalization**. This step is crucial in ensuring that all input features have similar scales, which is instrumental in finding the optimal performance and successful convergence. There are multiple methods to normalize the data, we will use Min-Max scaling technique, which is a type of normalization that will replace each value with an adapter smaller number between zero and one.

```
# Normalize data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(price_list.reshape(-1, 1))
```

Code 4.1: Data normalization

The next step involves **dividing the dataset into training and testing arrays.** This process of splitting the dataset into training and testing arrays is a crucial step in the machine learning and predictive modelling process. The split is necessary to properly measure how well the model works on new information, and to get a better idea of how well it can be applied to different situations. The typically used ratio is 80:20, with 80% of the dataset being allocated to the training array and the remaining 20% reserved for the testing array.

```
# Train and test split
test_ratio = 0.2
training_ratio = 1 - test_ratio
train_size = int(training_ratio * n_days)
test_size = int(test_ratio * n_days)

# Create training and test data
train = scaled_data[0:train_size]
test = scaled_data[train_size - test_size:train_size]
```

Code 4.2: Dataset split into train and test arrays

The following step consist of **building and training the models.** It involves utilizing deep learning libraries described such as TensorFlow and Keras. The process for training the LSTM network and GRU network is similar and will now be discussed.

First, define the number of LSTM layers, hidden units, and activation functions by the specified requirements. Afterwards, the model can be trained using the normalized data that has been divided into training and validation sets. To improve the model's performance, parameters like learning rate, batch size, and epochs need to be further optimized. The same steps are repeated for the GRU (Gated Recurrent Unit) model, which is a distinct form of a recurrent neural network. The code of both models is described in **Code 6.2** and **Code 6.3** for LSTM and GRU respectively. After the model has been developed, it is crucial to load it quickly as we aim to pre-generate each model daily without making the user wait, as demonstrated in **Code 6.4**. All models will be saved in a folder named "models" with a modular name format: *"models/lst_model_"* + *symbol* + *".h5"* and the corresponding historical metric results to ensure rapid loading.

Once the training phase is completed, the next stage involves **testing and making predictions.** The model generates predictions using the test dataset, which are then evaluated by comparing them with the actual values. To measure the accuracy and precision of the models, several metrics are used, such as Mean Squared Error (MSE), Root Mean Square Error (RMSE) and training loss. We will go more into detail of the results in section 5.

```
##############################
# Make predictions and test #
##############################
train_predict_lstm = lstm_model.predict(train)
test_predict_lstm = lstm_model.predict(test)

train_predict_gru = gru_model.predict(train)
test_predict_gru = gru_model.predict(test)
```

Code 4.3: Testing and making predictions

Before accurately evaluate the performance of the models, it is important to **calculate the metrics:** additional evaluation metrics should be calculated such as root mean absolute error (RMSE), mean absolute percentage error (MAPE). These metrics will provide an exhaustive insight into the model's predictive power and accuracy.

Finally, **visualizing the results** by graphing the predicted values against the actual values, as it is essential to understand the performance of the model. Line plots or candlestick charts are used to visualize the results. Through these visualizations, patterns and trends in the stock market data can be identified.

This entire process has been repeated multiple times until we found the optimal model performance. There has been many areas and distinct parameters of improvement which include different layer architectures, activation functions, regularization techniques, or hyperparameters. In section 5, we will explore in-depth the adjustments made to the parameters for determining the optimal model.

4.1.2 Unify backend development and frontend templates

The bridge between the two main architecture layers is the controller, is an internal part of the Django framework’s design. This fundamental element includes defining URLs and handling the administration of views controllers. The role of the controller is of importance since it guarantees smooth communication between the involved components.

In **Code 4.4**, we demonstrate how the dashboard request is managed through the controller. When a GET request is received, the controller does not perform any action; however, when a POST request comes in, it invokes the *get_stock_price* method, a function found in the "main" where all the implementations of the neural networks have been handled. After that, it processes the data into a list, applying a reverse operation and limiting the number of days. We choose 365 days for frontend visualization, making it more convenient for users to observe the predicted data, which only accounts for 20% of the entire dataset. Finally, the stock_prediction_lstm method is called, responsible for loading the pre-generated model.

```python
def dashboard(request):
    context_dict = {}
    if request.method == 'POST':
        symbol = request.POST.get("symbol")
        n_days = 365
        data = main.get_stock_price(symbol)

        date_list = list(data.keys())
        close_price_list = list(data.values())
        date_list.reverse()
        close_price_list.reverse()
        close_price_list = close_price_list[-n_days:]
        date_list = date_list[-n_days:]
        results = main.stock_prediction_lstm(symbol, n_days)
```

Code 4.4: Dashboard controller view

4.2 Frontend development

As we have mention in the analysis and design section. In this section, we will talk about all the Front-end development of our web application, as we mention with the lean methodology used. Initially, the templates were coded in plain HTML, and as the project progressed, CSS and JavaScript styles were incorporated. The primary design comes from the CSS framework Bootstrap, while the charts are created using the JavaScript library ChartJs [7].

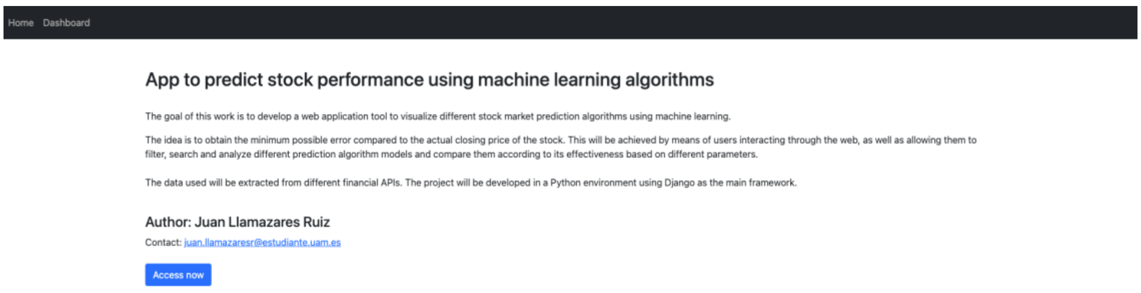
The most challenging aspect of frontend development is effectively presenting a neural network and stock forecasting to users who lack technical expertise. It is crucial to illustrate and clarify the details, it is very important to focus on user experience while trying to scale the number of users on the web application.

In this section, the primary templates that have been implemented will be demonstrated and explained.

4.2.1 Home view

The first template is the home view, the first view the user will access on the website. it contains a short description about the project and some contact information. It has an action button to access the dashboard view. It is crucial, from the first impressions the user understands the utility of the website and the importance of having a support/contact form in case it is needed.

The illustration is shown in **Figure 4.1**.



App to predict stock performance using machine learning algorithms

The goal of this work is to develop a web application tool to visualize different stock market prediction algorithms using machine learning.

The idea is to obtain the minimum possible error compared to the actual closing price of the stock. This will be achieved by means of users interacting through the web, as well as allowing them to filter, search and analyze different prediction algorithm models and compare them according to its effectiveness based on different parameters.

The data used will be extracted from different financial APIs. The project will be developed in a Python environment using Django as the main framework.

Author: Juan Llamazares Ruiz

Contact: juan.llamazaresr@estudiante.uam.es

Figure 4.1: Home view

4.1.1 Dashboard view

Moving forward with the viewpoints, we encounter the dashboard view, which serves as the primary interface, it includes all the information the user needs to study and analyse the predictions, as well as frequently used questions link in case there is any question. The view will receive two types of requests. Plain get request will only show the search input box and some extra information. The plain get initial request is shown in **Figure 4.2**.

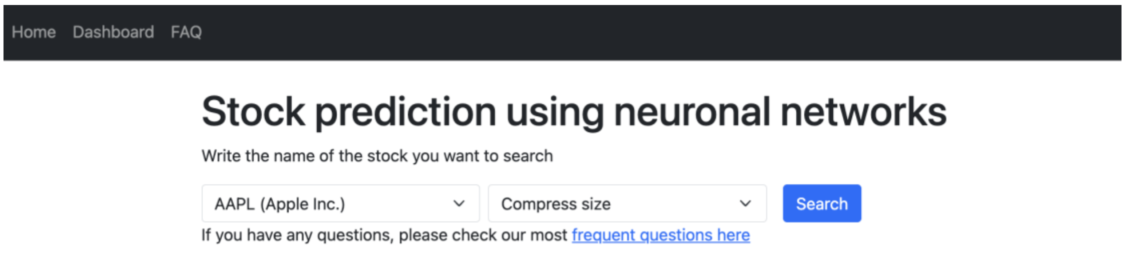
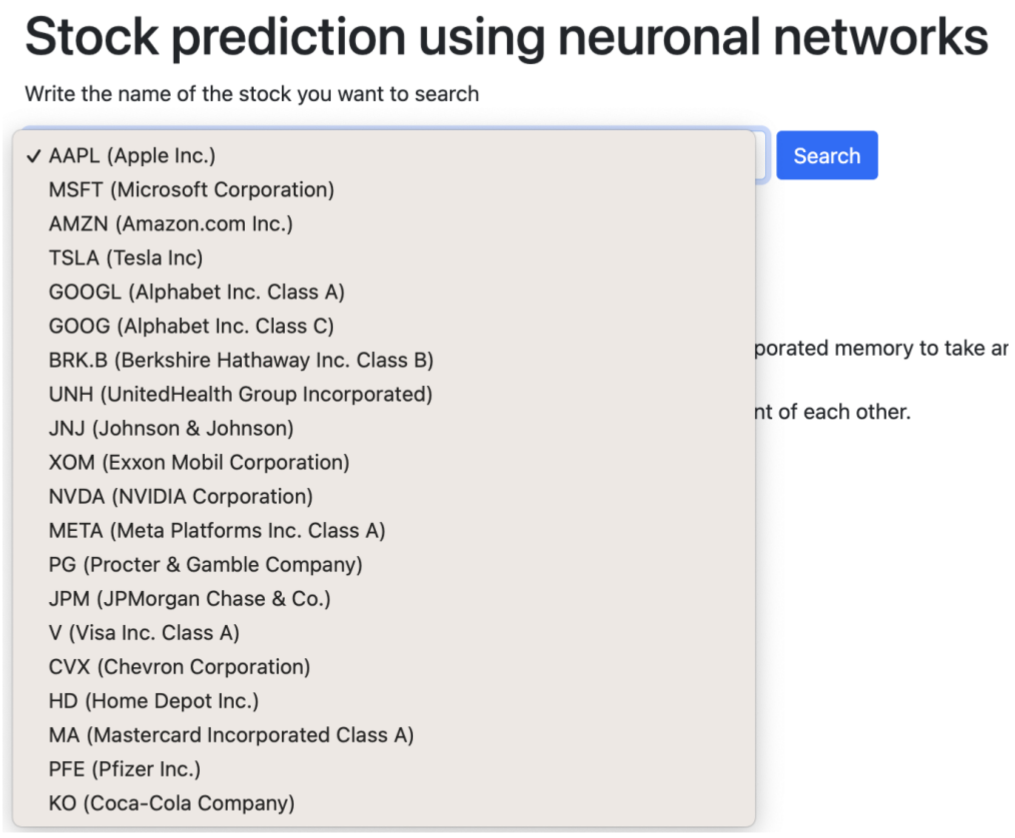

Home
Dashboard
FAQ
Stock prediction using neuronal networks
Write the name of the stock you want to search
AAPL (Apple Inc.)
Compress size
Search
If you have any questions, please check our most frequent questions here

Figure 4.2: Dashboard view

The search input box will have a list of all the possible assets. In section 5 we will go into detail of the decision to list the assets instead of allowing it all. The chosen assets are listed in **Table 1**. The input selection inbox will display all the possible options, allowing the user to choose the most suitable choice to continue with the process. Once the "Search" button is clicked, the form will be submitted, and a request will be sent to the backend. In **Figure 4.3**, the input search box illustration is presented.



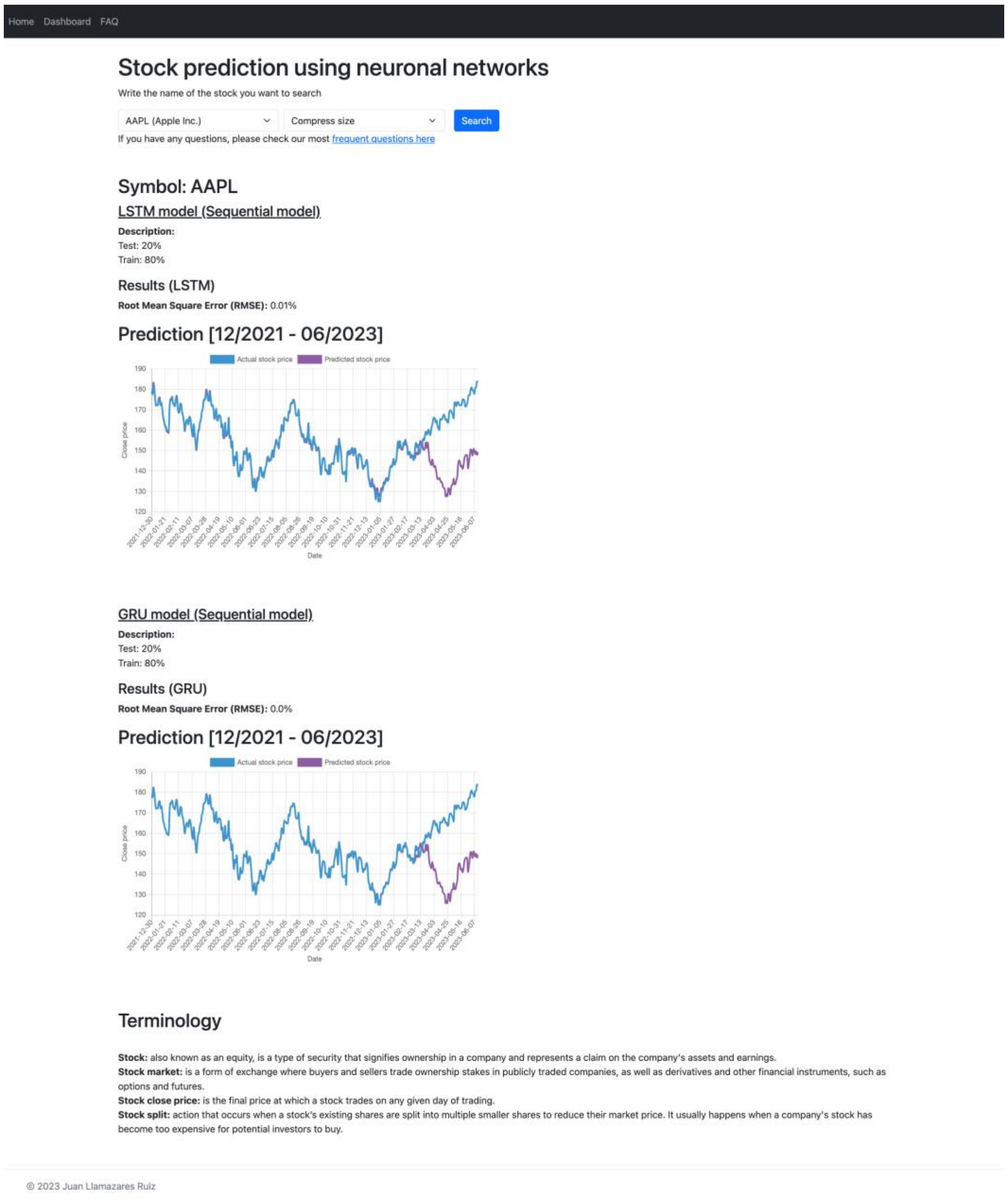
Stock prediction using neuronal networks

Write the name of the stock you want to search

- ✓ AAPL (Apple Inc.)
- MSFT (Microsoft Corporation)
- AMZN (Amazon.com Inc.)
- TSLA (Tesla Inc)
- GOOGL (Alphabet Inc. Class A)
- GOOG (Alphabet Inc. Class C)
- BRK.B (Berkshire Hathaway Inc. Class B)
- UNH (UnitedHealth Group Incorporated)
- JNJ (Johnson & Johnson)
- XOM (Exxon Mobil Corporation)
- NVDA (NVIDIA Corporation)
- META (Meta Platforms Inc. Class A)
- PG (Procter & Gamble Company)
- JPM (JPMorgan Chase & Co.)
- V (Visa Inc. Class A)
- CVX (Chevron Corporation)
- HD (Home Depot Inc.)
- MA (Mastercard Incorporated Class A)
- PFE (Pfizer Inc.)
- KO (Coca-Cola Company)

Figure 4.3: List of available assets

Once the POST request calls the back end, and if the stock model is generated, it will return an array of data, including the symbol, the number of days, and the stock price data. The code then forms a list of the keys in the data variable (date list) and the values (close price list). The date list and close price list are then both reversed, and the close price list is limited to the number of days specified. The stock prediction LSTM and GRU results are then stored in variables with the metrics obtained. The view then changes with the data obtained as shown in **Figure 4.4**.



Stock prediction using neuronal networks

Write the name of the stock you want to search

If you have any questions, please check our most frequent questions here

Symbol: AAPL

LSTM model (Sequential model)

Description:
Test: 20%
Train: 80%

Results (LSTM)

Root Mean Square Error (RMSE): 0.01%

Prediction [12/2021 - 06/2023]

GRU model (Sequential model)

Description:
Test: 20%
Train: 80%

Results (GRU)

Root Mean Square Error (RMSE): 0.0%

Prediction [12/2021 - 06/2023]

Terminology

Stock: also known as an equity, is a type of security that signifies ownership in a company and represents a claim on the company's assets and earnings.
Stock market: is a form of exchange where buyers and sellers trade ownership stakes in publicly traded companies, as well as derivatives and other financial instruments, such as options and futures.
Stock close price: is the final price at which a stock trades on any given day of trading.
Stock split: action that occurs when a stock's existing shares are split into multiple smaller shares to reduce their market price. It usually happens when a company's stock has become too expensive for potential investors to buy.

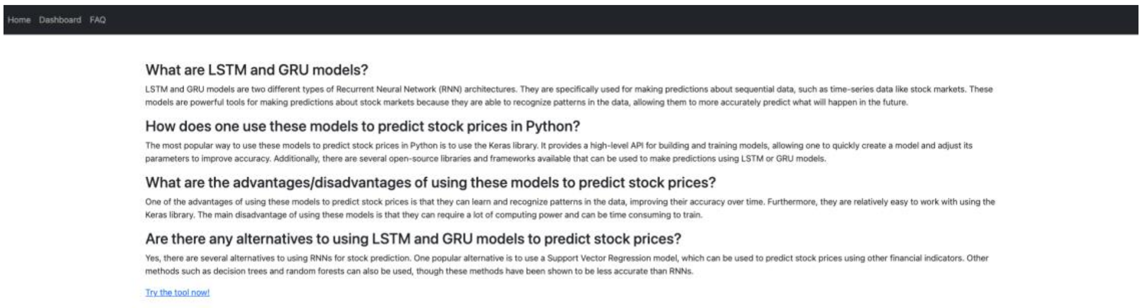
© 2023 Juan Llamazares Ruiz

Figure 4.4: Full dashboard view

The displayed view features various charts and textual information, representing two distinct neural networks being utilized. Each network is outlined with critical details such as the numbers assigned for testing and training, the specific date range, and the RMSE (root mean square error) metric data. Following these visual representations, there is an informative section that elaborates on several key financial terms to enhance comprehension of the web application's functions.

4.1.2 Frequently asked question’s view

Developing a user experience FAQ (Frequently Asked Questions) as part of the project is essential for understanding technical and financial terms, as it makes it easier for users to understand the use case of the tool. The reason for it, is that it will decrease the customer service requests and support needs. The view is shown in **Figure 4.5**.



What are LSTM and GRU models?

LSTM and GRU models are two different types of Recurrent Neural Network (RNN) architectures. They are specifically used for making predictions about sequential data, such as time-series data like stock markets. These models are powerful tools for making predictions about stock markets because they are able to recognize patterns in the data, allowing them to more accurately predict what will happen in the future.

How does one use these models to predict stock prices in Python?

The most popular way to use these models to predict stock prices in Python is to use the Keras library. It provides a high-level API for building and training models, allowing one to quickly create a model and adjust its parameters to improve accuracy. Additionally, there are several open-source libraries and frameworks available that can be used to make predictions using LSTM or GRU models.

What are the advantages/disadvantages of using these models to predict stock prices?

One of the advantages of using these models to predict stock prices is that they can learn and recognize patterns in the data, improving their accuracy over time. Furthermore, they are relatively easy to work with using the Keras library. The main disadvantage of using these models is that they can require a lot of computing power and can be time consuming to train.

Are there any alternatives to using LSTM and GRU models to predict stock prices?

Yes, there are several alternatives to using RNNs for stock prediction. One popular alternative is to use a Support Vector Regression model, which can be used to predict stock prices using other financial indicators. Other methods such as decision trees and random forests can also be used, though these methods have been shown to be less accurate than RNNs.

Try the tool now!

Figure 4.5: Frequently asked questions view

5

Testing and results

Since one of the objectives of this work is to evaluate the prediction of the models developed, we will go into the detail on all the tests carried out to evaluate the efficacy of our web application tool. The approach to find the model has been adjusted to the time and effort of the project. Therefore, the results will probably not be as accurate as if the project was fully dedicated into finding the best model. Later, some other possible improvements and alternatives will be explained.

As we previously stated, the significance of testing and its outcomes has been crucial in the project's development. Given that the project involves designing a web application, the testing will involve in the improvement of the technology used in the frontend. Without testing different parameters and machine learning techniques, the results themselves would not be as accurate as the obtained ones.

5.1 Metrics

The evaluation of the metrics is divided into two parts. First, we evaluate the quality provided by the two models LSTM and GRU. The most popular metrics used for it are Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE). MAPE metric measures the size of the error in percentage terms. A MAPE value between 0 and 10 indicates accurate predictions, while anything above 20 indicates a problem with the model.

```
def mape(y_test, y_pred):
    y_test, y_pred = np.array(y_test), np.array(y_pred)
    return np.mean(np.abs((y_test - y_pred) / y_test)) * 100
```

Code 5.1: Method to calculate MAPE

RMSE (Root Mean Squared Error) is typically used if the errors are normally distributed and have a constant variance. A good value for RMSE depends on the specific application and the model being used. Generally, a value under 0,5 is a good result.

```
def rmse(y_true, y_pred):
      return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axis=-1))
```

Code 5.2: Method to calculate RMSE

5.2 Recurrent neuronal network model tests

The aim is to determine the optimal architecture for a Recurrent Neuronal Network (RNN) model to predict future stock prices.

To evaluate the testing of the neuronal network, a visual library for testing, specifically the Python library Matplotlib, has been implemented. It illustrates the performance of the different models evaluated and represents the effectiveness of the stock prediction process through graphical plots.

Going into the architecture detail, a crucial optimization technique utilized is known as Adam optimizer [12], which increases the efficacy of neural network weights through continuous iterations through the training data.

5.2.1 Long Short-Term Memory

For training and testing the LSTM model, an Adam optimizer [12] was used. Additionally, 100 epochs and a batch size of 32. The details of the layer structure are described in **Code 3.3**.

After different parameter and function changes, we found the best model to predict future stock prices values. This layer architecture and parameters will be used with all the assets used for the project. It is not the best approach, but it will work to the intention of the web tool.

For most of the models created the training loss and MAPE metric has decreased drastically from the first 0-30 epochs, after that number of epochs, it has slowly decreased. The downward trend in the visual representation can be observed in **Figure 5.1** and **Figure 5.2**.

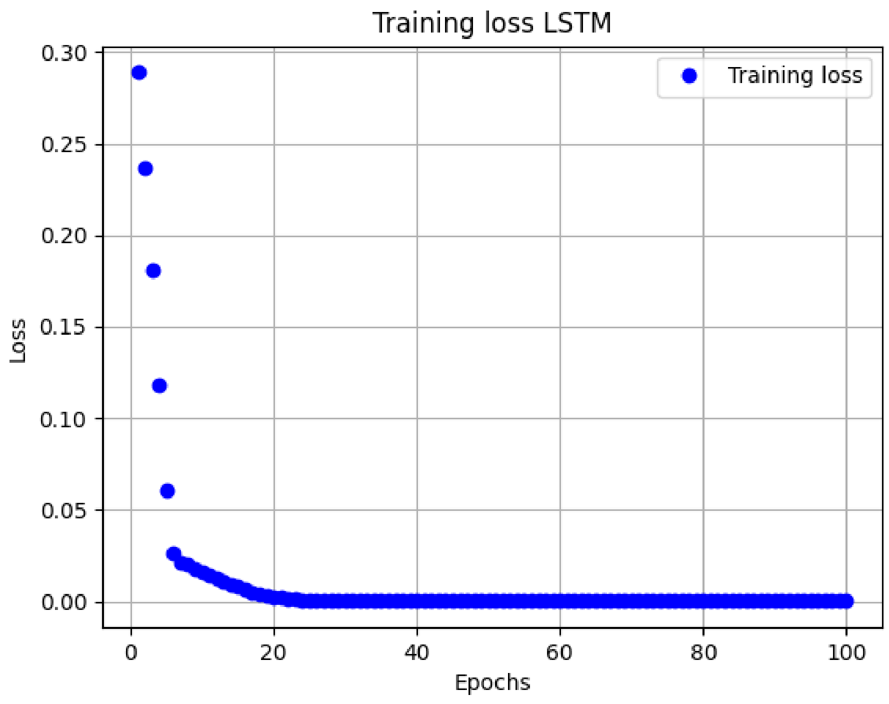


Figure 5.1: Training loss from Long Short-Term Memory

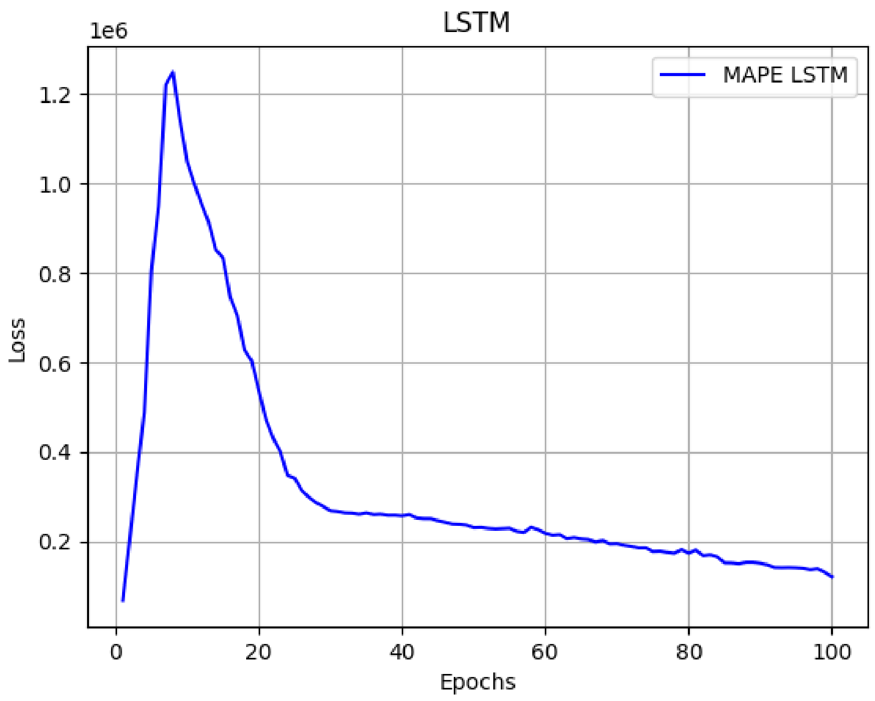


Figure 5.2: MAPE from Long Short-Term Memory

In **Code 5.3**, it is observed that the training phase takes around 12 milliseconds for every epoch. The root mean square error decreases, eventually hitting a value of 0,0092.

```
Epoch 95/100
10/10 - 0s - loss: 9.4108e-05 - root_mean_squared_error: 0.0097 - 12ms/epoch - 1ms/step
Epoch 96/100
10/10 - 0s - loss: 9.0036e-05 - root_mean_squared_error: 0.0095 - 12ms/epoch - 1ms/step
Epoch 97/100
10/10 - 0s - loss: 8.7545e-05 - root_mean_squared_error: 0.0094 - 11ms/epoch - 1ms/step
Epoch 98/100
10/10 - 0s - loss: 8.4107e-05 - root_mean_squared_error: 0.0092 - 11ms/epoch - 1ms/step
Epoch 99/100
10/10 - 0s - loss: 8.4127e-05 - root_mean_squared_error: 0.0092 - 12ms/epoch - 1ms/step
Epoch 100/100
```

Code 5.3: LSTM last training epochs

5.2.2 Gated Recurrent Unit

The GRU model has been compiled using the Adam optimizer [12] with 100 epochs and a batch size of 32. The structure is described in **Code 3.4**.

On the other hand, we have the results from training the Gated Recurrent Unit model. From initial observations, the GRU as expected, appears to be trained more rapidly in comparison to the Long Short-term Memory. In under 10-15 epochs, there is a significant decrease in training loss, and the MAPE falls below 20 within 20 epochs. Following this, both metrics follow a linear decreasing trend. The results are illustrated inf **Figure 5.3** and **Figure 5.4**.

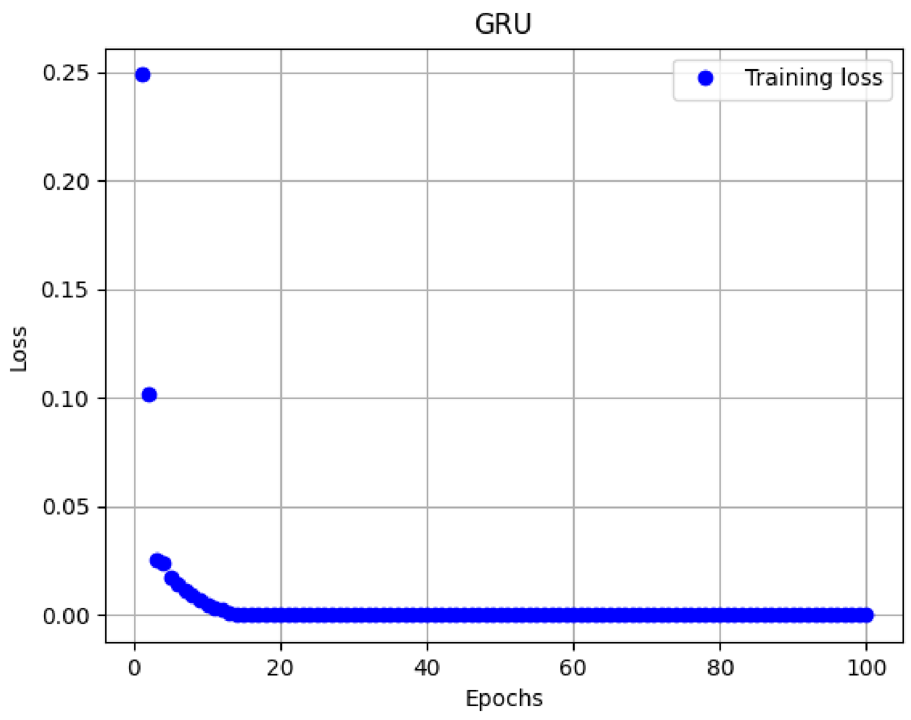


Figure 5.3: Training loss from Gated Recurrent Unit

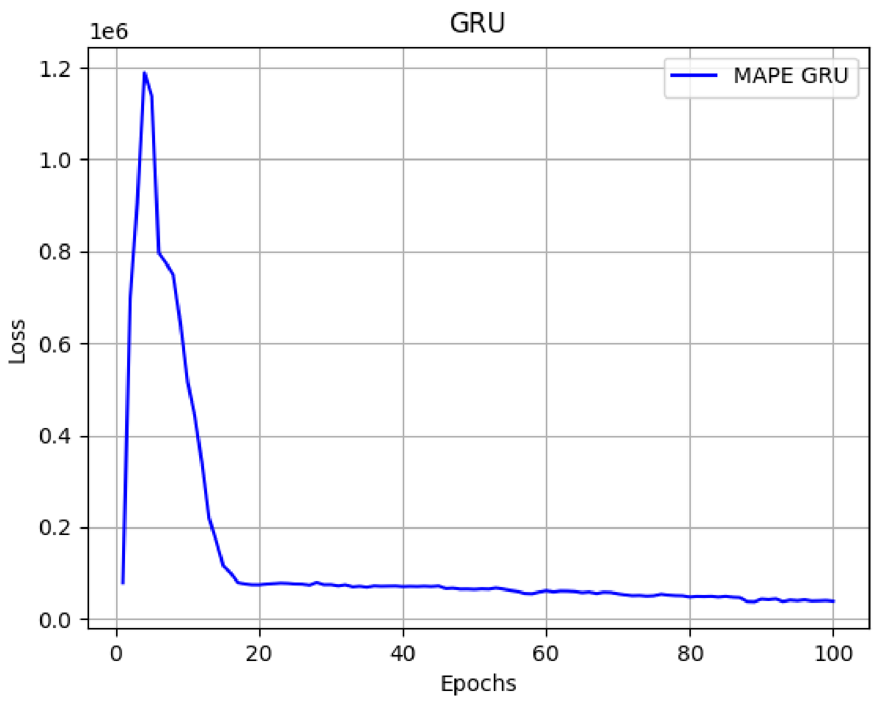


Figure 5.4: MAPE from Gated Recurrent Unit

Additionally, when examining **Figure 5.4**, the training process takes around 13 milliseconds for each epoch, and as the training progresses, the root mean square error continues to diminish until it eventually reaches a value of 0,032.

```
Epoch 95/100
10/10 - 0s - loss: 1.0621e-05 - root_mean_squared_error: 0.0033 - 12ms/epoch - 1ms/step
Epoch 96/100
10/10 - 0s - loss: 1.1555e-05 - root_mean_squared_error: 0.0034 - 12ms/epoch - 1ms/step
Epoch 97/100
10/10 - 0s - loss: 1.0973e-05 - root_mean_squared_error: 0.0033 - 12ms/epoch - 1ms/step
Epoch 98/100
10/10 - 0s - loss: 1.1966e-05 - root_mean_squared_error: 0.0035 - 12ms/epoch - 1ms/step
Epoch 99/100
10/10 - 0s - loss: 1.1714e-05 - root_mean_squared_error: 0.0034 - 13ms/epoch - 1ms/step
Epoch 100/100
10/10 - 0s - loss: 1.0100e-05 - root_mean_squared_error: 0.0032 - 13ms/epoch - 1ms/step
```
Code 5.4: GRU last training epochs

5.2.3 Conclusion results

Comparing the outcomes of both models can be a difficult task, as several aspects can impact their performance. The financial market has experienced numerous fluctuations in recent years, primary caused by market trends and conditions. These changes can potentially affect the outcomes of time series networks that rely on historical closing prices. When it comes to technical factors, both networks have similar outcomes in terms of the time taken for training and the accuracy of the predictions made.

Another point of view of comparing results, particularly with the LSTM model, is to increase the number of epochs for training as it appears there is still room of improvement. However, the issue with this approach is that when applied to a larger scale of assets, it could significantly increase the training time without necessarily yielding a substantial difference in the final forecasting outcome.

5.3 User experience tests

At the end, comprehensive backend testing will ensure the accuracy and reliability of the results that will later be visualized at the frontend. It is important to recognize the significance of user experience, they will be the real clients of the project. The initial aspect to consider is that I have done the testing myself and being aware that comprehending the project's objective and outcomes may likely be insufficient. Therefore, usability and user experience will be essential for larger-scale development. A couple of decisions were made to enhance the user experience, with two of them detailed below.

The initial decision can be seen in **Figure 5.5**, which displays the count of dates from the last day minus 365 days. Now, each date and its corresponding closing price are shown. The significant improvement involves filtering the data displayed on each axis of the chart, resulting in improved graph readability.

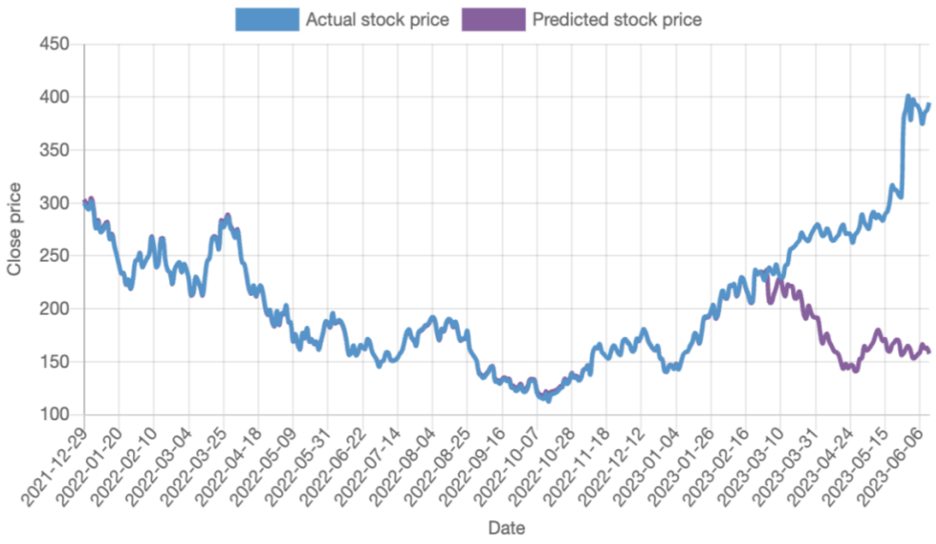


Figure 5.5: Example chart to improve user experience

To enhance the frontend performance, we have incorporated a filter that enables users to access either full or compressed data. The complimentary version displays information from the past year, while the premium edition would provide data in its entirety. However, the full data size feature has not been implemented in the application yet due to the absence of pricing plans and incomplete integration of the API with the full-time data. The search filter is shown in **Figure 5.6**.

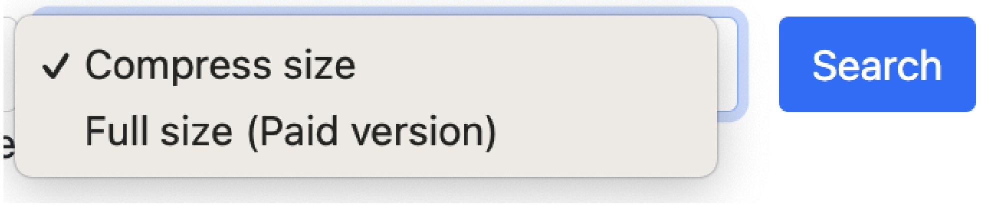


Figure 5.6: Size filter parameters

5.4 Challenges encountered

During the development of the project, some challenges have been found while trying to unify the financial part with the technical part.

The first and biggest found difficulty is the creation of the neuronal network models, as it takes an enormous consideration of time, and letting the user wait that long to generate a model it not a proper solution since we want to keep their attention. Therefore, we found a solution, the models are pre-generated. They will be saved as “.h5” files, which is the typical file format to store and distribute trained models in the HDF5. The “.h5” file allows efficient storage and retrieve of large models, making it very usable for sharing and deploying trained neural networks. A “.json” file will also be used to store and load the metrics used for testing.

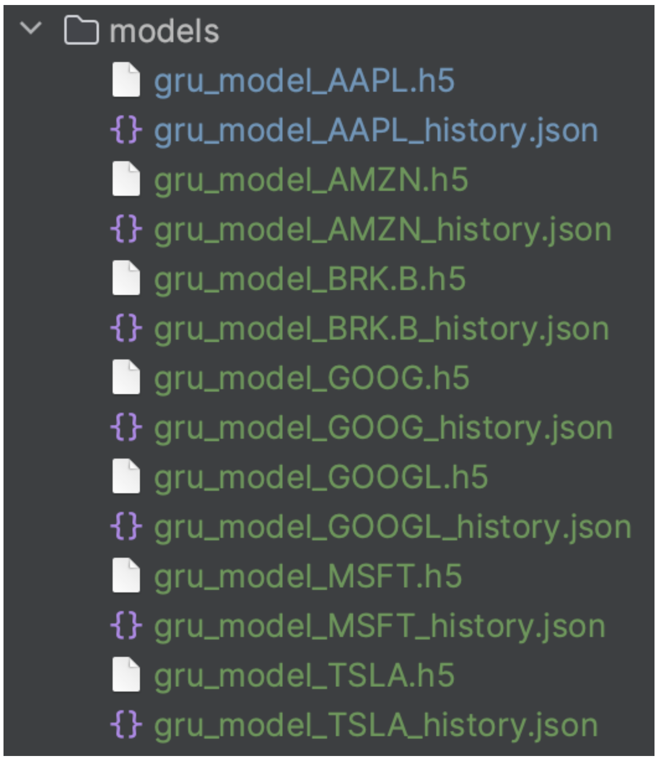

models
gru_model_AAPL.h5
gru_model_AAPL_history.json
gru_model_AMZN.h5
gru_model_AMZN_history.json
gru_model_BRK.B.h5
gru_model_BRK.B_history.json
gru_model_GOOG.h5
gru_model_GOOG_history.json
gru_model_GOOGL.h5
gru_model_GOOGL_history.json
gru_model_MSFT.h5
gru_model_MSFT_history.json
gru_model_TSLA.h5
gru_model_TSLA_history.json

Figure 5.7: Model storage structure

A sample of the JSON results structure can be found in **Code 5.5**. This structure not only proves useful for utilizing metadata at a later stage but also offers a valuable resource for future research looking for network performance and development.

```
{
   "loss":[
      7.640528565389104e-06,
      7.6318056017044e-06,
      ...
   ],
   "root_mean_squared_error":[
      0.0027625723741948605,
      ...
   ],
   "mean_absolute_percentage_error":[
      35901.578125,
      ...
   ]
}
```

Code 5.5: JSON file model structure

Another problem found is the stock market has thousands of different assets and our models can be scaled in relation to the size and complexity. The short solution is that only the biggest 20 stocks in the US market will be added, listed in **Table 1**. An appropriate solution could involve identifying the most suitable and efficient model but finding a balance between speed and accuracy is crucial and very difficult to find.

An additional issue encountered when attempting to deploy the website to web applications such as Heroku is that most free trials do not permit the storage of numerous models created in local files. Instead, they necessitate the use of a database, which this project has not utilized. One potential solution is to employ a file system service to store the models, allowing them to be accessed through requests made from the backend.

A final issue that impacts the outcomes derived from the API is the stock split error., It occurs due to the limitations of the free tier API being used. To resolve this, finding a more advanced and comprehensive API solution or upgrading to a paid version of the existing API could ensure an enhanced and more accurate data analysis experience. As can be observed in **Figure 5.8**, there is a significant jump in the historical closing price from one day to the next, which consequently leads to a non-accurate trained model

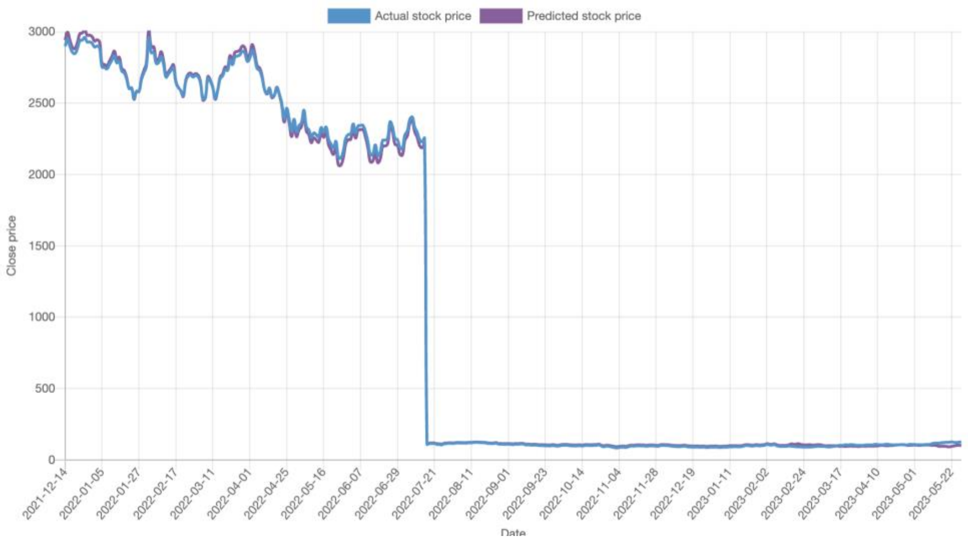


Figure 5.8: Split error

5.5 Testing improvements

Throughout the testing phase, we discovered various enhancements, with most of them relying on project budget or time dedication. There are several potential solutions for boosting the performance of a neural network, which include:

- Improve the financial API, since the free tier API used for the project did not consider splits and other data, which lead to have limited results. By incorporating a better source of data, it will improve the accuracy of the prediction.
- Improve neuronal network using hyperparameter tuning techniques to optimize the overall architecture.
- Using an alternative deep learning framework like Pytorch [13], in contrast to Keras, is ideal for developers looking to rapidly build, train, and assess their models. Pytorch offers a faster option and some debugging capabilities [9].

6 Conclusions

6.1. Conclusions

To conclude with the project, we have successfully created a web application to allow users and other individuals interact and experience with a tool to predict some financial assets, in this case stocks with recurrent neuronal networks. It can also teach investors another approach of understanding the future market trends using machine learning technology.

Talking about the neuronal networks used on the project, we can conclude that stock prediction based only on the historical price's movement will not be the only factor that affects the future price movement. However, using this recurrent neuronal network like LSTM or GRU will provide in a way the stock trend for the future days. Time series networks play a crucial role in identifying patterns over time and predicting future events effectively, particularly when the market demonstrates a linear trend or specific patterns.

The conclusion of this technical bachelor thesis is that stock prediction using neural networks should not solely be relied upon. While neural networks can be used to analyse the past data and trends to generate predictions, these predictions should be considered in combination with an analysis of other market trends and financial data. Using a combination of both methodologies will increase the accuracy of the predictions being made. Instead of relying solely on the output generated from a web application tool, a comprehensive analysis of the markets and stocks should be conducted. It will help to provide a deeper insight into the stocks and the trends of the market which should then be used to generate the most accurate predictions.

6.2. Future work

There are various options to improve this project, we will go examine some of the most effective ways to keep the development of the web application tool. Based on the objective, there are various lines of improvement and progress, for example:

- **Enhance user experience**, by focusing more on the front-end aspect of the web application. This can be achieved dedicating more time into users, such as implementing an authentication method, or displaying additional information already given through the financial API.

- As discussed in the testing and results section, there is still a lot of space of improvement in the neuronal network. In my opinion, it is currently the worst aspect of the system, given that these neural networks will always contain errors from external sources, particularly since they are time series forecasting networks. Therefore, it is crucial to dedicate more effort to improving the neuronal network.
- **Improve input data** and focus on additional financial numbers that may impact the future the asset's future value. This approach demands greater financial knowledge, but it remains a viable choice for improvement. Incorporating more input data, such as historical volume, daily highest and lowest values, and even going into company results like earnings could significantly boost the results obtained.
- Another alternative is delving into **autoencoders**, which are a type of neuronal network where the input and the output data are identical. One benefit is that they don't need any human assistance, such as data labelling. The architecture provides an encoder that converts the input into a lower-dimensional representation. Autoencoders can be individually trained on various independent stock market returns and then incorporated into other end-to-end neural networks while retaining the ability to be globally optimized through back-propagati

Bibliography

References

[1] (). *Long short-term memory (LSTM) architecture*. Available: https://en.wikipedia.org/wiki/Long_short-term_memory.

[2] (). *Gated recurrent unit (GRU) architecture*. Available: https://en.wikipedia.org/wiki/Gated_recurrent_unit.

[3] (). *Python*. Available: https://www.python.org/.

[4] (). *Django framework documentation*. Available: https://docs.djangoproject.com/en/4.2/.

[5] (). *JavaScript documentation*. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScrip.

[6] (). *HTML tutorials*. Available: https://www.geeksforgeeks.org/html/.

[7] (). *ChartJs documentation*. Available: https://www.chartjs.org/docs/latest/.

[8] (). *Bootstrap documentation*. Available: https://getbootstrap.com/docs/5.3/getting-started/introduction/.

[9] (). *REST API (RESTful API)*. Available: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API.

[10] (). *Alphavantage API*. Available: https://www.alphavantage.co/.

[11] T. B. Shahi *et al*, "Stock Price Forecasting with Deep Learning: A Comparative Study," *Mathematics,* vol. 8, *(9),* 2020. . DOI: 10.3390/math8091441.

[12] (). *Adam optimizer*. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[13] (). *Keras vs Tensoflow vs Pytorch*. Available: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article#:~:text=TensorFlow%20is%20an%20open%2Dsourced,because%20it's%20built%2Din%20Python.

Terminology

Technical terms

- **API** [9]**:** Application Programming Interface is a set of functionalities and protocols for facilitating communication between two or more computer systems.
- **Uniform Resource Locator (URL):** It refers to a web address or link that indicates the position of an online resource.
- **RESTful interface**: is an API designed with the principles of Representational State Transfer, utilizing standard HTTP methods for communication between clients and servers in a stateless, scalable, and simple structure.
- **POST request**: is an HTTP request method used to send data to a server for processing. It is typically used when creating or updating a resource, such as submitting a form on a website.
- **GET request:** is an HTTP request method used to send data to a server for processing. It is typically used when creating or updating a resource, such as submitting a form on a website.
- **Machine Learning (ML):** is a subfield of artificial intelligence. It examines the design and construction of algorithms capable of learning from data, even without specific programming. These ML algorithms can autonomously recognize hidden patterns and tendencies, generate predictions, and form conclusions, all without outside input.
- **Hierarchical Data Format version 5 (HDF5):** is a file format designed to store and organize large amounts of numerical data.
- **JSON (JavaScript Object Notation):** is a user-friendly, light data exchange format commonly used for data transmission between servers and web applications, as well as data storage and organization.

Financial terms

- **Stock**: also known as an equity, is a type of security that signifies ownership in a company and represents a claim on the company's assets and earnings.
- **Stock market**: is a form of exchange where buyers and sellers trade ownership stakes in publicly traded companies, as well as derivatives and other financial instruments, such as options and futures.
- **Stock close price:** is the final price at which a stock trades on any given day of trading.
- **Stock split:** action that occurs when a stock's existing shares are split into multiple smaller shares to reduce their market price. It usually happens when a company's stock has become too expensive for potential investors to buy.

Appendices

Appendice A

The appendix of this bachelor's thesis contains a variety of supplemental figures, tables and blocks of code that have been cited within the project.

```python
from keras.models import Sequential
from keras.layers import Dense, GRU
from keras.layers import LSTM
```

Code 6.1: Imports from library Keras used for the neuronal networks

```python
def create_lstm_model(train, metrics):
    lstm_model = Sequential()
    # First layer
    lstm_model.add(LSTM(units=50, return_sequences=True, input_shape=(train.shape[1], 1)))
    # Second layer
    lstm_model.add(LSTM(units=50))
    lstm_model.add(Dense(1))
    # Compiling the RNN (Recurrent Neuronal Network)
    lstm_model.compile(optimizer="adam", loss='mse', metrics=metrics)
    # Fitting the RNN to the Training set
    history = lstm_model.fit(train, train, epochs=100, batch_size=32, verbose=2)
    return lstm_model, history
```

Code 6.2: Method to create LSTM neuronal network

```python
def create_gru_model(train, metrics):
    gru_model = Sequential()
    # First layer
    gru_model.add(GRU(units=50, return_sequences=True, input_shape=(train.shape[1], 1)))
    # Second layer
    gru_model.add(GRU(units=50))
    gru_model.add(Dense(1))
    # Compiling the RNN (Recurrent Neuronal Network)
    gru_model.compile(optimizer="adam", loss='mse', metrics=metrics)
    # Fitting the RNN to the Training set
    history = gru_model.fit(train, train, epochs=100, batch_size=32, verbose=2)
    return gru_model, history
```

Code 6.3: Method to create GRU neuronal network

```python
try:
    if new_model:
        lstm_model, history_lstm = create_lstm_model(train, metrics)
        lstm_model.save("models/lstm_model_" + symbol + ".h5")

        json.dump(history_dict_lstm, open("models/lstm_model_" + symbol + "_history.json", "w"))

        gru_model, history_gru = create_gru_model(train, metrics)
        gru_model.save("models/gru_model_" + symbol + ".h5")
        history_dict_gru = history_gru.history
        json.dump(history_dict_gru, open("models/gru_model_" + symbol + "_history.json", "w"))
    else:
        try:
            lstm_model = keras.models.load_model("models/lstm_model_" + symbol + ".h5")
            history_dict_lstm = json.load(open("models/lstm_model_" + symbol + "_history.json", "r"))
        except OSError:
            lstm_model, history = create_lstm_model(train, metrics)
            lstm_model.save("models/lstm_model_" + symbol + ".h5")
            history_dict_lstm = history.history
            json.dump(history_dict_lstm, open("models/lstm_model_" + symbol + "_history.json", "w"))

except:
    print("[ERROR] Creating/Reading model")
```

Code 6.4: Method to store / load neuronal networks

List of chosen assets from the stock market.

Company name	Symbol
Apple Inc	AAPL
Microsoft Corporation	MSFT
Amazon.com Inc	AMZN
Tesla Inc	TSLA
Alphabet Inc. Class A	GOOGL
Alphabet Inc. Class C	GOOG
Berkshire Hathaway Inc. Class B	BRK.B
UnitedHealth Group Incorporated	UNH
Johnson & Johnson	JNJ
Exxon Mobil Corporation	XOM
NVIDIA Corporation	NVDA
Meta Platforms Inc. Class A	META
Procter & Gamble Company	PG
JPMorgan Chase & Co	JPM
Visa Inc. Class A	V
Chevron Corporation	CVX
Home Depot Inc	HD
Mastercard Incorporated Class A	MA
Pfizer Inc	PFE
Coca-Cola Company	KO

Table 1: Assets selected from the stock market

Appendice B

How to install and run the project

All the programming code can be found within a GitHub repository, and you can access it by visiting the provided link below:

Repository: https://github.com/Juanllamazares/TFG

Tutorial

To successfully install and run the project, please follow the steps outlined below:

Before proceeding, ensure that you have Python 3 installed on your system. Once confirmed, open a terminal or command prompt, and navigate to the project directory.
To install all the necessary Python 3 dependencies, run the following command:

Step 1: Install Python 3 Dependencies

```
pip3 install -r requirements.txt
```

This command will automatically install all the required packages and libraries needed for the project to run smoothly.

Step 2: Initiating the Server

To initiate the server, you have two options:

- **Option 1:** Using Makefile. If you have Makefile support, simply run the following command:

```
make run
```

- **Option 2:** Using Django's manage.py script, If you do not have Makefile support, execute the following command in the terminal or command prompt:

```
python3 backend/manage.py runserver
```

Note: In the above command, "backend" represents the directory containing the Django project. Adjust the path if necessary.

<u>Step 3:</u> Database Migration. The project should not have migrations as there is no database, but in case there are any pending database migrations or the default Django migrations, execute the following command to apply them:

```
python3 backend/manage.py migrate
```

This command will ensure that the database schema is up to date with the latest changes in the project.