UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

22|23

Undergraduate Thesis

Modelado y predicción de series temporales financieras con redes neuronales

Daniel Brito Sotelo

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC

www.uam.es

Autonomous University of Madrid

Autonomous University of Madrid
HIGHER POLYTECHNIC SCHOOL

Bachelor's Degree in Computer Science

Undergraduate Thesis

Modelado y predicción de series temporales financieras con redes neuronales

1920_2147_COSE

Author: Daniel Brito Sotelo
Advisor: David Domínguez

September 2022

Daniel Brito Sotelo
Modelado y predicción de series temporales financieras con redes neuronales

Daniel Brito Sotelo
C\ Francisco Tomás y Valiente Nº 11

PRINTED IN SPAIN

Acknowledgments

To my mum, who, for six months had to endure with my constant rambling about this study.

Resumen

En estos últimos años los avances en el área del aprendizaje automático han conseguido grandes resultados en problemas que hace décadas parecían imposibles. Uno de estos problemas es el de la predicción en el área de las finanzas.

El mundo de las inversiones y finanzas está presente constantemente en nuestras vidas, apareciendo tanto en televisión como en artículos en internet. La extensión de las inversiones y su desmistificación ha permitido a empresas emergentes recibir la financiación que necesitan, y a la gente sacar rentabilidad a sus ahorros.

El objetivo de este trabajo es emplear información disponible en diferentes fuentes de internet, ratios financieros, métricas de la salud de una empresa, etc. para construir modelos de aprendizaje automático, que puedan predecir o clasificar el precio futuro de una acción. Todos los modelos propuestos se probarán sobre datos históricos para decidir si pueden formar estrategias de inversión que obtengan rentabilidades positivas.

Finalmente compararemos el desempeño de nuestros modelos frente a otros instrumentos populares y comunes como los fondos indexados, bonos del estado o las cuentas de ahorros de alto rendimiento.

PONER RESULTADOS. PONER MODELOS.

Palabras clave

Aprendizaje automático, finanzas, series temporales, inversion, classificacion

Abstract

In recent times machine learning has achieved significant breakthroughs in many areas that seemed impossible just decades ago. On the other hand the investment world is an ever present highly debated one, that is present throughout our lives, appearing constantly in the news and online articles. It has allowed for new and upcoming businesses to receive much needed financing, while giving people a way to make their savings grow.

The objective of this study is to use widely available investment data, such as prices, financial ratios, metrics, etc. to build models, such as recurrent neural networks, like LSTM, and classification models, SVM, MLP or Random Forest, that can predict or classify a stock's future returns. All proposed models are tested on past data to find out if they are able to form strategies with positive returns over long periods of testing.

Finally, these results will be compared to other common and popular investment strategies like index funds, government bonds or high-yield savings accounts.

Keywords

Machine learning, finance, time series, recurrent neural network, investment, clasification.

Table of Contents

Lists

List of algorithms

List of codes

List of equations

List of figures

List of tables

1

Introduction

The study of stock markets and the search for sound and succesful investment strategies is a practice as old as the actual stock echanges themselves. Ever since the inception of the first investment funds a wide variety of strategies have been developed. Furthermore, with the rapid development of technology, many investors have been adopting approaches based on statistics and technology. Due to the large success that machine learning has seen in various uses such as natural language processing, image recognition, product recommendations, in this work we take a look at its application in the financial sector.

the aim of this work is to introduce machine learning techniques in this field and test their effectiveness, highliting possible drawbacks or limitations they might have.

1.1. Motivation

Market prediction is a field that is ever present in our lifes. Will this company grow? Is there going to be another market crash? Will this company become the next Apple? These questions are ever present in investment forums, publications and newspapers, the responses are varied, and the reasons behind the predictions are equally inconsistent.

What's more, nowadays, with the advances of technology, investing has become more available than ever, there exist a myriad of online brokers and investment apps, that in a bid to become more attractive than their competitor offer little to none comissions. With the click of a button you can buy Apple's stock, something that 15 years ago would have taken a few calls with brokers and even more effort in finding a reliable broker. Thus, investment has emerged as a real and accessible way of helping people's savings grow.

Therefore it would be interesting to see if the information available online to the average person, could be used to improve investing decisions and how that can benefit the 'small' investor, ensuring that they have access to a reliable way to shield their savings from inflation, rising costs, or to plan their retirement.

1.2. Objectives

The main objective of this work is to introduce machine learning into the field of stock investment, and to try to build succesful investment strategies based on the results from the proposed models.

On that sense we can divide our work into two different approaches, the first one is analogous to technical analysis. Technical analysis is an investment physolophy that argues that past prices and their patterns can be used to predict future returns. We will do so using recurrent neural networks, which have seen wide use in the prediction of time series. This strategy focuses heavily on purchasing and selling assets with a very short time frame between operations, which can be a week, a day, or even in many cases, less than a day, in what are called intraday trades.

The other strategy presented in this work could be considered a part of fundamental analysis, this valuation style, used by funds such as Berkshire Hathaway or the Magellan Fund [1] is based on the long term, and focuses heavily on studying a companies fundamentals to determine if it is undervalued, valued fairly or overvalued. We will gather a plethora of financial ratios, and measurements of a companies debt, income and revenues, to build a classification model that can predict stocks that will grow over a certain threshold in a year's time.

Lastly we will choose the most successful strategy and perform some backtests on it. Which consist on testing it ind different time periods, observing their returns and performance, and comparing them to how other popular financial products, such as state bonds or index funds, performed during the same time frame.

1.3. Structure

- Chapter 1: Introduction: We introduce the subject that this work will be based on, we will detail the motivation behind it as well as the objectives we'd like to achieve.
- Chapter 2: State of the art: We will take a look at similar works in the field, explaining their achievement briefly, as well as introducing theoretical concepts relevant to our machine learning models and concepts that will appear further on in our work.
- Chapter 3: Stock price prediction using recurrent neural networks: We will attempt to predict one day ahead stock prices using a recurrent neural network.
- Chapter 4: Stock return classification. Clasification of stocks into two cathegories depending on their yearly return.
- Chapter 5: Conclusions. Presentation of the conclusions obtained from the previous experiments, comparison of approaches with other investment strategies.

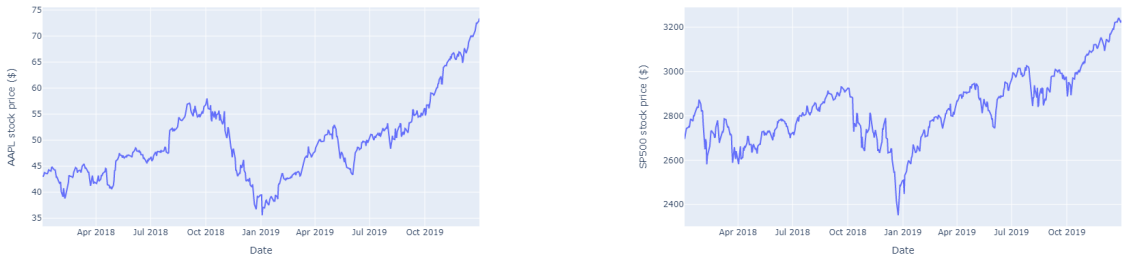
2

State of the Art

In this chapter we will look into the most used machine learning models used in predicting stock prices or stock returns. Furthermore, we will also cover some important theoretical concepts related to the financial valuation of companies, and their financial ratios, which we will be using in our classification models.

2.1. Temporal Series

A temporal series is a set of data points indexed by the time they occurred. Examples of temporal series include electricity consumption, volcanic activity charts, or, as we focus in this work: financial time series. There exist a myriad of time series in the financial sector. Perhaps the most known are, the graph of a stock price's over time, or the price of an index such as the S&P500, NASDAQ, etc.



(a) Apple's stock price.

(b) S&P500 price.

Figure 2.1: Examples of time series of Apple's and the S&P500 index's price.

In financial contexts it is quite common to see them represented as in 2.1, or via a candlestick chart. This type of chart encodes in a single graph, the Opening, Closing, High and Low price of the assest in each time period, this is visible in 2.2.

Figure 2.2: APPL candlestick chart.

2.2. Models

Now let's briefly explore the different models used in the problem of predicting price as a temporal series.

2.2.1. Recurrent Neural Networks for Temporal Series

For the problem of predicting stock prices given past prices in a short term frame, we will be using Recurrent Neural Networks (RNN). Due to their architecture they have seen heavy use in predicting sequential data. For example both [2] and [3] a LSTM recurrent neural network is used in the field of Natural Language Processing (NLP) for inference, while other's, like [4] use them for precipitation prediction.

The architecture of a recurrent neural network can be observed in 2.3, where the loop on the diagram of the left can be unrolled to produce the diagram on the right.

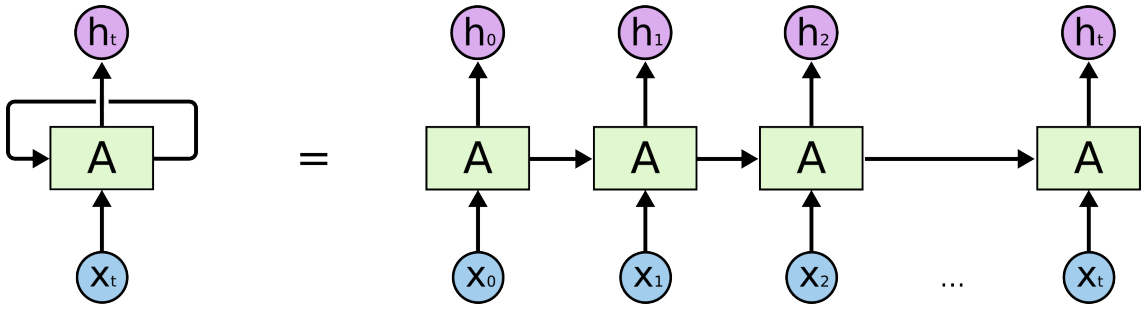


Figure 2.3: Recurrent Neural Network architecture.

The fact that recurrent neural networks have this loop, means that information is allowed to persist.

This is the key aspect that has led them to be used for sequential data, as can be observed by their application to problems of speech recognition [5].

Vanishing Gradient problem in Recurrent Neural Networks

The biggest drawback that recurrent neural network's suffer from is the vanishing gradient in long term dependencies. The value of the hidden layer output at time t is determined by a combination of the input at that same timestep t and the hidden layer value at the previous timestep $t-1$, thus it can be expressed as

$$h_t = \sigma(Uh_{t-1} + Wx_t + b_h) \tag{2.1}$$

where, U and W are weight matrices and b_h the bias parameter, while σ is the activation function.

The problem comes from how the activation functions are defined, the most common ones being $sigmoid$ and $\tanh$ are

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3}$$

And their respective derivatives can be expressed in terms of the original functions as:

$$\sigma'(x) = \sigma(x)(1-\sigma(x)) \tag{2.4}$$

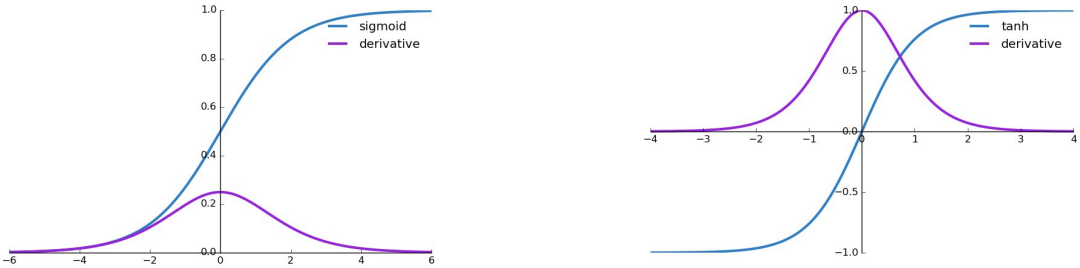
$$\tanh'(x) = 1 - \tanh^2(x) \tag{2.5}$$

Graphically we can see their representation in 2.4, and we can clearly observe how both derivatives have values in the $(0, 1)$ range, in the case of the sigmoid even smaller: $(0, 0{,}25)$. This explains why the $\tanh$ is generally favoured instead of sigmoid.

Due to the fact that gradient descent is performed to update the weights in order to learn, the update of the weights at t timesteps ago is a product of t gradients. This product leads the gradient of the error to converge to very small values. This means that the model isn't able to learn long term dependencies well, since the weights of those barely get updated in the learning process.

Choosing a ReLU 2.6 or Leaky ReLu as activation function can help contain this behaviour, since the derivative is 1 for values larger than 0.

$$\text{ReLU}(x) = \max(0, x) \tag{2.6}$$

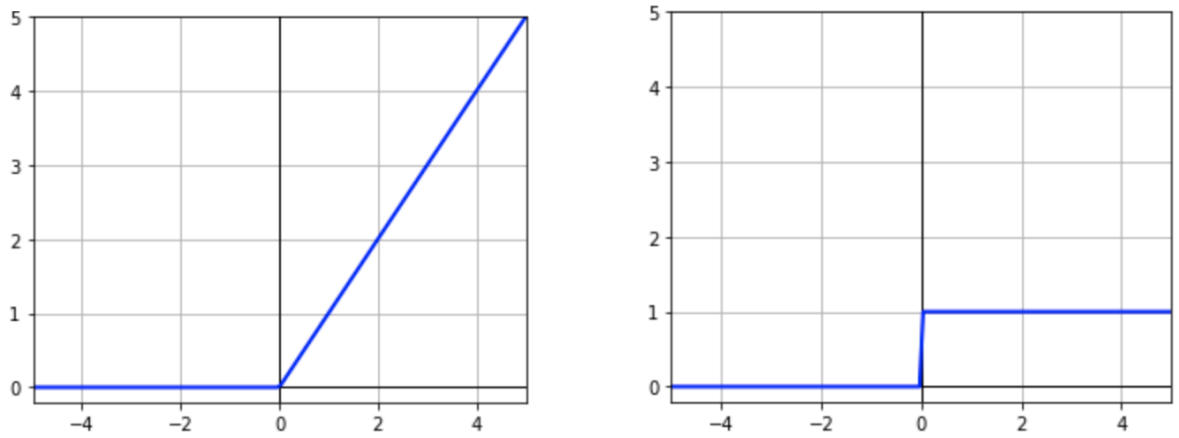


(a) Sigmoid activation and it's derivative.

(b) $\tanh$ activation and it's derivative.

Figure 2.4: Sigmoid and $\tanh$ activation functions and their derivatives

$$\text{LeakyReLU}(x) = \max(ax, x) \text{ for } a > 0 \tag{2.7}$$



(a) ReLU activation function.

(b) Derivative of the ReLU.

Figure 2.5: Relu activation function and its derivative.

But LSTMs or Long Short Term Memory networks don't suffer from this problem. We will now see why this is the case.

2.2.2. LSTM Networks

LSTMs, or Long Short Term Memory, were fist introduced by Hochreiter and Schidhuber in [6] directly as a fix for the problem of recurrent neural networks' inability to store long term information, a problem which Hochreiter also found and documentated in [7]. Let's briefly observe how LSTM neural networks address this problem.

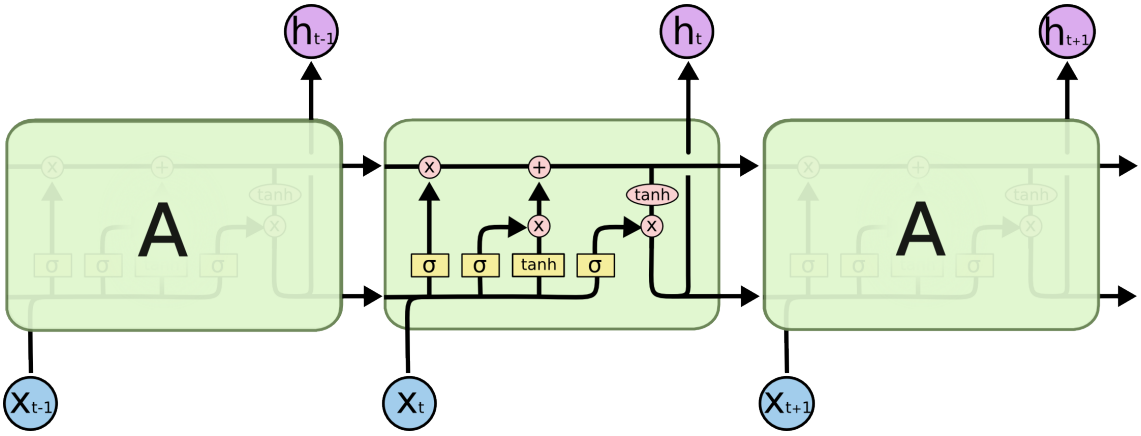


Figure 2.6: Long Short Term Memory architecture.

In 2.6 the elements represented with the symbol σ are called gates. They serve to modulate the amount of information let through in each case, outputting a value between zero and one, 0 meaning we forget all the information coming in and 1 we keep all of it.

There are three gates, the forget gate which decides how much information we will keep from the previous hidden state. While the input and output gate, which decide on the ingoing and outgoing information respectively.

This way, a network can learn to keep information that occurred several timesteps ago intact

2.3. Financials and Stock Valuation

Value investing is the phylosophy of some well known investors such as Peter Lynch, Warren Buffet or Charlie Munger. It is based on picking stocks that are trading for less than their intrinsic or real value, and using several metrics and financials to gauge this instrinsic value. The second part of this work will try to use machine learning models to mimic a value investors logic when picking stocks.

There are some examples of financial papers where the relation between certain ratios and the future returns have been studied like [8], where the author exposes that the Price-to-Earnings (P/E), Book-to-Market (B/M) and Dividend Yield have a slight ability to predict stock returns. While [9] demonstrates that choosing companies with strong financial statements and high B/M can be used to

form profitable strategies. Yielding 23 % annualized return in a span of 20 years between 1976 and 1996.

2.3.1. Prediction of stock returns using classification techniques

The number of research papers which use classification methods to generate investment strategies is not high. But there exist some, such as [10], where hybrid machine learning models are used to predict the daily return direction of the SPDR S&P500 ETF ($SPY) using a series of financial and economic variables, which include returns of companies inside the S&P500 index, other indices around the world, economic indicators, etc.

2.3.2. Backtesting

Backtesting is the method that evaluates a trading strategy by testing it against historical data and observing the returns it would have produced. The idea behind it is that if a wide array of backtest with different timeframes and inputs are tested, the results on past data will carry on into the future, so a model that performed well in the past is more likely to do so in the future than one that didn't show this good performance on past data. Ideally the backtesting strategies should span a variety of timeframes, which should always be long enough, so the chance of a random model performing well is minimized. Also longer timeframes, allow us to test our algorithms against different market conditions that can knowlingly or unknowlingly affect it's performance. Whenever possible, and in the case of machine learning models, the strategy should also be tested againsts stocks that it wasn't trained on.

2.4. Opportunity Cost

Opportunity cost is the benefit lost when choosing between different investment options. The concept of opportunity cost will be used to compare our backtested strategy's performance against other strategies. There exist and infinite amount of possible strategies, and a great number of those are assured to outperform any great strategy that an investor can come up with. For example, one very popular investment that many individuals use is monthly investments into an S&P500 index fund. If you were to invest $100\$$ montly between the years 2012 and 2018, you would obtain an annualized return of $14{,}073\,\%$, having invested a total of $7200\$$. While investing the same $100\$$ only on Amazon stocks would yield a $33{,}51\,\%$ annualized return, in both cases we have ignored taxes and fees. But to be fair we will compare our models with some of the most common investment vehicles used in the US, which will be an S&P500 index fund, US Treasury bonds, and high yield savings accounts.

2.5. Alpha

The *alpha* of a stock or investment, is a measure of their performance when put into context with a chosen benchmark index. It is one the most important measures to compare strategies that span different time periods and to assess the real meaning behind single strategy's return. For the purposes of this work we will be considering the benchmark index to be the S&P500.

$$alpha = \text{performance of model} - \text{S\&P500 performance} \tag{2.8}$$

The reason why it is important to judge the return of an investment is that investing in an S&P500 index fund is seen by many, especially non institutional investors as a relatively safe investment, that obtains moderate returns with a low risk, and needing virtually no research from the investor.

Let's see it's usefulness via a simple example.

Say we have two investments strategies, both a year long for simplicity. The first spans the dates between January 2019 and January 2020 and has a $15\,\%$ *return, while the second is between January 2007 and January 2008 and has a return of* $5\,\%$*. At first glance one might think that the former is superior, since it boasts a* $10\,\%$ *higher return, but if we take into account that the returns of an S&P500 index fund between January 2019 and January 2020 were of* $25{,}727\,\%$*, while between January 2007 and January 2008 it yielded* $-3{,}188\,\%$*, with this in mid the summary of both investments can be seen in 2.1.*
Now the results seem a lot different than we first thought, and we can safely conclude that the investment made between 2007 and 2008 is better than the one between 2019 and 2020.

Frame 2.1: Example to show how the alpha of two investments with different timeframes is used to compare them

Dates	Return	S&P500	$alpha$
01/2007-01/2008	$5\,\%$	$-3{,}188\,\%$	$8{,}188\,\%$
01/2019-01/2020	$15\,\%$	$25{,}727\,\%$	$-9{,}273\,\%$

Table 2.1: Summary of two hypothetical investments with their respective alpha.

2.6. Editor or IDE

In terms of development environments for machine learning there are a great number of alternatives, either local or online collaborative. Some examples include: *Visual Studio Code*, *PyCharm*, *JupyterLab* or, what we will be using, *Google Colab*. The reason being that we can take advantage of it's GPU hardware acceleration capabilities, [11], as well as the collaborative nature of it. The main drawback being that you have to be connected to the internet to make use of it.

3

Recurrent Neural Networks for Stock Price Prediction

In this section we will be building a database of historical prices for a stock, and then training recurrent neural networks on these past prices, finally we will visualize the results and discuss their viability. For these models we will be utilizing the following *Python* libraries: *keras, keras_tuner* and *tensorflow*, the data processing will be done using *pandas, numpy* and *sklearn.preprocessing*. Meanwhile for the extraction of the historical prices we will use the *requests* library.

We will be sticking to one stock to perform all the predictions, the chosen one is Apple's, with ticker $AAPL. The results exhibited in this section will, therefore, vary depending on which stock or financial product we decide to choose.

3.1. Data Extraction

There are many different resources on the internet which offer stocks historical prices. Such as Morningstar, Investing.com or Yahoo Finance. For our case study we will be using the latter. [12]

To fecth the data from the website, there are some python packages and libraries that can aid us, althought they frequently suffer from problems connecting to the yahoo finance API, which is sometimes unavailable for long periods of time. So for the sake of stability we will perform a GET petition directly to the webpage and filter the response to extract the table with the prices. The result after being loaded into a dataframe can be observed in 3.1. The data ranges from the first of January 2010 to the first of January 2022.

For the rest of the study we will focus on the closing price of the stock in question, so we discard the rest.

3.2. Data Preparation

Firstly we will transform our data to logarithmic scale, which will enhance the learning capabilities of the gradient descent that we apply later on.

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-22	44.325001	44.445000	44.150002	44.250000	42.147121	108434400
2018-10-10	56.365002	56.587502	54.012501	54.090000	52.111137	167962400
2020-06-09	83.035004	86.402496	83.002502	85.997498	84.818840	147712400
2019-07-12	50.612499	51.000000	50.549999	50.825001	49.537205	70380800
2019-01-16	38.270000	38.970001	38.250000	38.735001	37.448109	122278800

Table 3.1: Sample of Apple's historical prices taken from Yahoo Finance.

There are no missing, NaN or infinite values so there is no work to be done in that aspect.

We won't be scaling the data either, this technique is used across different types of machine learning applications to make data points in different data ranges into a common one, which helps the learning algorithms better learn information about our datasets. The most common examples are *StandardScaler, Normalizer, MinMaxScaler* from *sklearn.linear_model*. However with stock prices, although they do have a minimum price of $0\$$, they don't have a maximum value, theoretically they could always reach higher values than they have ever presented up until today. For example, from 1980 to 1985 the highest value that the Bank of America stock reached was $4{,}484\$$, while the next year it reached $6{,}983\$$.

This means that since the scaling is applied separately to the train and test subsets, we would be testing our model on data with a scale never seen before by our model which yields worse results.

To build the dataset that we will use to train our model, we apply the *sliding window* method. We will segment the signal in superposed windows of 60 days. So, in our case, for we will build the window of the price on the 60 previous days. A visual representation of this method can be seen in 3.1.

Now let's look at the metrics used to measure the effectiveness of the models we will propose.

3.3. Metrics

To asses how accurate the prediction of future prices is, we will be using the following metrics, widely employed for continuous variables.

- MSE, Mean Squared error, which is calculated with the following expression

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

- MAE, Mean Absolute error,

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|^2. \tag{3.2}$$

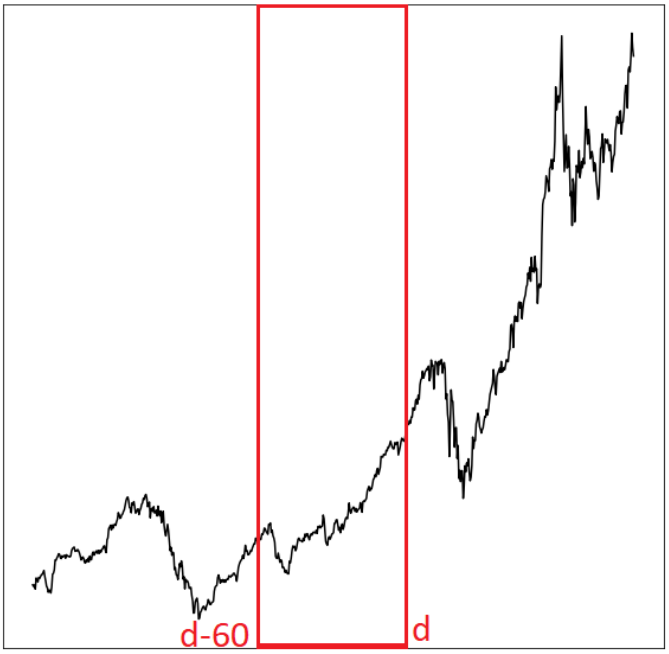


Figure 3.1: Sliding Window.

- RMSE, Root Mean Squared error,

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}. \tag{3.3}$$

Where n is the number of observations, $\hat{y}_i$ is the vector of predictions for observation i and y_i is the expected output for observation i.

3.4. Models

All of the proposed models will be using the LSTM layer discussed in 2.2.2, we will be varying the number of layers and their parameters, as well as performing a *RandomSearch* tuning function from the *keras_tuner* to find the optimal hyperparameters.

One Recurrent Layer

Firstly we propose a model with a total of 3 layers, Firstly one LSTM with 64 neurons and tanh as its activation function and two Dense layers with 20 and one neuron respectively.

We train this model for 200 epochs, using Adam's optimizer with a learning rate of 10^{-3} and obtain the results in 3.2.

In 3.2 we plot the predictions for both train and test to visualize the results.

Metrics	Train	Test
RMSE	$1{,}9 * 10^{-2}$	$3{,}19 * 10^{-2}$
MSE	$3{,}6284 * 10^{-4}$	$1 * 10^{-3}$
MAE	$1{,}44 * 10{-}2$	$2{,}68 * 10^{-2}$

Table 3.2: Metric Results of training.

(a) Train Results between 01/01/2018 to 01/01/2021.

(b) Test Results between 01/03/2021 and 01/01/2022

Figure 3.2: Results for model with one recurrent layer of 64 neurons.

Keras Tuner

Now we will try different hyperparameter configuration to try to improve the results obtained by this model. To do so we make use of the KerasTuner, a hyperparameter optimization framework of the Keras deep learning API. In particular we will be using RandomSearch.

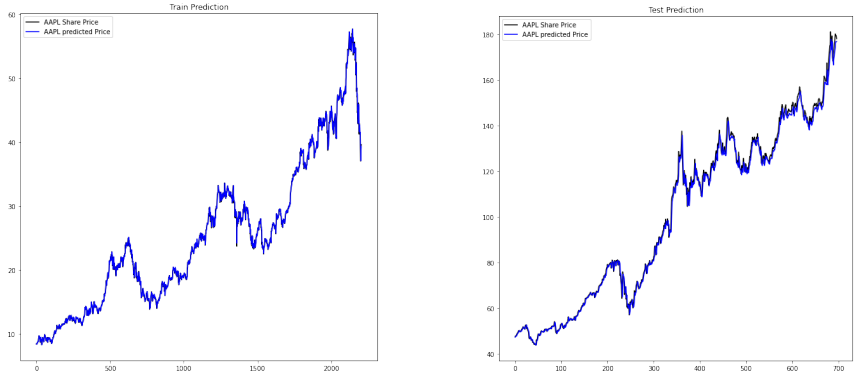
We will be iterating the number of neurons of the LSTM layer between 32, 64, 72, 80 and 96, and between ReLU and tanh activation functions, the first Dense layer will try the same with the same neurons configurations as the LSTM. Finally, the learning rate is left as 10^{-3}.

The best performing model obtained has the following layers and parameters: LSTM layer with 80 neurons and tanh, Dense with 80 neurons also, and the final Dense layer wasn't iterated on, so it still has 1 neuron.

The results can be seen in 3.3 and visualized via graphs in 3.3.

Metrics	Train	Test
RMSE	$1{,}74 * 10^{-2}$	$2{,}27 * 10^{-2}$
MSE	$3{,}0395 * 10^{-4}$	$5{,}1437 * 10^{-4}$
MAE	$1{,}23 * 10{-}2$	$1{,}67 * 10^{-2}$

Table 3.3: Results of model obtained by RandomSearch Keras Tuner.



(a) Train Results between 01/01/2018 to 01/01/2021.

(b) Test Results between 01/03/2021 and 01/01/2022

Figure 3.3: Results of model obtained by RandomSearch Keras Tuner.

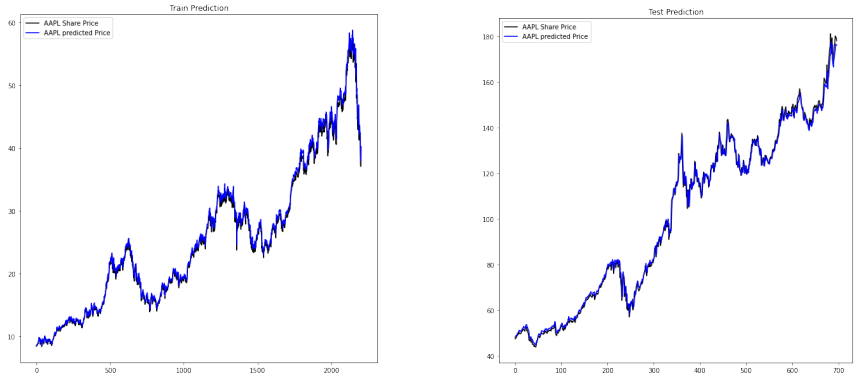
2 Recurrent Layers

As a final test we will add another LSTM layer to see if it helps improve our model, we will build a model with two LSTM layers with 80 neurons each and tanh activation function, then we add a Dense layers, with 40 neurons and another final Dense layer with only one neuron to produce the output.

After training for 200 epochs, we get the results in 3.4 and 3.4. We should note that the training in the case of 2 layers starts to get quite slower than before.

Metrics	Train	Test
RMSE	$2{,}58 * 10^{-2}$	$2{,}4 * 10^{-2}$
MSE	$6{,}6474 * 10^{-4}$	$5{,}7384 * 10^{-4}$
MAE	$2{,}12 * 10{-}2$	$1{,}81 * 10^{-2}$

Table 3.4: Results of recurrent model with 2 LSTM layers of 80 neurons.



(a) Train Results between 01/01/2018 to 01/01/2021.

(b) Test Results between 01/03/2021 and 01/01/2022

Figure 3.4: Results for a recurrent model with 2 LSTM layers.

3.5. Conclusions

From the error tables and graphs we can conclude that the best performing model is the one obtained when using the Keras tuner, the a model with 1 LSTM layer with 80 neurons and two Dense layers with 80 and 1 neurons respectively.

We can see that the LSTM networks apparently have great results as the errors obtained are pretty low, and the graphs superpose well. But we have to remember that we are making only 1 day ahead predictions, that is predicting the price for the next day, given the last 60. If we were trying to make, say, 5 day ahead predictions, the results would be potentially much worse, as the errors would get carried on and amplify over time.

Also our 1 day ahead predictions aren't as accurate as we might percieve, in 3.5 we zoom into the test results of figure 3.3. As we can see, the predicted prices is almost identical to the real prices shifted 1 day backwards. In essence, this means that every day, our model is predicting the price of tomorrow

to be roughly the same as the price today. Which is far from being a satisfactory investment strategy.

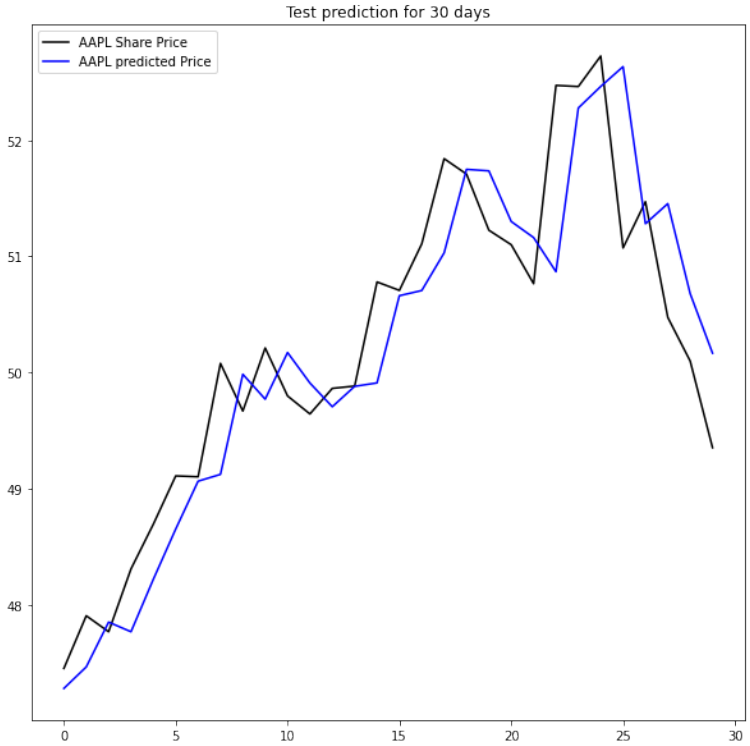


Figure 3.5: Test prediction of recurrent neural network for 30 days.

We will further confirm our suspicion by running a backtesting strategy on the best performing model.

3.5.1. Backtesting

This backtest will be based upon simple buy and sell operations, so we won't be going into options, like calls or puts, nor will we be exploring shorting stocks, swaps, etc.

The strategy is quite simple, for every day, we predict, using our trained model, the price of the day ahead. If this predicted price is higher than the current price of the stock, we buy, if it isn't we don't perform any action.

If we do invest we will hold this investment for a maximum of 7 days. If we let the algorithm hold indefinitely, it would turn into a buy and hold strategy instead of a short term one like we are exploring.

Furthermore we will set both a stop loss order and a take profits order during those 7 days for 5 % of the initial investment. Meaning that during that time if the value of the investment rises 5 % then we would sell with a return of 5 %, while if the investment drops more than 5 % we would also sell immediately, losing 'only' 5 %.

The pseudocode for the backtesting algorithm can be seen in 3.1.

input : model: RandomForestClassifier model, test_data: data to perform backtest on
output: number of trades, gross return, money invested, return on investment

```
predictions ← model.predict(test_data);
acc_ret ← 0.0;
invested ← 0.0;
n_trades ← 0;
for day in test_data do
    if prediction at day is 1 then
        If prediction is that stock will rise we invest ;
        ret += calculate_return(Closing price at day, Closing price next 7 days);
        invested+ = (Closing Price at day);
        n_trades+ = 1;
        if ret > 0,0 then
            n_succ_trades+ = 1 ;
        end
        acc_ret+ = ret;
    end
end
return acc_ret, invested, n_trades, n_succ_trades
```

Algorithm 3.1: Backtesting algorithm to test recurrent model's return.

input : starting_price: price that we paid for the stock, rest_prices: the future prices of this stock, N: number of days we wait until we sell the investment
output: return: amount of money made or lost on this trade.

```
perf ← 0.05;
for price in rest_prices[:N] do
    if price <= (1 − perf) ∗ starting_price then
        If in the following N days the price drops more than perf %, we sell;
        return price − starting_price;
    end
    if price >= (1 + perf) ∗ starting_price then
        If in the following N days the price grows more than perf %, we sell;
        return price − starting_price;
    end
end
If in the following N days the price didn't grow or drop more than perf %, we sell;
return price − starting_price;
```

Algorithm 3.2: Function to calculate return of an investment given the prices it will have on the next N days.

We apply this backtest for different lengths of time and we get the results in 3.5.

Date Period		Invested	Trades	Return	Return (%)	S&P500	alpha
17/12/2021	01/01/2022	0$	0	0$	0 %	3,1497 %	−3,1497 %
18/11/2021	01/01/2022	332,55003$	2	8,39$	2,523 %	1,31 %	1,2127 %
11/08/2021	01/01/2022	628,71$	4	11,64$	1,8514 %	7,28816 %	−5,4367 %
01/06/2021	01/01/2022	1862,6$	13	44,97$	2,4143 %	13,42539 %	−11,011 %
23/10/2020	01/01/2022	8620,07$	67	106,24$	1,2324 %	37,5366 %	−36,304 %

Table 3.5: Backtesting results for recurrent neural network.

4

Prediction of Stock Return as a Classification problem

The next part of our work will focus on building a classification model whose objective is to predict if a stock's yearly return will exceed a certain threshold. We have chosen this yearly timeframe following the strategy of value investing, which argues that an undervalued stock will eventually rise to its intrinsic value, on the long term.

For the extraction and preprocessing of the dataset we will use the following *Python* libraries for data treatment, analysis, creation of machine learning models and neural networks: *pandas*, *numpy*, *sklearn*, *keras*, *tensorflow*.

While for the visualization and plotting of data we use *matplotlib* and *seaborn*.

4.1. Data Extraction

Firstly, we will be creating a dataset consisiting on a series of financial ratios, profit and debt metrics, margins and other metrics. in 4.1 we can see a summary of them. These values aren't updated continuously like prices do, instead they only get recalculated when new earning reports get published, normally, that happens every quarter. This means that in 10 years we would only get 40 data points. If we only considered data for one company it would leave us with a far to small database, so we are forced to expand it somehow.

The solution we opted to take was to, instead on focusing on one company like we did previously in 3, we will be taking a few of them and joining their datasets to create the final one. In particular, we will be taking data from all S&P500 companies. The ratios and financials we mentioned will be taken from the website macrotrends.com between 2009 and 2022, since it is the furthest that this webpage recopilates said data.

Apart from the data of this data, we will also add another two columns to our dataset, corresponding to the stock ticker, and the price of said ticker on that day, which will be taken from Yahoo Finance as in 3.

The resulting dataset has 23843 rows and 32 columns.

Ratios	Metrics	Margins	12 Month Trailing
Price to Earnings	Gross profit	Profit margin	Earnings per share diluted
Price to Sales	Operating income	Net profit margin	Operating Income
Price to Book	EBITDA	Operating margin	ebitda
Price to FCF	net income	EBITDA margin	Gross profit
Current Ratio	Earnings per Share diluted		
Quick Ratio	Shares Outstanding		
Debt to Equity	Long term debt		
Return on Assets	Total liabilities		
Return on Investment	Total shareholder equity		
Return on Investment	Total assets		
Return on Equity	Earnings per Share basic		
Return on Tangible Equity			

Table 4.1: Table of ratios and financials used for our classification model.

4.2. Data Preparation

Unlike in 3.2 this time we do have entries with NaN data, we will replace all them with 0 values, so we don't lose entries of valuable information.

Now in order to define a target variable, we set an outperformance threshold level, in our case 12.5 %, this value is chosen since it will later on balance our target classes so they are roughly equivalent in size. Therefore, that all stocks that after 1 year present a change in price of more than 12.5 % will labeled as positive (class 1), the rest will be labeled as negative (class 0).

Similarly to what happens in 3.2, we won't scale or normalize or scale our data. This time some of our financial ratios, margins, measures of profits and liabilities, might seem like they move within a range, others clearly don't, like *gross-profit, total-assets, total-liabilities, EBITDA*, etc. Some of the features that seem to have a common scale between different stocks, and which don't change in scale overtime, don't, however, move in a set scale. A great example of it is the P\E ratio, the average P\E ratio of the S&P500 has been between 13 and 15, but there have been many instances where companies exhibit, far larger values than this, for example on the first quarter of 2021, Tesla $TSLA had a P\E ratio of roughly 360.

The correlation matrix of the variables chosen as features is 4.1

In any binary clasification problem it is really important to have balanced classes, or to implement a method like class weights in case they aren't. Having chosen the threshold as we did before, we have ensured that they are in fact balanced, the countplot for it is 4.2.

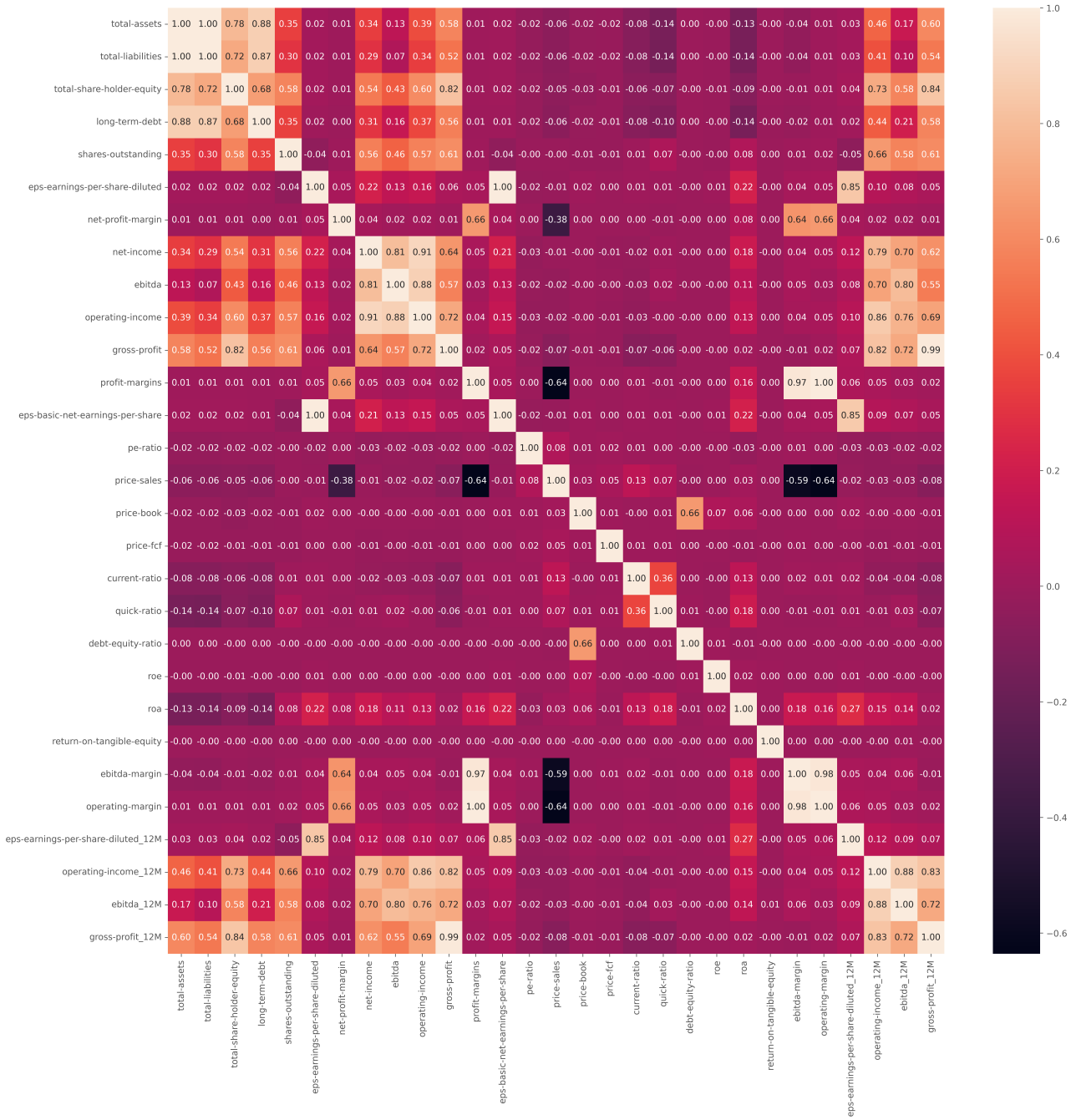


Figure 4.1: Correlation matrix for the variables of classification model.

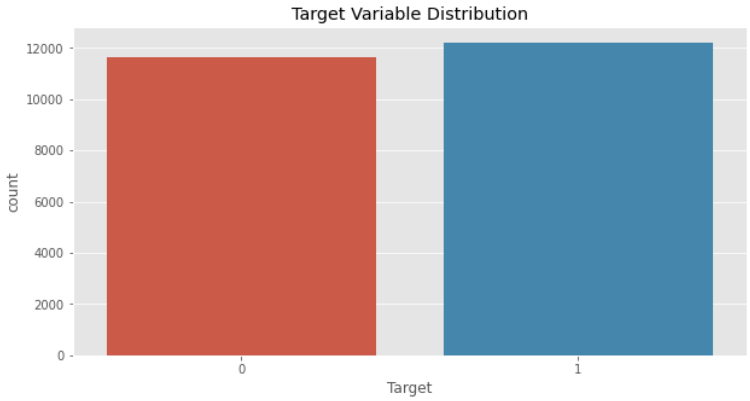


Figure 4.2: Target variable distribution for yearly stock return classification.

4.3. Metrics

To evaluate our model we will be using both accuracy and precision.

Accuracy

Accuracy is probably the most common metric for classification problems, it is defined in 4.1 as the number of times the model correctly classifies an input over the total number of tries.

$$Accuracy = \frac{\text{correct predictions}}{\text{Total number of predictions}}. \tag{4.1}$$

The main drawback with accuracy is the fact that in problems where the target classes are heavily imbalanced, we might get great accuracy results, distorted by that great imbalance. Due to this limitation we must also look into other metrics to validate the result of the accuracy.

Precision

The precision metric is used to measure the effect that false positives have in our positive predictions, that is a low precision means that when our model predicts a positive result, it makes a lot of mistakes. While a high precision, of close to one, means that if our model predicts an input to be positive, there is a high probability that this prediction is in fact true, however, it might incorrectly classify as negative some inputs that are positive.

In our case, a stock rising in price over a 12.5 % threshold in a year will be the condition for a stock to be classified as positive. Because the investment backtests we will perform will only buy stocks when we predict said class, minimizing the precision is equivalent to minimizing the number of trades with potential of obtaining a negative return.

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}. \tag{4.2}$$

Other metrics used in problems of this type are *recall, F1 score*, etc. But in our case they aren't as relevant as these two, and ultimately, the most important metric we will be the backtesting results.

Confusion Matrix

The confusion matrix for a binary classification problem is a bidimensional matrix with two columns, the columns correspond to the predicted classes while the rows are the true classes.

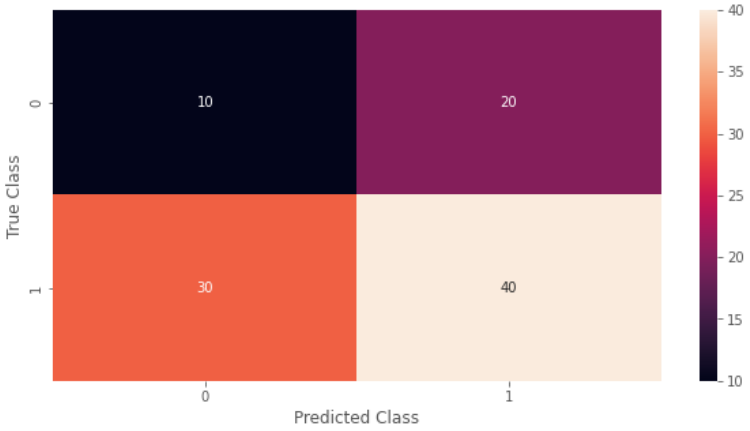


Figure 4.3: Example of a confusion matrix.

In the confusion matrix the values in the off-diagonal elements are: False Positives for the top right corner, and False Negatives for the bottom left corner.

4.4. Models

We will be trying some of the most common classification models, like *MLP, SVM* and *RandomForestClassifier* from the *sklearn* package.

4.4.1. Multi Layer Perceptron

It is a simple and widely used feedforward neural network, with the following diagram, the number of hidden layers varies, but the algorithm to perform the weight adjustments is gradient descent. We will be using *sklearn.neural_network.MLPClassifier*.

Where the exit of each neuron is a weighted sum of the output values of the neurons in the previous layer, following the next equation:

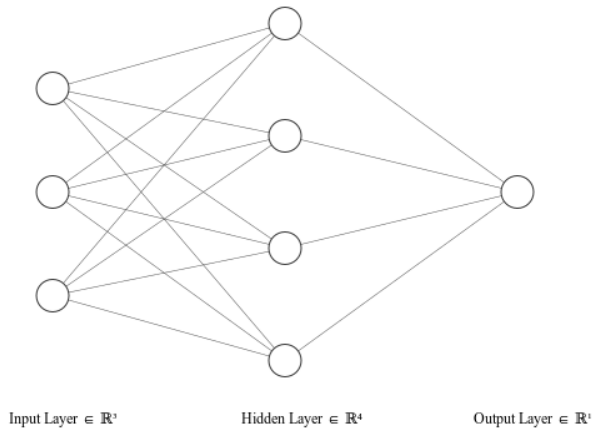


Figure 4.4: Multi layer perceptron diagram.

$$X^k = \sigma\left(b^{k-1} + W^{k-1}X^{k-1}\right). \tag{4.3}$$

where:

- X^k, is the vector of outputs of neurons in layer k.
- b^{k-1}, the bias for layer k
- W^{k-1}, the weight matrix.
- σ, the activation function.

4.4.2. Support Vector Machine

A support vector machine is a machine learning model that tries to find the hyperplane in n dimensions that better distinctly classifies a set of data points. For example, in a two dimensional space it would try to form straight lines that separate the data in classes. The function in *sklearn* that creates this model is *svm.SVC*.

4.4.3. Random Forest

This type of classifier consists of many decision trees, in which every tree has random features in it. Furthermore each tree is trained on a different set of data. These approaches are taken in order to create uncorrelated trees, then the predictions of each tree are put together and the one that has occurred more is the result, effectively being a voting mechanism between all the decision trees. We will be using *sklearn's* version of it, *sklearn.ensemble.RandomForestClassifier*.

The main advantage of this model is that the bagging algorithm that chooses the different sets of

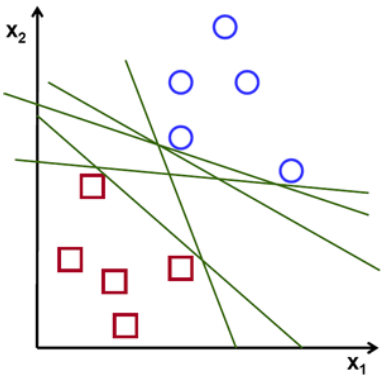


Figure 4.5: Support Vector Machine diagram in 2 dimension space.

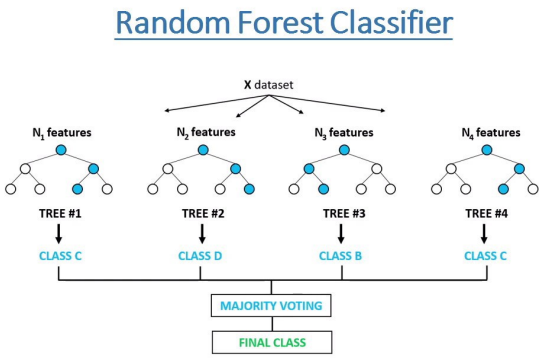


Figure 4.6: Random Forest diagram.

data to use in each tree ensures that overfitting doesn't happen.

4.5. Results

We will evaluate the presented machine learning algorithms with the metrics discussed in . But before getting into the results we must assess what results will be satisfactory. Knowing that investing is an extremely difficult topic to make predictions on, we will want our models to outperform at least the model that blindly predicts the majority class in the train dataset. That is, if our model wasn't able to learn any information from the features provided it would tend to predict the target class that it observes the most, so this should be the real baseline score that we want to always be over. In our case this value is $39{,}90941117262204$. From here on, we will refer to this model as the *dummy model*.

For all the experiments we will perform the train test split will be always the same, performed by sklearn's *train_test_split* method, with a test size of 25 % and correspondingly a train size of 75 % of the whole dataset, it is extremely important that the train data and test data do not have entries for stocks on the same time periods, as this would be tainting our test results.

4.5.1. Multi Layer Perceptron

In the case of the multi layer perceptron we try different parameter configurations.

We will be iterating the number of neurons in the hidden layers and the number of hidden layers used, as well as the tipe of activation function used. The other parameters are set, *solver*: adam, *learning_rate*: constant and *alpha*: $0{,}0001$. The parameter grid is:

- *hidden_layer_sizes*: (10,), (50,), (100,), (10,50), (50,50), (100,50), (50,50,50), (50,100,50).
- *activation*: $\tanh$ or ReLU.

After running, the best performing model has the following hyperparameters: *activation*: tanh and *hidden_layer_sizes*: (50, 100).

The results and confusion matrix, 4.2 and 4.7 respectively, are positive, presenting a rough $17{,}246\,\%$ outperformace of the *dummy* model, and a precision score that backs the accuracy results,

	Train	Test
Metrics		
Accuracy	$82{,}47399619729336\,\%$	$54{,}97399765140077\,\%$
Precision	$84{,}8952361096633\,\%$	$43{,}65904365904366\,\%$

Table 4.2: Results of Multi Layer Perceptron (MLP) for yearly stock return binary classification.

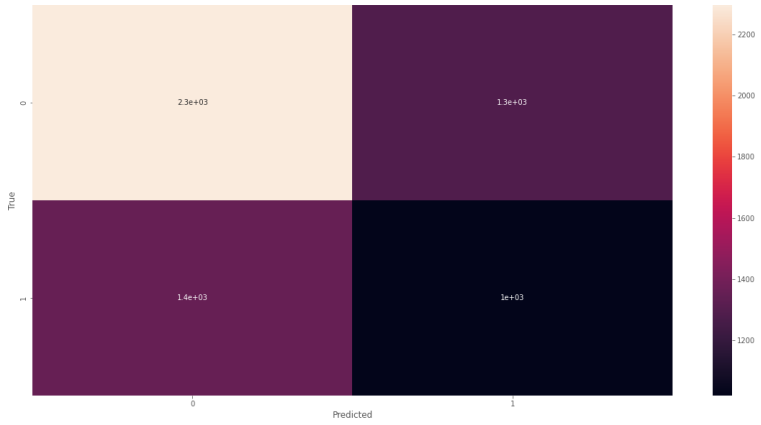


Figure 4.7: Confusion matrix of Multi Layer Perceptron (MLP) for yearly stock return binary classification.

4.5.2. Support Vector Machine

From the support vector approach the results in 4.3 and 4.9 are not satisfactory, as we barely overcome the results of the dummy model.

Metrics	Train	Test
Accuracy	$55{,}704059948551624\,\%$	$42{,}84516020801879\,\%$
Precision	$55{,}783220174587775\,\%$	$40{,}25777103866566\,\%$

Table 4.3: Results of Support Vector Machine for stock yearly return binary classification.

And the corresponding confusion matrix is 4.9.

The outperformance versus the dummy model is of a mere $3\,\%$. The precision is quite high so that means that the predictions are skewed to one side, and with the accuracy being so low, we can't consider the support vector machine as an effective model given the current data.

4.5.3. Random Forest

For the RandomForestClassifier after trying different parameter configurations we settled for a simple one that kept most of the default parameters intact, only changing *n_estimators* to be 300, the *randomstate* is kept constant to make the results reproducible in successive iterations.

The results obtained can be seen in 4.4.

And the corresponding confusion matrix is 4.9.

The results exihibited by the RandomForestClassifier are quite superior to the previous ones, spe-

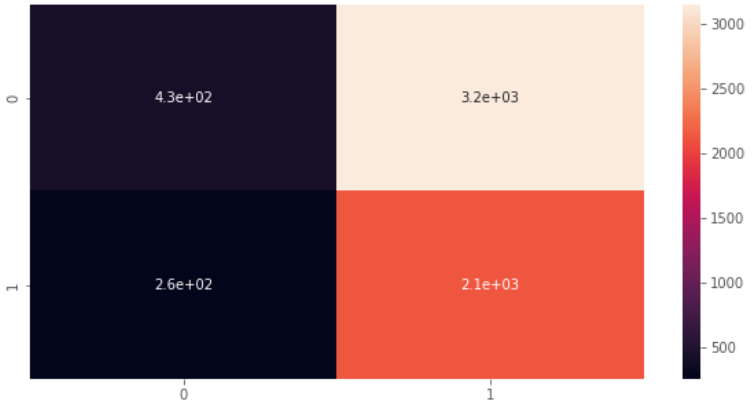


Figure 4.8: Confusion matrix of Support Vector Machine for stock yearly return binary classification

Metrics	Train	Test
Accuracy	$98{,}26641315289118\,\%$	$63{,}89867471900688\,\%$
Precision	$97{,}42899850523169\,\%$	$56{,}002115282919085\,\%$

Table 4.4: Results of RandomForestClassifier for stock yearly return binary classification.

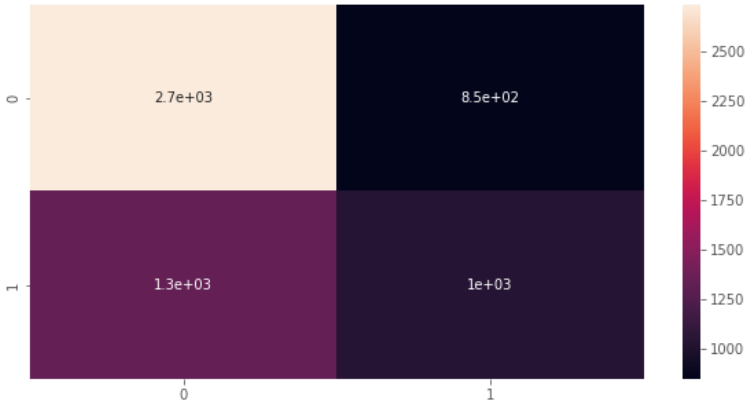


Figure 4.9: Confusion matrix of RandomForestClassifier for stock yearly return binary classification

cifically it would give us an outperformance against the dummy model of $23{,}3853380305\,\%$. Although the accuracy value might be seen as low, since only $43{,}75\,\%$ of predictions of growth over $12{,}5\,\%$ are correct, we must remember that even predictions where we are wrong, a stock's yearly value can still rise, just for a value less than the threshold, giving us a positive return, we will see that this is the case in the section 4.6.1, where we will backtest our strategy.

4.6. Conclusions

From all the tested models we obtain varying degrees of satisfactory results, although clearly the better one is the one exihibited by the RandomForestClassifier. Having $63{,}3\,\%$ accuracy results and even $43{,}75\,\%$ precision, efectively meaning our predictions of a stock's yearly return being over $12{,}5\,\%$ are right $43 - ,75\,\%$ of the time. This does not, however, mean that any strategy built around it will guarantee extraordinary results.

In order to truly understand the viability of the machine learning model we have to, as done in 2.3.2, perform a series of backtests, with varying date periods and stocks of different companies. These backtested results will then be compared to the S&P500 for the same periods of time and further conclusions will be drawn after those.

4.6.1. Backtesting

The backtest in this section will follow the same rules as the one in 2.3.2. Our model will predict whether a stocks yearly return will be over $12{,}5\,\%$, if we predict it will do so, we will invest in it, and sell that stock in a year's time. If our model predicts a stock won't go over that threshold we will not invest in it. The pseudocode for this algorithm can be found in 4.1.

We will be performing the backtest on the testing data, since it is the largest period of time that we can obtain, that our model has not seen before. This period corresponds to the dates between 30/06/2021 AND 31/07/2021. The results of the backtest experiments on the different models presented are visualized in 4.7.

With this results we can consider the model based on classification to be widely superior to the one utilized a recurrent neural network. Obtaining even roughly a $24\,\%$ return over a 3 year period.

The role of Dividends

A dividend is a payment made to the shareholders of a company, as a distribution of its earnings. Not every company pays dividends to it's shareholders, as it is an internal business decision. Their role is relevant since they don't reflect in a company's stock price. For example, at the time of writing this,

input : model: RandomForestClassifier model, test_data: data to perform backtest on
output: number of trades, gross return, money invested, return on investment

```
predictions ← model.predict(test_data);
acc_ret ← 0.0;
invested ← 0.0;
n_trades ← 0;
foreach day in test_data do if prediction at day is 1 then
    If prediction is that stock will rise we invest ;
    ret += (Closing Price at day next year) − (Closing Price at day);
    invested+ = (Closing Price at day);
    n_trades+ = 1;
    if ret > 0,0 then
        n_succ_trades+ = 1 ;
    end
    acc_ret+ = ret;
end
;
return acc_ret, invested, n_trades, n_succ_trades
```

Algorithm 4.1: Backtesting algorithm to test clasification model's return.

Date Period		Invested	Trades	Return	Return (%)	S&P500	alpha
30/06/2021	31/07/2022	380310,05$	2087	−8635,83$	−2,2707 %	−3,8908 %	1,6201 %
31/01/2021	31/07/2022	469687,89$	2612	−1401,86$	−0,2985 %	9,445 %	−9,74 %
30/06/2020	31/07/2022	621523,5964$	3669	48228,95$	7,76 %	33,22 %	−25,463 %
31/07/2019	31/07/2022	818744,01$	5276	112069,81$	13,69 %	38,58 %	−24,89 %

Table 4.5: Backtesting results for Support Vector Machine.

Date Period		Invested	Trades	Return	Return (%)	S&P500	alpha
30/06/2021	31/07/2022	92544,84$	555	−1828,68$	−1,98 %	−3,8908 %	1,915 %
31/01/2021	31/07/2022	116549,43$	714	887,48$	0,76 %	9,445 %	−8,70 %
30/06/2020	31/07/2022	169030,10$	1184	21172,78$	12,53 %	33,22 %	−20,70 %
31/07/2019	31/07/2022	243087,64$	1888	57755,91$	23,76 %	38,58 %	−14,82 %

Table 4.6: Backtesting results for RandomForestClassifier.

Date Period		Invested	Trades	Return	Return (%)	S&P500	alpha
30/06/2021	31/07/2022	180036,274693$	926	−3717,2875$	−1,98 %	−2,065 %	1,83 %
31/01/2021	31/07/2022	217125,689866$	1148	252,972$	0,1165 %	9,45 %	−9,33 %
30/06/2020	31/07/2022	299286,826675$	1648	27140,23$	9,07 %	33,22 %	−24,15 %
31/07/2019	31/07/2022	393539,08$	2405	59347,68$	15,08 %	38,58 %	−23,50 %

Table 4.7: Backtesting results for Multi Layer Perceptron.

Apple pays dividens, so, an investment made between 2019 and 2020, would yield a 82.31 % return solely on the stock price rising, but the actual return would be higher due to these dividends.

Notice that in our backtests we haven't taken these dividends into account, so the return of our strategies would be slightly higher, also, many index funds of the S&P500 vary in the way they handle dividends payments, some reinvesting them directly into more shares while others hand them out. This makes the figures we found to be slightly off due to no factoring these dividends, although this deviation is not large.

Comparison to other investments

We compared our investments to the performance of the S&P500, which is, a quite demanding benchmark to meet, even for active managed mutual funds. This can be observed in the SPIVA reports [13] [14], which measures actively managed funds against their index benchmarks, finding that, for example in 2021, where the S&P500 grew $28{,}7\,\%$, although funds obtained positive returns, a staggering $79{,}6\,\%$ of them obtained worse results than the S&P Composite 1500 index fund, which combines the S&P500, S&P MidCap 400 and S&P SmallCap 600, which during the year 2021 it obtained a $28{,}45\,\%$ return.

These findings put the results obtained by our models in another light, since it highlights that even institutions as prestigious and which employ hundreds of people like mutual funds fail to outperform their respective benchmark. It is also highly likely that our model would outperform some of those actively managed funds.

If we were to compare our results to other less volatile investments, such as US Treasury Bonds with a yearly maturity, which on average since their inception, return a 5-$6\,\%$. Therefore, we can see that our backtesting results, when tested in a long term period, beat the treasury bond returns. The comparison is even more favourable if the pit our results against the average high-yield savings account's interest rates, which can be as high as $1{,}5\,\%$.

These results are great, since normally, government bonds and high-yield savings accounts are considered one of the safest investments there are, and we would expect the model we create, which

inherently carries more risk than them, to achieve higher results.

Limitations of our Backtesting

We have concluded that our results are quite satisfactory, however, we have to take into account a few limitations that the tests that we run have had.

Firstly, we are unable to test our models in recessionary periods, where the economy is in decline, like for example, 2008. This would give our conclusions more robustness and would also help us tweak our models further so the results were better accross differnet scenarios.

Another aspect of investment that we are completely disregarding are commissions, although there are a myriad of online brokers that charge little to no comission, they are an aspect that exists, and will always affect how an investment is viewed, even investing in passive index funds carries a small comission that the broker earns.

Lastly, we are not taking into account taxation on our benefits, but this is an aspect that depends on the individual type of investment and the country from where the investment is made.

5

Conclusions and Future Improvements

Throughout this work we have been able to explore two totally different approaches for the same base problem.

In the recurrent case we started out by fetching from Yahoo Finance a very simple dataset, only contaning Open, High, Low and Closing prices from a stock. Applying the sliding window method to it we transformed into a more complex dataset in which each day contained data from 60 days before, giving us a dataset that could be used to train a recurrent neural model, in particular a Long Short Term Memory (LSTM) neural network.

Furthermore we investigated how different parameter configurations affected the results, employing *keras_tuner's RandomSearch* tuning algorithm to improve on our experimental results. We obtained results which at first glanced seemed quite good, but taking a deeper look into them, we found some fundamental flaws in them. We argued that they wouldn't form a good investing strategy, were they to be used. Finally, we confirmed this suspicion by backtesting the best performing model in a series of timeframes, obtaining results which were far from satisfactory.

This led us to try a completely different approach. Firstly we changed the type of problem we were dealing with, by transforming it into a binary classification one. Also, strechting out the timeframe over which we invested, to a year, we could involve financial metrics and ratios that reflected information of the companies we were investing in. We also realized that for this part we couldn't as before, restrict ourselves to only one stock, as the dataset wouldn't have the appropiate size to train a machine learning model. Therefore, we expanded to use all the stocks in the S&P500 index.

In order to avoid having imbalanced clases we chose the yearly return threshold accordingly, helping us down the road when training our models. In that sense, we tried different popular models, like the multilayer perceptron, support vector machine and random forest classifier. We tested these three models over different times and with different lengths, and the returns obtained are quite adequate. We Conclude that the random forest model was the best performing. Upon comparison with other finantial products, we find that they are higher than safer options like high yield savings accounts and US treasury bonds, but in the majority of tested scenarios, worse than investing in an S&P500 index fund. Research also suggests us that although we underperformed this benchmark index, a large majority of

actively managed funds in the US suffer the same fate as us, giving our results a good outlook. This proves that the features we used have a certain predictive power over the future price of a stock.

5.1. Future Work

The amount of future improvements that can be done on our work is quite large, and it can be separated in areas.

5.1.1. Dataset

Perhaps the most critical aspect of this and many machine learning works is quality data. On that regard, we could further expand our classification dataset to include more features, such as the percentage change of the used ratios and metrics from the previous year, this would give us a slight tendency on how the company changed in stance from the last year.

Furthermore, the addition of macroeconomic variables that can affect a stocks price or the performance of a company, can be helpful in enhancing our results, especially in big recessionary, or high inflation situations. For example, we could add unemployment rates, interest rates, GDP growth, or the credit default rates experienced in the country of each company.

Another dimension that the data can be enlarged upon, is to add stocks from outside the S&P500 index. Other indexes from the US can be explored, such as the NASDAQ, Dow Jones, S&P MidCap 400, etc. Or, even, to add stocks from outside the US altogether, in this case, one should take into account the rules and regulations that apply on the prospective markets to be added to our model, since some can have very hard entry barries, like the Chinese market. For simplicity's sake we could consider European based ones, such as the Euro Stoxx 500, FTSE 100, DAX, SMI, etc.

The third dimension that we can consider is the timeframe that our data is taken from, currently our dataset contains data from 2009 to 2022, which is a period of relative steady growth in the S&P500 index, so our model hasn't had the opportunity of being trained on recessionary periods. Being able to expose our model to those periods will make it more robust.

5.1.2. Models

We could also maximize the efficiency of our machine learning models by doing more thorough parameter tuning, although this approach is unlikely to improve the current results significantly.

Finally, another option is to explore more complex models, such as hybrid ones, that intregrate sentiment analysisi of financial news from different sources like articles from Yahoo Finance, Morningstar

or Twitter.

5.1.3. Backtesting

The backtesting section of our work that could be expanded. By performing more extensive backtesting we could determine the actual risk vs. return performance of our models. We should aim at diversifying the moments and which companies are included in these backtests. Firstly, periods with economic and political instability should be looked into, as well as employing our model on stocks of companies that it hasn't ever been trained on. This would serve to validate the limited results we obtained, and potentially conclude that they extrapolate to other situtations, or, find out what situations they perform the worst in, to decide if the addition of new features is warranted, and in that case, what type of features should be added.

Bibliography

[1] A. Chokkavelu, "How peter lynch destroyed the market," 10 2016.

[2] J. J. Shuohang Wang, 2015. Learning Natural Language Inference with LSTM.

[3] Q. Chen, X. Zhu, Z. Ling, S. Wei, and H. Jiang, "Enhancing and combining sequential and tree LSTM for natural language inference," *CoRR*, vol. abs/1609.06038, 2016.

[4] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," *CoRR*, vol. abs/1506.04214, 2015.

[5] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[7] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München," 1991.

[8] J. Lewellen, "Predicting returns with financial ratios," *Journal of Financial Economics*, vol. 74, no. 2, pp. 209–235, 2004.

[9] J. D. Piotroski, "Value investing: The use of historical financial statement information to separate winners from losers," *Journal of Accounting Research*, vol. 38, pp. 1–41, 2000.

[10] X. Zhong and D. Enke, "Predicting the daily return direction of the stock market using hybrid machine learning algorithms," *Financial Innovation*, vol. 5, 12 2019.

[11] T. Pessoa, R. Medeiros, T. Nepomuceno, G.-B. Bian, V. Albuquerque, and P. P. Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, vol. PP, pp. 1–1, 10 2018.

[12] "Yahoo finance." `https://finance.yahoo.com/quote/AAPL/history?p=AAPL`. Accessed: 2021-2022.

[13] B. Liu and S. Gayrav, "Spiva u.s. year-end 2021," 10 2022.

[14] T. Edwards, G. Anu, C. Lazzara, J. Nelesen, and D. Di Gioia, "Spiva u.s. scorecard mid-year-2022," 9 2022.

APPENDICES

Index


UAM
UNIVERSIDAD AUTONOMA
DE MADRID