UNIVERSITÀ DEGLI STUDI DI TRENTO

DIPARTIMENTO DI INGEGNERIA e SCIENZE DELL'INFORMAZIONE
Master in Artificial Intelligence Systems

AI for Finance
Professor Flavio Bazzana

Academic Year 2021/2022
Trento, Italy

Miotto Sara, 232086

Contents

Chapter 1

Goals

1.1 Goals

What I want to do in this project is to learn **how to create** an efficient portfolio of the Italian Stock Exchange and to learn **how to predict** the trends of the portfolio in view of exhibiting it to an hypothetical investor.
In order to create my portfolio I have tried to consider companies from **different sectors** so as to differentiate it.
In fact stocks within a single industry tend to have higher correlation than stocks which belong to various industries, and so, combining stocks, would reduce the volatility of the portfolio itself.
However, before this diversification, the index I based my research on, was **EBITDA**.
EBITDA, or earnings before interest, taxes, depreciation, and amortization, is a measure of a company's overall financial performance and so it is also a measure of profitability.
It is calculated in a straightforward manner, with information that is easily found on a company's income statement and balance sheet.
There are two formulas used for this scope, one that employs operating income and the other, instead, net income:

- $EBITDA = NetIncome + Taxes + InterestExpense + Depreciation\&Amortization$
- $EBITDA = OperatingIncome + Depreciation\&Amortization$

It usually has the aim of analyzing and comparing profitability among companies and industries.
So, in summary, I have chosen the 10 companies with the best value of EBITDA throughout 2021. As I took into account a timeframe of 10 years, I had to discard companies which had a time span not deep enough.
Here is the list:

- Telecom Italia SpA (TIT)
- TERNA (TRN)
- Hera (HER)
- DiaSorin (DIA)
- Piaggio (PIA)
- Brembo (BRE)
- IGD SIIQ (IGD)
- Amplifon (AMP)
- Zignago Vetro (ZV)
- Gruppo FNM (FNM)

The belong sectors are specifically: telecommunication, electricity, water supply, production of motorcycles, pharmaceutical products, accessories for motorcars and engines, financial lease, medical and orthopedic products, glass manufacturing, land transports.
As we can see it is quite differentiate as companies come from **various scenarios**.

Chapter 2

Data

2.1 Data description (data cleaning and descriptive statistics)

To collect the data related to the chosen companies I have used **AIDA**.
AIDA is a database which contains economic, financial and qualitative information of Italian stocks (almost 2,127,764).
It is very practical to be used as it allows us to select groups of companies according to a wide range of quantitative and qualitative indicators.
First of all, it was necessary to **clean the data**. Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset.
In statistics, missing data occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and could have a significant effect.
In case of a missing daily value, this is set to the previous valid value.
On the other hand outliers are data point that differs significantly from other observations. Of course they might cause serious problems in statistical analyses.

2.2 Description Statistics

So I have calculated for the **returns** of each individual **stock**:

- the expected values
- the matrix of variances
- the matrix of covariances

Here is the matrix of covariances obtained:

	TIT	TRN	HER	DIA	PIA	BRE	IGD	AMP	ZV	FNM
TIT	0.000564	0.000133	0.000118	0.000057	0.000142	0.000146	0.000178	0.000095	0.000080	0.000135
TRN	0.000133	0.000177	0.000111	0.000064	0.000079	0.000083	0.000087	0.000082	0.000048	0.000076
HER	0.000118	0.000111	0.000228	0.000049	0.000077	0.000077	0.000101	0.000084	0.000049	0.000077
DIA	0.000057	0.000064	0.000049	0.000344	0.000063	0.000056	0.000050	0.000098	0.000034	0.000036
PIA	0.000142	0.000079	0.000077	0.000063	0.000448	0.000153	0.000138	0.000082	0.000084	0.000109
BRE	0.000146	0.000083	0.000077	0.000056	0.000153	0.000383	0.000131	0.000089	0.000073	0.000112
IGD	0.000178	0.000087	0.000101	0.000050	0.000138	0.000131	0.000459	0.000085	0.000078	0.000127
AMP	0.000095	0.000082	0.000084	0.000098	0.000082	0.000089	0.000085	0.000343	0.000058	0.000090
ZV	0.000080	0.000048	0.000049	0.000034	0.000084	0.000073	0.000078	0.000058	0.000231	0.000065
FNM	0.000135	0.000076	0.000077	0.000036	0.000109	0.000112	0.000127	0.000090	0.000065	0.000569

Figure 2.1: Covariances Matrix

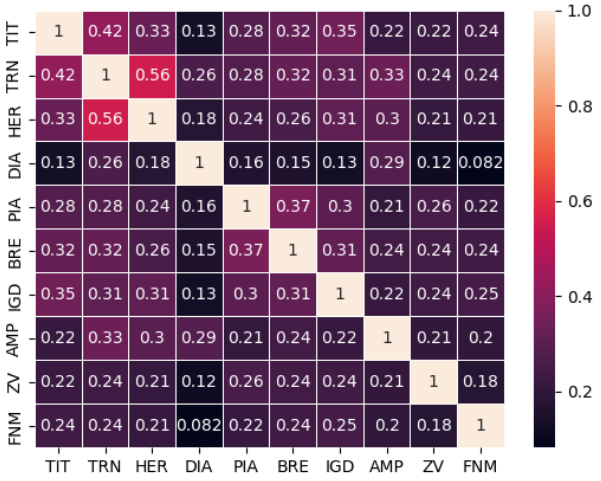


Figure 2.2: Correlation Matrix

We can figure out that stocks are weakly correlated between each other.
This fact is important to happen in order to avoid that a single stock would be able to affect the whole portfolio.
Once this is verified it is possible to forecast with sufficient approximation the future values of the stock.

Chapter 3

Forecasting model

3.1 Description of the forecasting model

As already anticipated, what I want to **forecast** is the trend of my portfolio both with an **econometric model** (ARIMA) and with a **recurrent neural network**(LSTM).
In the dedicated paragraphs they will be both accurately described.
Before I had to **create my weighted portfolio** and to prepare the data for my forecasting models.
In order to create a reliable portfolio, I have followed the so-called Model Portfolio Theory by **Markowitz**.
Markowitz's purpose was the one of eliminating idiosyncratic risk, or the risk inherent in each investment because of the investment's own unique characteristics.
One of the key components of this theory is the concept of **diversification**. In fact most investments are either high risk and high return or low risk and low return. Investors should try to find a balanced mix of the to two based on their own **tolerance towards risk**.
As a consequence, modern portfolio theory can be used in order to build diversified portfolios which aim at maximizing their returns without unacceptable levels of risks.
Risks and returns of an investment should not be taken into account on their own but they have to be evaluated by how much they affect the **overall risk** and return of the portfolio.
That is to say that the performance of a single investment is less important than how this investment affects the **whole** portfolio.
As a consequence with a well-balanced and calculated portfolio, if some of the assets fall due to market conditions, others should rise an equal amount in compensation.
Everything is based on **statistical measures**, such as variance and correlation.
As the basic assumption of this theory is the fact that investors may prefer a low risk, this model suggests to invest in multiple asset classes.
Regarding the expected return of the portfolio, it is calculated as a weighted sum of the returns of the individual assets.
The portfolio's risk is a function of the variances of each asset and the correlations of each pair of assets.

3.2 Implementation

Now let's see specifically how to turn theory into practice.
First of all I started by defining some functions to simulate random weights to each stock in the portfolio.
After I have calculated the portfolio's overall annualised returns and annualised volatility. There are proper functions for this in python libraries as we can see from the code.
I want to make a remark: in order to compute annualised calculations I took into account 252 as the number of trading days in one year.
In order to compute the standard deviation of the portfolio I have used the following formula:

$$\sigma_{portfolio} = \sqrt{w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + 2w_1 * w_2 * Cov_{1,2}}$$

Using matrix calculation, we get the part inside the square root in the above formula.
Risk-adjusted return refines an investment's return by measuring how much risk is involved in producing that return, which is generally expressed as a number or rating.

We express the risk-adjusted return using the **Sharpe ratio**.
This ratio describes how much excess return you are receiving for the extra volatility that you endure for holding a riskier asset. The Sharpe ratio can be expressed as:
$\frac{r_p - r_f}{\sigma_p}$, where r_p is the expected portfolio return, r_f is the risk free rate and σ_p is the portfolio standard deviation.
At this point I wanted to find the needed argument values for our functions. I easily took **daily returns**. The mean daily returns, the covariance matrix of returns are needed to calculate portfolio returns and volatility.
Finally, the risk-free rate has been taken from Italian **BTP** 10Y in 2021, which is commonly assumed as a reference point to value the risk within the Italian market.
The idea behind this is that the historical price data is from 2012-2021, and if I assume that I implement this analysis at the start of 2022, the most updated BTP rate is from the start of 2022.
The return of the value of the BTP is assumed to be equal to the 1%.
After that I have generated random portfolio and got the results and weights for the corresponding result (25,000 random portfolios).
Then by locating the one with the highest Sharpe ratio portfolio, I have observed the maximum Sharpe ratio portfolio and I have done similar steps for minimum volatility portfolio.
If we are willing to take higher risk for higher return, one that gives us the best risk-adjusted return is the one with maximum Sharpe ratio.
I also had to compute the so-called **efficient frontier**: points along this line give you the lowest risk for a given target return.
So I have found the optimal portfolio by simulating many possible random choices and pick the best ones, either minimum risk or maximum risk-adjusted return.
The **sum** of all the weights should be equal to 1 as one cannot allocate more than 100% of the budget in total.
Bounds is giving another limit to assign random weights, by saying any weight should be inclusively between 0 and 1 as one cannot give minus budget allocation to a stock or more than 100% allocation to a stock.
We can also define an optimising function for calculating minimum volatility portfolio. This time we really do minimise objective function. We want to minimise volatility by trying different weights.
Here is reported the allocation of my portfolio that I have obtained by running my code. Note that I have found two series of weights, one corresponding to the **minimum volatility and minimum return**, the other to the **maximum of both volatility and return**.
As the values of volatility were much close between the two series, I have made an **"aggressive" choice**, by selecting the one with highest volatility but also highest return.
Maximum Sharpe Ratio Portfolio Allocation:

- Annualised Return: 0.23
- Annualised Volatility: 0.17

Minimum Volatility Portfolio Allocation:

- Annualised Return: 0.18
- Annualised Volatility: 0.16

	TIT	TRN	HER	DIA	PIA	BRE	IGD	AMP	ZV	FNM
allocation	0.62	2.74	5.18	18.84	0.6	13.15	0.57	19.53	20.48	18.29

I suppose the value of my portfolio to be equal to 1000000 at the beginning (01-01-2012) and I start investing into the stocks following the weights previously calculated. In this way I extract the initial number of actions as well as the historical series of the values of my portfolio.

Chapter 4

Econometric analysis

4.1 Econometric analysis

As econometric model I have decided to use **ARIMA** model as it is widely used for the purpose of forecasting.
ARIMA, acronym for Auto Regressive Integrated Moving Average, is actually a class of models that explains a given time series based on its own past values, and so its own **lags and the lagged forecast errors**. In this way the equation can be used to forecast future values.
An ARIMA model is characterized by 3 terms: p, d, q, where p is the **order of the AR** term, q is the **order of the MA** term and d is the **number of differences** required to make the time series stationary.
So the first step to build an ARIMA model is to make the time series **stationary**, as the term Auto-Regressive in ARIMA means it is a linear regression model that uses its own lags as predictors. Linear regression models work best when the predictors are not correlated and are independent of each other.
As I have already said, the most common approach in order to obtain a stationary time series is to difference it, and so subtracting the previous value from the current value. Sometimes more than one differences may be needed.
The value of d, therefore, is the minimum number of differences needed to make the series stationary.
Let's go deeper in the mathematical formulation of ARIMA. It is composed of two parts: AR and MA.
A pure Auto Regressive (AR only) model is one where **Yt depends only on its own lags**:
$Y_t = \alpha + \epsilon_t + \phi_1\epsilon_{t-1} + \phi_2\epsilon_{t-2} + ... + +\phi_q\epsilon_{t-q}$
Likewise a pure Moving Average (MA only) model is one where **Yt depends only on the lagged forecast errors**:
$Y_t = \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + ... + \beta_0 Y_0 + \epsilon_t$
$Y_{t-1} = \beta_1 Y_{t-2} + \beta_2 Y_{t-3} + ... + \beta_0 Y_0 + \epsilon_{t-1}$
An ARIMA model is one where the time series was differenced **at least once to make it stationary** and you combine the AR and the MA terms. So the equation becomes:
$Y - t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \beta_p Y_{t-p}\epsilon_t + \phi_1\epsilon_{t-1} + \phi_2\epsilon_{t-2} + ... + +\phi_q\epsilon_{t-q}$
In order to find the right order of differences we have to notice that it is the minimum difference required to get a near-stationary series which roams around a defined mean and the **ACF plot** (auto-correlation function which gives us values of auto-correlation of any series with its lagged values) reaches to zero fairly quick.
The next step is to identify if the model needs any AR terms. You can find out the required number of AR terms by inspecting the **Partial Autocorrelation** (PACF) plot.
Partial autocorrelation can be imagined as the correlation between the series and its lag, after excluding the contributions from the intermediate lags. So, PACF sort of conveys the pure correlation between a lag and the series.
Partial autocorrelation of lag (k) of a series is the coefficient of that lag in the autoregression equation of Y.
Any autocorrelation in a stationarized series can be rectified by adding enough AR terms. So, we initially take the order of AR term to be equal to as many lags that crosses the significance limit in the PACF plot.
For the number of MA terms we rely on the ACF plot. An MA term is technically, the error of the lagged forecast.
After having determined the values of p, d, q we can proceed by fitting the model.

4.2 Analysis

First of all, I have started by doing the differences in order to make my data static. I have carried out for this purpose the **Dickey–Fuller** test. It essentially tests the null hypothesis that a unit root is present in an autoregressive time series model.
All in all it appears to be stationary after the first difference and so the value of d is set to 1.

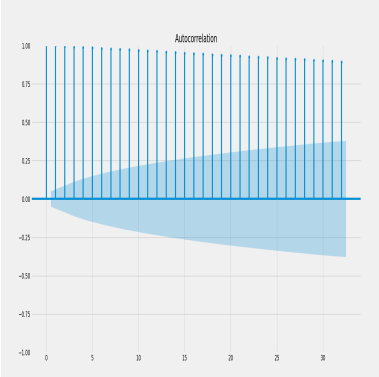


Figure 4.1: Autocorrelation

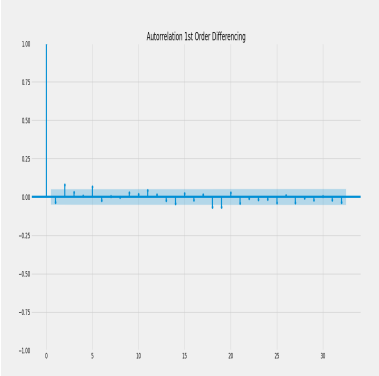


Figure 4.2: Autocorrelation first order differencing

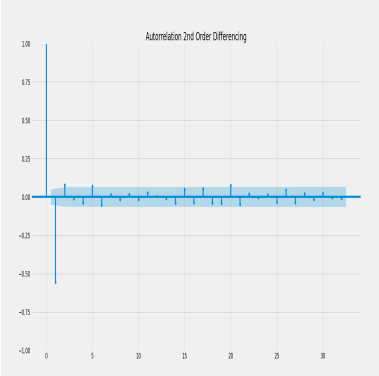


Figure 4.3: Autocorrelation 2nd order differencing

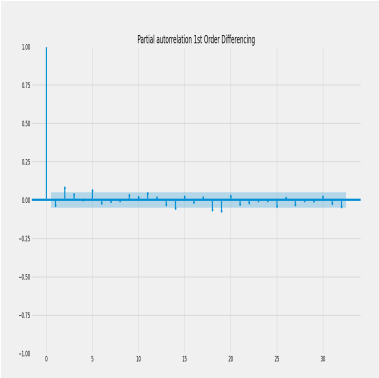


Figure 4.4: partial autocorrelation first order

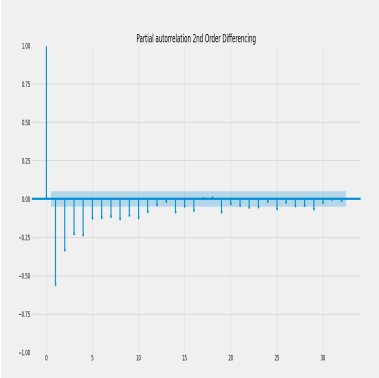


Figure 4.5: partial autocorrelation 2nd order differencing

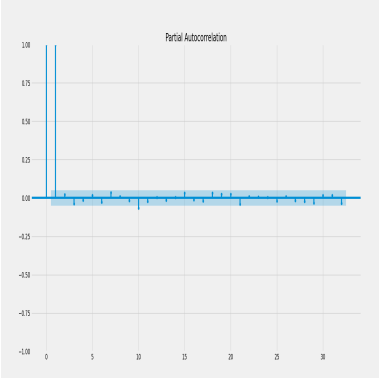
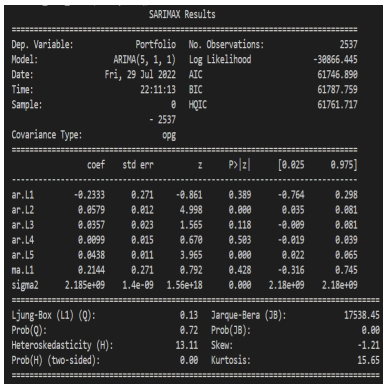


Figure 4.6: partial autocorrelation

So p and q are also calculated by observing the graphs. They result to be respectively equal to 5 and to 1.

Now that we have the values we can fit the model:



SARIMAX Results

Dep. Variable:	Portfolio	No. Observations:	2537
Model:	ARIMA(5, 1, 1)	Log Likelihood	-30866.445
Date:	Fri, 29 Jul 2022	AIC	61746.890
Time:	22:11:13	BIC	61787.759
Sample:	0	HQIC	61761.717
	- 2537		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.2333	0.271	-0.861	0.389	-0.764	0.298
ar.L2	0.0579	0.012	4.998	0.000	0.035	0.081
ar.L3	0.0357	0.023	1.565	0.118	-0.009	0.081
ar.L4	0.0099	0.015	0.670	0.503	-0.019	0.039
ar.L5	0.0438	0.011	3.965	0.000	0.022	0.065
ma.L1	0.2144	0.271	0.792	0.428	-0.316	0.745
sigma2	2.185e+09	1.4e-09	1.56e+18	0.000	2.18e+09	2.18e+09

Ljung-Box (L1) (Q):	0.13	Jarque-Bera (JB):	17538.45
Prob(Q):	0.72	Prob(JB):	0.00
Heteroskedasticity (H):	13.11	Skew:	-1.21
Prob(H) (two-sided):	0.00	Kurtosis:	15.65

Figure 4.7: ARIMA model

Model is not able to capture residual errors:

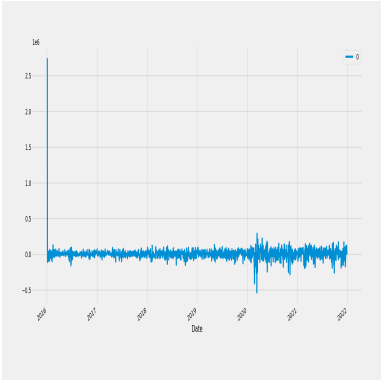


Figure 4.8: Residual errors

It represents the density of these errors. As we can see errors follow a Gaussian's shape.

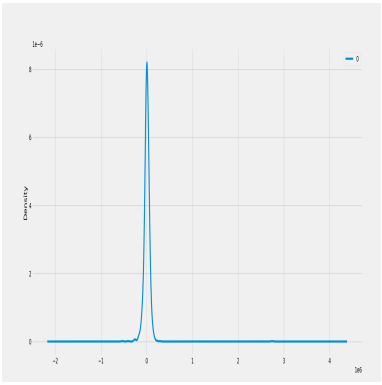


Figure 4.9: Density of residual errors

So we divide the dataset into two sections: train (80%) and test (20%) set.
We collect the test values within predictions and we compare each of them with the corresponding true values from the test set (which are store in the historical series).

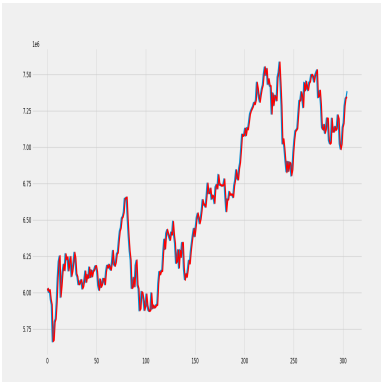


Figure 4.10: trainin and test predictions

Now we want to predict future 30 values: we take the first one from the historical data and the predict value is inserted in the historical series and used to predict the following one.

As we can see the prediction does not seem to work that good.

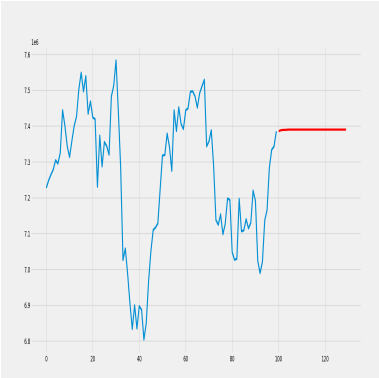


Figure 4.11: Forecasting

I have tried also to set the values of p, d and q manually in order to find improvements, but without satisfying results.
This could be caused by the enormous variability of data in the previous period. My prediction seems to follow a trend which **does not consider the fluctuations**.

Chapter 5

Neural Network

5.1 Neural Network

Now I want to do the **forecasting by using a Neural Network**.
For time-series predictions the LSTM is very common, so I have opted for this choice.
These sequential problems are problems comprised of a single series of observations and a model is required to learn from the series of past observations to predict the next value in the sequence.
A few words about LSTM:
LSTM belongs to the category of **recurrent neural network** which is any networks that contains a cycle within its network connections, meaning that the value of some unit is directly, or indirectly, dependent on its own earlier outputs as an input.
In practice, it is quite difficult to train RNNs for tasks that require a network to use information distant from the current point of processing.
Despite having access to the entire preceding sequence, the information encoded in hidden states tends to be fairly local, more relevant to the most recent parts of the input sequence and recent decisions.
A second difficulty with training RNNs arises from the need to backpropagate the error signal back through time. The concatenation of the derivation one has to compute to accomplish this task results in the so-called issue of vanishing gradients.
LSTMs divide the context management problem into two sub-problems: removing information no longer needed from the context, and adding information likely to be needed for later decision making.
LSTMs do this by first adding an explicit context layer to the architecture (in addition to the usual recurrent hidden layer), and through the use of specialized neural units that make use of gates to control the flow of information into and out of the units that comprise the network layers.
These gates are implemented through the use of additional weights that operate sequentially on the input, and previous hidden layer, and previous context layers. The gates in an LSTM share a common design pattern: a feed-forward layer, a sigmoid activation function, followed by a pointwise multiplication with the layer being gated. The choice of the sigmoid as the activation function arises from its tendency to push its outputs to either 0 or 1.
There are three gates: input gate (allow incoming signal to alter the state of the memory cell or block it), forget gate (can modulate the memory cell's self-recurrent connection, allowing the cell to remember or forget its previous state, as needed) and output gate (allow the state of the memory cell to have an effect on other neurons).

5.2 Analysis

To implement my Neural network I have used **Keras library**.
First of all I have processed my data in a way such that they could have been accepted by my LSTM. To do that I have used a proper function which has transformed features by **scaling** each one of them to a given range, between 0 and 1.
So I have **divided** my dataset in order to have a training set (65%), a validation set (35%) and a test set (35%).
Then for each set I have created a variable x (my inputs) and a variable y (my target/label).
X contained the first 4 elements and y the 5th element, and these values were assigned by moving one

step at a time.
Before passing x and y to the LSTM they needed to be **reshaped** in order to have the three entries accepted by keras: **samples, time steps, features**.
After that I have implemented my sequential model, a very easy one, composed by three LSTMs with 50 units apiece and a final dense layer.
In order to choose the number of units I have used the following formula:
$N_h = \frac{N_s}{\alpha*(N_i+N_o))}$
N_i is the number of input neurons, N_o the number of output neurons, N_s the number of samples in the training data,and N_o represents a scaling factor that is usually between 2 and 10.
Regarding the number of hidden layers, I have used just one. In fact generally, 2 layers have shown to be enough to detect more complex features. 1 hidden layer work with simple problems, like this one.
The specific loss function chosen is the **mean squared error** and the optimizer used is **Adam**.
The MSE tells how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line and squaring them.
Adam is instead a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments.
I have also added **Dropout** and **L2 Loss** as regularization technique, always to reduce the risk of overfitting.
Every LSTM layer should be accompanied by a Dropout layer. This layer will help to prevent overfitting by ignoring randomly selected neurons during training, and hence reduces the sensitivity to the specific weights of individual neurons. 20% is often used as a good compromise between retaining model accuracy and preventing overfitting.
L2 loss, or weight decay, penalizes only the weights of the affine transformation at each layer and works like this:
$L(\theta) = \sum L(f(x_i, \theta), y_i) + \frac{\lambda}{2} \sum ||w_l||^2$
I have also try to add **layer normalization**.
Layer Normalization directly estimates the normalization statistics from the summed inputs to the neurons within a hidden layer. The normalization does not introduce any new dependencies between training cases.
It works well for RNNs improving both the training time and the generalization performance. However in this case accuracies were better without this normalization.
As activation function I have used the **ReLU**:
$f(x) = max(0, x)$
Here is the compilation of the model:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 4, 50)             10400

dropout (Dropout)            (None, 4, 50)             0

lstm_1 (LSTM)                (None, 4, 50)             20200

dropout_1 (Dropout)          (None, 4, 50)             0

lstm_2 (LSTM)                (None, 50)                20200

dropout_2 (Dropout)          (None, 50)                0

dense (Dense)                (None, 100)               5100

dense_1 (Dense)              (None, 1)                 101

=================================================================
Total params: 56,001
Trainable params: 56,001
Non-trainable params: 0
```

Figure 5.1: LSTM

So I have fitted my model, using also a **validation set** in order to avoid over-fitting and made it ran for 100 epochs with a batch size of 16.
Then I have used the function "model.predict" to have both the training predictions and the test predictions.
I have reconverted these data into their original form in order to plot both train and test predictions into a graph:

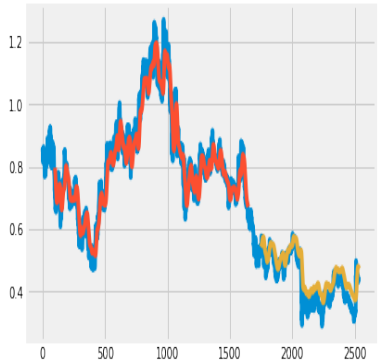


Figure 5.2: train and test predictions

Finally I have tried to predict the values day by day for a time frame of 30 days. Here are reported the **forecasting** for my portfolio:

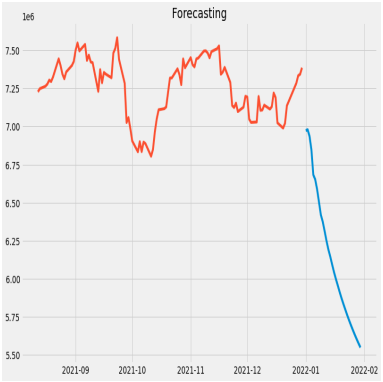


Figure 5.3: Portfolio

I have also tried to analyse the predictions of the trend for **each single stock**:

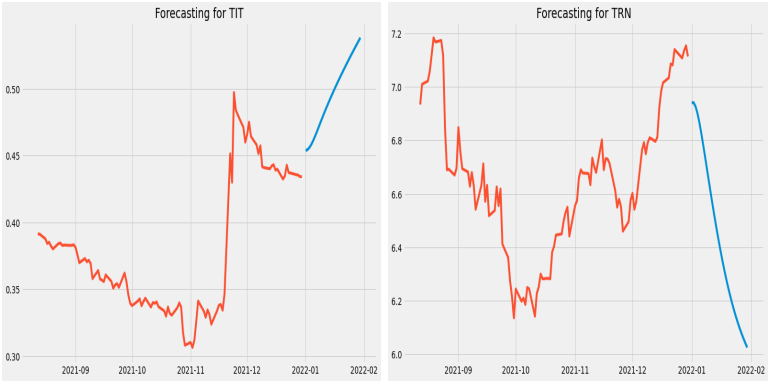


Figure 5.4: Telecom and TERNA

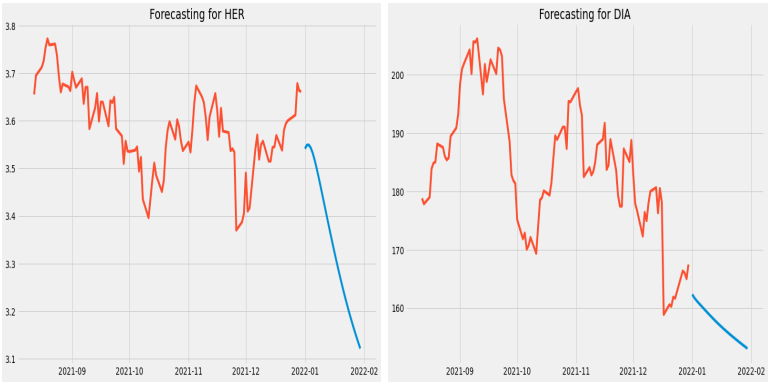


Figure 5.5: Hera and DiaSorin

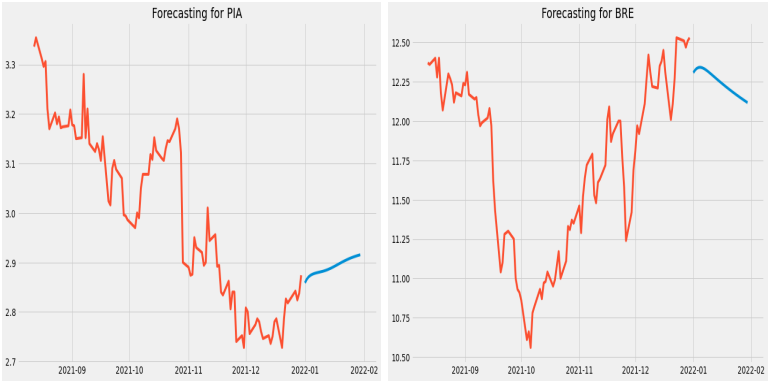


Figure 5.6: Piaggio and Brembo

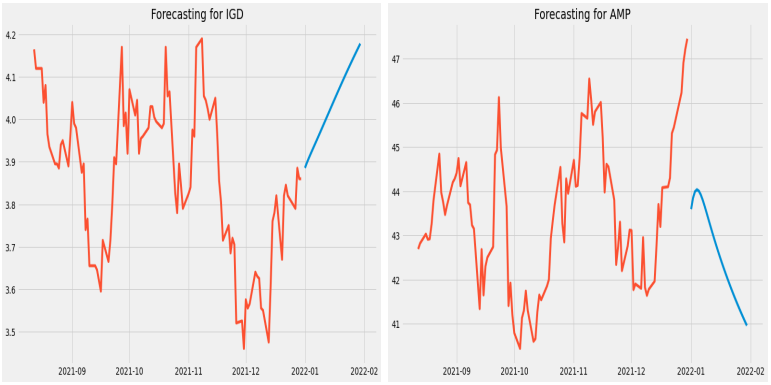


Figure 5.7: IGD SIIQ and Amplifon

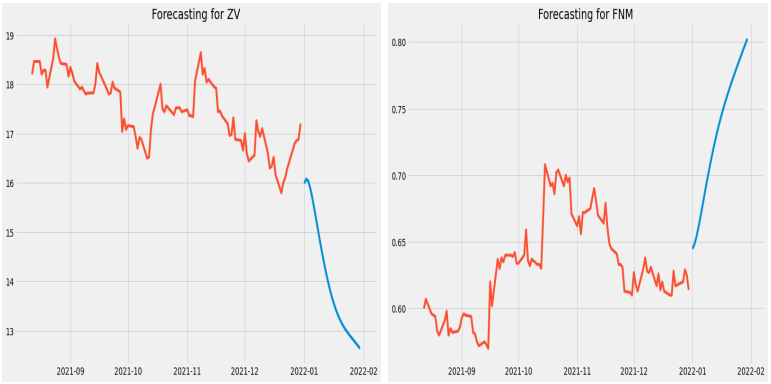


Figure 5.8: Zignago Vetro and Gruppo FNM

We can notice that also here the predictions do not seem so optimal. The LSTM tries to find a tendency and to follow it.
I have tried with different values of the parameters, especially I have modified the size of the observation window, as well as the number of hidden layers, of units or of epochs within the network, but I have recognised significant improvements.

Chapter 6

Conclusions

6.1 Conclusions

All in all, neither the econometric model nor the LSTM seems to predict that good the future trend of the portfolio.
This may be due to the fact that my model does not consider exogenous variables. These variables are essential when we want to tackle real data as I have also learnt from the previous projects carried out during this course.
The historical series of data has a strongly fluctuated behaviour and depends deeply on external facts which are unpredictable from any kinds of models, and the contemporary story has taught us this: covid19 and now the Ukraine war are having a huge impact on global market and this is reflected into raises and falls within the graph.
Fluctuations are also made more evident from another phenomenon: nowadays AI models are used almost everywhere in finance, they are all fed with the same data and predict more or less the same behaviours. So if there is a certain trend all operators tend to follow it amplifying the phenomenon.

Chapter 7

Appendix

7.1 Appendix

I report here below the link to the GitHub repository of my code. I have also uploaded my python code in moodle within an apposite folder.
Code:
https://github.com/saramiotto/AI-for-Finance-final-project.git