Università di Pisa

DIPARTIMENTO DI FISICA
Corso di Laurea Magistrale in Fisica

Relazione progetto di Computing Methods for Experimental Physics and Data Analysis

Autori: Angelo Lasala, Daniele Maria Di Nosse, Raffaele Paradiso

Anno Accademico 2020-2021

Indice

1 Introduction

It's well-know that the prediction of future stock market prices is a difficult task. The famous Efficient Market Hypothesis (EMH), in it's various forms (strong, semi-strong, weak) claims at different degrees that all past information can't be used to outperform the market, since it is instantly processed and reflected in the current price. In other word, prices follow a random path in time. With the advent of machine learnings algorithms like Artificial Neural Networks, a lot of authors [1] tried to use them to capture all the possible (non-linear) information that can break the EMH. This study is aimed to do a similar task using ANN like LSTM (Long-Short Term Memory), DNN (Deep Neural Network), RAF (Random Forest) and CNN (Convolutional Neural Network), comparing one against the other. Then, with three pre processing methods (Discrete Wavelet Transform, Principal Component Analysis and Autoencoder) we have tried to improve their performances and/or computing time. Every ML algorithms is then used to set up a trading strategy that can outperforms the market.

2 Data

The data we used was the close prices SP500's companies in the period spans from the 01/01/1995 to 31/12/2020. Since SP500 index experienced some changes during this time, we chose to keep data only from companies that were always been tracked. Starting from 652 companies, the ones that survived were 365. We downloaded this data from Yahoo Finance, after got the tickers from Wikipedia [1].
Let P_t^s be the price of the s-company at time t. From this data we obtained the dataframe of m period returns computing

$$R_t^{m,s} = \frac{P_t^s}{P_{t-m}^s} - 1 \tag{1}$$

The periods m over which we computed the returns represent the features we used as input of our models.
We set our problem as a binary classification one instead of a regression, so from the dataframe of returns we created another one containing only 0 and 1 in the following way: for every day, every entry in the returns dataframe becomes 0 if it's less than the median of returns of that day and 1 otherwise.

2.1 Creation of input and target data

For our implementation we relied on keras with tensorflow as backend, so we had to construct tensors of the right shape to be the input of the models. To do so we set our problem in this way: first, we splitted our data in 17 study periods, each of which containing 1308 days. For each period we put aside 30% of it as test and use the rest as training. Then we normalized the data with a StandardScaler normalization,

$$\tilde{R}_t^{m,s} = \frac{R_t^{m,s} - \mu_{train}^m}{\sigma_{train}^m} \tag{2}$$

but we applied it only on the train set in such a way to avoid any look-ahead bias. We chose so a 1-day-sliding window of 240 days (approximately one year of trading days) in order to construct the input sequences and select the one day ahead from the binary test set as our target. We have done this for each companies, so we end up with a structure as shown in Figure 1.

[1] https://en.wikipedia.org/wiki/List_of_S%26P_500_companies#S&P_500_component_stocks

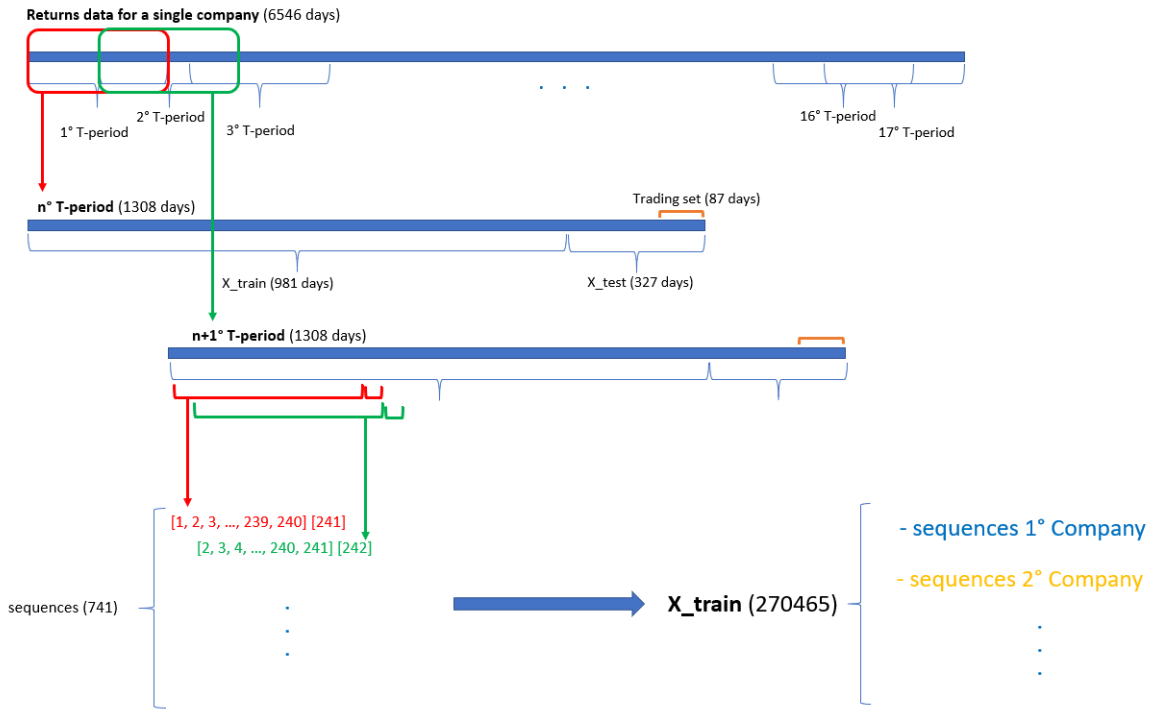

Figura 1: Input sequences

3 Methodology

The pipeline we used is quite simple and straightforward:

1. Select features from the input sequences of 240 data.
2. Set a grid of hyperparameters for the particular algorithm in exam.
3. Train the model on the 80% of the train set and use the remaining set as validation.
4. Choose the set of hyperparameters that gives the best validation accuracy.
5. Use the model trained to make predictions on the test set and construct ROC curves (Receiver operating characteristic curves)
6. Use these predictions to create the portfolios over the actual 87 (equal to the lenght of the test set minus 240) trading days in each period.
7. Check with a simple back-testing the performance in terms of accumulative returns. At this stage also a randomness test (Welch's t-test) is computed.

3.1 Algorithms

Once all the sequences were made, the first split between the procedure happened, according to the behaviour of the different algorithms we implemented. Since we were dealing with a classification problem, we set the loss function to binary cross entropy for all of them.

3.2 LSTM - Long Short-Term Memory

Long Short-Term Memory networks belong to the family of Recurrent Neural Networks (RNN) that are very suitable for time-series analysis thanks to their capacity of sharing parameters across several time steps. LSTM is an improved version of a classical RNN that suffers the problem of gradient vanishing or exploding. LSTM networks are composed by an input layer, one or more hidden layers and an output layer. The main characteristic of these network can be found in the hidden layer consisting of the so called memory cells. Each of them has three gates that are able to manage the information that comes from the previous cell state s_{t-1} + the input at that step x_t and decide which part of this information will contribute to the cell state s_t. See Figure 2 for details.

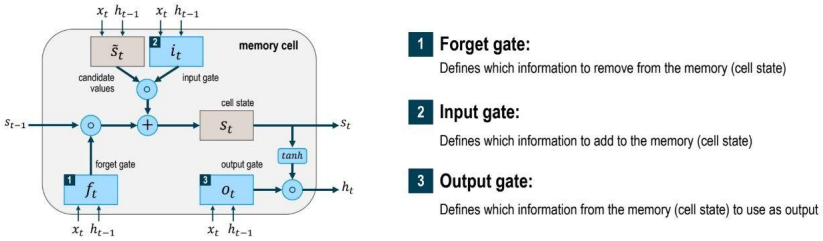

Figura 2: LSTM cell structure took from [2]

In this case the features selection was simple: all the 240 steps in the sequences were kept. We trained several models, varying both the architecture and the parameters of the layers. We found that the models are high susceptible to overfitting, obtaining validation accuracies of the order of 0.50 or below in every periods for architectures with just two LSTM layers. We discarded then these models and decided to keep the simpler ones composed by an input layer, a dropout layer set at 0.2, an LSTM layer with 25 hidden neurons, another dropout layer set at 0.2 as well and a final output layer with a single neuron with a sigmoid as activation function, in order to obtain a value between 0 and 1 that expresses the classification probability. This architecture is shown in Figure 3. With two dropout layers we were able to reduce significally the overfitting. Following [3] we used as optimizer RMSprop, a mini-batch version of rprop, but we obtain slightly better performance with Adam. One interesting thing about optimizers used was that for RMSprop we had to set also the clipvalue hyperparameter (the default is NULL) to avoid None values for the loss and accuracy due to too much large (or to much small) weights updates. So, every time that the gradient exceeds 0.5 or 0.3 (they are the two values choosen), it's clipped back.

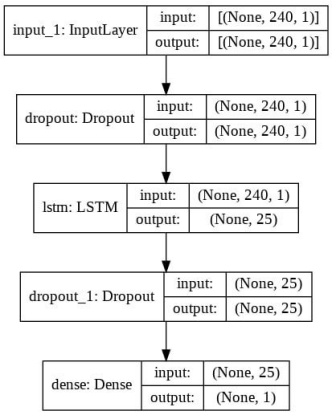

Figura 3: LSTM architecture of our best model.

3.3 CNN - Convolutional Neural Network

Convolutional Neural Network (CNN) allows us to extract several features inside the input data locally. CNNs are developed for two-dimensional image data, although they can be used for one-dimensional data such as sequences of text and time series. When operating on one-dimensional data (convolutional 1D), the CNN reads across a sequence of lag observations and learns to extract features that are relevant for making a prediction. As the LSTM, the input are composed by 240 returns about previous days and the target is the binary label of the following day. For our purpose the hyperparameters are selected following the "nature" of the problem, so the kernel size is almost one month of trading days (20) and the stride is set to one week (5). The first two model have respectively 8 and 15 filters while the third has got a more complicated structure, in contrast to the first ones, where there is only one maxpooling layer, here there is also a minpooling layer connected the convolutional layer to allow the model to learn at the same time the minimum and maximum value inside the kernel.

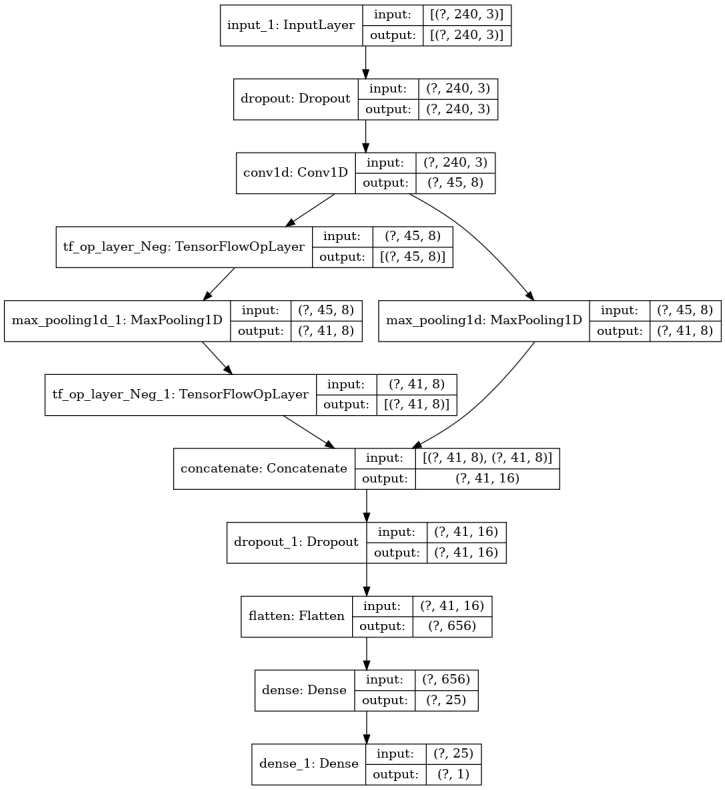

Figura 4: Structure of multiheaded model

Then, for each models, a flatter layer condenses all the extracted features in a one dimensional array that passes to a dense layer composed by 25 nodes, its outputs are connected with a single nodes that returns the binary classification. Lastly, because of the massive presence of noise in train and validation loss, two dropout layers are added to the models: the former following the input layer and the latter between convolutional and pooling layers.

3.4 DNN - Deep Neural Network

Deep Neural Network (DNN) is the simpler deep architecture belongs the family of ANN. Basically it is composed by several full-connected layers with different numbers of nodes.
Unlike LSTM and CNN, this architecture can't learn neither globally nor locally features from the input data, so in this case the features selection was made "by hand". From the 240 days of input data we selected 31 features corresponding to the m period returns with $m \in [1, 2, ..., 20] \cup [40, 60, ..., 240]$. There isn't a general rule to decide the right numbers of layers and nodes but, as literature suggests, the suitable way to tackle the problem is analyse the input data with respect to the output. For this reason the choice fell on a model containing 4 hidden layers with the number of nodes starting from 31 and descreasing to 5, through 15 and 10 as in [4]. The optimizer used was Adam.

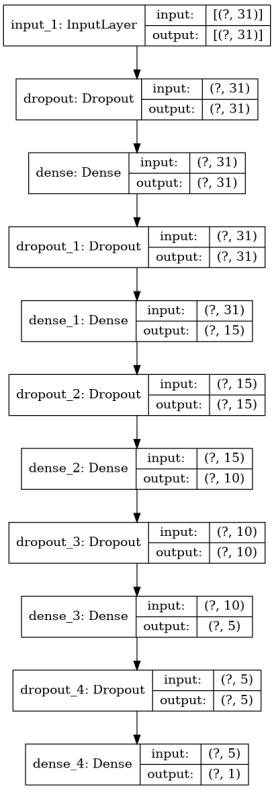

Figura 5: Architecture of the best DNN model trained

3.5 RAF - Random Forest

Random Forest (RAF) are an ensemble learning method for classification or regression that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.
Simply speaking, random forests are composed of many deep yet decorrelated (because of the random features selection) decision trees built on different bootstrap samples of the training data.
Normally trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, have low bias, and very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.
The main reason to use random forest to benchmark the others models is because it represents the state-of-the-art of machine learning model that requires almost no tuning and can delivers very good results, furthermore RAF is invariant under scaling and various other transformations of feature values and is robust to inclusion of irrelevant features. Like the DNN this model can't learn or extract features from the input data.
Also in this case the features selection was made "by hand" and the features was the same of the DNN i.e. from the 240 days of input data we selected $F = 31$ features corresponding to the m period returns with $m \in [1, 2, ..., 20] \cup [40, 60, ..., 240]$.
The final output is a committee of T trees and classification is performed as majority vote.
The chosen parameters for the random forest were:

1. The maximum depth of every tree: $J = [15, 20, 20, 25, 25]$
2. The number of trees (estimators): $T = [1000, 1000, 500, 1000, 2000]$
3. The value of the random feature selection: $m = \sqrt{F}$
4. The function that measure the quality of the split: gini index (this parameter is tree-specific)

4 Preprocessing

Even if the results from the previous models were pretty good (and, furthermore, not random), we asked ourselfs if there was a way to handle data before putting them into the algorithms. We decided so to apply two preprocessing tools: Discrete Wavelet Transforms for the decomposition of the time signal of price returns and

the well-know Principal Component Analysis that used together with the DWT gave us an astonishing improvement of the results for the LSTM and CNN algorithms as the next sections will show.

4.1 PCA - Principal Component Analysis

Principal Component Analysis is a common way to reduce the dimensionality of the data through a projection onto a less-dimensional space composed by the so-called principal components, i.e. the eigenvectors of the covariance matrix of the data.
Our implementation articulates as follow: Starting from a 365-dimensional space (equal to the number of companies we tracked from the available data of the SP500 index in our entire period of study), we choosed to keep the first 250 components that correspond to the eigenvectors of the covariance matrix with the greatest eigenvalues. This resulted in a preserved variance of 0.94. Then, since these orthogonal vectors are linear combinations of the ones from the higher dimensional space, we selected the features that most contribute to them, resulting in 175 companies.

4.2 DWT - Discrete Wavelet Transform

Discrete Wavelet Transform (DWT) is a kind of time series decomposition that is able to extract information from both time and frequency domain simultaneously at different scales. Computationally speaking, it's implemented through a convolution of the signal with a low pass filter and a high pass filter, respectively output the so-called approximations and details coefficients on that particular scale [5].
In our study, because of the objective is to forecast the returns data of the following day, we have applied the DWT to returns data over 3 time scales in order to keep only the "short-scale" information. Figure XX and Figure XX depict respectively the DWT of price and returns (the one we used), in particular they contains the three details coefficients and the approximation on the last time-scale.

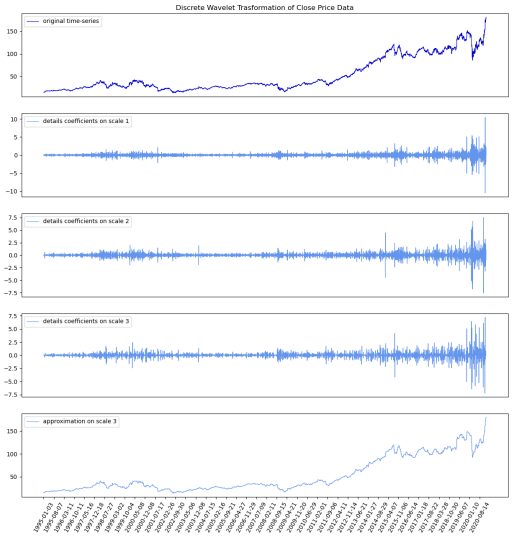

Figura 6: DWT of price data over 3 scales.

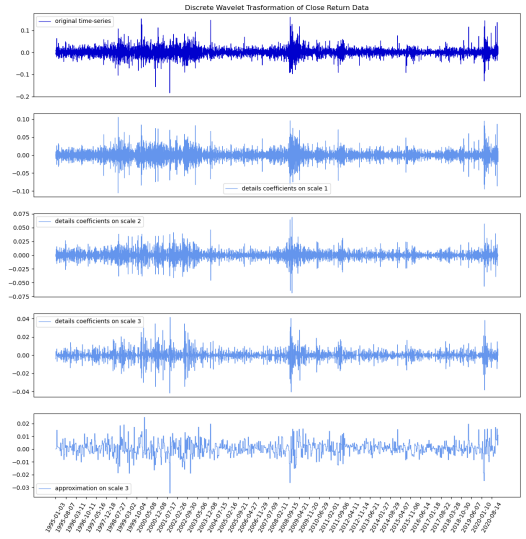

Figura 7: DWT of return data over 3 scales. The input for LSTM and CNN were data in the last three images

4.3 Autoencoder

An autoencoder is a neural network that is trained to attempt to copy its input to its output.
Internally, it has a hidden layer h (MiddleLayer in Figure 8) that connects two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$. An autoencoder succeeds in learning if $g(f(x)) = x$ everywhere. Usually they are restricted in a ways that allow them to copy only approximately and so the model is forced to prioritize which aspects of the input should be copied. One way to obtain useful features from the autoencoder is to constrain h to have smaller dimension than x. An autoencoder whit this property is called undercomplete. The learning process is performed minimizing a loss function (like mean squared error as in our case) $L(x, g(f(x)))$. In our study, because the input for the random forest and DNN were the features selected "by hand", we delegated the choice of the features to the Autoencoder in order then to compare the results

obtained with them to the ones from our a priori choice. Figure 8 describes the architecture of our Encoder: as it 8 shows, the last layer of the Encoder contains 31 nodes, resulting so in the same number of features as in RAF and DNN.

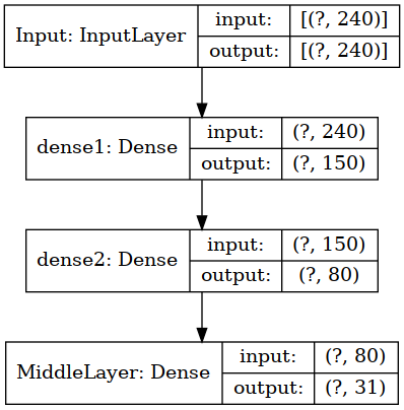

Figura 8: Structure of encoder part of autoencoder

The following images represent how the Autoencoder performed on a single sequence. Of course there is a loss of information but, as the third image shows, the differences between reconstructed and original signal are small compared to the original one. So, in the encoded data, there is the overwhelming majority of information.

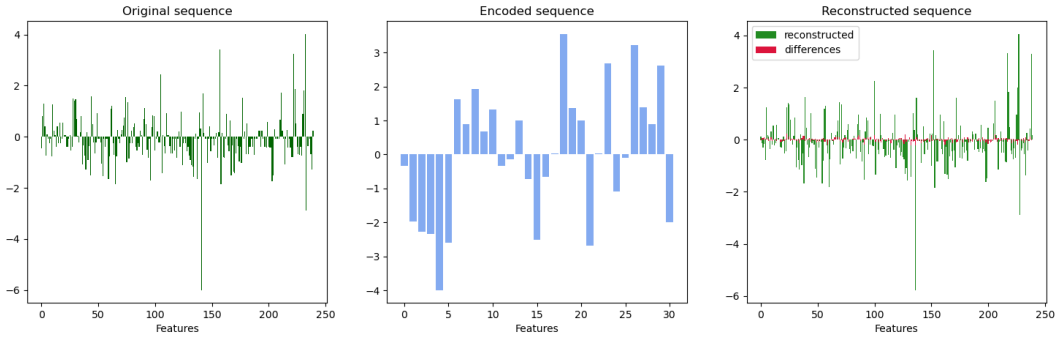

Figura 9: Original, Encoded and Reconstructed signal

5 Results

In the present section we show the results obtained with the previously explained models.
The trading strategy we set was to sort the predictions from each of our models in a crescent order and choose the first 10 of them for shorting and the last 10 for longing. In according to this, for all the models the randomness test was assessed by randomly sample from our basket of companies to trade 10 of them for shorting and 10 of them for longing. This sampling was done for 500 times (our monkeys number) and the distribution of mean daily monkeys' returns together with the distribution of accumulative monkeys' returns were computed. A Welch's t-test was made in order to compared the random mean daily returns distribution to the ones that have come up from our models. With a significant level of 0.05, none of them are even near to be random, except for PCA-Autoencoder-DNN over the first period of study.
The accumulative returns were computed by setting an initial amount of money of 1000$ with the choice of use 10% of it for everyday investment. In other words, each day we picked a part of the previous end-day money and created an equal-balanced portfolios over the 20 companies we traded. Let m_t be the amount of money at time t and $\mathbf{r_t}$ be the vector containing all the 20 companies returns ad time t

$$m_{t+1} = m_t + \frac{1}{20}\sum_{i=1}^{20} 0.1 \times m_t \times r_{ti} \tag{3}$$

In the following we present ROC curves and the evolution of the accumulative returns during the 17 periods of study. One thing to keep in mind is that the formers are computed over all the predictions the models made, while the latters show results only on the top 10 and the bottom 10 companies. Excpet for the LSTM,

ROCs told us that the model randomly choose class 1 and class 0: AUC mean spans 0.5 in just one standard deviation. When we restricted our attention on 20 companies and check what this strategy should gave us if performed in the past, the situation flipped to a significantly non random predictions.

5.1 CNN

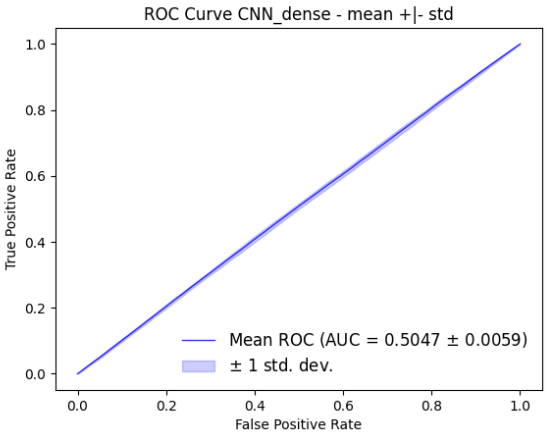

Figura 10: ROC curve and AUC distribution for CNNdense.

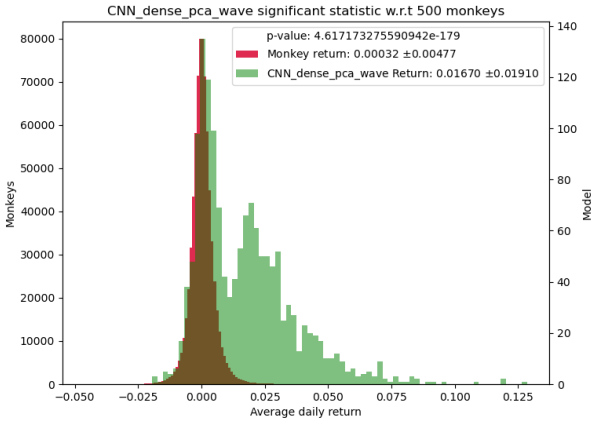

Figura 11: Distribution of mean daily returns for CNNdense compared to the one of our team of 500 monkeys.

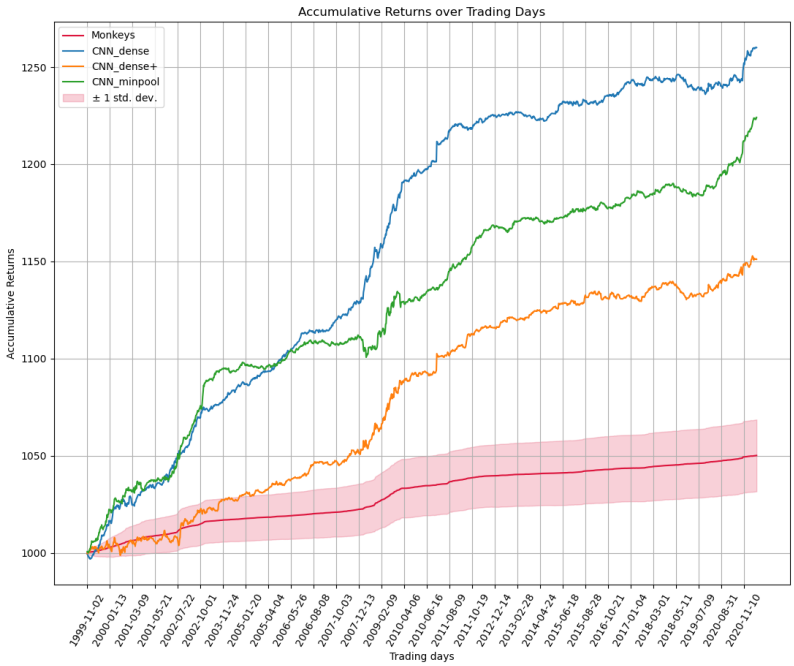

Figura 12: Accumulative returns for CNN models compared to the ones our team of 500 monkeys.

5.2 LSTM

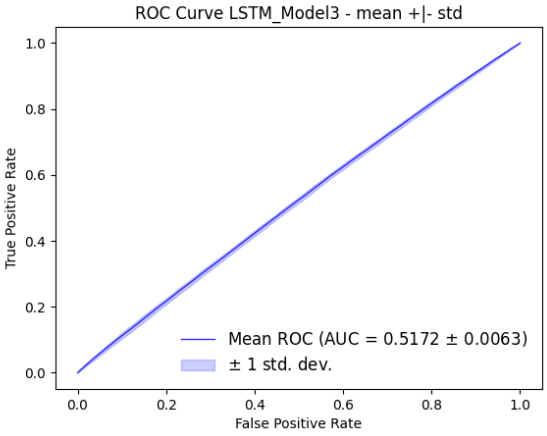

Figura 13: ROC curve and AUC distribution for LSTM Model 3.

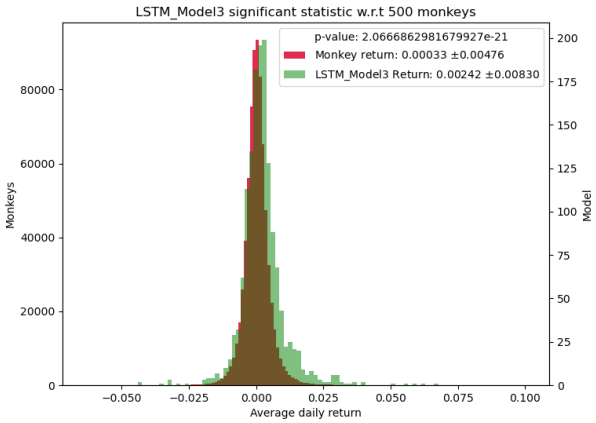

Figura 14: Distribution of mean daily returns compared to the one of our team of 500 monkeys.

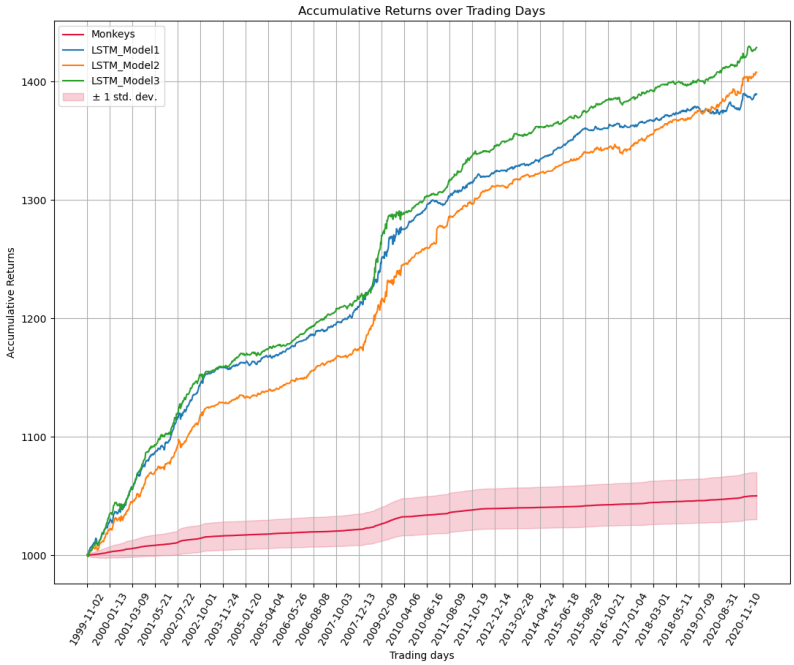

Figura 15: Accumulative returns for LSTM models compared to the ones our team of 500 monkeys.

The Long Short-Term Memory network that gave us the best accumulative returns has been LSTMModel3. As already sad it has 25 hidden neurons in the LSTM layer and the optimizer is RMSprop with a clip value of 0.3. The smaller clip value gave the network the opportunity to weight more equally the long-term interaction respect to the short-term ones. Although the difference between the models wasn't so much significant, this can be an heuristically explanation to the graph.

5.3 RAF

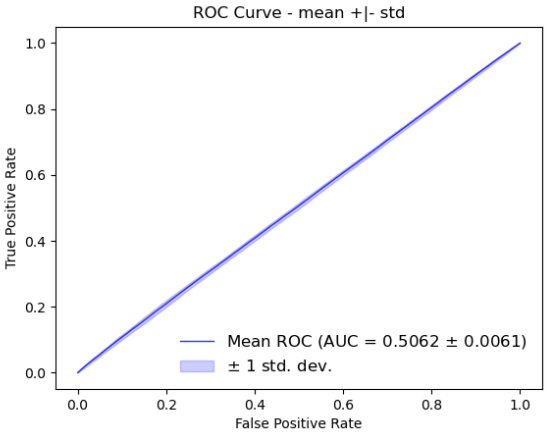

Figura 16: ROC curve and AUC distribution for RAF Model 3.

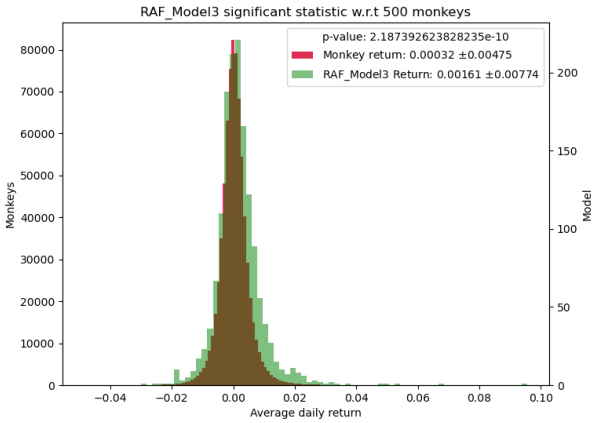

Figura 17: Distribution of mean daily returns compared to the one of our team of 500 monkeys.

The Random Forest model that best performed of the back testing has been the RAFModel3 having 1000 estimators with a maximum depth of 15

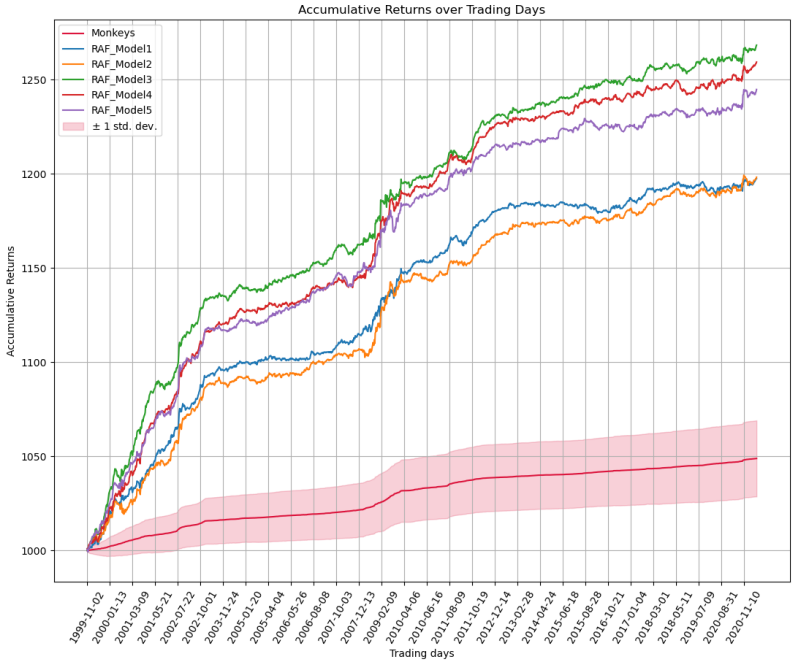

Figura 18: Accumulative returns for RAF models compared to the ones our team of 500 monkeys.

5.4 DNN

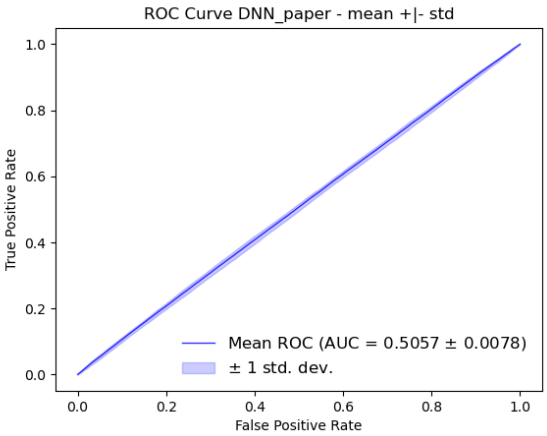

Figura 19: ROC curve and AUC distribution for DNN paper.

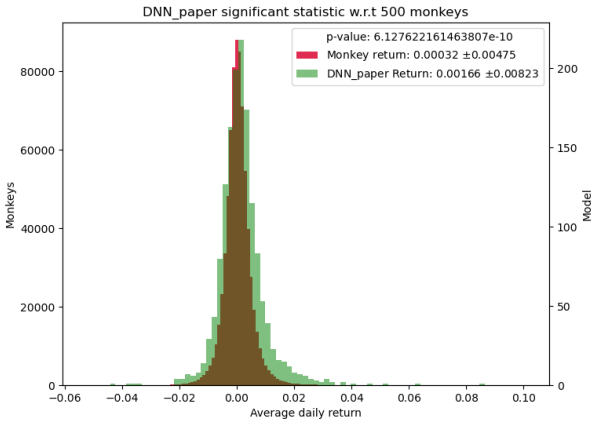

Figura 20: Distribution of mean daily returns compared to the one of our team of 500 monkeys.

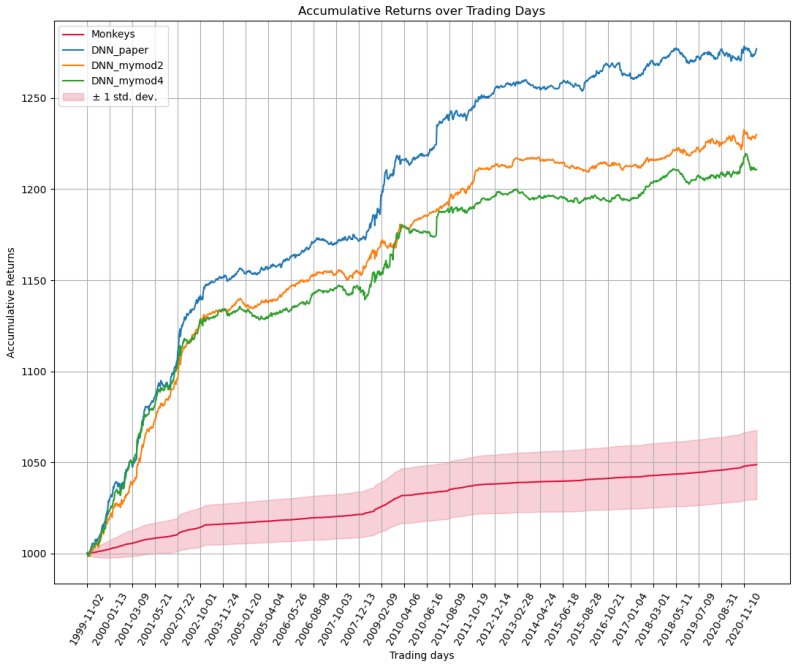

Figura 21: Accumulative returns for DNN models compared to the ones our team of 500 monkeys.

5.5 PCA-DWT-CNN and PCA-DWT-LSTM

Since DWT enlarges the dataset fed into the algorithms, the implementation of PCA reduces at the beginning the amount of data but with the advantage of maintaining the most of variance of them. This match was able to keep the computational time at the same scale of the non preproccessed models (or even less) giving also an improvement of a magnitude that we didn't expect at all. The wavelet we used in the DWT has been 'Haar' that gave us information of three different time scale. As already said we used the first three details coefficients (high frequency information on their scale) and the third approximation (containing all the remaining signal).

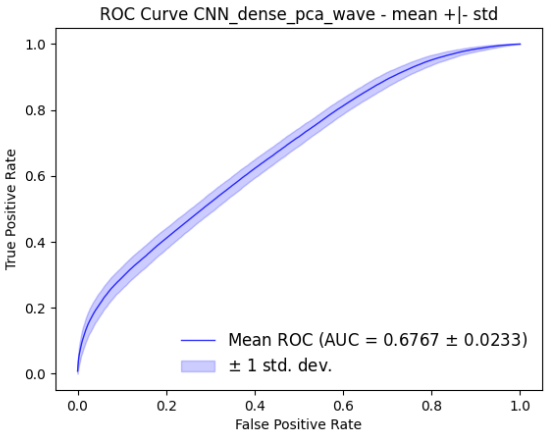

Figura 22: ROC curve and AUC distribution for PCA-DWT-CNN.

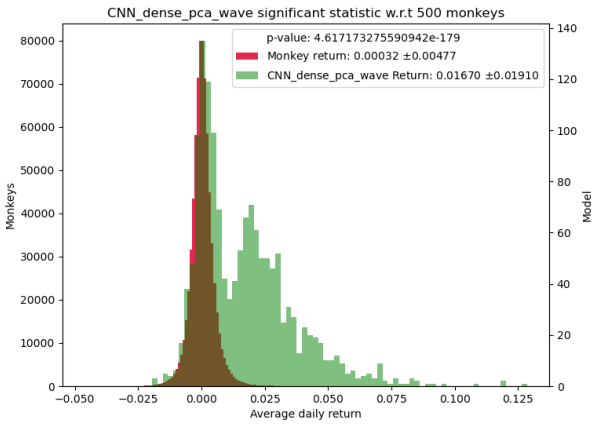

Figura 23: Distribution of mean daily returns for PCA-DWT-CNN compared to the one of our team of 500 monkeys.

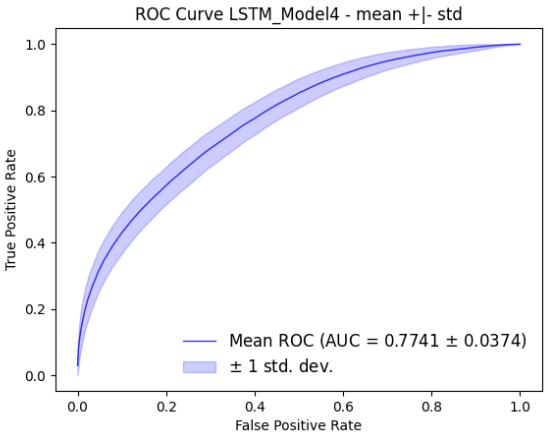

Figura 24: ROC curve and AUC distribution for PCA-DWT-LSTM paper.

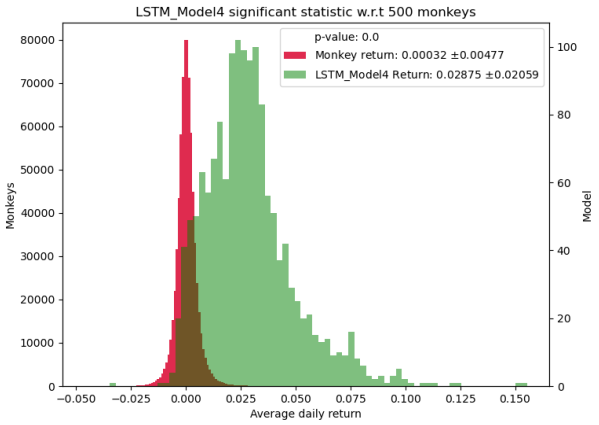

Figura 25: Distribution of mean daily returns for PCA-DWT-LSTM compared to the one of our team of 500 monkeys.

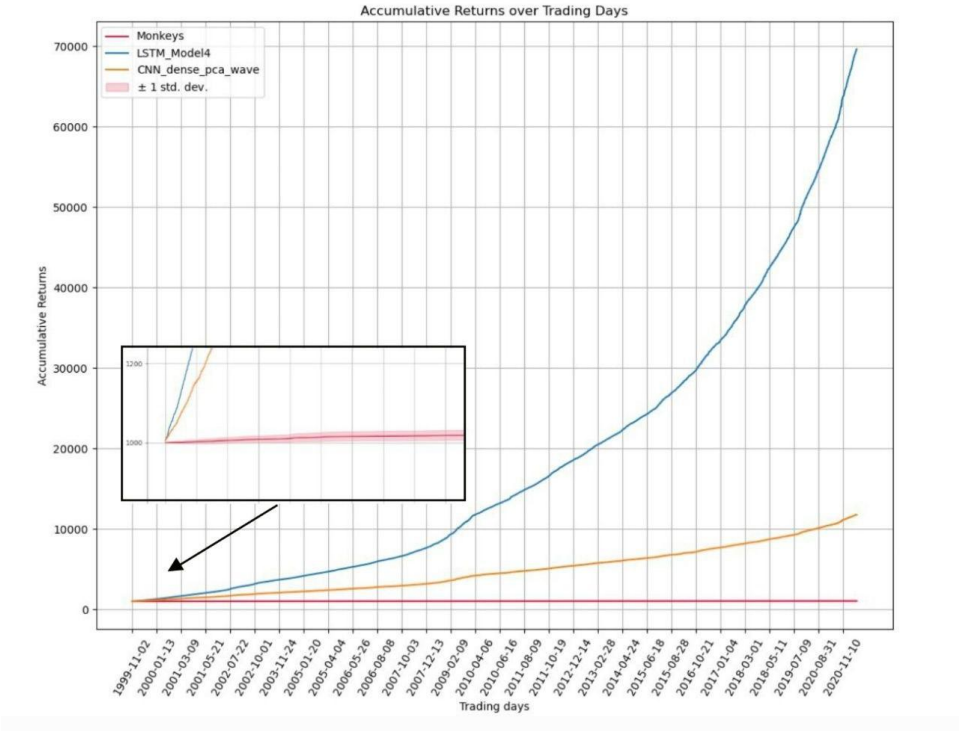

Figura 26: Accumulative returns for PCA-DWT-CNN and PCA-DWT-LSTM models with a close up to monkeys results.

5.6 PCA-Autoencoder-RAF and PCA-Autoencoder-DNN

To improve the results of RAF and DNN models we tried to substitute the feature selection "by hand" to an automatic one. Even if the encoded sequences seems good enough to our purpose, the elevate computational time (about 22 h) was prohibitive for the application over the all 17 study periods. Nevertheless this issue, we tried to compare the results of this approach with results of previous models over the first study period.

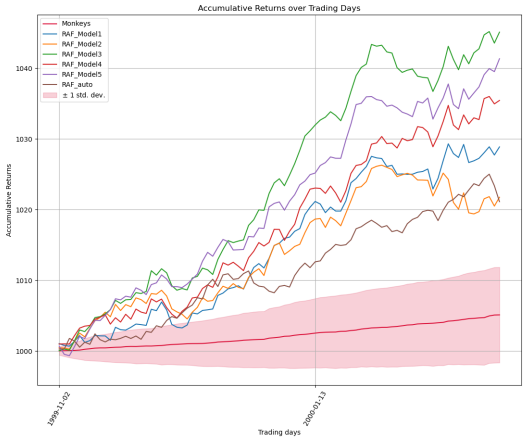

Figura 27: Accumulative returns PCA-Autoencoder-RAF

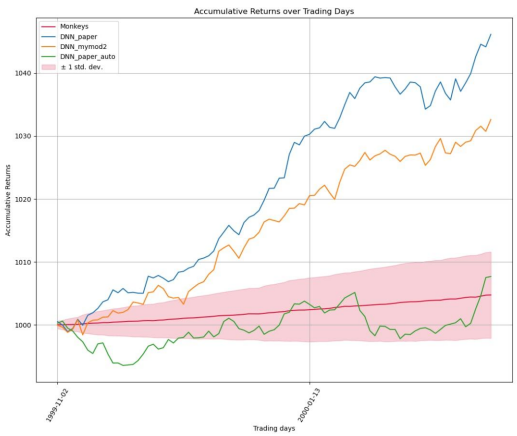

Figura 28: Accumulative returns PCA-Autoencoder-DNN

As can be seen, the accumulative returns obtained with the autoencoder were worst then the previous models, in particular for DNN in which it totally lies in the randomness region.

6 Conclusion

As the previous images show, the algorithms that best performed , without preprocessing, both in terms of ROC and accumulative returns was LSTM. This isn't a surprising result, since LSTM are the state of the art algorithm for handling time series forecasting. The surprising fact, instead, is the win of the DNN over the

RAF, in contrast to the results of [3]. This is can be due to the different handling of the data from the S&P500 index. For more details see [3].
Without any doubt, DWT+PCA+LSTM is the winner out of this challenge.
Since our tests showed also that PCA+LSTM returns poorer results then just the LSTM alone (we didn't show this here), we think that DWT+LSTM can perform even better, but at the cost of a greater computational time.
Even if this is beyond our purposes, it's interesting to note how even the monkeys were able to experience a significantly increase in the accumulative returns in the period of the 2007-2009 crisis. We think that the main contribution to this increase comes from the companies on which monkeys short. During a crisis all the prices fall because the major trend is to sell, so if you randomly pick a stock and go short, you will probably end with a positive return. Anyway, it must be said that we didn't take into account any transaction cost or other complications such finding a lender for shorting.
Even with these limitations, we can conclude that, according to the literature we investigated, EMH is an idealization of the market that can be predicted with the help of clever preprocessing and ANNs.

7 Bibliography

(1) Omer Berat Sezer, Mehmet Ugur Gudelek, Ahmet Murat Ozbayoglu, Financial time series forecasting with deep learning : A systematic literature review: 2005–2019, Applied Soft Computing, Volume 90, 2020, 106181, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2020.106181.
(2) Graves, A., 2014. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.
(3) Thomas Fischer, Christopher Krauss, Deep learning with long short-term memory networks for financial market predictions, European Journal of Operational Research, Volume 270, Issue 2, 2018, Pages 654-669, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2017.11.054.
(https://www.sciencedirect.com/science/article/pii/S0377221717310652)
(4) Christopher Krauss, Xuan Anh Do, Nicolas Huck, Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500, European Journal of Operational Research,
Volume 259, Issue 2, 2017, Pages 689-702, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2016.10.031.
(https://www.sciencedirect.com/science/article/pii/S0377221716308657)
(5) Jothimani, Dhanya and Yadav, Surendra S. and Shankar, Ravi, Discrete Wavelet Transform-Based Prediction of Stock Index: A Study on National Stock Exchange Fifty Index (2015). Journal of Financial Management and Analysis, Vol. 28(2), 2015, Available at SSRN: https://ssrn.com/abstract=2769529