IBM Stock Price Predictor

Bruno de Vargas Zimpel

July 2019

Abstract

The proposed challenge was to create a system to act as a predictor for IBM stock price based on Machine Learning techniques. The development of such system is briefly described at this report. This predictor has two regression models: one based on Random Forest Regressor algorithm and another based on Long Short-Term Memory Neural Networks.

The entire project was developed using Python programming language. The first analysis and study took place on Jupyter Notebooks, later passing to python scripts running directly on terminal of a Debian 10 OS. The link for the repository of the project can be found **here**.

1 Problem and data

The first step of the development was to understand the available data and the problem itself. The dataset used on the project was imported from Quandl [1] and is illustrated by the Figure 1. A brief description of each columns is located below.

None	ticker	date	open	high	low	close	volume	ex-dividend	split_ratio	adj_open	adj_high	adj_low	adj_close	adj_volume
0	IBM	2018-03-27	153.95	154.8697	151.160	151.91	3810994.0	0.0	1.0	153.95	154.8697	151.160	151.91	3810994.0
1	IBM	2018-03-26	151.21	153.6570	150.280	153.37	4038586.0	0.0	1.0	151.21	153.6570	150.280	153.37	4038586.0
2	IBM	2018-03-23	152.25	152.5800	148.541	148.89	4389015.0	0.0	1.0	152.25	152.5800	148.541	148.89	4389015.0
3	IBM	2018-03-22	155.00	155.2499	152.000	152.09	4617371.0	0.0	1.0	155.00	155.2499	152.000	152.09	4617371.0
4	IBM	2018-03-21	156.57	158.2000	155.920	156.69	3240695.0	0.0	1.0	156.57	158.2000	155.920	156.69	3240695.0

Figure 1: Available data from the original dataset

- **ticker**: Unique series of letters used to identify to which company these data makes reference to. In this case, **IBM**.
- **date**: Trading day that the variable is related to.
- **open**: The opening price (in US$) at which a stock was traded on a trading day.
- **high**: Highest price the stock reached on a trading day.
- **low**: Lowest price the stock reached on a trading day.
- **close**: The last price at which a stock was traded during a trading day.
- **volume**: Number of stock transactions made on a given trading day.
- **ex-dividend**: Describes if the stock was traded without the value of the next dividend[1] payment.
- **split_ratio**: A stock split is a corporate action in which a company divides its existing shares into multiple shares. Basically, companies choose to split their shares so they can lower the trading price of their stock to a range deemed comfortable by most investors and increase liquidity of the share. A 1:1 split ratio means the share wasn't splitted.
- **adj_variable**: Represents a more accurate value for the variable that takes its initial value as a start point, but considers others factors such as dividends, stock splits and new stock offerings.

[1] Dividend is the distribution of reward from a portion of the company's earnings and is paid to a class of its shareholders.

So the goal here is to predict the value of **adj_close** on a given day. The available data covers the trading days (week days) from January 2^{nd} of 1962 until March 27^{th} of 2018, and the evolution over time of the adj_close value can be seen on Figure 2:

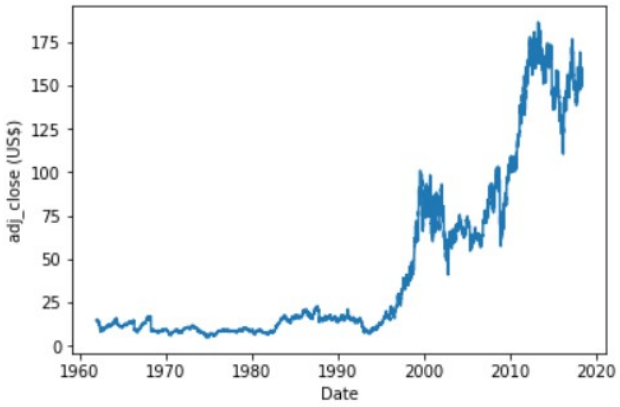


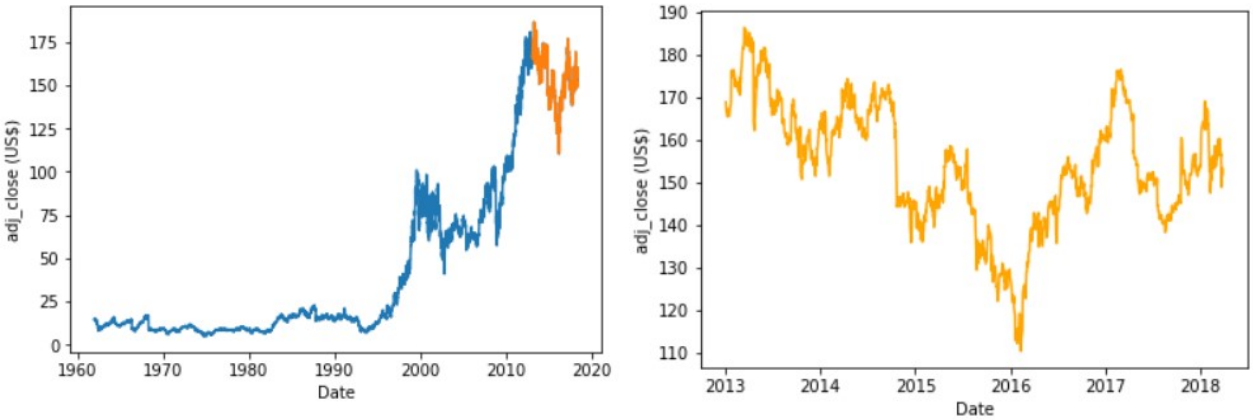
Figure 2: adj_close price value over time

The next section covers the data preprocessing step of the project development.

2 Data preprocessing

2.1 Considered range of time

The first decision was to limit the date considered for the project on the last five years. So old prices wouldn't influence our model too much and we would still have a considerable amount of data to work with. The Figure 3 shows the new graph of the evolution of adj_close value over the selected period.



(a) adj_close price value over time since the beginning

(b) adj_close price value over time from 2013 onwards

Figure 3: Evolution over time of adj_close values considered on the development

2.2 Features

There are several implementations online, such as [4], [5] and [6], for example, that use all of the variables described on Figure 1 as features for the model. However, on the understanding that most of those variable values become available at the same time that the adj_close value on a given trading day, it's not fair to use them as features for the predictor.

Instead, there are only two features that were used: the date itself and new one created: the moving average of window N of the last N values of adj_close before the trading day that we are expecting to predict.

For the date, we considering a continuous stream of values. So January 2^{nd} of 2013[2] is the day 0 and March 27^{th} of 2018 is the day 1910.

The second feature, the moving average of windows N of the last N values of adj_close, added the first hyperparameter for both regression models: N. Which had its value optimized on the cross validation process, described in Section 4 to be $\boldsymbol{N=5}$.

So, from now on, we could split our dataset on X (features dataset) and Y (labels dataset). The X dataset is composed by the two columns: days and average, and represents the input for the models. The Y has one column: the adj_close values for each day, and represents the output for the models.

Finally, both X and Y datasets were scaled so each of them have a same scale without distorting differences in the range of values. The scaler chosen remove the mean and scale to unit variance each of the columns of each dataset.

3 Algorithms used

3.1 Random Forest Regressor

This algorithm is based on the merging of multiples decisions trees and an ensemble technique called **bagging**.

A decision tree is an algorithm by itself that, by "asking questions" to the training data, mapes a series of decisions to be made given the input value. Those questions are usually on the form "Is this value greater/lesser than X?". Based on the answer, its created a new branch, until it forms a structure that recalls a tree. Figure 4 shows an example of a decision tree.

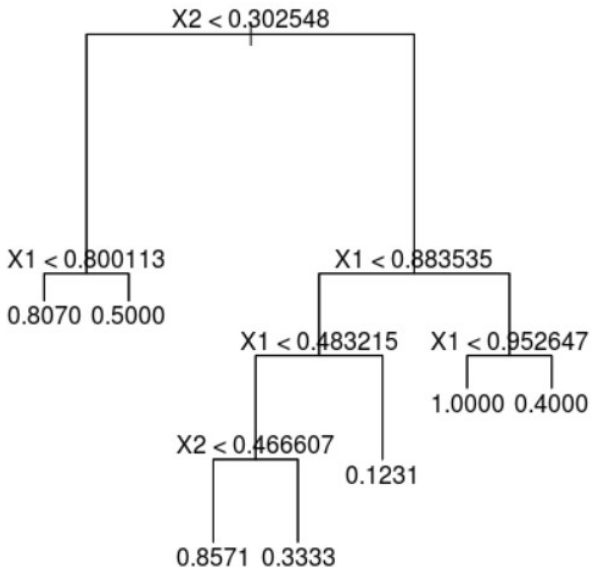


Figure 4: Example of decision tree

The training phase is the phase where it's gonna be defined the questions and values that will be made to create such branches. The step where we make the first question its called the root node and, when we reach a prediction, we reach the leaf node.

On a Random Forest Regressor algorithm, we have a series of decision trees, where each of them is trained with a subset of the training dataset. This technique is called **bagging**. Each subset has the same size and the objective is to bring more diversity to the whole algorithm based on differences on each decision tree, making the prediction more robust and the overfit problem less likely. Figure 5 illustrated the main idea behind the algorithm.

[2]Since January 1^{st} is a holiday, it's not considered as a valid trading day.

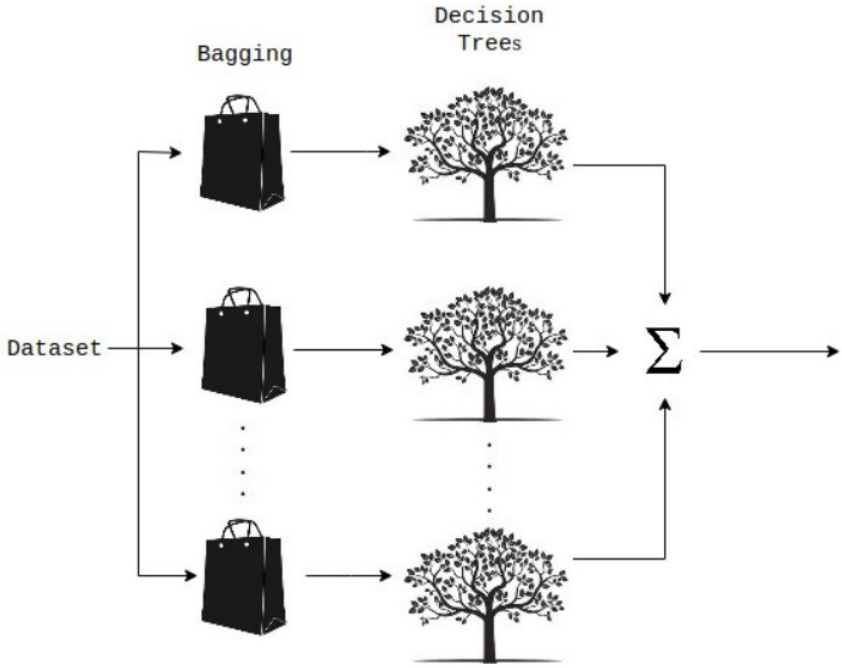


Figure 5: Main idea behind the Random Forest algorithm

3.2 Long Short-Term Memory Neural Networks

Long Short-Term Memory (LSTM) Neural Networks are a variant of Recurrent Neural Networks (RNN), which is a type of neural network designed to recognize patterns in sequences of data. They take time and sequence into account, which means they have a temporal dimension, sometimes referenced as "memory".

Different from a feedforward neural networks, that takes into account a single input to predict a single output, the RNN takes into account the history of predictions, so it has two sources of data: the new one and the recent past, through a link between its output and its input.

However, RNNs fail to recognize importance between outputs and events that occurred many time steps before, which could lead to an inappropriate prediction. Besides that, the memory characteristic also appears on the gradient calculation in reverse during back-propagation. So small gradients tend to get smaller and large gradients get larger as they move through the RNN, known as the vanishing and exploding gradients problems respectively.

Those problems are mitigated on the LSTM variant, on which there are complicated mechanisms in each unit, designed to make the memory more selective to past data that are really important and forget (dropout) useless data.

These mechanism are called gates, and they define how much of each data sample of that time step will "pass through", and are controlled with activation functions, such as sigmoid and hyperbolic tangent.

Each layer of a LSTM Neural Network are composed by a cell of units. Each unit has an update gate, a forget gate, and an output gate. The update and forget gates determine whether each element of the memory cell is updated. The output gate determines the amount of information to output as activations to the next unit of that cell. Figure 6 illustrates the gates on a single LSTM unit.

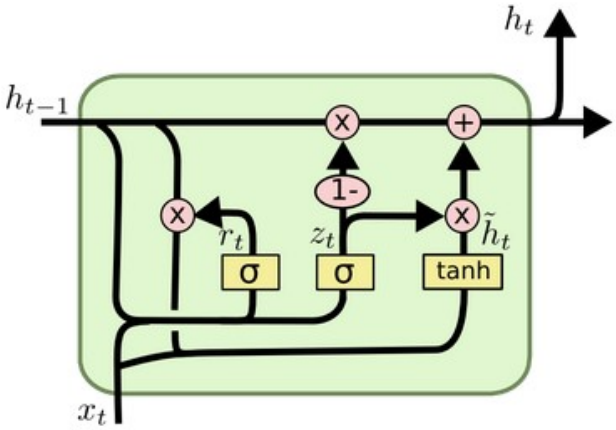


Figure 6: Representation of a single LSTM unit

4 Training pipeline

The original dataset was splitted on training (60%), validation (20%) and testing (20%) datasets. Figure 7 shows the original adj_close value over time separated on the three datasets.

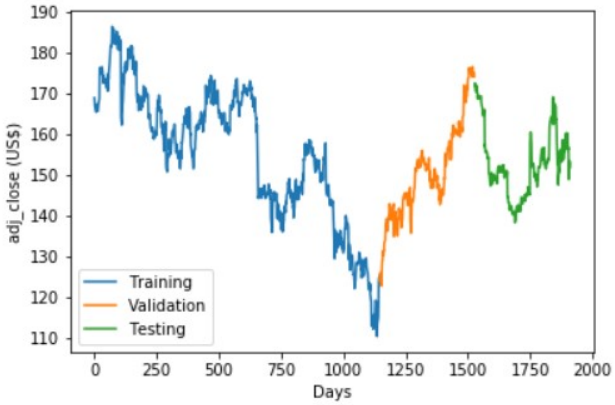


Figure 7: adj_close price value over time separated on train, validation and test datasets

The main error metric that was taken into account was the Mean Square Error (MSE), that calculates the average square difference between the predicted data and the expected value.

This metric is vastly used on regression problems. Others metrics also used for these problems are Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). However, compared to RMSE, MSE is easier to use and, compared to MAE, MSE is more sensitive to outliers. Since Stock Prediction hardly follows any standard, it's important that we take outliers into account.

For both models, the first step was to train the model using the training dataset and a set of hyperparameters initialized with values described on the next section.

After that, those hyperparameters were tuned using the validation dataset, to avoid overfitting. For each one them, the value that minimizes the MSE was chosen for the final model.

The final step was to test the model with the tuned hyperparameters on the testing dataset.

The next two sections shows which hyperparameters of each model were tuned on the validation step.

4.1 Random Forest Regressor

On this algorithm, there are only two hyperparameters to be optimized. The first one is the windows size N of the moving average, and the second one is the number of estimators on the Random Forest, *i.e.* the number of trees on the Forest. The list of values tested for each of the hyperparameters are described on the Equations

1 and 2:

$$N = \{5, 10, 20, 50, 100\} \tag{1}$$

$$num\_est = \{50, 100, 500, 1000\} \tag{2}$$

On Table 1, we have the comparison between the values of the hyperparameters used on training and the final values after tuning - the ones used for testing.

Table 1: Values of the hyperparameters before and after tuning

Hyperparameter	Value before tuning	Value after tuning
N	5	5
Number of estimators	1000	100

4.2 Long Short-Term Memory Neural Networks

On the LSTM algorithm, there were more hyperparameters to be optimized. Each one of them is described below:

- Windows size N of the moving average: already explained above. The values considered for the optimization are described by the Equation 3.

$$N = \{5, 10, 20, 50, 100\} \tag{3}$$

- Epochs of training: the number times that the learning algorithm will work through the entire training dataset. The values considered for the optimization are described by the Equation 4.

$$epochs = \{1, 10, 20, 30, 40, 50\} \tag{4}$$

- Batch size: represents the number of samples processed before the model is updated. The values considered for the optimization are described by the Equation 5.

$$batch\_size = \{8, 16, 32, 64, 128\} \tag{5}$$

- Number of units per cell: number of hidden units in each LSTM layer, which also described the output dimensionality of that layer. The values considered for the optimization are described by the Equation 6.

$$num\_units = \{10, 50, 64, 128\} \tag{6}$$

- Dropout probability: Probability at which outputs of the layer are dropped out. Where 1 means there are no dropouts, 0 means that all of the output layer is discarded. The values considered for the optimization are described by the Equation 7.

$$dropout\_prob = \{0.5, 0.6, 0.7, 0.8, 0.9, 1\} \tag{7}$$

- Optimization algorithm: algorithm used for the update of the model's parameters. The list considered for the optimization are described by the Equation 8.

$$optm\_list = \{adam, sgd, rmsprop, adagrad, adadelta, adamax, nadam\} \tag{8}$$

Since there are more hyperparameters to optimize, we separated this process on steps. First it was optmized the windows size N. After that, we used this value and experimented different values of *epochs* and *batch sizes*. The next step was to optimized the *number of units per cell* and the *dropout probability*. Finally, it was chosen the best optimization algorithm.

On Table 2, we have the comparison between the values of the hyperparameters used on training and the final values after tuning - the ones used for testing.

Table 2: Values of the hyperparameters before and after tuning

Hyperparameter	Value before tuning	Value after tuning
N	5	5
Epochs	1	30
Batch Size	1	64
Number of units	50	50
Dropout probability	0.5	0.7
Optimization	adam	nadam

5 Final results

To avoid outliers executions, all the values here described of MSE are the median of three executions with the same dataset and hyperparameters.

5.1 Random Forest Regressor

On the validation dataset, the change on the MSE after the optimization is shown on the Table 3.

Table 3: Values of the MSE on the validation dataset before and after tuning

Metric	Value before tuning	Value after tuning
MSE	0.064	0.057

On the testing dataset, the final MSE obtained was:

$$\text{MSE} = 0.06$$

Plotting the original testing data and the prediction on the same graph, we could get a visual result, shown on Figure 8.

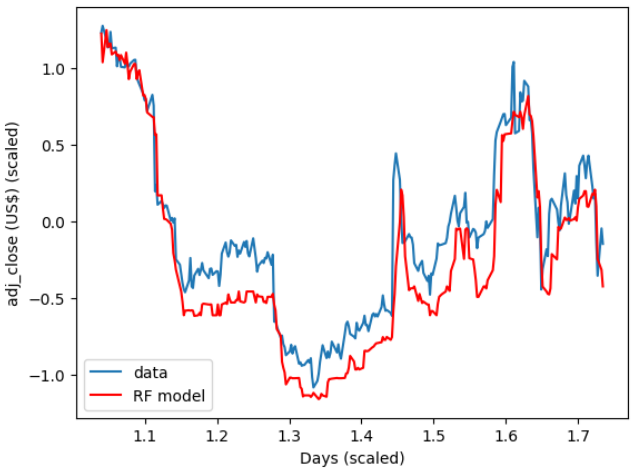


Figure 8: Comparison between original testing data and the prediction made with the Random Forest Algorithm tuned model

5.2 Long Short-Term Memory Neural Networks

On the validation dataset, the change on the MSE after the optimization is shown on the Table 4.

Table 4: Values of the MSE on the validation dataset before and after tuning

Metric	Value before tuning	Value after tuning
MSE	0.087	0.044

On the testing dataset, the final MSE obtained was:

$$\text{MSE} = 0.042$$

Plotting the original testing data and the prediction on the same graph, we could get a visual result, shown on Figure 9.

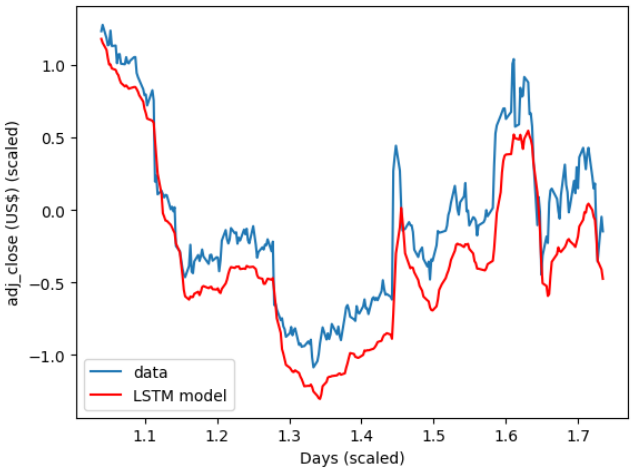
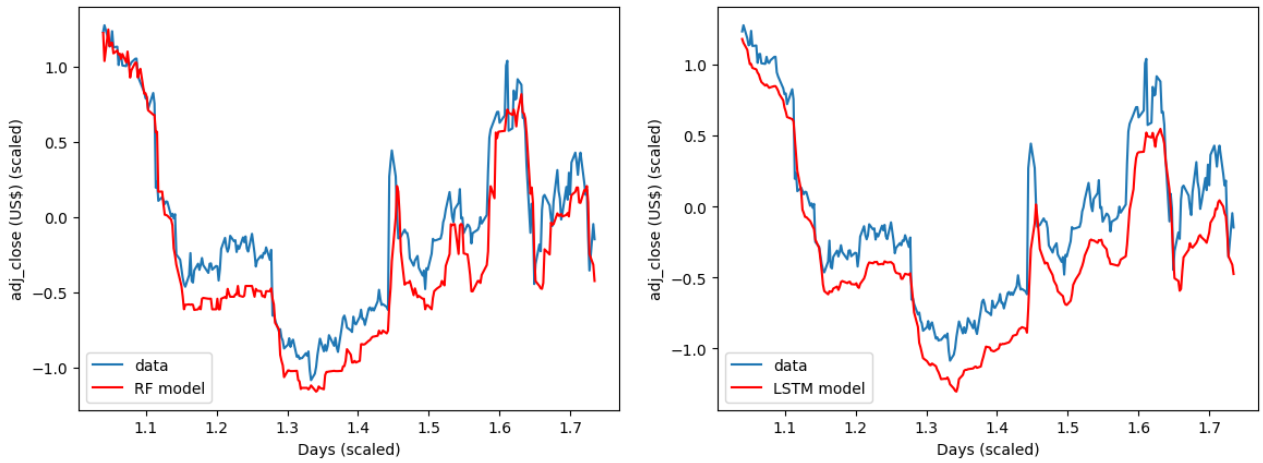


Figure 9: Comparison between original testing data and the prediction made with the LSTM Algorithm tuned model

5.3 Final comments

Comparing both results side by side, we can see that the Random Forest algorithm, even considered as a simpler one, got the better results:



(a) Random Forest (b) LSTM

Figure 10: Comparison between the final results of both models

Beyond the MSE metric, we could also point out the visual accuracy of the prediction made by the Random Forest model: the prediction follows the original data in several points. This is shown specially on the X range of 1.05 to 1.15.

However, regardless of the model, one problem common to both of them is that they do not predict well when a stock price rises after some time falling, as can bee seen on several ranges of X, for example, 1.15 to 1.3 or 1.3 to 1.45.

This issue could be sorted out by creating a new feature that takes this into account. The proposition for this new feature, however, requires a more deep analysis of the data prior to implementation.

Besides training the model with a larger dataset, the LSTM algorithm could receive future improvements and other tests could be made, for example, different architectures for the neural network than the one used on this project. This effort is justified since there are several stock price predictors that explore such technique, for example [4], [5] and [13].

References

[1] Quandl: Financial, economic and alternative data. https://www.quandl.com/.

[2] Investopedia: Sharper insight, better investing. https://www.investopedia.com/.

[3] Bea Bischoff. Adjusted closing price vs. closing price. https://budgeting.thenest.com/adjusted-closing-price-vs-closing-price-32457.html, may 2019.

[4] Asutosh Nayak. Predicting stock price with lstm. https://towardsdatascience.com/predicting-stock-price-with-lstm-13af86a74944, mar 2018.

[5] Rohith Gandhi. Predicting stock prices — comparison of different algorithms. https://medium.com/datadriveninvestor/predicting-stock-prices-comparison-of-different-algorithms-ab2b8fd42514, may 2018.

[6] Yibin Ng. Machine learning techniques applied to stock price prediction. https://towardsdatascience.com/machine-learning-techniques-applied-to-stock-price-prediction-6c1994da8001, jan 2019.

[7] Georgios Drakos. Decision tree regressor explained in depth. https://gdcoder.com/decision-tree-regressor-explained-in-depth/, may 2019.

[8] Georgios Drakos. Random forest regression model explained in depth. https://gdcoder.com/random-forest-regressor-explained-in-depth/, jun 2019.

[9] Krishni Hewa. A beginners guide to random forest regression. https://medium.com/datadriveninvestor/random-forest-regression-9871bc9a25eb, nov 2018.

[10] 3.2.4.3.2. sklearn.ensemble.randomforestregressor. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[11] Christopher Olah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, aug 2015.

[12] Skymind. A beginner's guide to lstms and recurrent neural networks. https://skymind.ai/wiki/lstm.

[13] Leonardo dos Santos Pinheiro and Mark Dras. Stock market prediction with deep learning:a character-based neural language model for event-basedtrading. *Proceedings of Australasian Language Technology Association Workshop*, 2017.