# 3
# The Forecasting Task

The forecasting task performed by the predictors aims to forecast the current day minimum and maximum values. These values differ according to the kind of trade considered. They are the stock prices themselves if we intend to simulate the investor's decision making regarding buy and sell trades. On the other hand, if the focus is to guide the investor into Pairs Trading, they refer to the spread values between the considered asset pair.

In our stock market time series forecasting, we propose two different forecasting schemes. One uses only interday data, while the other one uses both interday and intraday features. We test both schemes by using all the proposed predictors.

To better explain the forecasting process, we subdivide this chapter into three essential parts. In section 3.1, we describe how the available dataset is preprocessed before the forecasting itself. The section 3.2 explains in detail the two forecasting schemes. And finally, in the section 3.3, we present the proposed predictor algorithms and benchmarks, exposing the peculiarities of each.



## 3.1
## Dataset Standardization

In order to soften undesirable irregularities in the dataset and keep its values between 0.0 and 1.0, the dataset is preprocessed using two different standardization methods consecutively.

First we use the Standard Score method [32], calculated by the equation:

$$z_i = \frac{(x_i - \mu_i)}{\sigma_i}, \tag{3.1}$$

where $x_i$ represents the i-th sample vector of a dataset $(X, Y)$, $\mu_i$ is the mean of $x_i$ and $\sigma_i$ its standard deviation.

Then, the values are normalized using the Min-Max algorithm [33]. However, the values to be predicted may transcend the limits of minimum and max-

imum shown in the input data. Hence, to ensure that all values are well represented by the predictors, we use the Min-Max algorithm by scaling the values to the range 0.20 to 0.80. We choose these limits since a daily variation, either in stock prices or in indicators, above 25% is very unusual. The following equation shows how the Min-Max standardization is performed:

$$y_i = \frac{z_i - min_i}{max_i - min_i}(cmax - cmin) + cmin, \tag{3.2}$$

where the values will be in the range $[cmin = 0.2,\ cmax = 0.8]$, and $min_i$ and $max_i$ are respectively the minimum and maximum values of the i-th sample vector $z_i$, after the Standard Score standardization.

Obviously, once the predictor has completed the forecasting, the output data require post-processing by performing the inverse operations.

## 3.2 Forecasting Schemes

The proposed stock market time series forecasting explores two different forecasting schemes. The simplest one, here called interday forecasting, uses just interday data. It takes advantage of only previous day values and the current day opening value. The second scheme, on the other hand, also uses values obtained during the current day. It carries out several predictions throughout the same day. Hence, it is called intraday forecasting. More precisely, every hour a new prediction is performed, which also includes the input data minimum and maximum values already known in the current day up to that instant in time.

Since the Brazilian stock exchange hours of operation are officially from 10am to 17pm, we build seven predictions throughout the day in the intraday forecasting. In the absence of more data referring to the current day beyond the opening value, we carry out the first prediction exactly as the interday scheme does. However, in the rest of the predictions, the current day opening value in the input data is replaced by the minimum or the maximum value already reached, so that the forecasting quality improves during the day. We choose intervals of one hour between the predictions to allow the user to use the system even in the absence of a Algorithmic Trading [34].

## 3.3 Forecasting Algorithms

This section describes in detail the concerned forecasting algorithms. In subsection 3.3.1, we present the proposed predictor algorithms, by explaining how the training is conducted in each. And in subsections 3.3.2 and 3.3.3, we present respectively the selected benchmarks and the Oracle used to simulate the optimal solution.

### 3.3.1 The Proposed Predictors

The goal of many problems in Machine Learning area is modeling the complex relationship present in a system between input variables and their respective outputs, overcoming the leak of a theoric model [35].

There are respected models for regression that comprise linear combinations of fixed basis functions, such as Bayesian Regression models. Although these models have useful analytical and computational properties, their practical applicability is limited by the curse of dimensionality. Thus, in order to apply such models to large-scale problems, it is necessary to adapt the basis functions to the data [36].

The proposed predictors, described in detail below, are some sucessfull methods to solve this kind of problem.

SVR and ANN are non-linear methods that have emerged as solutions to overcome the linear methods limitation.

SVR address this by first defining basis functions that are centered on the training data points and then selecting a subset of these during training [36].

ANN presents an alternative approach, by using parametric forms for the basis functions in which the parameter values are adapted during training. According to [36], Feed-forward Neural Network are the most successful model in the context of pattern recognition.

Finally, being a linear technique, PLSR is a less powerful regression method. It holds up in an attempt to overcome the classical methods by performing the regression incrementally only in a set of input variables. PLSR works by extracting a latent structure which uses orthogonal factors as linear combinations of the input features [7].

### Partial Least Squares Regression

Developed by Wold et al. [37, 38], Partial Least Squares Regression is a linear method proposed to solve regression problems whose dataset presents a big number of features in relation to the number of samples. Compared to Ordinary Least Squares Regression (OLSR), a form of classical linear regression which uses all the input variables simultanely, PLSR reaches more robust results once it performs the regression incrementally only on a set of input variables

PLSR usage has increased in the last three decades, receiving special attention in symposiums specialized in regression algorithms [39, 40]. The success obtained by the use of PLSR in chemometrics, its initial application area [41, 42], encourages its application in other areas such as, process check, marketing, image processing and stock market [43].

Among the most notable advantages of using PLSR, we must mention its good performance when applied to features that are highly correlated, the easy understanding of the implicit model and the fast processing.

Here, we use PLSR based on the work of [35], as an attempt to also get good predictions for stock market time series with a linear model.

Suppose we have a mean centered dataset $[X, Y]$ where $X$ are the independent variables and $Y$ the dependent variables. PLSR extracts a latent structure using orthogonal factors as linear combinations of features. Instead of working in the feature space, by using the covariance matrix given by:

$$XX^T = Cov(X, X) = Var(X), \tag{3.3}$$

like Principal Component Analysis (PCA), PLSR adds dependent variables and uses the modified matrix given by:

$$XY^TY^TX = Cov(X, Y). \tag{3.4}$$

The key idea is to find a feature space that better describes both dependent and independent variables, to predict the dependent variables. The overall PLSR process can be described in algorithm 1.

And once we have learned the PLSR model, we can fit to a new dataset $X^{'}$ for predicting the $Y^{'}$ responses. We can do this by using the variables $w$, $b$ and $p$ learned from the model, as shown by the algorithm 2.

**Algorithm 1** PLSR Training Algorithm

$X_1 \leftarrow X$
$Y_1 \leftarrow Y$
**for** $i$=1 to $k$ **do**
  $w_i \leftarrow X_i^T Y$
  // normalization
  $w_i \leftarrow w_i/(w_i w_i^T)^{1/2}$
  $t_i \leftarrow X_i w_i$
  // computing the regression coefficients
  $b_i \leftarrow Y_i^T t_i/t_i^T t_i$
  $p_i \leftarrow X_i^T t_i/t_i^T t_i$
  // residual processing
  $X_{i+1} \leftarrow X_i - t_i p_i^T$
  $Y_{i+1} \leftarrow Y_i - b_i t_i$
**end for**

**Algorithm 2** PLSR Testing Algorithm

$X_1^{'} \leftarrow X^{'}$
$Y^{'} \leftarrow 0$
**for** $i$=1 to $h$ **do**
  $t_i^{'} \leftarrow X_i^{'} w_i$
  // residual prediction
  $Y^{'} \leftarrow Y^{'} + t_i^{'} b_i^T$
  // residual processing
  $X_{i+1}^{'} \leftarrow X_i^{'} + t_i^{'} p_i^T$
**end for**



## Support Vector Regression

Used in both classification and regression problems, the Support Vector (SV) algorithm is a nonlinear generalization of the Generalized Portrait (GP) algorithm developed in Russia in the sixties by [44, 45].

In its current form, SVM was largely developed at AT&T Bell Laboratories at 90 decade by Vapnik and co-workers within the statistical learning theory and the structural risk minimization [46, 47, 48, 49, 50, 51]. Due to this industrial context, SVM presented a sound orientation towards real-world applications since its conception [52]. Their theory was quickly proven to be very successfully on many classification and regression applications, such as Optical Character Recognition and Object Recognition tasks [50, 53, 54, 55], its first two foci, and time series prediction applications [56, 57, 58, 59].

The SV learning consists of using a kernel representation of the data and then formulating the problem as a convex optimization problem [7]. Usually the convex optimization problem can be modeled using a quadratic programming technique, for which the dual problem is solved.

According to Bishop [36], the main advantage of SV algorithm is that, although the training involves nonlinear optimization, the objective function is convex, and so the solution of the optimization problem is relatively straightforward. High generalization and good performance for high dimension space of features are other advantages of using this technique that we must mention. Here, we use the LIBSVM implementation published by Chang and Lin [60].

Suppose we have a training dataset consisting of $\{(x_1, y_1), ..., (x_n, y_n)\} \subset \chi \times \mathbb{R}$, where $\chi$ denotes the space of the input patterns in a higher dimension of ($\chi = \mathbb{R}^d$). The SVR proposed here, also called $\varepsilon-$SVR [61], aims to find a function $f(x)$ that has at most $\varepsilon$ deviation from the target $y_i$ for all the training data, and at the same time is as flat as possible. Thus, errors that are less than $\varepsilon$ deviations are not penalized.

Now, for an easier understanding, suppose we have a linear function $f(x)$, taking the form:

$$f(x) = \langle w, x \rangle + b, \text{ with } w \in \chi,\ b \in \mathbb{R}, \tag{3.5}$$

where $\langle ., . \rangle$ denotes the dot product in $\chi$. Flatness here means that one seeks a small $w$. We ensure this by minimizing the norm $\| w \|^2 = \langle w, w \rangle$. And so, for approximating $f(x)$ with an $\varepsilon-$precision, we formulate this problem as the following convex optimization model:

$$\text{Minimize } \frac{1}{2} \| w \|^2 \tag{3.6}$$

$$\text{Subject to } \begin{cases} y_i - \langle w, x_i \rangle - b & \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i & \leq \varepsilon. \end{cases}$$

This model is correct and achieves the proposed objective. However, it brings a tacit assumption that the convex optimization problem is feasible, that is, it assumes that such a function $f(x)$ actually exists that approximates all pairs $(x_i, y_i)$ with $\varepsilon-$precision. Sometimes, however, this may not be the case [52], or we also may want to allow for some errors that can be irrelevant to the problem.

In order to solve this problem, analogously to the "soft margin" loss function [62] adapted to SVM by Cortes and Vapnik [48], we introduce the slack

variables $\xi_i$ and $\xi_i^*$ to relax the optimization problem constraints. Hence we arrive at the formulation 3.7, published by [61].

$$\text{Minimize} \quad \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*) \tag{3.7}$$

$$\text{Subject to} \begin{cases} y_i - \langle w, x \rangle - b & \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i & \leq \varepsilon + \xi_i^* \\ \xi, \xi_i^* & \geq 0, \end{cases}$$

where the constant $C > 0$ determines the trade-off between the flatness of $f(x)$ and the amount up to which deviations larger than $\varepsilon$ are tolerated [52]. This corresponds to dealing with a so called $\varepsilon$-insensitive loss function $|\xi|_\varepsilon$ described by:

$$|\xi|_\varepsilon := \begin{cases} 0 & \text{if } |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & \text{otherwise.} \end{cases}$$

As long as the dimensionality of $w$ is much higher than the number of observations, the optimization problem formulated in the model 3.7 can be solved more easily in its dual formulation [63, 64]. This is exactly the situation here. Moreover, the dual formulation provides the key for extending SV algorithm to nonlinear functions. As described in [65], we use the standard dualization method utilizing Lagrange Multipliers and after some simplifications, we get the following formulation:

$$\text{Maximize} \begin{cases} -\frac{1}{2} \sum_{i,j=1}^{n} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle x_i, x_j \rangle \\ \\ -\varepsilon \sum_{i=1}^{n} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i (\alpha_i - \alpha_i^*) \end{cases}$$

$$\text{Subject to} \begin{cases} \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) & = 0 \\ \alpha_i, \alpha_i^* & \in [0, C]. \end{cases}$$

From this formulation, to make the SV algorithm nonlinear, we could simply preprocess the training patterns $x_i$ by a map $\Phi : \chi \rightarrow \mathcal{F}$ into some feature space $\mathcal{F}$, as described in [66, 67] and then apply the standard SV algorithm. However, this approach can easily become computationally infeasible for both polynomial features of higher order and higher dimensionality [52].

We show that the SV algorithm only depends on dot products between

patterns $x_i$. Hence, in order to find a computationally cheaper model, we use the implicit mapping via kernels proposed by [68] to solve non-linear problems, where we take advantage of $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$ rather than computing $\Phi(x)$ explicitly. This kernel mapping allows us to restate the SV optimization problem as

$$\text{Maximize} \begin{cases} -\frac{1}{2} \sum_{i,j=1}^{n} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K(x_i, x_j) \\ \\ -\varepsilon \sum_{i=1}^{n} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i (\alpha_i - \alpha_i^*) \end{cases}$$

$$\text{Subject to} \begin{cases} \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) & = 0 \\ \alpha_i, \alpha_i^* & \in [0, C]. \end{cases}$$

For a more detailed formulation, we refer the reader to [52, 69, 70, 71].



### Artificial Neural Networks

Inspired by the structure and functional aspects of biological neurons, ANN is a non-linear learning method that easily deals with irregularities [13, 25], and uncertain, incomplete or insufficient data [19], by using a connectionist approach to computation.

Recognized as being the designers of the first ANN, McCulloch and Pitts [72] demonstrate that combining many simple processing units together could lead to an overall increase in computational power.

We use a three layers feed-forward ANN, trained with the Backpropagation algorithm, to predict either the minimum or the maximum current day value. The neurons' values depend on which value we hope to predict.

For each neuron, we use the logistic activation function. That is why we care about adjusting the dataset, by scaling its values to the range 0.0 to 1.0. The formula for the logistic function is:

$$f(x) = \frac{1}{(1 - e^{-x/c})}, \tag{3.8}$$

where $c$ term is used to alter the shape of the function, either stretching or compressing the function along the horizontal axis. The figure 3.1 illustrates the S-shape of the logistic activation function.

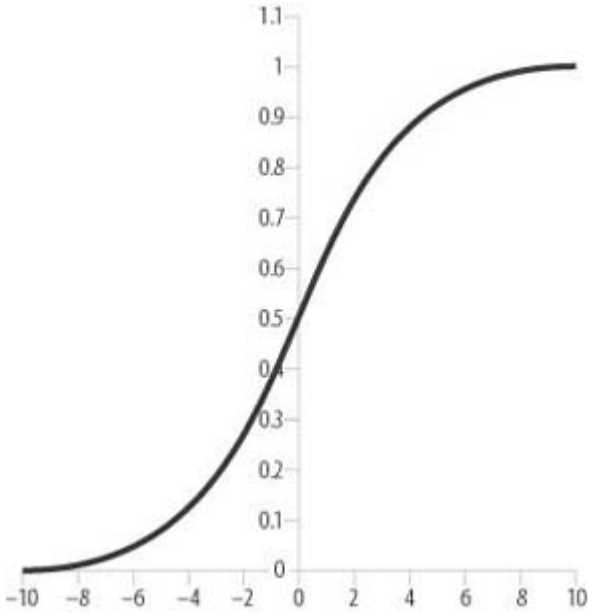


Figure 3.1: Logistic Activation Function



Every connection from one neuron to another has an associated weight. Weights in an ANN are analogous to the synaptic connection in a biological neural network. The weights affect the strength of a given input and can be either inhibitory or excitatory. It is the weights that truly define the behavior of an ANN. Further, the task of determining the value of these weights is the subject of training or evolving an ANN [73].

The input to a neuron is then the sum of the products of each input's weight connecting that neuron multiplied by its input value plus a bias term. Bias terms shifts the net input along the horizontal axis of the activation function, which effectively changes the threshold at which a neuron activates. The net result is called the net input to a neuron. The equation 3.9 shows how the net input to a given neuron $j$ is calculated from a set of input neurons $\{x_1, ..., x_n\} \subset \mathbb{R}$.

$$n_j = \sum_{i=1}^{n} n_i w_{ij} + b_j w_j, \tag{3.9}$$

where $n_i w_{ij}$ is the product of the source neuron value and the weight of the connection that binds the source neuron to that one whose value is being calculated, $b_j = \{0, 1\}$ is the bias value and $w_j$ is the bias weight. While in our ANN the bias value is always 1, the bias weights are adjusted through training, just like all other weights. This essentially allows the neural network to learn the appropriate thresholds for each neuron's activation [73].

To train an ANN, we feed it a set of inputs, which generates some output.

To compare this calculated output to the desired output for a given set of inputs, we need to calculate the error. This enables us to not only determine whether the calculated output is right or wrong, but also to determine the degree to which it is right or wrong.

Here, we use is the Mean Square Error (MSE), which is the average of the squared difference between the calculated and the desired output. The equation 3.10 shows how the MSE is calculated for the training set.

$$\varepsilon = \frac{1}{n^o} \sum_{i=1}^{n^o} (cv_i^o - dv_i^o)^2, \tag{3.10}$$

where $cv_i^o$ and $dv_i^o$ are respectively the calculated and desired output values, corresponding to the i-th of the $n^o$ output neurons present in each observation.

The training process aims to get this error value as small as possible by iteratively adjusting the weight values which connects all ANN neurons [73]. To calculate the weights adjustment, each iteration requires that we also calculate the error associated to each neuron in the output and hidden layers. We calculate the error $\delta_i^o$ for the i-th output neuron as follows:

$$\delta_i^o = (cv_i^o - dv_i^o)\ f'(cv_i^o), \tag{3.11}$$

where $(cv_i^o - dv_i^o)$ is the difference between the calculated and desired values for the i-th output neuron and $f'(cv_i^o)$ is the derivative of the activation function for its calculated value. Thus, we can rewrite this equation replacing the derivative of the logistic function to get this final equation for the output neuron error calculation:

$$\delta_i^o = (cv_i^o - dv_i^o)\ cv_i^o(1 - cv_i^o). \tag{3.12}$$

For hidden-layer neurons, the error equation is somewhat different. It is a function of the error associated with each output-layer neuron to which the hidden neuron connects multiplied by the weight for each connection. This means that to calculate the error and, subsequently, to adjust weight, you need to work backwards from the output layer toward the input layer. We calculate the error $\delta_i^h$ for the i-th hidden neuron as follows:

$$\delta_i^h = \sum_{j=1}^{n^o} \delta_j^o w_{ij}\ f'(cv_i^h), \tag{3.13}$$

where $\delta_j^o$ is the error for the j-th output neuron and $w_{ij}$ is the weight of its connection with the i-th hidden neuron, whose calculated value appears in the derivative $f'(cv_i^h)$. Again, calculating the derivative of the activation function we get the final equation 3.14.

$$\delta_i^h = \sum_{j=1}^{n^o} \delta_j^o w_{ij}\ cv_i^h(1 - cv_i^h) \tag{3.14}$$

Since the input layer neurons values are given, obviously, there are no errors associated with them.

Once we have calculated the errors for the output and hidden layers neurons, we can proceed to calculate suitable adjustments for each weight in the network. The equation 3.15 shows the adjustment applied to each weight.

$$\Delta w = \rho \delta_i cv_i + \alpha(\Delta w'), \tag{3.15}$$

where $\rho$ is the learning rate, $\delta_i$ is the error associated with the considered neurons and $cv_i$ its value. $\Delta w'$ is the weight adjustment calculated to the previous iteration and $\alpha$ is the momentum factor. The new weight is simply the old weight plus $\Delta w$.

In the ANN training, we set the learning rate to 0.01. In order to increase its convergence speed and reduce the risk of instability, we set the momentum factor to 0.5 [73]. The training is limited to 2000000 iterations. All these parameters are defined empirically.



### 3.3.2 The Benchmarks

To better understand our system's potential, we compare its predictions to four benchmarks described in the following sub-subsections, besides the Oracle presented in the next subsection.

Both forecasting schemes are implemented identically when using either our predictors or benchmarks. Hence, in the intraday forecasting, the benchmarks also use either the minimum or the maximum value already achieved during the current day as features, depending of which bound value is being predicted.

#### Carbon Copy Strategy

Carbon Copy Strategy (CCS) is a Naive Bayes predictor, the simplest among the benchmarks considered here. It estimates the minimum and maximum values for the current period by simply copying the previous period values, or in other words, he believes that the value to be predicted for the current period is equal to that desired in the last period.

#### Simple Moving Average

Proposed by O'Connor and Madden [27], Simple Moving Average (SMA) predictors are three more elaborate benchmarks. Relying on the Technical Analysis teachings, they predict the minimum and maximum values for the current period using the SMA of these values over the previous 5, 10 and 15 periods.

### 3.3.3 Oracle

Finally, Oracle is the name given here to the optimal solution for the forecasting task. It is a kind of crystal ball simulating a predictor that accurately provides all the desired values.

Through Oracle predictions, we can estimate the potential profit of a more robust predictor and, moreover, analyze how far away our predictors are from the optimal solution.