

University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Damian Dąbrowski

Student no. 439954

Heorhii Lopatin

Student no. 456366

Ivan Gechu

Student no. 439665

Krzysztof Szostek

Student no. 440011

Large language models for forecasting market behaviour

Bachelor's thesis
in COMPUTER SCIENCE

Supervisor:
dr Piotr Hofman
Instytut Informatyki

Warsaw, May 2024

Abstract

This thesis concerns research into the use of machine learning and large language models in market analysis, focusing on market predictions.

Keywords

machine learning, large language models, time series forecasting, market prices

Thesis domain (Socrates-Erasmus subject area codes)

11.4 Sztuczna inteligencja

Subject classification

D. Software

D.127. Blabalgorithms

D.127.6. Numerical blabalysis

Tytuł pracy w języku polskim

Duże modele językowe w przewidywaniu giełdy

Contents

1. Introduction	5
1.1. Overview	5
1.2. Contributions	5
1.3. Outline	5
2. Related work	7
3. Other models	9
3.1. Random forest	9
3.1.1. results	9
3.2. Logistic regression	10
3.2.1. Results	11
3.3. Support vector machine	11
3.3.1. Parameters	11
3.3.2. Results	12
3.4. Multilayer Perceptron	12
3.4.1. Structure of an MLP	12
3.4.2. Forward Propagation	13
3.4.3. Backpropagation and Training	14
3.4.4. Results	14
3.5. Convolutional neural network	14
3.5.1. Architecture of Convolutional Neural Networks	15
3.5.2. Results	15
3.6. Residual neural network	16
3.6.1. ResNet Architecture	16
3.6.2. Results	16
4. Methodology	17
5. Main results	19
6. Forecasting applications	21
7. Conclusion	23
A. Visualisation	25

Chapter 1

Introduction

1.1. Overview

In the world of stock markets a major problem is the apparent incalculability of the complex network of factors e.g. how stock prices of one company affect those of another. As the environment of stock markets becomes more and more complex, the ability to analyse and confidently predict its future becomes of crucial importance for traders, investors and researchers.

With the recent advent of generative AI and the demonstrable power of Large Language Models a question arises of how these can be used to accurately analyse and predict time series market prices in different environments. Therefore, this thesis presents our work on the subject.

1.2. Contributions

1.3. Outline

First, we look at what work has already been done in the field of LLM time series prediction, in particular what techniques of fine-tuning and input data transformation were used. Then we look at how different, smaller machine learning models deal with time series prediction.

We describe the datasets we used for testing small models and LLMs.

Subsequently, we discuss our own methodology; different applied methods and techniques of input reprogramming, use of prompts and context, and LLM fine-tuning. Next, we present the results we have achieved on the chosen datasets (and compare them to some other known solutions).

Finally, we speculate on the significance of our work, its potential applications in forecasting price time-series.

Chapter 2

Related work

Chapter 3

Other models

Here we present our results from trying to use the following models to extrapolate a time series. Classification models output in binary categories: increase or decrease in value, and are therefore less precise.

3.1. Random forest

A random forest is a machine learning model for classification and regression tasks introduced by Leo Breiman in 2001. A random forest is an ensemble of individual tree predictors, each of which depends on a random vector, chosen independently and with the same distribution for all trees. The results from individual trees are then aggregated into the overall result of the model - for classification tasks it is the mode of individual classifications and for prediction tasks it is the mean average of individual predictions.

The error of forest prediction converges as such as the quantity of trees in the forest increases. The error of forest prediction depends negatively on the accuracy of individual trees and correlation between them.

The `num_lags` parameter is the width of data taken for individual predictions. The larger the `num_lags` the more of past data the model takes into account.

The `n_estimators` parameter (number of estimators) describes the number of trees grown in the forest. Increasing the number of trees increases the accuracy of the model, but it also increases the computational cost.

The `max_features` parameter (number of features) specifies the the number of features of the data considered for a split at each node while growing an individual tree. A higher value of `max_features` may capture more information about the data at the ris of overfitting and decreased randomness of the model.

The `criterion` parameter describes the function used by the model to calculate a quality of a split at a given node

3.1.1. Results

The model was trained and tested on the simple property sales dataset. The model was trained to classify whether the next price will be higher or lower than the previous one, based on the sequence of prices spanning the last `num_lags` days. The combinations of following parameters were tried

- `num_lags`: [1, 5, 10, 13, 25, 40, 50]

- `n_estimators`: [20, 50, 100]
- `max_features`: [2, 4, 8]
- `criterion`: ["gini", "entropy", "log_loss"]

The accuracy of the models ranged from 0.749 to 0.827. The main influence seems to be the lag number - the optimal being around 10. Then slightly better were those models with `max_features` equal to 4, and number of trees greater or equal to 50. The criterion didn't seem to play a significant role. See figure 3.1 below.

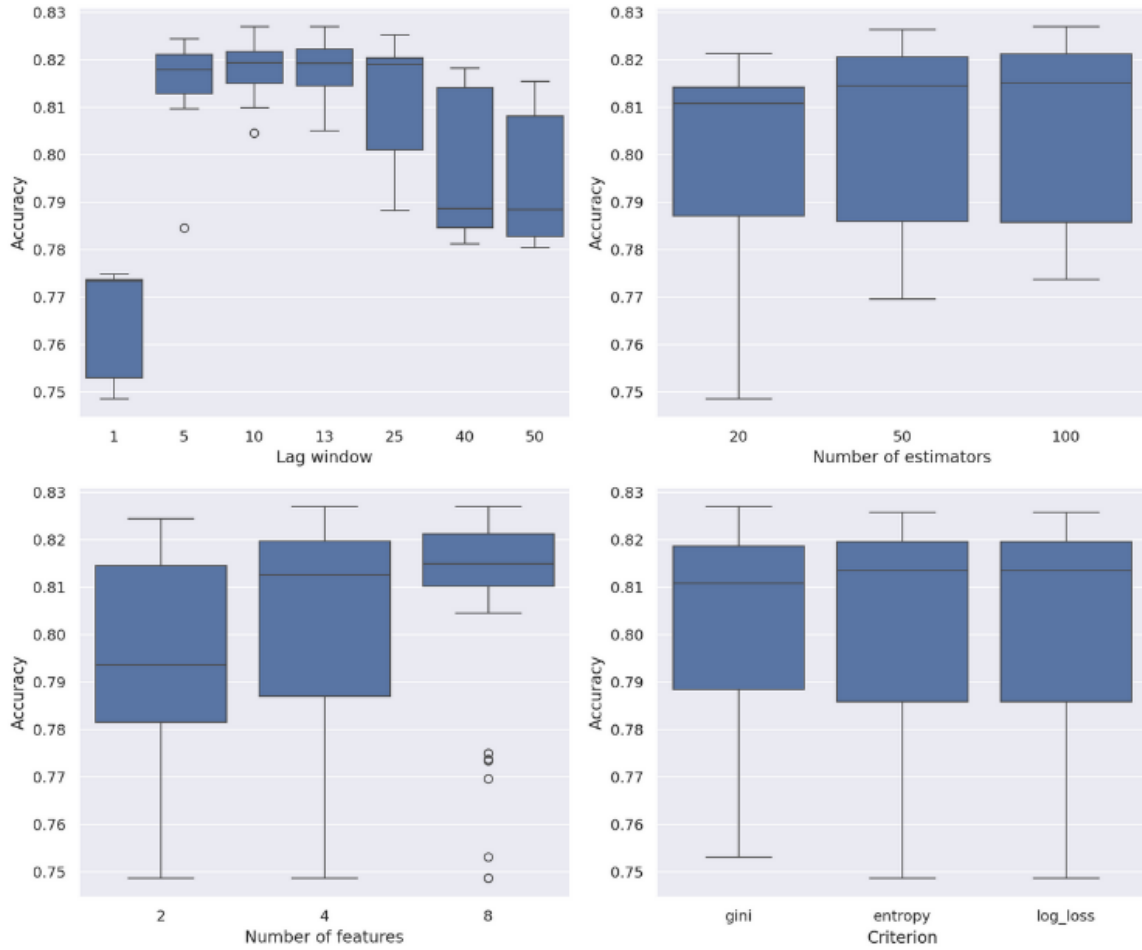


Figure 3.1: Results of random forest experiment.

3.2. Logistic regression

Logistic regression is a statistical model used for binary classification. It calculates the probability of whether a datapoint classifies to one class. During training of the model a sigmoid function of features of data is calculated that best fits the provided training dataset.

More precisely, the model calculates the function $Y(X) = \frac{1}{1+e^{-z}}$, where

- $z = B_0 + B_1 \cdot X_1 + \dots + B_n \cdot X_n$,

- X_1, \dots, X_n are features of data X ,
- B_0, B_1, \dots, B_n are parameters of the model.

Function Y assumes values only in the range $(0, 1)$. If $Y(X) \geq 0.5$, the model classifies the datapoint as 1 (in our model below - the price will be higher). If the converse is true, the datapoint is classified as 0 (the price will be lower).

During training the parameters B_0, B_1, \dots, B_n are chosen using Maximum Likelihood Estimation function, so that the results of the Y function best fit the training dataset.

3.2.1. Results

Two models have been trained. The first one is trained to predict if the next price in the dataset will be higher than the previous one, provided with a sequence of prices spanning over the last `num_lags` days. The second one does the same but operates on monthly averages instead of single records (for this, the dataset was averaged over subsequent months). The first model proved to be more precise with an accuracy of 82% for an optimal value of k of around 40 (Fig 3.2a). At the same time the second model only scored 75% (Fig 3.2b).

However, simply predicting whether the price will be higher or lower is much easier than predicting the actual next price. Additionally, logistic regression assumes an independence of datapoints from each other, wherefore it performs poorly for time series datasets. Therefore this model is insufficient for our purpose.

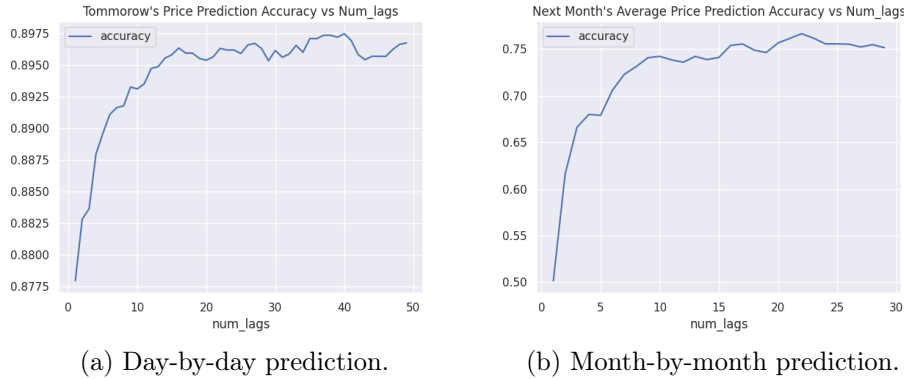


Figure 3.2: Plots of two models of logistic regression.

3.3. Support vector machine

Support vector machines are a widely popular model for machine learning classification problems, due to their high generalization abilities and applicability to high-dimensional data.

The model construes datapoints as high-dimensional vectors and finds the best hyperplane that divides the two classes the data is classified into. The goal of training is to find the hyperplane with the greatest margin - that is, the greatest distance from the closest vectors, called the support vectors.

3.3.1. Parameters

- **Kernel:** The model uses a kernel function to transform the space of data points which are not separable by a hyperplane, into one where they are separable.

- **C value:** The C value specifies how accurate the model should be, that is how much it should avoid misclassifications, versus how wide the margins should be. A lower value of C corresponds to wider margins.
- **Gamma:** The gamma value specifies how much influence individual training datapoints should have. The `scale` value of gamma means that gamma is scaled automatically to fit the size of the dataset.

3.3.2. Results

The model was tested using two metrics: Mean Squared Error and R-squared.

For the support vector machine there were overall nine models tried:

- three kernels: `rbf`, `poly`, `sigmoid`
- one gamma: `scale`
- three C values: 0.1, 1.0, 10.0

The ‘poly’ kernel worked much better than both ‘rbf’ and ‘sigmoid’, which both worked equally badly. Overall, though, the statistics for every model were terrible. The value of ‘ C ’ has had very little impact and only on ‘rbf’ kernel.

Support Vector Machine model doesn’t perform well in this task, as can be seen in the results (Fig 3.3). Overall, Support Vector Machines don’t perform well with large datasets, which will be part of our endeavour.

3.4. Multilayer Perceptron

The Multilayer Perceptron (MLP) is a feedforward neural network, that is made up of multiple layers of nodes in a directed graph, where each node from one layer is connected to all the nodes from the previous one. MLPs are widely used in pattern recognition, classification, and regression problems due to their ability as networks to model complex nonlinear relationships in the input data. An MLP consists of an input layer of neurons, one or more hidden layers, and an output layer. Each node, except for those in the input layer, is a neuron that uses a nonlinear activation function to combine inputs from the previous layer and an additional bias term.

3.4.1. Structure of an MLP

An MLP is made up of the following components:

- **Input Layer:** The first layer of the network, which receives the input data to be processed. Each neuron in this layer represents a feature of the input data.
- **Hidden Layers:** One or more layers that perform computations on the inputs received and pass their output to the next layer. The neurons in these layers apply activation functions to their inputs to introduce nonlinearity.
- **Output Layer:** The final layer that produces the output of the network.

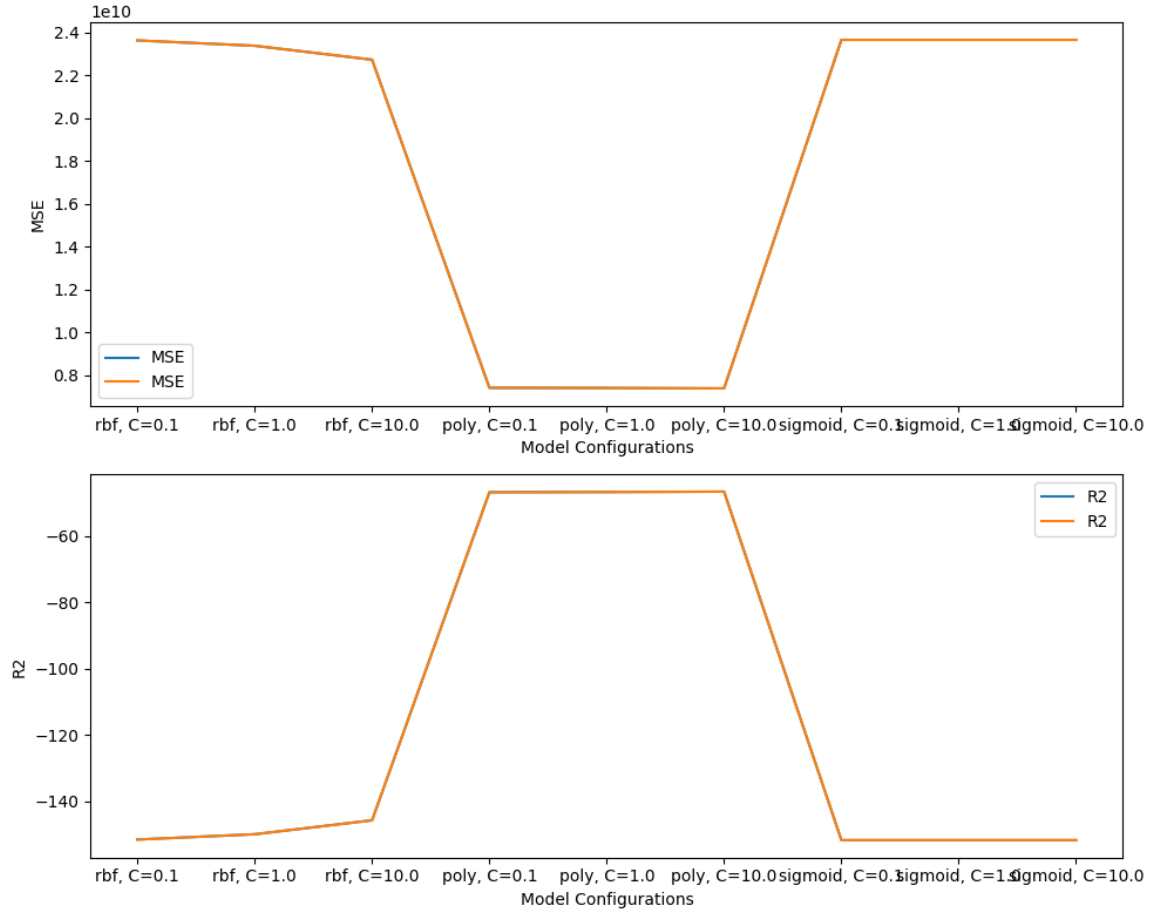


Figure 3.3: Results of support vector machine experiment.

3.4.2. Forward Propagation

The process of computing the output of an MLP is called forward propagation. In this process, the input data is passed through each layer of the network, transforming the data as it moves through. The output of each neuron is computed as follows:

$$a_j^{(l)} = \phi \left(\sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)} \right) \quad (3.1)$$

where

- $a_j^{(l)}$ is the activation of the j -th neuron in the l -th layer,
- ϕ denotes the activation function,
- $w_{ji}^{(l)}$ represents the weight from the i -th neuron in the $(l-1)$ -th layer to the j -th neuron in the l -th layer,
- $b_j^{(l)}$ is the bias term for the j -th neuron in the l -th layer,
- $a_i^{(l-1)}$ is the activation of the i -th neuron in the $(l-1)$ -th layer.

3.4.3. Backpropagation and Training

To train an MLP, the backpropagation algorithm is used. This algorithm adjusts the weights and biases of the network to minimize the difference between the actual output and the expected output. The process involves computing the gradient of a loss function with respect to each weight and bias in the network, and then using these gradients to update the weights and biases in the direction that minimizes the loss. The loss function measures the error between the predicted output and the actual output. The update rule for the weights is given by:

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} \quad (3.2)$$

where

- η is the learning rate,
- $\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}}$ is the partial derivative of the loss function \mathcal{L} with respect to the weight $w_{ji}^{(l)}$.

Similar updates are made for the biases.

Through iterative training involving forward propagation, loss calculation, and backpropagation, the MLP learns to approximate the function that maps data inputs to desired predictions.

3.4.4. Results

The model has been trained and tested on the google dataset. The results can be seen below (Fig 3.4).

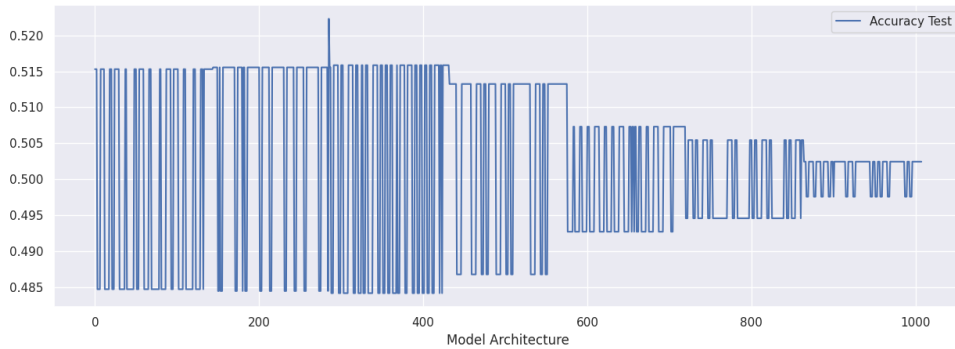


Figure 3.4: Results of multilayer perceptron experiment.

3.5. Convolutional neural network

Convolutional Neural Networks (CNNs) are a class of deep neural networks, highly effective for analyzing visual imagery. They employ a mathematical operation called convolution, which allows them to efficiently process data in a grid-like topology, such as images.

3.5.1. Architecture of Convolutional Neural Networks

A typical CNN architecture comprises several layers that transform the input image to produce an output that represents the presence of specific features or class labels. The most common layers found in a CNN are:

Convolutional Layer

The convolutional layer is the fundamental building block of a CNN. It applies a set of learnable filters to the input. Each filter activates specific features at certain spatial positions in the input. Mathematically, the convolution operation is defined as follows:

$$f(x, y) = (g * h)(x, y) = \sum_m \sum_n g(m, n) \cdot h(x - m, y - n)$$

where $f(x, y)$ is the output, g is the input image, h is the filter, and $*$ denotes the convolution operation.

Activation Function

Following convolution, an activation function is applied to introduce non-linearity into the model. The Rectified Linear Unit (ReLU) is commonly used:

$$f(x) = \max(0, x)$$

Pooling Layer

The pooling layer reduces the spatial dimensions (width and height) of the input volume for the next convolutional layer. It operates independently on every depth slice of the input and resizes it spatially. A common operation for pooling is max pooling:

$$f(x, y) = \max_{(a, b) \in D} g(x + a, y + b)$$

where D is the set of coordinates in the input, and g is the input image.

Fully Connected Layer

Towards the end of the network, fully connected layers are used, where each input node is connected to each output by a learnable weight. This layer classifies the image into various classes based on the learned high-level features.

Output Layer

The final layer of a CNN is a softmax layer that outputs a probability distribution over the classes, indicating the likelihood of the input image belonging to each class.

3.5.2. Results

The CNN model has been trained and tested on the simple sales data.

3.6. Residual neural network

As the number of layers in classic CNN grows, the problem of vanishing or exploding gradient appears. It means that gradients become very small (or very large in case of exploding) as they are propagated through the layers of the network.

Residual neural network (ResNet) is a type of CNN that is less subject to this problem. It was developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in 2015.

3.6.1. ResNet Architecture

ResNet uses a technique called skip connections. Skip connection bypasses several layers, so that input of one layer is added to the output of some further layer. Blocks of such connections are called residual and the network is formed by stacking residual blocks.

The idea is that instead of learning the original mapping of input to output, residual blocks learn the difference between output and input, which is called the residual function.

3.6.2. Results

This model has been trained on GBP/CAD dataset in two versions: not pretrained and pretrained.

Chapter 4

Methodology

Chapter 5

Main results

Chapter 6

Forecasting applications

Chapter 7

Conclusion

Appendix A

Visualisation