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

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 analise 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 analise 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 aplied 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

Preliminary definitions & guidelines

2.1. Macros

2.2. Notation

- By goal or problem or task we mean the overall task of the thesis, which is develop a machine learning model suitable for predicting prices on the financial market, based on a history of data.

2.3. Datasets

Here we describe the various datasets we used for the training and testing of our models. Each description includes the following:

- **Source:** the name and source from where we took the dataset. Including a link or a way to access and download the dataset.
- **Collection method:** a description of how the data in the dataset was gathered and over what time span.
- **Motivation:** a description of what the data in the dataset represent, what their purpose is, how they were gathered and why it is valuable to our research.
- **Size:** a description of how many datapoints the dataset contains, how many features each datapoints has and the overall size of the file.
- **Features:** a description of the features each datapoint of the dataset has and what the features represent.
- **Quality:** a description of the features and drawbacks of the overall dataset e.g. how dependent are the features between themselves.
- **Plot:** a plot of the numeric features of the dataset, its visual representation.

2.3.1. House sales

- **Source:** House Property Sales Time Series (raw_sales.csv) :
https://www.kaggle.com/datasets/htagholdings/property-sales

- **Collection method:** Property sales data was gathered for the 2007-2019 period for one specific region. (not concrete proofs)(TODO: * region was not specified)
- **Motivation:** A simple dataset for the basic models. A proper, manageable size, once filtered by property and number of bedrooms. Has very few features, making it easier to understand and process.
- **Size:** Consists of 29600 datapoints. Once filtered for property being house, number of bedrooms equal to 3, around 11300 datapoints remain.
- **Features:** The dataset has very few features. Every datapoint consists of:
  - **datesold**: Date at which the sale was made.
  - **postcode**: 4 digit postcode of the suburb where the property was sold (given for reference only).
  - **price**: Price for which the property was sold.
  - **propertyType**: Property type i.e. house or unit.
  - **bedrooms**: Number of bedrooms: 1, 2, 3, 4 or 5.

  The dataset was filtered by propertyType and bedrooms, and the postcode was discarded.
- **Quality:** The prices rise, while nonlinearly, with time.
- **Plot:**

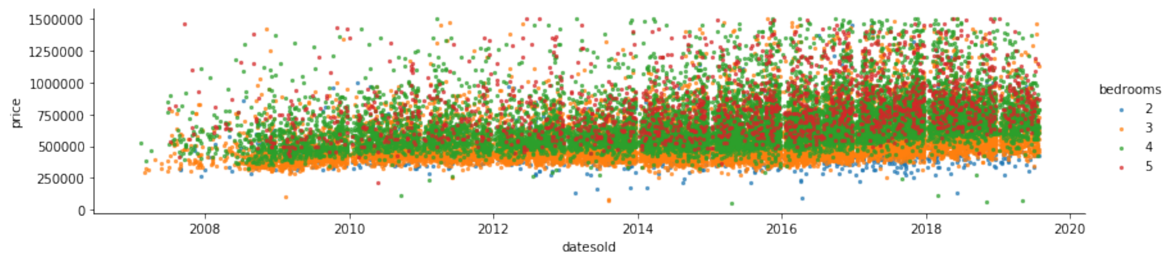


Figure 2.1: A plot of the House Sales dataset.

2.3.2. Google Stock

- **Source:** Google Stock Dataset
  https://www.kaggle.com/datasets/r1shabhgupta/google-stock-price-daily-weekly-and-monthly-2023/data
- **Collection method:** Collected from the stock market (probably). Has daily data and weekly and monthly summaries.
- **Motivation:** More complex dataset, multiple time series numeric features, but fewer datapoints.
- **Size:** Daily set has 2510 datapoints. Weekly set has 520 datapoints. Monthly set has 120 datapoints.

- **Features:** Every datapoint has the following features:
  - **Date**: the date on which the stock price was recorded.
  - **Price**: the opening price of the stock on the given date.
  - **High**: the highest price at which the stock was traded during the day.
  - **Low**: the lowest price at which the stock was traded during the day.
  - **Close**: the closing price of the stock on the given date.
  - **Volume**: the total number of shares traded on the given date.
  - **Adj Close**: the adjusted closing price of the stock on the given date.
- **Quality:**
- **Plot:**

2.3.3. Apple

- **Source:** Apple Stock Prices
  https://www.kaggle.com/datasets/suyashlakhani/apple-stock-prices-20152020
- **Collection method:** Collected from the stock market (probably).
- **Motivation:** Even more features then the Google Stock dataset, more interesting.
- **Size:** The dataset consists of 1258 datapoints.
- **Features:** Every datapoint has the following features
  - **close** - Closing price
  - **high** - Highest price of the day
  - **low** - Lowest Price of the day
  - **open** - Opening price of the day
  - **volume** - Volume of stock traded
  - **adjClose** - Closing stock price in relation to other stock attributes/actions
  - **adjHigh** - Highest stock price in relation to other stock attributes/actions
  - **adjOpen** - Opening Stock price in relation to other stock attributes/actions
  - **adjVolume** - Trading volume in relation to other stock attributes/actions
  - **divCash** - Cash dividend
  - **splitFactor** - Stock split

2.3.4. Gold

- **Source:** Gold rates
  https://www.kaggle.com/datasets/hemil26/gold-rates-1985-jan-2022
- **Collection method:** This data was collected from https://www.gold.org/goldhub and then cleaned. Has daily data and annual summaries.
- **Motivation:** The dataset contains lots of datapoints and lots of features.

- **Size:** Annual dataset has 43 datapoints. Daily dataset has a bit over 10 thousand datapoints.
- **Features:** Each datapoint has the following features: date and rates in six curriencies: USD (American), INR (Indian), AED (Arabian), EUR (European), GBP (South Georgian) and CNY (Chinese).

2.3.5. Electricity

- **Source:**
- **Motivation:**
- **Size:** The dataset has 26304 datapoints.
- **Features:** Each datapoint consists of a date and 320 numbered columns and an OT column.
- **Collection method:**
- **Quality:**
- **Plot:**

2.3.6. Sine

- **Source:** sine.csv - we generated this dataset ourselves
- **Motivation:** This is a non-trivial, but predictable, dataset for testing whether LLM responds to the data.
- **Size:** Dataset consists of 1000 datapoints.
- **Features:** Each datapoint consists of date, x and target, where target = sin(x).
- **Collection method:** Dataset was generated by a script.
- **Quality:**
- **Plot:** A simple sine wave.

2.3.7. Time Series Practice Dataset

- **Source:** https://www.kaggle.com/datasets/samuelcortinhas/time-series-practice-dataset/data?select=tra
- **Motivation:** As the Sine dataset it is a fictional dataset to more easily tune LLM training, but more complicated than Sine and significantly larger.
- **Size:** This dataset contains 230 thousand simulated time series datapoints covering 10 years (2010-2019). The are no null values.
- **Features:** The features include date, store id, product id and number sold. The train.csv covers the years 2010-2018 and the test.csv covers 2019 only. The are 7 unique stores and 10 unique products.
- **Collection method:** This time series data was created using multiple features including various long term trends, year-long seasonality patterns, weekday/weekend effects and noise. Moreover, the products and the stores are supposed to be weakly correlated.

2.4. What metrics we used

Every metric should be described by

- what it measures,
- how it is calculated,
- what its result represents,
- how valuable it is,
- when it can be applied.

2.4.1. Mean Squared Error

The Mean Squared Error (MSE) is a metric used for evaluating the accuracy of a machine learning model, especially in regression tasks. It quantifies how close a model's predictions are to the actual values.

Definition

MSE calculates the average of the squares of the errors. The error is the difference between the values ($\hat{y}_i$) predicted by the model and the values (y_i) it should have predicted.

Calculation Steps

The metric is computed as follows.

1. For each prediction the error is computed by subtracting the predicted value from the actual value.
2. Each error is then squared in order to ensure that positive and negative errors do not cancel each other out and in order to emphasize larger errors.
3. The average of these squared errors is then computed to obtain the MSE.

Mathematical Formulation

The mathematical formula for MSE is given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

where:

- n is the number of data points,
- y_i is the actual value for the ith datapoint,
- $\hat{y}_i$ is the predicted value for the ith datapoint,

Interpretation

- MSE is a non-negative number where a value of 0 indicates perfect predictions.
- Larger MSE values indicate worse model performance.
- MSE emphasizes larger errors due to the squaring of each term, which can be both advantageous and disadvantageous depending on the application.

2.4.2. R^2 Score

The R^2 score, or the coefficient of determination, is a statistical measure used in regression analysis to assess how well fitted a model is. It indicates the proportion of the variance in the dependent variable that is predictable from the independent variable(s).

Definition

The R^2 score is defined as the ratio of the variance explained by the model to the total variance. It is a measure of how well the observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model.

Calculation

The R^2 score is calculated using the following formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{2.2}$$

where:

- y_i is the actual value for the ith data point,
- $\hat{y}_i$ is the predicted value for the ith data point,
- $\overline{y}$ is the mean of the actual values,
- n is the number of data points.

Interpretation

- An R^2 score of 1 indicates that the regression model perfectly fits the data.
- An R^2 score of 0 means that the model does no better, than would a prediction using the mean.
- Negative R^2 values can occur when the chosen model fits worse than a horizontal line representing the mean of the response.

Limitations

The R^2 metric has some limitations. It does not necessarily imply causation, nor does a high R^2 score mean that the model is the best choice for prediction. It's also important to note that adding more predictors to a model can artificially inflate the R^2 value, even if those predictors are not statistically significant.

2.5. How we describe models

Below, in the chapter "Other models" we describe several models we tried to use for the task of predicting prices. The descriptions include the following:

- **Description:** a short description of how the model works and how it is trained.
- **Motivation:** what the model is usually used for and why we chose to try it out.
- **Features and limitations:** some advantages and benefits of the model, as well as its disadvantages and drawbacks.
- **Parameters:** the description of the parameters of the model and how they affect its training.
- **Metrics:** how we measured the results of the training and testing of the model.
- **Data used:** what combinations of parameters of the model we tested and on what datasets we trained and tested the model.
- **Preprocessing:** how the datasets used were preprocessed for training and testing of the model.
- **Analysis:** an analysis of our results of our training and testing of the model compared with the results obtained in literature.
- **Picture:** a picture or a plot demonstrating the results obtained from testing the model.

2.6. Literature review

Literature review contains:

- A list of approaches to the problem.
- For each approach, its basic description and its significance to our goal.
- Its features and drawbacks compared to our goal.
- Its differences, when compared to our goal.
- Whether our own results validated the results of the article.

Chapter 3

Literature review

Chapter 4

Related work

Chapter 5

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.

5.1. 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 risk 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

5.1.1. 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 5.1 below.

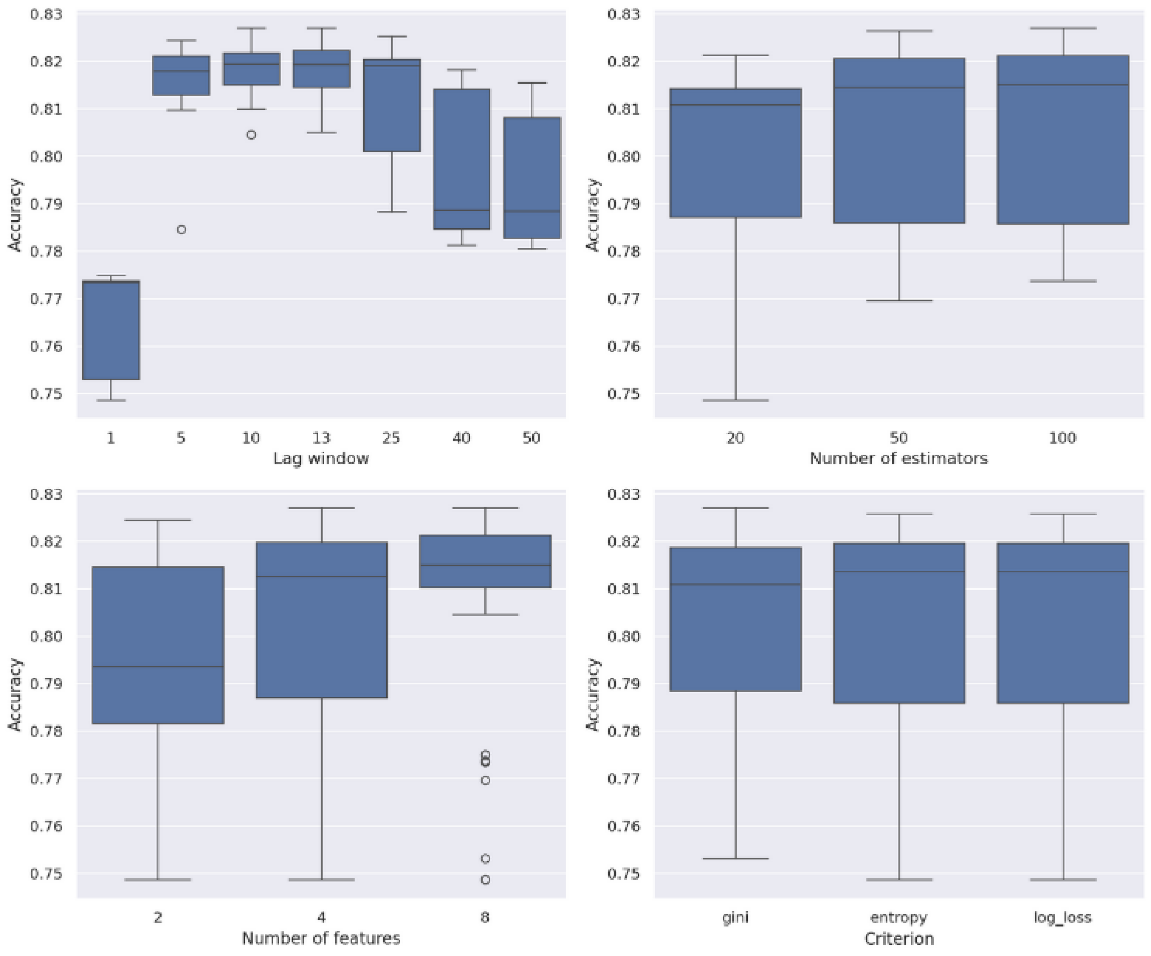


Figure 5.1: Results of random forest experiment.

5.2. 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

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

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

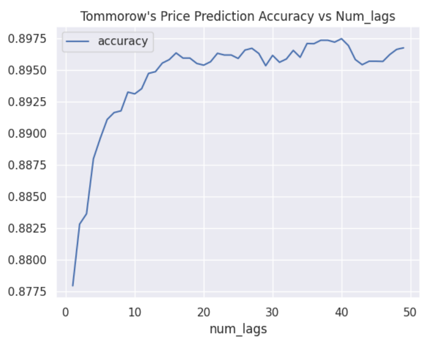
Function Y assumes values only in the range $(0, 1)$. If $Y(X) >= 0.5$, the model classifies the datapoint as 1 (in our model below - the price will be higher). If the converse is true, the datapoint is classified as 0 (the price will be lower).

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.

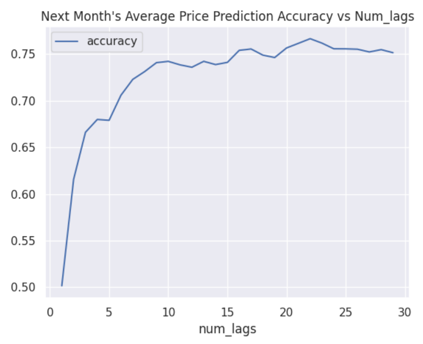
5.2.1. Results

Two models have been trained. The first one is trained to predict if the next price in the dataset wil 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 subsuquent months). The first model proved to be more precise with an accuracy of 82% for an optimal value of k of around 40 (Fig 5.2a). At the same time the second model only scored 75% (Fig 5.2b).

However, simply predicting whether the price will be higher or lower is much easier then 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.



(a) Day-by-day prediction.



(b) Month-by-month prediction.

Figure 5.2: Plots of two models of logistic regression.

5.3. 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.

5.3.1. 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.

5.3.2. 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 5.3). Overall, Support Vector Machins don’t perform well with large datasets, which will be part of our endeavour.

5.4. 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.

5.4.1. 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.

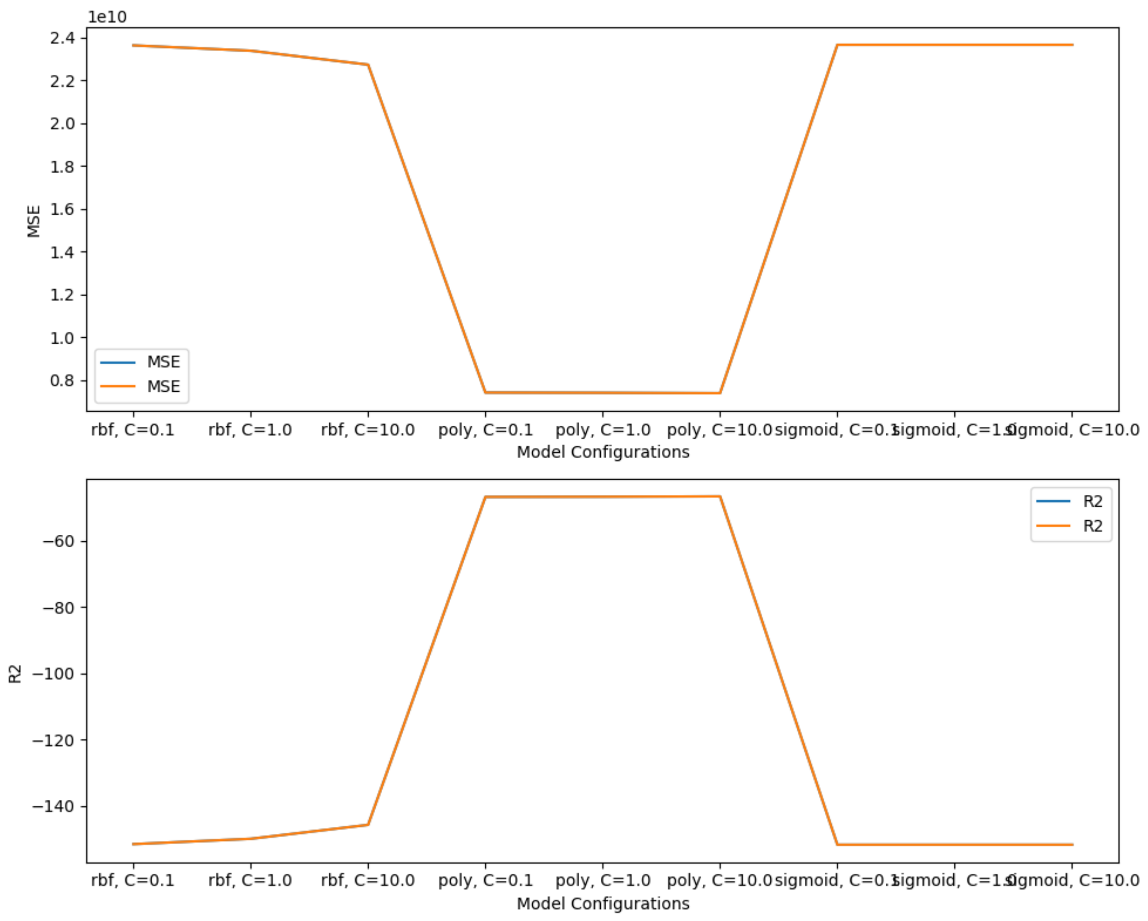


Figure 5.3: Results of support vector machine experiment.

5.4.2. 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:

$$a_j^{(l)} = \phi\left(\sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}\right) \tag{5.1}$$

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.

5.4.3. 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:

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} \tag{5.2}$$

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.

5.4.4. Results

The model has been trained and tested on the google dataset. The results can be seen below (Fig 5.4).

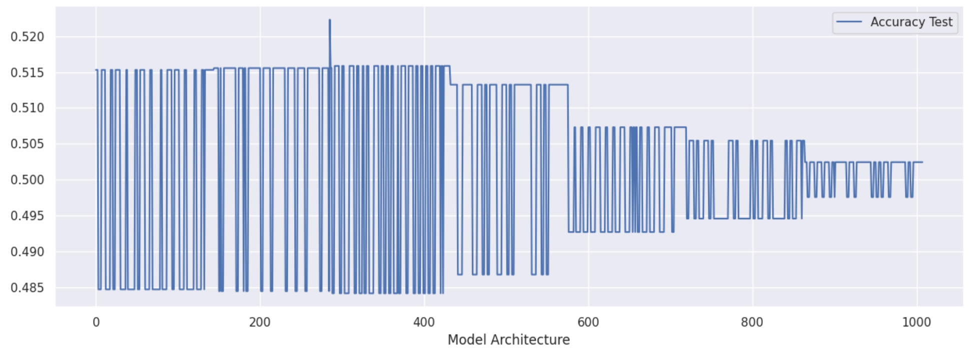


Figure 5.4: Results of multilayer perceptron experiment.

5.5. 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.

5.5.1. 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.

5.5.2. Results

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

5.6. 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.

5.6.1. 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.

5.6.2. Results

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

Chapter 6

Large Language Model

In this chapter we present the basic theory behind Large Language Models. We then introduce the LLaMA model, which we've been using. We then describe the embedding technique we've used and we present our results.

6.1. Overview

A Large Language Model (LLM) is an advanced artificial intelligence system designed to understand and generate human-like text based on patterns it learns from enormous amounts of textual data. These models utilize deep learning techniques, particularly neural networks, which consist of layers of interconnected neurons that (layers) process information sequentially, one by one. The predominant architecture underpinning most contemporary LLMs is the Transformer (see below), noted for its self-attention mechanism that enables the model to assess the importance of different words in a sentence irrespective of their positional relationship.

The training of an LLM involves presenting it with a large, diverse corpus of text, during which it adjusts its internal parameters to minimize the difference between its predictions and the actual data. This process, known as supervised learning, equips the model with a probabilistic understanding of language, enabling it to predict a continuation of a given piece of input text.

Once trained, LLMs can perform a variety of language-based tasks such as translation, summarization, question answering, and text generation. Their capability to generate coherent and contextually relevant text makes them invaluable for applications in natural language processing (NLP), including conversational agents, content creation tools, and sophisticated text analysis. We will see how it deals with analysing and predicting time series data.

6.2. The Transformer

The Transformer is a type of neural network architecture introduced in the seminal 2017 paper *"Attention is All You Need"* by Vaswani et al. It has since become foundational for many cutting-edge natural language processing (NLP) models due to its efficiency and effectiveness in handling data sequences, such as text.

6.2.1. Key Features and Components

- **Attention Mechanism:** The Transformer implements a self-attention mechanism that allows the model to semantically connect different words in a sentence, regardless of their positions. For instance, in the sentence "The cat that sat on the mat chased the mouse," the model can directly relate "cat" and "chased" without processing intervening words sequentially.
- **Layer Structure:** Composed of a stack of identical layers, each Transformer layer has two main sub-layers: a multi-head self-attention mechanism and a fully connected feed-forward layer. Both sub-layers are equipped with residual connections (see ResNet description) followed by layer normalization.
- **Multi-Head Attention:** This component allows the model to focus on different positions of the input sequence simultaneously. It divides the attention mechanism into multiple "heads", where each head focuses on a different aspect of the input sequence, enabling the model to learn from input information on many different syntactic and semantic levels simultaneously.
- **Positional Encoding:** To compensate for the inherent lack of sequential data processing in Transformer model, *positional encodings* are added to input to inject information about the position of each word in the sequence. This way the Transformer can also take into account the order of the input tokens.
- **Encoder and Decoder:** The Transformer model typically features an *encoder* to process the input text and a *decoder* to generate the output text, a design particularly effective for machine translation. The encoder transforms an input sequence into a series of continuous representations (i.e. high-dimensional vectors), which the decoder then converts into an output sequence. The decoder also utilizes self-attention to reference previously generated words, ensuring contextually aware generation.

6.3. LLaMA model

6.3.1. Introduction

LLaMA or **Large Language Model Meta AI** is an innovative large language model developed by Meta AI. The model is part of a new generation of neural networks designed for a wide range of natural language processing (NLP) applications, from simple text generation to complex decision-making tasks.

6.3.2. Key Features

- **Model Variants:** LLaMA is available in various sizes, offering flexibility for deployment in different environments. These variants range from smaller models that can run on limited-resource settings to larger versions designed for high-performance servers.
- **Training Data:** The model has been trained on a diverse dataset, including a vast corpus of text from books, websites, and other text sources.
- **Application Scope:** Due to its scalable nature, LLaMA is suited for a variety of applications, such as conversational agents, content generation, summarization, and more advanced tasks like sentiment analysis and machine translation.

- **Accessibility:** Meta AI has made strides towards making LLaMA accessible to researchers and developers, which promotes further innovation and development within the community.

6.3.3. Introduction to LLaMA-2

LLaMA-2 represents the next generation of Large Language Models developed by Meta AI, building upon the foundation established by the original LLaMA model. This enhanced model incorporates significant advancements in architecture, training, and application, making it more efficient and powerful in handling a wider range of natural language processing tasks.

6.3.4. Advancements in LLaMA-2

LLaMA-2 introduces several key improvements that enhance its performance and usability:

- **Improved Training Techniques:** LLaMA-2 employs advanced training methodologies that increase its learning efficiency and the richness of its language understanding. These techniques also help in reducing biases and improving the model's ability to generalize across different contexts.

- **Expanded Model Sizes:** While maintaining scalability, LLaMA-2 offers a range of model sizes, enabling users to select a model that best fits their computational budget and performance needs.

6.4. Time-series Embedding

We focus on reprogramming an embedding-visible language foundation model, here Llama-2, for general time series forecasting without requiring any fine-tuning of the backbone model. Specifically, we consider the following problem: given a sequence of historical observations $X \in \mathbb{R}^{N\times T}$ consisting of N different 1-dimensional variables across T time steps, we aim to reprogram a large language model $f(\cdot)$ to understand the input time series and accurately forecast the readings at H future time steps, denoted by $\hat{Y} \in \mathbb{R}^{N\times H}$, with the overall objective to minimize the mean square errors between the ground truths Y and predictions, i.e., $\frac{1}{H}\sum_{h=1}^{H} \|\hat{Y}_h - Y_h\|_F^2$.

The method encompasses three main components: (1) input transformation, (2) a pre-trained and frozen LLM, and (3) output projection. Initially, a multivariate time series is partitioned into N univariate time series, which are subsequently processed independently (Nie et al., 2023). The i-th series is denoted as $X(i) \in \mathbb{R}^{1\times T}$, which undergoes normalization, patching, and embedding prior to being reprogrammed with learned text prototypes to align the source and target modalities. Then, we augment the LLM's time series reasoning ability by prompting it together with reprogrammed patches to generate output representations, which are projected to the final forecasts $\hat{Y}^{(i)} \in \mathbb{R}^{1\times H}$.

We note that only the parameters of the lightweight input transformation and output projection are updated, while the backbone language model is frozen. In contrast to vision-language and other multimodal language models, which usually fine-tune with paired cross-modality data, this use of model is directly optimized and becomes readily available with only a small set of time series and a few training epochs, maintaining high efficiency and imposing fewer resource constraints compared to building large domain-specific models from scratch or fine-tuning them. To further reduce memory footprints, various off-the-shelf techniques (e.g., quantization) can be seamlessly integrated for slimming this.

6.5. Our methodology

6.6. Results

Chapter 7

Methodology

Chapter 8

Main results

Chapter 9

Forecasting applications

Chapter 10

Conclusion

Appendix A

Visualisation

Bibliography

[] https://arxiv.org/pdf/2310.19717v1.pdf

[Bea65] Juliusz Beaman, *Morbidity of the Joll function*, Mathematica Absurdica, 117 (1965) 338–9.

[Blar16] Elizjusz Blarbarucki, *O pewnych aspektach pewnych aspektów*, Astrolog Polski, Zeszyt 16, Warszawa 1916.