Wids Final Summary

24B3928

February 2025

1 ARIMA Model

Regression refers to a statistical method used to model and analyze relationships between variables. It helps in understanding how a dependent variable (outcome) changes when one or more independent variables (predictors) change.

1.1 AR models

A precisely defined auto regressive (AR) model of order $p \geq 0$, denoted as AR(p), is given by:

$$x_t = \sum_{j=1}^{p} \phi_j x_{t-j} + w_t, \tag{1}$$

where w_t, for $t = 0, \pm 1, \pm 2, \pm 3, \ldots$, is a white noise sequence.

The coefficients $\phi_1, ..., \phi_p$ in (1) are fixed (non-random), and we assume $\phi_p \neq 0$ (otherwise the order here would effectively be less than p).

We also define a back shift operator as

$$Bx_t = x_{t-1}$$

and hence we can write AR model equivalently as

$$\phi_p(B) w_t = \epsilon_t \tag{2}$$

where $\phi_p(B)$ is the AR polynomial:

$$\phi_p(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p, \tag{3}$$

and ϕ_i are the AR coefficients. White noise refers to a sequence of random variables w_t that are independently and identically distributed (i.i.d.) with zero mean and constant variance, typically expressed as:

$$\mathbf{E}[w_t] = 0 \tag{4}$$

1.2 Moving Average (MA) Models

A moving average (MA) model is conceptually "dual" to an autoregressive (AR) model. Instead of modeling x_t as a function of its past values, an MA model represents x_t as a linear combination of past white noise terms.

Formally, an MA model of order $q \geq 0$, denoted as MA(q), is defined as:

$$x_t = w_t + \sum_{j=1}^{q} \theta_j w_{t-j} \tag{5}$$

where:

- w_t is a white noise sequence with mean zero and constant variance.
- θ_j are the moving average coefficients.
- The value of x_t depends on the current and past white noise terms.

The MA model captures short-term dependencies in the time series by modeling how past errors influence the present.

1.3 Comparison of AR and MA Models

Both autoregressive (AR) and moving average (MA) models are fundamental in time series analysis. They differ in how they capture dependencies:

- **Autoregressive (AR) Model:** Models x_t as a function of its past values. It is useful when past observations have a direct influence on the current value.
- **Moving Average (MA) Model:** Models x_t based on past forecast errors (white noise terms). It is useful when shocks in the data influence future values.

Advantages of AR Models:

- Suitable for long-term dependencies.
- Can be interpreted as a regression model on past observations.

Advantages of MA Models:

- More stable for certain datasets since past white noise terms are bounded.
- Useful for capturing short-term correlations without requiring stationarity constraints.

1.4 ARIMA Model

The **AutoRegressive Integrated Moving Average (ARIMA)** model combines three components:

- **Autoregression (AR)**: A model that uses the dependent relationship between an observation and a number of lagged observations.
- **Integration (I)**: Differencing the raw observations to make the time series stationary.
- **Moving Average (MA)**: A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

1.4.1 Mathematical Formulation of ARIMA(p, d, q)

Let y_t be the observed time series data. The ARIMA model is formulated as follows:

Differencing for Stationarity If the time series is non-stationary, we apply differencing d times:

$$w_t = \Delta^d y_t = (1 - B)^d y_t \tag{6}$$

where B is the backshift operator: $By_t = y_{t-1}$, and Δ is the difference operator $\Delta y_t = y_t - y_{t-1}$.

Autoregressive (AR) Component The AR part of order p is given by:

$$\phi_p(B) w_t = \epsilon_t \tag{7}$$

where $\phi_p(B)$ is the AR polynomial:

$$\phi_p(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p, \tag{8}$$

and ϕ_i are the AR coefficients.

Moving Average (MA) Component The MA part of order q is given by:

$$w_t = \theta_q(B) \epsilon_t \tag{9}$$

where $\theta_q(B)$ is the MA polynomial:

$$\theta_q(B) = 1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q, \tag{10}$$

and θ_i are the MA coefficients.

Final ARIMA Model Equation Combining the AR and MA components, we obtain:

$$\phi_p(B)(1-B)^d y_t = \theta_q(B)\epsilon_t \tag{11}$$

where:

- $\phi_p(B)$ represents the AR component,
- $(1-B)^d$ represents the differencing for stationarity,
- $\theta_q(B)$ represents the MA component,
- $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ is white noise.

This equation provides a complete description of an **ARIMA(p, d, q)** process.

Stationary Requirement the (ϕ) values must satisfy certain conditions to prevent the influence of past values from growing uncontrollably. For example

$$|\phi_1| < 1$$

the series remains stationary because past values gradually fade out over time.

$$|\phi_1| \geq 1$$

the series becomes non-stationary:

- If $\phi_1 = 1$, the series is a random walk:

  $$x_t = x_{t-1} + w_t$$

  which has an ever-increasing variance.

- If $|\phi_1| > 1$, the series explodes exponentially.

1.5 Optimization

We used a simple grid-search method to get the lowest value of error. In the grid search method we only iterated over all the probable values for the (p, q, d) triplet.

```
import numpy as np
from statsmodels.tsa.arima.model import ARIMA

def grid_search_arima(series, p_range, d_range, q_range):
    best_aic = np.inf
    best_order = None
    for p in p_range:
        for q in q_range:
            for d in d_range:
                model = ARIMA(series, order=(p, d, q)).fit()
                if model.aic < best_aic:
```

```
                    best_aic = model.aic
                    best_order = (p, d, q)
    return best_order

best_order = grid_search_arima(training_df, p_range=range(5),
    d_range=range(3), q_range=range(5))
print("Optimal ARIMA order:", best_order)
```

Listing 1: Grid Search for Optimal ARIMA Order

2 LSTM Neural Networks

Take from:-"https://colah.github.io/posts/2015-08-Understanding-LSTMs/"

LSTM networks are a special type of Recurrent Neural Networks which overcome the following problem of RNNs:- "long-term dependencies".

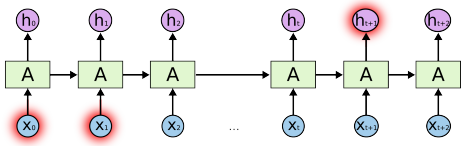

Figure 1: Recurrent Neural Network

Say we take the example of a node of a recurrent neural network ,then as that gap grows, RNNs become unable to learn to connect the information.

In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer i.e.

$$h_t = tan^{-1}(Wh_{t-1} + Wx_t + b)$$

where:

- h_t is the hidden state at time step t,
- h_{t-1} is the hidden state from the previous time step,
- x_t is the input at time step t,
- W_h and W_x are weight matrices,
- b is a bias term,
- tanh is the activation function that introduces non-linearity.

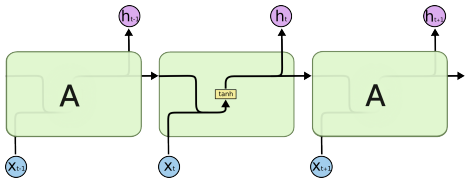

Figure 2: Simple Recurrent Neural Network

The Solution LSTMs (Long Short-Term Memory networks) are a type of recurrent neural network (RNN) designed to address long-term dependencies in sequential data. Unlike traditional RNNs, LSTMs incorporate a structured repeating module with multiple interacting layers.

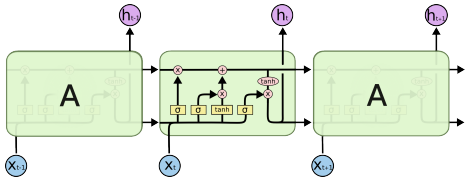

Figure 3: Simple Recurrent Neural Network

At the core of LSTMs is the **cell state**, a horizontal line that allows information to flow with minimal modification. This is controlled by three types of gates:

- **Forget Gate:** This gate determines what information from the previous cell state should be discarded. It consists of a sigmoid activation function that takes the previous hidden state h_{t-1} and the current input x_t and outputs a value between 0 and 1 for each element in the cell state. A value close to 0 means forgetting the information, while a value close to 1 means retaining it:
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
- **Input Gate:** This gate decides which new information should be added to the cell state. It consists of two parts:
  - A sigmoid activation function that determines which values to update:
  $$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
  - A tanh activation function that generates candidate values to be added to the state:
  $$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The new cell state is updated as:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

- **Output Gate:** This gate determines what the next hidden state should be, which also serves as the output of the LSTM unit. It consists of:
  - A sigmoid activation function to decide which parts of the cell state should contribute to the hidden state:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

  - A tanh activation function applied to the cell state, followed by element-wise multiplication with the output gate's result:

$$h_t = o_t \cdot \tanh(C_t)$$

3 Sentiment Analysis

We used TextBlob to determine the sentiment of a give news,it returned a float value ranging from -1 to 1 that represents the sentiment of the text:

- 1 indicates a positive sentiment.
- -1 indicates a negative sentiment.
- 0 indicates a neutral sentiment

Although, I didn't got deep into the working of TextBlob, I took it as a model for classification problem using machine learning, obviously with the devil lurking in the details.

4 Final Implementation

In the Final week of my project, we trained our LSTM neural network on both stock price data as well as the sentiment of relevant news. We first extended our data set to include a sentiment column, and then trained the network over close price and sentiment.

5 Challenges

- The data collection for the final week was very difficult.
- The multi variable training for the LSTM network was challenging.

Overall it was a very good experience and I am looking forward to more upcoming projects.