Advanced Data Analysis and Machine Learning (2023-2024)
Computational Engineering
LUT University

Google Stock Dataset: Week 3

Alejandro López Vargas
Sina Saeedi
Pau Ventura Rodríguez

November 20, 2023

1. Introduction

Visit our GitHub to gain access to the script.
In this project, we aim to analyze the historical data of Google's (GOOGL) stock price and create a predictive model for it. The dataset includes the following variables: Date, Open, High, Low, Close, Volume, and Name. The 'Name' variable is constant and represents the company's name, which is Google in this case, making it unnecessary for our analysis.
Here are brief explanations about each of the variables:

1. **Date:** This variable represents the specific dates corresponding to each data point in the dataset. It serves as the timeline for the stock's historical performance.

2. **Open:** The opening price of the Google stock on a particular date. This is the price at which the stock begins trading at the beginning of the trading day.

3. **High:** The highest price at which Google stock traded during a given day. It provides insights into the highest level the stock reached within a trading session.

4. **Low:** The lowest price at which Google stock traded during a given day. It indicates the lowest level the stock reached within a trading session.

5. **Close:** The closing price of Google stock on a specific date. It is the final price at which the stock is traded for the day, representing the market sentiment at the close of the trading session.

6. **Volume:** The total number of shares of Google stock traded on a given day. Volume is a crucial indicator of market activity, providing insights into the level of interest and participation in the stock.

Project's Goal:
The main objective of this project is to develop a predictive model that can anticipate future closing prices(Close) of Google stock based on historical patterns and trends. By drawing insights from the information contained in the Open, High, Low, Close, and Volume variables, we aim to create a reliable model that can assist investors and traders in making reasonable decisions about Google stock.

2. Preprocessing

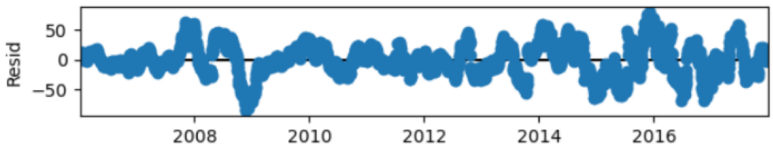
Before performing any visualization, we have to convert the Date column on the dataframe to format Datetime in order to give it an ordinal value.
The column "Name" only has one unique value, which does not give us any information on the target and will then be dropped.
Our dataset has the columns Open, Close, High, Low, and Volume. However, if we want to predict the Closing price, we will not have the High, Low, or Volume variables. However, what we will have is the values for those variables on the previous timestamp, so we can use on each sample the lagged version of Low, High, and Volume instead of the original one.

Do you have continuous measurements with the same frequency? If not, what procedures can you employ to have the data sampled at the same rate.
The stock market is only open on business days, which means we do not have data for all days of the year. Furthermore, some holidays did also not register data like Christmas, which we will create in order to have an equally spaced dataframe. Pandas allows us to create these extra samples by using the command: df.asfreq('b'), which will fill their values with NaN.

Is your data synchronous across all the variables? If not, what procedures can you utilize to bring the data in the same timesteps?
Yes, all of our features are gathered daily. Open is gathered at the beginning of the day, and all others are determined at the end of the day using the stock price fluctuation over the day.

Do you have missing values in your data? if yes, are there multiple consecutive missing values? what are some strategies for filling in the missing values?
We have missing values as we created them when we added the extra days to keep the frequency. Unfortunately, we have 3 samples that have consecutive missing values. As all those samples consist of only clusters of 2 consecutive samples, we will use Frontfilling on the first instance and Backfilling on the last instance.
To fill up NaN values, we can use Backfilling (setting up the values with the next datapoint's values), Frontfilling (setting up the values with the previous datapoint's values), Meanfilling (assigning the mean value of that feature, which in this case will not yield good results due to the increasing tendency). In this case, we will use Backfilling.

Can you use the STL decomposition to spot and eliminate possible outliers?
We can use STL to spot and eliminate possible outliers using residuals. Residuals should be white noise, which means that they have zero mean, constant variance and are stationary. If our residuals match this definition, then we can use the normal distribution quantiles to check those samples that are unlikely to happen, assigning them as outliers. Our residuals for all variables have zero mean, but the variance is not constant:



For the "Low" feature for example, before 2008 the variance was very low compared to the variance between 2008-2009. We can also check stationarity using the Augmented Dickey–Fuller test. We got a zero-valued p-value for all residuals, which means that they are not stationary.

Do a mini literature-review. (i) on dividing a long time-series into sub-samplings (multiple sequences). How can the sub-sequencing be done? How does seasonality affect sub-sequencing? Can we consider trends and seasonality in LSTM models? (ii) what are some typical standardization methods for LSTM.
According to: link1 and link2
LSTMs expect 3d-inputs of shape [samples, n_time_steps, n_features]:

- Samples. Number of batches/sequences.
- Time Steps. Number of time steps per sample/sequence.
- Features. Number of features per sample.

LSTMs don't like sequences of more than 200-400 time steps, so the data will need to be split into samples. This is because, apart from having high training times, long sequences might cause vanishing gradients while back-propagating. There are many ways to do this, depending on the problem. Let's say we have a time series sequence of [1,2,3,4,5,6,7,8]:

- Sequences As-Is: Just using the sequence without splitting (i.e. [1,2,3,4,5,6,7,8]). This might cause vanishing gradient problems and long training times.
- Truncate Sequences: Consists of selectively removing time steps from the beginning or the end of input sequences. This would result in a considerable loss of information. For example, we would keep: [3,4,5,6].
- Non-overlapping Sub-sampling: Consists of dividing the sequence into multiple non-overlapping sub-sequences of the same shape. For example, we could split the array into: [[1, 2],[3,4],[5,6],[7,8]]. However, we need to make sure that the samples on each sub-sequence are larger than the seasonality periodicity, as otherwise we might lose information. A downside of this method is that we are only keeping certain information (how does 1 and 2 relate) but we lose some other information (how does 2 and 3 relate) as they are in different sub-sequences.
- Overlapping sub-sampling: Consists of dividing the sequence into multiple overlapping sub-sequences of the same shape. For example, we could split the array into: [[1,2,3,4], [2,3,4,5], [3,4,5,6], [4,5,6,7], [5,6,7,8]]. This would fix the previous

method's problem of losing information, but it drastically augments the amount of training data. We can also use different stride, for example: [[1,2,3,4], [3,4,5,6], [5,6,7,8]], reducing the number of samples.

- Using Truncated Backpropagation Through Time: Rather than updating the model based on the entire sequence, the gradient can be estimated from a subset of the last time steps. This would allow all sequences to be provided as input and execute the forward pass, but only the last tens or hundreds of time steps would be used to estimate the gradients and used in weight updates.

We do not have a large dataset (less than 3500 samples), so we can use the Overlapping sub-sampling technique with samples of shape 350, as we previously saw yearly seasonality.

Can we consider trends and seasonality in LSTM models?
Yes, the LSTM(Long Short-Term Memory) models can be used to capture trends and seasonality in time series. LSTMs are a type of recurrent neural network that is perfect for learning patterns over time. LSTMs process data sequentially, considering the order of the input sequence. This enables them to capture seasonality. In addition, LSTM models have hidden stats that process the data and store the information in the long-term memory after that, and this feature enables the model to learn and remember long-term patterns such as seasonality.

What are some typical standardization methods for LSTM?
Normalization is a common preprocessing step in machine learning that helps the model converge faster and sometimes improves performance. Here are some of the typical standardization methods for LSTM:

1. Z-score Normalization: Calculate the mean and standard deviation of each feature in the training set and standardize each feature by subtracting the mean and dividing by the standard deviation. For the test data, we apply the same mean and standard deviation we found from the train data set for normalization.

2. Min-Max scaling: Scale the features to a specific range, often [0, 1]. This method is useful when we have a clear understanding of the data range and its maximum and minimum values. It is not suitable for time series or any other data type that the test data might have a higher maximum or smaller minimum than the train data.

3. Robust Scaling: This method is less sensitive to outliers compared to Z-score normalization. It uses the interquartile range (IQR) to scale the features. The formula is: xscaled=(x-Q1)/(Q3-Q1) where Q1 is the first quartile and Q3 is the third quartile.

4. Log Transformation: If the data has a skewed distribution, a log transformation can be applied to make it more symmetric.

5. Batch Normalization: Batch normalization is applied directly to the inputs of each layer during the training process. It normalizes the input of a layer by subtracting

the mean and dividing by the standard deviation, which is computed across the mini-batch.

For our model, we are going to use the Z-score normalization since it is applicable for time series data and is easy to use and interpret.