Anomaly Detection and Forecasting of Time Series : A comparative study on LSTM and Xgboost

By :
Sandeep Vemulapalli

Contents

1. Introduction

1.1 Motivation and Objectives

Anomaly detection is an essential task in time series as it can help identify unusual patterns or events that deviate from expected behaviors. It is particularly vital in domains such as financial fraud detection, intrusion detection in cyber security and predicting equipment failures in industrial processes. While it is a significant area of research, anomaly detection in time series faces challenges in distinguishing normal variations from actual variations,trends, handling missing values and dealing with huge volumes of data. Time series data is very susceptible to changes and can exhibit patterns that are not immediately apparent in other data types. Hence, new methods, state of the art tools and techniques are required to address these obstacles and improve the reliability and accuracy of anomaly detection algorithms.

1.2 Overview of the Problem

The goal of the project is to detect anomalies using LSTM and then use Xgboost, LSTM to forecast closing values of the apple stock data after removing anomalies detected. The main purpose of this project is to improve accuracy and compare the performance of the models used.

1.3 Related Work

Anomaly detection is a feature engineering technique and data scientists spend most of their time in this aspect. This has been a topic of research in the intellectual community for many years. Various techniques have been proposed like statistical methods, Machine learning Algorithms and deep learning models. For Example, statistical methods such as ARIMA and Holt-winters methods are widely used for forecasting. However, these methods have limitations of their own like handling complex patterns in data.
Machine Learning algorithms such as SVM, Random Forest have also been used for anomaly detection in time series but they may not perform well in situations where there are missing values or outliers.
Deep Learning algorithms like LSTM that can capture complex dependencies have been proposed recently and they have shown promising results when data contains trends, seasonality and noise.

2. Data Collection and PreProcessing

2.1 DataSet

I have used Alpha Vantage API to retrieve historical stock data for the ticker symbol "AAPL"(Apple Inc.) and stored it in a CSV file. The function Time_Series_Daily_adjusted is used, requests.get() function is used to send the API request to the endpoint URL with the defined parameters and the API key parameter has been set to the Private Key. The obtained parsed data in JSON format has been processed and converted to CSV format using to_csv() function.

The data frame contains Date, Open, High, Low, Close, Adjusted close, Volume, Dividend amount and split coefficient from 24-10-2003 till 24-03-2023. In total there are 4887 instances/entries in the dataframe. Open attribute gives the opening price of stock on a traded day, High gives the highest value of the stock on a traded day, Low gives the lowest value of stock on a traded day. Adjusted_close is the closing price of the stock on a given day, adjusted for any corporate actions such as stock splits, dividends, or other distributions. This adjusted price is used to calculate the return on investment (ROI) of the stock, as it takes into account the effects of these corporate actions on the price of the stock. Split Coefficient: The ratio of the new shares to the old shares resulting from a stock split. For example, if a stock split 2-for-1, the split coefficient would be 0.5, indicating that each old share was split into 2 new shares. The split coefficient is used to adjust the historical prices of the stock for the effect of the split, so that the prices can be compared across the split without distorting the actual returns.

2.2 Data Cleaning and Transformation

Since I am modeling the close price of the stock, I have removed the other columns. Additionally, the Date column has not been in the dataframe, so I had to rename the column with

Date and additionally, I have indexed my data with the date for ease of doing the EDA and modeling.

3. Exploratory Data Analysis

3.1 Trends, Seasonality and Randomness

Time Series data is a collection of observations over time. When analyzing a time series, it is often useful to break it down into its underlying components: trends, seasonality and noise.

3.1.1 Trends

Trends are long term in the data over time and these can be upward or downwards, linear or nonlinear. Identifying trends can be very helpful especially when forecasting or understanding the impact of external factors.

3.1.2 Seasonality

Seasanality refers to regular patterns that repeat over a fixed time interval which can be yearly, monthly and even weekly.

3.1.3 Noise

Noise also known as residuals or errors, represents the randomness that cannot be attributed to either trends or seasonality. Identifying and removing noise can help in making more accurate forecasts.

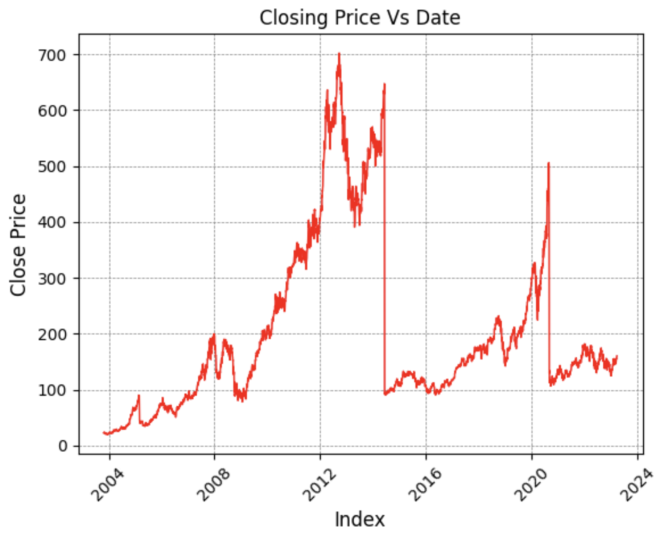


Fig 3.1 closing price vs time

3.2 STL Library

(Seasonal-Trend Decomposition Procedure based on Loess) is a statistical tool used for time series decomposition. It uses a method called locally weighted regression (Loess) to fit a smooth curve to the data then decompose the time series by subtracting the fitted components from the original data. From the below figures, it is clear the data does not contain clear trends, seasonality because the seasonal and noise components are nearly identical.

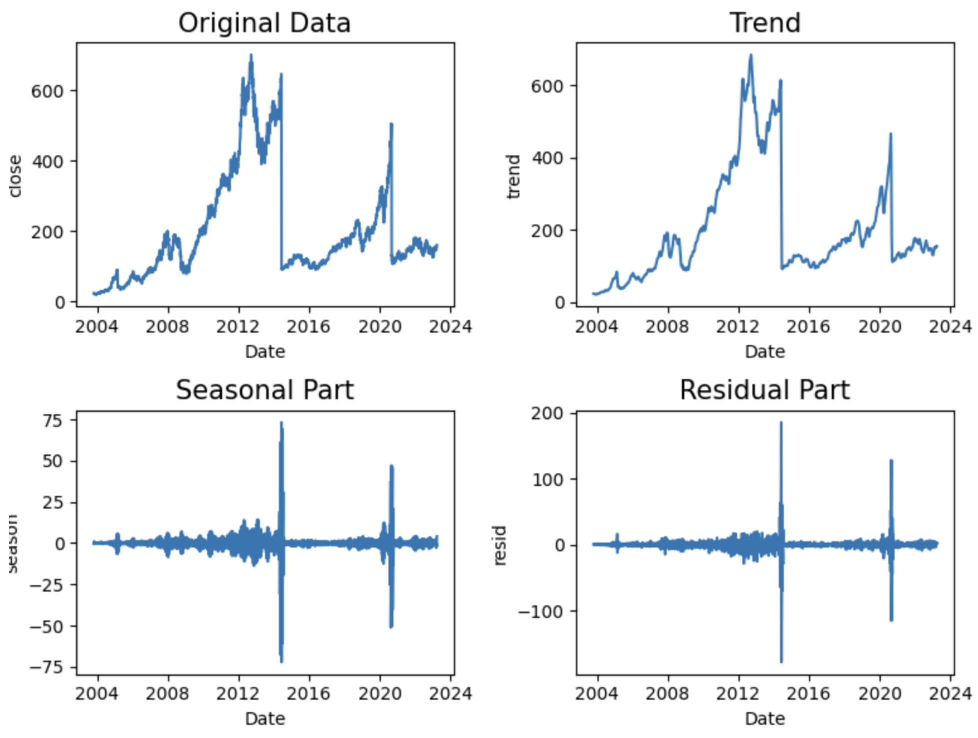


Fig 3.2.1 Trends, Seasonality decomposition of Data

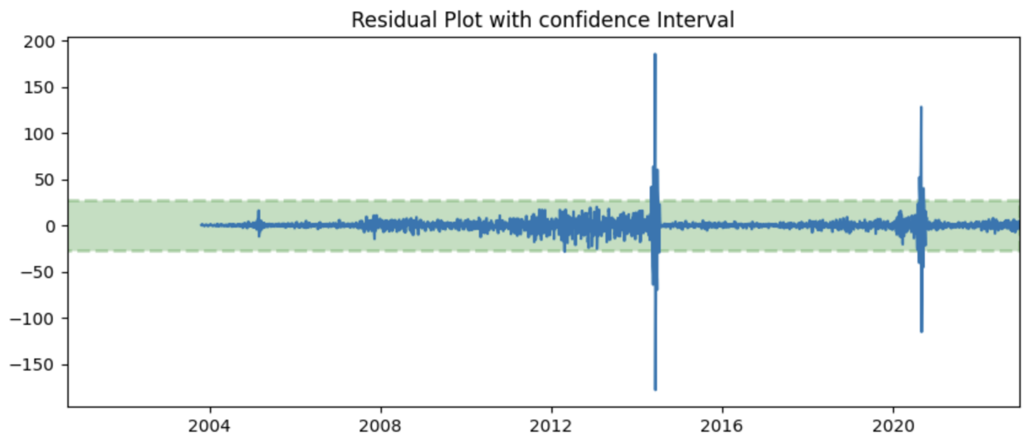


Fig 3.2.2 Residuals plotted with noise

3.3 Visualizations and Summary Statistics

The below Histogram shows the distribution of closing value of Apple stock over time. x-axis represents the close price and y-axis shows the frequency. It is skewed at the tail telling us that the majority of times, the value is between 100 to 200 dollars.

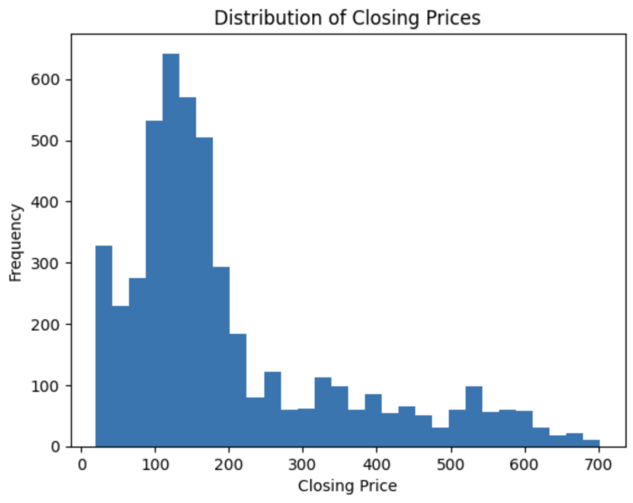


Fig 3.3 Histogram of closing price values

The below line graph is daily returns of stock over the time period. We can understand that on some dates the returns are negative and might have caused loss to the investors. Those time frames are exactly the dates when the stock price fell sharply from the previous day. They might fall under the anomalies category.

The next line graph shows the 30 day moving average. I have done this to smooth out the short term fluctuations and highlight long term trends. This plot is useful for visualizing the trend in the closing price of the stock over time, and to see how it compares to its 30-day rolling mean.

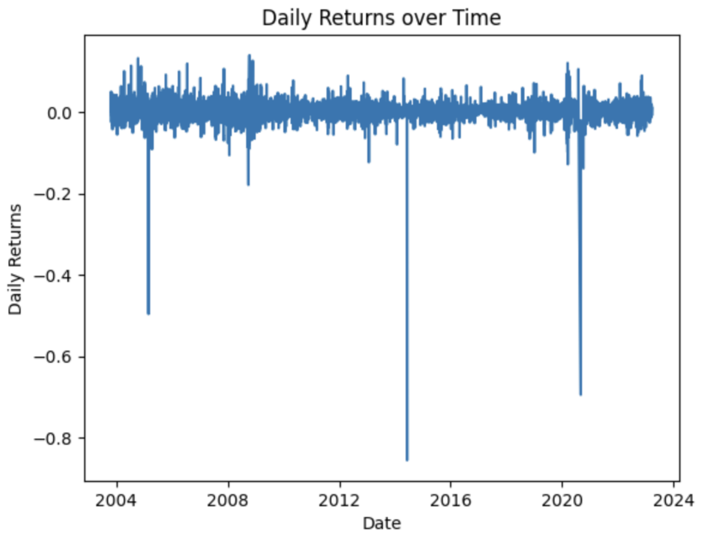


Fig 3.3.1 Daily returns of stock

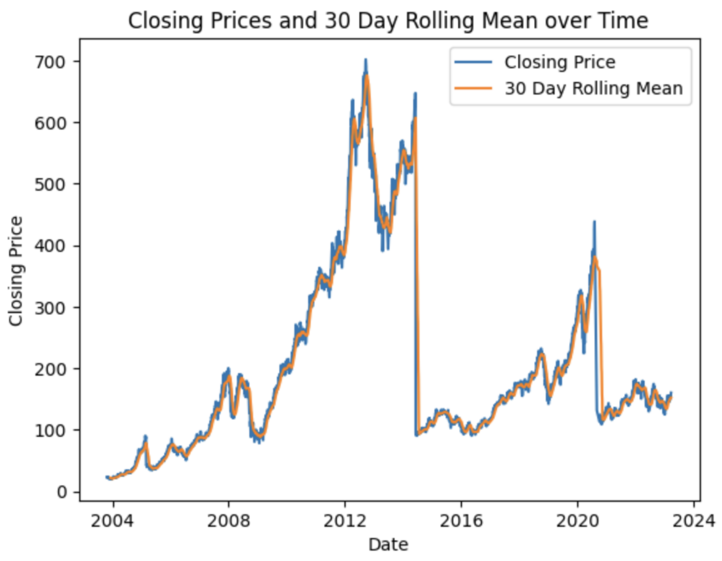


Fig 3.3.2 30 day rolling mean and Closing price

4. Feature Engineering

In the Project for forecasting using Xgboost section, I have added various features in order to use them as input for the model namely ema_9, sma_7,sma_14,sma_21 and sma_30 calculated by varying time periods from 9,7,14, 21, 30 respectively. Moving averages is a widely used tool in identifying trends and potential shifting points in closing stock price data as they help to smooth the data with significant variations.

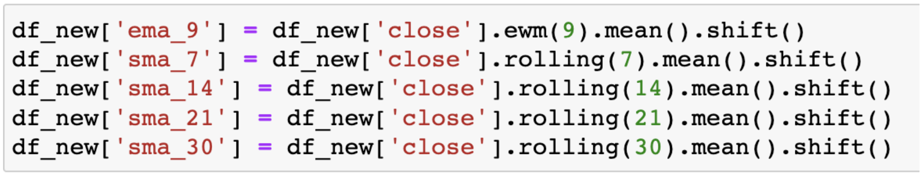
4.1 Exponential Moving Averages

The EMA calculation was done using the 'ewm' method which is defined as exponential Exponential Weighted moving average. Applying a greater weight to the recent data points compared with older data with exponentially decreasing weights is the strategy underlying principle behind this technique. The 'mean()' function calculates the average of the weighted data points resulting in a single EMA value for each time frame with that corresponding window.

4.2 Simple Moving Averages

The SMA calculation was done using the 'rolling' method, which generates a rolling window of the size taken and thus calculating average of the data points in the particular window using the '.mean()' function resulting in a single SMA value for every time frame.

Finally, for the above two methods after the mean is calculated, I have shifted the calculated moving averages one time period by one inorder to align the moving averages with stock price and to plot them alongside the closing column of the data. This step was purely done for the ease of gaining insights from plot and the whole set was shifted back by one step before model building. EMA is better suited for short term trends because of the fact it places higher weights on recent data and on the other hand SMA is suitable for long term trends due to its linear distribution of weights over a fixed time period.



```
df_new['ema_9'] = df_new['close'].ewm(9).mean().shift()
df_new['sma_7'] = df_new['close'].rolling(7).mean().shift()
df_new['sma_14'] = df_new['close'].rolling(14).mean().shift()
df_new['sma_21'] = df_new['close'].rolling(21).mean().shift()
df_new['sma_30'] = df_new['close'].rolling(30).mean().shift()
```

Fig 4.2 Features added for EMA and SMA

4.3 Relative strength Index

Relative strength Index(RSI) is one of the popular technical indicators used for measuring the strength of a stock's price. This is the most important feature created in order to boost the performance of the model. RSI is calculated by analyzing the magnitude of recent gains relative to the price losses and it ranges from 0 to 100. In the code snippet provided, a 14 day window was considered along with gains and losses objects taken separately.

4.4 Bollinger Bands

Bollinger bands are now being extensively used by analysts for identifying potential price movements in the financial sector by calculating the standard deviation of closing price over a time period and then plotting upper and lower bands around a moving average of the price. I have incorporated a similar approach by using the pandas library by first applying the 'rolling' method on the close column to calculate mean and standard deviation. I then used these values to calculate the upper and lower bands by adding and subtracting a multiple of the standard deviation from the rolling mean. One way to understand bands is if the price of an instrument moves close to or touches the upper band, it could be a sign that the instrument is overbought and may experience a pullback.

5. Approach and Algorithms:

5.1 Approach for Anomaly Detection

For Anomaly detection using LSTM and Forecasting using Xgboost, I have done scaling to normalize features so that they have a similar range of values, which can improve the performance and accuracy. I have used standardScalar from the Sklearn library.

5.1.1 LSTM Sequence

Creating sequential data is necessary for the LSTM model inorder to capture temporal dependencies and patterns. After the training and 20% testing split of the dataframe, I have taken 30 as window size followed by defining create_seq function which takes this window_size, input data X & target variable. For each index i in the range from 0 to the length of X minus steps_time, the function appends a slice of the X array from index i to i + steps_time to the Xs list. In this way the empty lists xs and ys will be populated for each iteration. Training data has shape (3879,30,1) and testing data has shape (948,30,1)

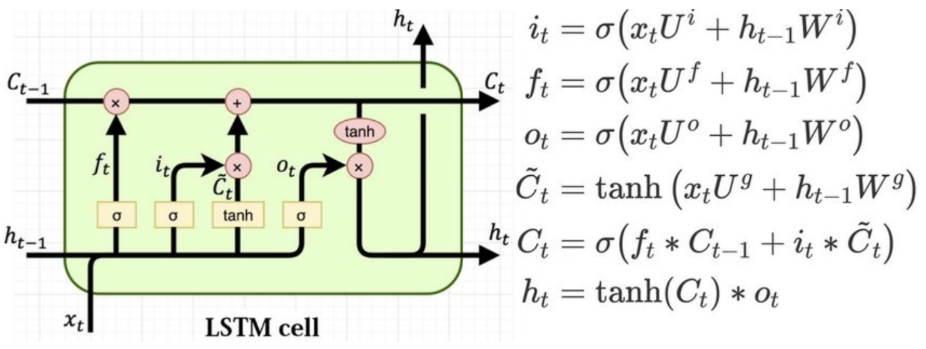
5.1.2 Algorithms for Anomaly detection:

5.1.2.1 LSTM

Long short term memory (LSTM) is a type of recurrent neural network that is commonly used for anomaly detection in time series data and forecasting tasks by learning specific patterns and relations which can then be mapped to testing data both during anomaly detection and forecasting.

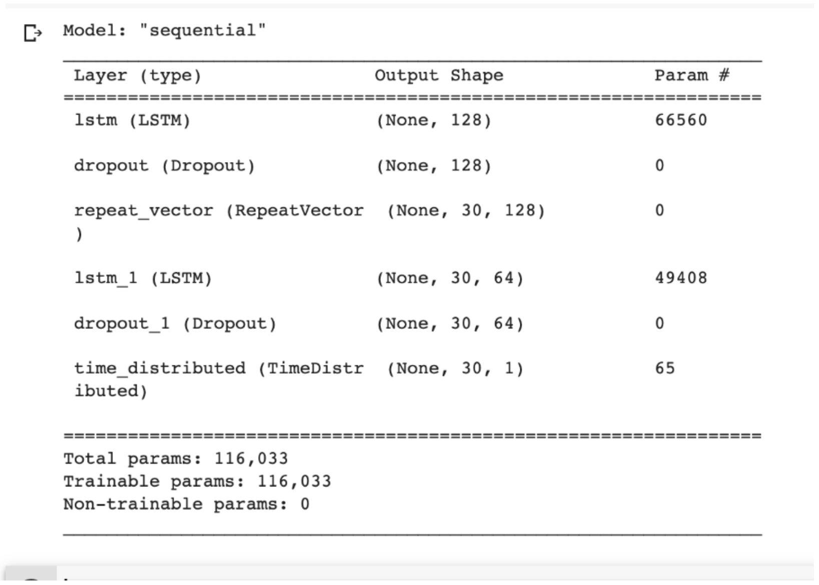
LSTM networks are equipped with three gates - input, output, and forget gates - that control the flow of information within the network. These gates act as a form of memory mechanism that allows the network to selectively remember or forget certain information at different time steps. The input gate determines which information from the current input should be incorporated into the cell state. The forget gate decides which information from the previous state should be kept or discarded. Finally, the output gate determines which information should be passed to the output. By using these gates, the LSTM network can selectively remember or forget information over time, allowing it to model long-term dependencies and perform well on tasks such as anomaly detection and forecasting.

The LSTM model architecture used consists of several layers. The first has 128 units and a rectified linear activation function (‘relu’). This was achieved using the ‘Sequential’ model class from the ‘keras.models’ module of tensorflow package. A drop out layer with rate 0.2 is added to prevent overfitting. The next layer is the RepeatVector layer which repeats the output of the first LSTM layer for each time step in the input sequence. Another LSTM layer with 64 units and the same activation function is applied after this to learn additional patterns followed by another dropout layer after this and a final Time distributed dense layer which applies a dense layer to each time step in the output sequence. The model was trained using '.fit' with mae (mean absolute error) loss function and adam optimizer. 'Earlystopping' callback from Keras was used if the validation loss does not improve for three consecutive epochs.



$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$
$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$
$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$
$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$
$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$
$$h_t = \tanh(C_t) * o_t$$

Fig 5.1.2 Structure of LSTM cell with equations describing gates, source


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 128)               66560

 dropout (Dropout)           (None, 128)               0

 repeat_vector (RepeatVector  (None, 30, 128)          0
 )

 lstm_1 (LSTM)               (None, 30, 64)            49408

 dropout_1 (Dropout)         (None, 30, 64)            0

 time_distributed (TimeDistr  (None, 30, 1)            65
 ibuted)

=================================================================
Total params: 116,033
Trainable params: 116,033
Non-trainable params: 0
_________________________________________________________________
```

5.1.2.2 Hyper Parameter Tuning

Hyper-parameter tuning is the process of selecting the best set of hyperparameters for a machine learning algorithm to optimize its performance. Hyperparameters are the parameters that are set before the training of the algorithm, such as the learning rate, number of hidden layers, number of neurons in each layer, activation functions, regularization parameters, etc. The goal of hyper-parameter tuning is to find the best combination of these parameters that results in the best model performance. This process is usually done by trying different combinations of hyperparameters and evaluating the model performance using a validation set or cross-validation. The best combination of hyperparameters is then selected based on the model's performance on the validation set.

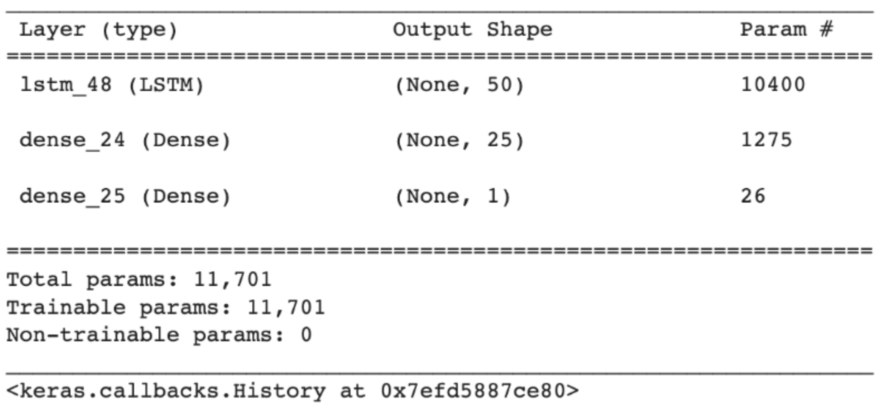
I have done hyper parameter tuning for a Keras-based regression model using GridSearchCV from scikit-learn. The model architecture includes two LSTM layers with dropouts, repeat vector, and time-distributed dense layers. The hyper-parameters optimized are the number of units, dropout rate, activation function, and optimizer. The best model achieves a mean absolute error (MAE) of -0.734907 using a combination of 128 units, 0.2 dropout rate, ReLU activation, and Adam optimizer. The performance is evaluated using a 5-fold cross-validation approach.

5.3 Forecasting

5.3.1 LSTM

For forecasting using LSTM, a 50 units input layer, ‘relu’ activation function and another hidden layer with 25 units and finally 1 unit for output layer was designed. Compared to above, this sequence is light in its architecture. The model was trained with mean squared error as the loss function to be optimized. The anomalies obtained in the above step were removed.
The approach involves splitting the input data into windows of size 10, and then using the previous 10 values to predict the next value. The training data is then split into X_train and Y_train arrays, where X_train is a 3D array of shape (number of samples, window size, 1) and Y_train is a 2D array of shape (number of samples, 1).

The model architecture consists of an LSTM layer with 50 units and a ‘ReLU’ activation function, followed by a dense layer with 25 units, and finally a dense output layer with 1 unit for the prediction. The model is compiled using mean squared error loss and the Adam optimizer.
The model is then trained for 10 epochs with a batch size of 10, and the training is done using the fit() method. The training progress is not displayed during the training process, as verbose=0 is set in the fit() method.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_48 (LSTM)              (None, 50)                10400

 dense_24 (Dense)            (None, 25)                1275

 dense_25 (Dense)            (None, 1)                 26

=================================================================
Total params: 11,701
Trainable params: 11,701
Non-trainable params: 0
_________________________________________________________________
<keras.callbacks.History at 0x7efd5887ce80>
```

Fig 5.3.1 Summary of LSTM model deployed

Xgboost

The Project uses Extreme Gradient boosting(Xgboost) as a regression model to predict the closing price. I have used Xgboost for the main reason that it is an ensemble machine learning technique that combines multiple decision trees to make final predictions. XGBoost can handle both regression and classification problems and is particularly useful when dealing with large datasets with a large number of features. It works by iteratively building decision trees and combining their outputs to make predictions. XGBoost also allows for tuning of hyperparameters to optimize model performance. I have used 10% validation and 10% sizes respectively. To train the Xgboost model, GridsearchCV is used to perform a grid search over a range of hyperparameters. Once the best combination of hyperparameters that results in the highest model performance are obtained, I fit the Xgboost model to the training data using these hyperparameters and have used the validation set to monitor the model's performance and prevent overfitting. Finally, I used the trained model to predict the closing prices on the test set. Overall, Xgboost is a powerful machine learning model that can handle large datasets with many features.

Hyper-Parameter Tuning

Using GridSearchCV, the parameters being tuned are the number of estimators, learning rate, maximum depth, and gamma value. The objective is to minimize the mean squared error (MSE) as the performance metric. After performing the grid search with cross-validation, the best parameters found were: gamma = 0.001, learning rate = 0.05, max depth = 8, and n_estimators = 100, with a corresponding best validation score.

6. Metrics and Model Evaluation

6.1 Anomaly Detection

6.1.2 LSTM

The model was trained on X_train and y_train data for 10 epochs with a batch size of 32 and a validation split of 0.1 with early stopping callback to prevent overfitting by monitoring validation loss. mean absolute error (MAE) loss between the predicted and actual values of the training data using the trained model. The MAE values are then plotted as a histogram, with the threshold value for reconstruction error set to the 95th percentile of the MAE values.

The diagram** Fig number ** shows the training and validation loss curves obtained during the training of the LSTM model. The training loss curve shows the variation of the loss function (i.e., error) of the model on the training data over each epoch. The validation loss curve shows the variation of the loss function on the validation data over each epoch. The plot helps to visualize the performance of the model during training and determine if it's overfitting or underfitting. We can see that validation loss is decreasing after epoch 2 but almost remaining constant after a certain stage indicating that the model is overfitting to the training data and not generalizing well to new data. But Overall, it is a good sign that the training loss is decreasing after epoch 1.

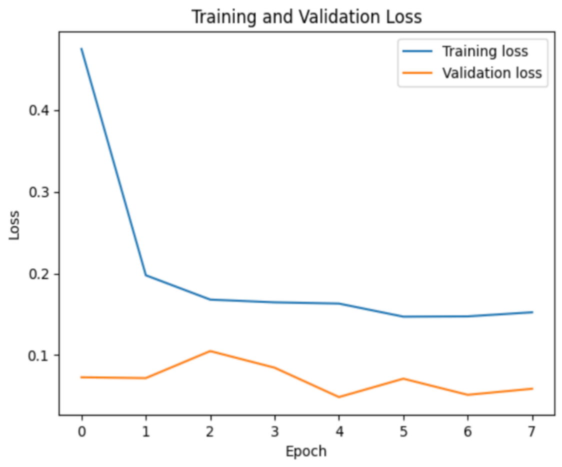


Fig 6.1.2 Training vs Validation loss

Mean Absolute Error: MAE is a commonly used metric in machine learning and statistical modeling to evaluate the accuracy of a model's predictions. It measures the average absolute difference between the predicted values and the actual values, where the absolute difference is the absolute value of the difference between the predicted and actual values. A lower MAE value indicates a better fit of the model to the data, as it means that the model's predictions are closer to the actual values.

In the context of time series forecasting, MAE is particularly useful for evaluating the accuracy of a model's predictions of future values, as it allows for a quantitative assessment of how well the model is able to forecast future values. Additionally, since MAE can also be used to identify outliers or anomalies in the data, as these can significantly increase the value of the MAE, I have used this as one of the evaluation metrics. I got an MAE of 0.113 calculated on the test set after training the LSTM autoencoder on the training set. Considering the margin for error and stock price for Apple is around 150 approximately, MAE of 0.113 is moderate.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} | y_i - \hat{y}_i |$$

Fig 6.1.2.1 Formula used for MAE calculation

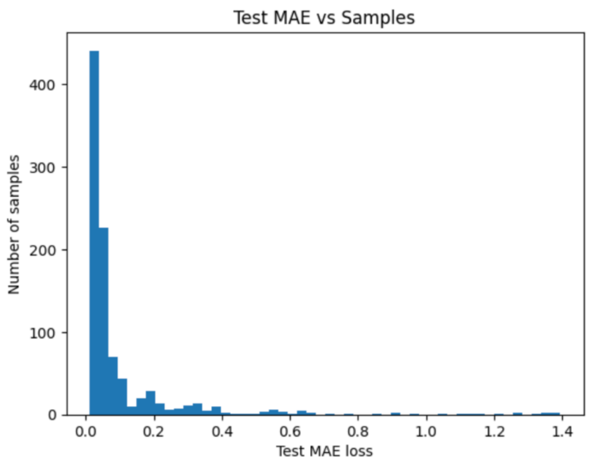


Fig 6.1.2.2 Test MAE histogram of samples

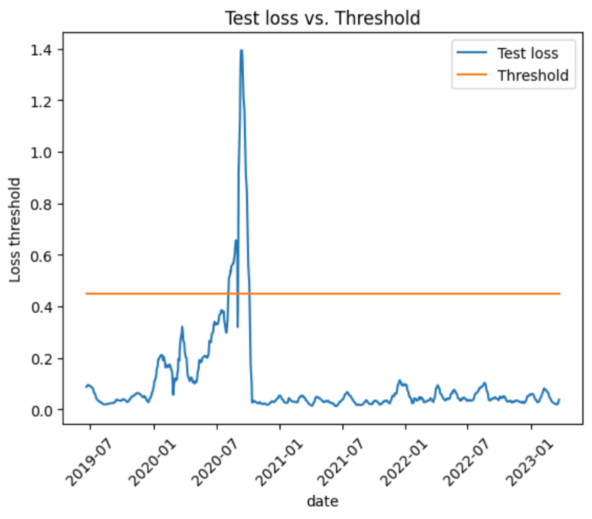


Fig 6.1.2.3 Testing Loss and Threshold for anomalies

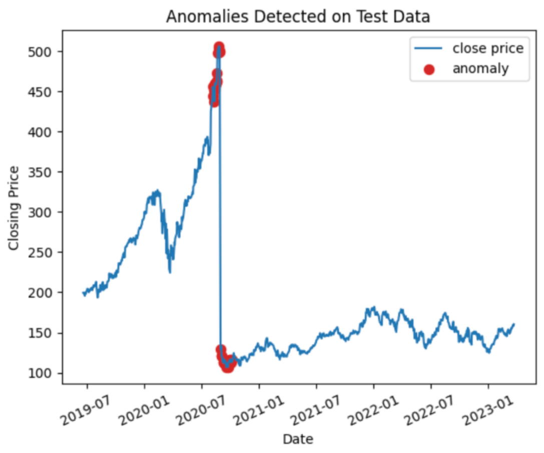


Fig 6.1.2.4 Anomalies detected for testing data

6.2 Forecasting

6.2.1 RMSE

Root Mean Squared Error (RMSE) is a commonly used metric to evaluate the performance of a machine learning model. It measures the average distance between the predicted and actual values, taking into account the magnitude of the error. RMSE is sensitive to outliers and penalizes large errors more severely than small ones. A lower RMSE value indicates better model performance, while a higher RMSE value indicates poorer performance. RMSE is a useful metric for regression problems where the goal is to predict a continuous variable.

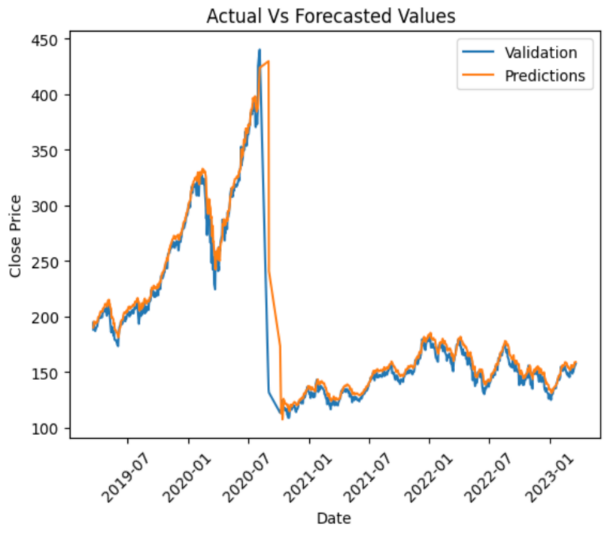


Fig 6.2.1 Actual vs Predicted values of closing price

6.2.2 Mean Absolute Percentage Error

Mean Absolute percentage error is a metric used in regression tasks to quantify the magnitude of the errors in the predictions as a percentage of the true values. In this case, the MAPE indicates the average percentage difference between the predicted and true values in the test set.

7. Results and Analysis

In the anomaly detection using LSTM, considering the margin for error and stock price for Apple is around 150 approximately, MAE of 0.113 is moderate.

In the forecasting using LSTM, the train RMSE value of 13.09 indicates that the LSTM model is able to predict the training data with an average error of 13.09. The validation RMSE value of 12.538 indicates that the model is able to predict the validation data with an average error of 12.538. As the RMSE values are relatively close and not too high, it can be concluded that the model is performing reasonably well in predicting the time series data.

Using Xgboost, The MAPE value obtained is 2.67%. This means that, on average, the model's predictions are off by 2.67% from the true values. A low MAPE value is generally desirable, as it indicates that the model is making accurate predictions. However, the interpretation of "low" or "high" MAPE values can vary in a general scenario.

The RMSE value of 4.98 in forecasting using Xgboost means that on average, the predicted stock prices deviated from the actual prices by $4.98. This could be considered a relatively low value, indicating that the model was able to accurately predict the stock prices.

Overall, Xgboost performed significantly well compared to LSTM.

8. Conclusion and Future Work

Based on the results obtained, it can be concluded that the Xgboost model used for predicting the Apple stock closing price performed reasonably well, with an RMSE of 4.98 and a mean absolute percentage error (MAPE) of 2.67%. This indicates that the model was able to capture the underlying patterns and trends in the data, making it suitable for use in real-world trading scenarios.

However, there is still room for improvement in the model's performance. One potential area of future work could be to explore the use of more advanced deep learning architectures, such as transformers, which have shown promising results in similar time series prediction tasks. Additionally, incorporating more external features, such as news sentiment data, could help to further enhance the model's predictive capabilities. Finally, the model could also be extended to predict longer-term trends, such as monthly or yearly prices, in order to provide more comprehensive forecasting insights.

9. Citations

9.1 Brownlee, J. (2017). How to develop LSTM networks for time series forecasting. Machine Mastery, https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

9.2 Jovian, (2020). Time series anomaly detection using Deep learning https://jovian.com/charmzshab-0vn/14-time-series-anomaly-detection

9.3 Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys (CSUR), 41(3), 1-58. https://dl.acm.org/doi/10.1145/1541880.1541882

9.4 Curiously, (2019) Time Series Anomaly Detection with LSTM Autoencoders using Keras in Python.https://curiousily.com/posts/anomaly-detection-in-time-series-with-lstms-using-keras-in-python/

9.5 Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc. https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

9.6 Susan Li (2020). Time Series of Price Anomaly Detection with LSTM, Towards Data science,https://towardsdatascience.com/time-series-of-price-anomaly-detection-with-lstm-11a12ba4f6d9

10. Appendix

Github Repository Link: https://github.com/sandy1597/890_Porject