Hanoi University of Science and Technology
School of Informations and Comunication of Technology

Capstone Project

Stock Price Prediction

Nguyen Quoc Thai - 20225456

Instructor: **Ms. Nguyen Thi Oanh**

January, 2025

Contents

1 Introduction

Stock prices are influenced by various factors like politics, the economy, and public sentiment. Predicting stock values is a challenging task that requires extensive research. Recent advancements in machine learning, specifically LSTM (a type of recurrent neural network) and KNN (a simpler yet effective machine learning algorithm), now enable us to build models for more accurate stock value predictions. This research aims to create and assess the predictive performance of these two models. We will use historical data, including prices, trading volume, and market sentiment, as input. The process involves data preparation (cleaning and analysis), scaling, splitting data for training and testing, evaluating model performance, and finally, making predictions.

In recent years , an increasing number of researchers have begun to explore stock price prediction methods based on data science. However, most of these studies focus on using machine learning algorithms to learn from historical data, neglecting the trend information of stock price changes. Therefore, this paper proposes a stock price prediction method, aiming to improve prediction accuracy by learning and under standing the trends of stock price fluctuations.

2 Stock Dataset

2.1 Data Collection

In this project, I use yfinance to collect the stock dataset from Yahoo Finance Website. The `data` variable now holds a Pandas DataFrame containing the historical stock data, including: `Open`, `High`, `Low`, `Close`, `Adj Close`, `Volume`

2.2 Data Preprocessing

- Data Cleaning:
  - Checked for missing values and handled them appropriately (e.g., imputation or removal).
  - Ensured data types are correct for each feature (e.g., dates as datetime objects, prices as floats).
- Feature Selection: Selected relevant features for modeling, primarily focusing on 'Close' prices and 'Date'.
- Data Transformation:
  - Converted the 'Close' prices into a suitable format for modeling.
  - Created a dataset matrix that captures the relationship between past and future prices.

2.3 Data Scaling

- Applied normalization or standardization techniques to scale the features, ensuring that the model training is efficient and effective.
- This step is crucial for algorithms like KNN and LSTM, which are sensitive to the scale of input data.

2.4 Data Splitting

Divided the dataset into training and testing sets:

- Approximately 70-80% of the data was allocated for training the models.
- The remaining 20-30% was reserved for testing and validating model performance.

2.5 Sequence Creation for LSTM

For the LSTM model, transformed the data into sequences:

- Each sequence represents a window of historical stock prices, allowing the model to learn temporal dependencies.
- Defined the input shape for the LSTM layer as [samples, time steps, features].

2.6 Final Dataset Preparation

- Ensured that the training and testing datasets are properly formatted and ready for model input.
- Verified that all preprocessing steps were completed, and the data is suitable for training the KNN and LSTM models.

3 Basic Concepts

3.1 Pandas

Pandas is a powerful Python library for working with structured data, such as tables and spreadsheets.

It provides efficient data structures (like DataFrames) and tools for data manipulation, analysis, and cleaning.

Pandas is considered a cornerstone for data analysis in Python due to its flexibility and extensive features.

3.2 NumPy

NumPy is the fundamental package for numerical computing in Python.

It provides high-performance arrays for efficient mathematical operations and scientific computing.

While primarily used for scientific purposes, NumPy can also be used as a general-purpose tool for handling multi-dimensional data.

3.3 Matplotlib

Matplotlib is a versatile library for creating 2D plots and visualizations in Python.

It's built upon NumPy arrays and integrates well with the broader scientific Python ecosystem.

Matplotlib offers a wide range of plotting options, including line plots, bar charts, histograms, and more.

3.4 Scikit-learn

Scikit-learn is a popular library for implementing various machine learning algorithms in Python.

It provides efficient and well-documented implementations of common algorithms for tasks like classification, regression, clustering, and more.

Scikit-learn is known for its user-friendly API and comprehensive documentation.

3.5 TensorFlow

TensorFlow is a comprehensive open-source platform for machine learning.

It offers a flexible ecosystem of tools and libraries for building and deploying machine learning models.

TensorFlow supports various workflows and provides APIs for both beginners and experienced users to develop and deploy machine learning solutions.

4 Models

4.1 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is employed to predict stock prices based on historical data. Initially, the dataset is preprocessed by extracting relevant features, specifically the 'date' and 'close' price columns, from the stock data. The 'close' prices are then transformed into a suitable format for modeling by creating a dataset matrix that captures the relationship between past and future prices. The dataset is split into training and testing sets, with a portion of the data reserved for validation. The KNN model is instantiated with a specified number of neighbors, in this case, two, and is trained using the training dataset. The model's performance is evaluated using the mean squared error metric, which quantifies the difference between the predicted and actual prices. Finally, predictions are made on the test dataset, providing insights into the expected stock price for a given test date. This methodology allows for a straightforward implementation of KNN, leveraging its simplicity and effectiveness in regression tasks.

4.2 Long Short Term Memory (LSTM)

The initial phase of building an LSTM model for predicting AAPL stock prices involves importing necessary libraries, including TensorFlow and Keras. The dataset is preprocessed to ensure it contains essential information such as opening and closing prices, adjusted close prices, and trading volumes. Once the data is split into training and testing sets, approximately 70–80% is allocated for training purposes. The data is then transformed into sequences, each representing a window of historical stock prices.

For the LSTM model architecture, we utilize the Keras Sequential API. Typically, this model includes an LSTM layer followed by one or more dense layers. The input shape of the LSTM layer is defined by the sequence length, which indicates the length of each input sequence, and the number of features used for prediction. To mitigate overfitting, dropout layers can be incorporated into the model.

The model is built using a loss function, commonly mean squared error (MSE), suitable for regression tasks, along with an appropriate optimizer such as Adam. Hyperparameters, including batch size and learning rate, are fine-tuned to enhance performance. Subsequently, the model is trained on the training

dataset to learn how to predict future stock prices based on historical data. Finally, evaluation metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are calculated to measure accuracy and generalization, with the model's performance being validated using the testing dataset. Effectively constructing robust LSTM-based stock price prediction models for AAPL necessitates careful model design.

- **Input Layer:** This layer receives the input data, which is typically reshaped to fit the LSTM's expected input format of `[samples, time steps, features]`.
- **LSTM Layer:** The core layer of the model that processes the input sequences and captures temporal dependencies in the data.
- **Dense Layer**: These layers are fully connected layers that follow the LSTM layer(s) and are used to produce the final output.
- **Dropout Layer**: Used to prevent overfitting by randomly setting a fraction of the input units to zero during training.
- **Output Layer:** The final layer that outputs the predicted value.

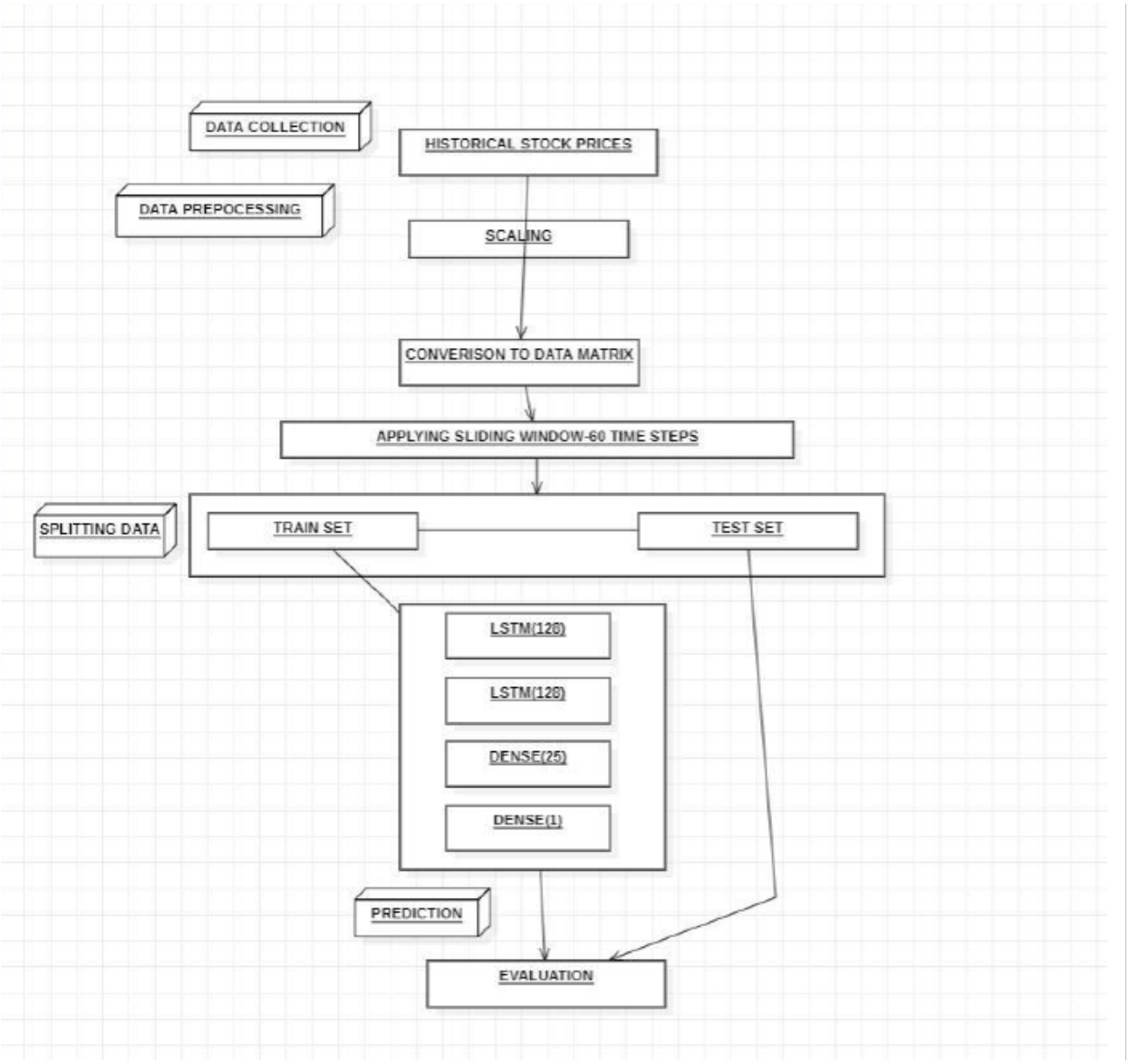

Figure 1:
LSTM Layer

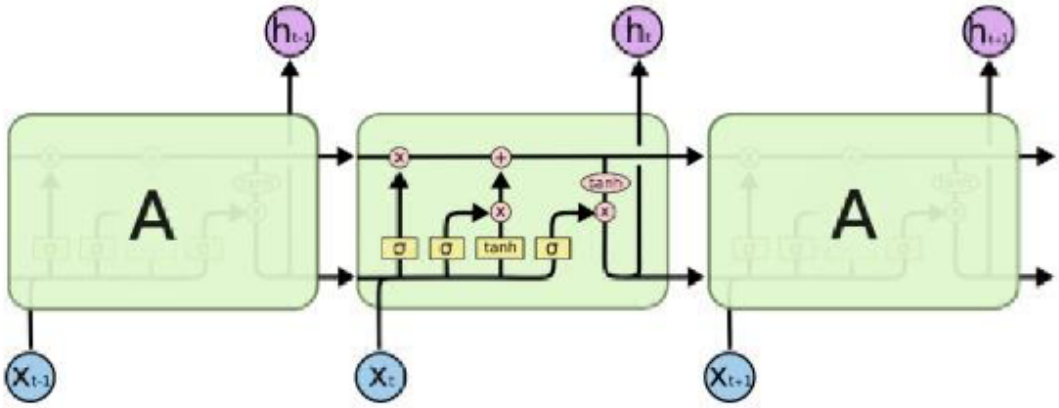

Figure 2:
LSTM Model

The LSTM architecture includes several key components, each with its own set of equations. Below are the main formulas used in an LSTM cell: - Forget Gate:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

When :

- (f_t): Forget gate activation (values between 0 and 1)
- (W_f): Weight matrix for the forget gate
- (h_{t-1}): Previous hidden state
- (x_t): Current input
- (b_f): Bias for the forget gate
- (σ): Sigmoid activation function

- Input Gate :

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

When :

- (i_t): Input gate activation
- (W_i): Weight matrix for the input gate
- (b_i): Bias for the input gate

- Candidate Cell State :

$$\tilde{C}t = \tanh(W_C \cdot [ht - 1, x_t] + b_C)$$

When :

- ($\tilde{C}_t$): Candidate cell state
- (W_C): Weight matrix for the candidate cell state
- (b_C): Bias for the candidate cell state
- ($\tanh$): Hyperbolic tangent activation function

- Cell State Update :

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

When :

- (C_t): Updated cell state
- (C_{t-1}): Previous cell state

- Output Gate :

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

When :

- (o_t): Output gate activation
- (W_o): Weight matrix for the output gate
- (b_o): Bias for the output gate

The model's weights were optimized using the Adam optimizer with a learning rate of 0.001. The mean squared error (MSE) was used as the loss function, and mean absolute error (MAE) was monitored as an additional evaluation metric during training.

4.3 Comparison of Models

The final metrics comparison for all models is summarized below:

Model	**Mean Squared Error (MSE)**
KNN	3.5277417
LSTM	9.447683353370865

5 Experiments

5.1 Evaluation Metric

To assess the performance of a regression model, the following metrics are commonly used:

1. Mean Squared Error (MSE)

Mean Squared Error (MSE) is a widely used metric for evaluating the performance of regression models. It measures the average of the squares of the errors, which are the differences between the predicted values and the actual values.

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

where:

- (n) = number of observations
- (y_i) = actual value (true value)
- $(\hat{y}_i)$ = predicted value by the model

6 Conclusion

In this project, we successfully developed and evaluated predictive models for forecasting stock prices using advanced machine learning techniques, specifically K-Nearest Neighbors (KNN) and Long Short-Term Memory (LSTM) networks. Through a systematic approach, we began by preprocessing the stock data to ensure it contained relevant features, such as closing prices and trading volumes.

The KNN model provided a straightforward yet effective method for regression, leveraging historical data to make predictions. We assessed its performance using Mean Squared Error (MSE), which allowed us to quantify the accuracy of the predictions and identify areas for improvement.

In contrast, the LSTM model demonstrated the capability to capture complex temporal dependencies inherent in stock price movements. By utilizing a sequential architecture, we were able to train the model on historical sequences of stock prices, enabling it to learn patterns over time. The incorporation of dropout layers helped mitigate overfitting, ensuring that the model generalized well to unseen data.

The evaluation metrics, including MSE , provided valuable insights into the models' performance, highlighting the strengths and weaknesses of each approach. The results indicated that while both models had their merits, the LSTM model outperformed KNN in terms of accuracy and predictive capability.

Overall, this project underscores the potential of machine learning techniques in financial forecasting, particularly in the context of stock price prediction. Future work could explore the integration of additional features, such as technical indicators or macroeconomic variables, and the application of more sophisticated models, such as ensemble methods or hybrid approaches, to further enhance prediction accuracy. The findings from this project contribute to the growing body of knowledge in financial analytics and provide a foundation for further exploration in the field of stock market prediction.