



WEST UNIVERSITY OF TIMIȘOARA
FACULTY OF MATHEMATICS AND COMPUTER
SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in
English

BACHELOR THESIS

SUPERVISOR:
Lect. Dr. Sancira Monica

GRADUATE:
Casian Jors

TIMIȘOARA
2024

WEST UNIVERSITY OF TIMIȘOARA
FACULTY OF MATHEMATICS AND COMPUTER
SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in
English

Intelligent Systems Comparison Based on Stock Market Analysis

SUPERVISOR:
Lect. Dr. Sancira Monica

GRADUATE:
Casian Jors

TIMIȘOARA
2024

Contents

1	Introduction	6
1.1	Overview	6
1.2	Motivation	6
1.3	Problem Statement	6
1.4	Objectives	7
1.5	Unique Contributions	7
1.5.1	Hyperparameter Optimization	7
1.5.2	Validation Techniques	8
1.6	Thesis Description	8
2	State of Art	9
2.1	Stock Market Description	9
2.1.1	Influences on Stock Prices	9
2.1.2	Market Types and Trends	9
2.2	Scope and Limitations	10
2.3	Similar Applications	10
2.4	Advantages and Disadvantages	11
2.5	Comparison of Similar Applications	12
2.6	Critical Analysis	12
3	Theoretical Basis	14
3.1	Technologies Used	14
3.1.1	Python Libraries	14
3.1.2	GitHub	14
3.2	Object Oriented Programming in Python	15
3.3	Predictive Models	15
3.3.1	Neural Networks	15
3.3.2	Classification	16
4	Architecture	17
4.1	Graphical User Interface	17
4.1.1	User Interactions	17
4.2	Model View Controller Design and Class Diagram	18
4.3	Sequence Diagram	20
4.4	Data Collection and Datasets	20
4.4.1	Data Sources	20
4.4.2	Data Retrieval	20
4.4.3	Data Columns	20
4.4.4	Sample Raw Data	21
4.4.5	Column Selection for Prediction	21
4.5	Data Processing	21
4.5.1	Data Cleaning	21
4.5.2	Data Serialization	22
4.5.3	Normalization	22
4.5.4	Dataset Creation	22
4.5.5	Data Splitting	22
4.6	Models Architecture	23

4.6.1	Model Development	23
4.6.2	Model Evaluation	23
4.6.3	LSTM	24
4.6.4	Random Forest	26
4.6.5	GRU	27
4.6.6	SVM	29
4.6.7	AdaBoost	31
5	Results and Analysis	33
5.1	Hyperparameter Optimization	33
5.1.1	Optimization Process	33
5.2	Validation Optimization	34
5.2.1	Random Grids for Models	35
5.2.2	Evaluation Process	35
5.2.3	Results and Comparison	35
5.3	Comparison Results over Different Datasets	35
5.3.1	Domains and Tickers	35
5.3.2	Evaluation Process	36
5.3.3	Results and Comparison	36
5.4	Experimental Setup	40
5.4.1	Hardware Configuration	40
5.4.2	Software Environment	40
5.4.3	Datasets	41
5.4.4	Model Configurations	41
5.4.5	Execution	42
6	Conclusion	43
6.1	Summary of Findings	43
6.2	Future Work	43
6.2.1	Implementing Hybrid Models for Increased Accuracy	43
6.2.2	Implementing Custom Models	43
6.2.3	Development of a More Sophisticated Real-Time Prediction System	44
6.2.4	Implementation of a Cloud-Based System	44
7	Bibliography	45

Abstract

The stock market is a complex system influenced by various factors, making accurate prediction a challenging task. This thesis presents a novel approach to stock market prediction, leveraging advanced machine learning techniques and extensive data analysis. The motivation behind this research stems from the need for more reliable and accurate prediction models that can aid investors in making informed decisions, thereby reducing financial risks.

Current methods in stock market prediction primarily utilize statistical techniques and basic machine learning models, which often fall short in capturing the intricate patterns and non-linear relationships within the data. This research addresses these shortcomings by integrating state-of-the-art technologies, such as deep learning and natural language processing, to enhance prediction accuracy [1].

The proposed model is built upon a comprehensive literature review, identifying gaps and areas for improvement in existing approaches. A detailed contextual background is provided, outlining the functioning of the stock market and the relevance of various technological tools. The methodology section describes the data collection process, including sourcing historical stock prices and market indicators, and preprocessing techniques to ensure data quality.

Various prediction methods are explored, with a focus on the implementation of neural networks and ensemble learning techniques. Validation of the model is conducted using rigorous performance metrics, including mean absolute error and root mean square error, to ensure robustness and reliability.

Experimental results demonstrate a significant improvement in prediction accuracy compared to traditional methods. The thesis concludes with a discussion on the implications of these findings, highlighting the potential for future research and application in real-world trading systems.

In summary, this research contributes to the field of financial forecasting by presenting an enhanced prediction model that integrates cutting-edge technologies, offering a valuable tool for investors and financial analysts.

1 Introduction

1.1 Overview

The stock market is a complex system where shares of publicly-held companies are issued, bought, and sold. It serves as a critical component of the global economy, providing companies with access to capital in exchange for giving investors a slice of ownership in the company. The stock market operates through exchanges, such as the New York Stock Exchange (NYSE) and NASDAQ, where participants can trade shares and other securities.

The stock market is not a place you can visit but refers to the trading (some physical, most online) of shares representing the partial owning of companies[7]. It serves as a platform for businesses to raise capital and an indicator of the overall economic health. Stock prices fluctuate due to various factors such as supply and company performance, demand, economic conditions, and sometimes seemingly irrational influences like investor sentiment.

1.2 Motivation

Accurate stock market prediction is crucial for investors, financial analysts, and policymakers due to its impact on investment decisions and economic planning. Traditional methods, such as fundamental and technical analysis, often fail to capture the complex, non-linear patterns in market data.

With the advent of machine learning and artificial intelligence, there is an opportunity to improve prediction accuracy by leveraging large datasets and sophisticated algorithms. This research is motivated by the need to enhance traditional analysis methods with modern computational techniques [1].

By developing a robust prediction model using advanced machine learning, this thesis aims to provide more precise and actionable insights into stock market behavior. The goal is to bridge the gap between traditional financial analysis and contemporary AI-driven approaches, ultimately offering a valuable tool for market participants to make informed decisions and manage financial risks more effectively.

1.3 Problem Statement

Despite the progress in financial forecasting, current stock market prediction models often suffer from limitations such as overfitting, poor generalization to new data, and inadequate handling of the complex interactions between market variables. Traditional statistical methods and basic machine learning models are frequently unable to adapt to the dynamic nature of financial markets, resulting in less reliable predictions.

Additionally, the integration of unstructured data sources, such as news articles and social media sentiment, remains a challenge. These sources can provide valuable context and enhance predictive power but are underutilized in many existing models.

This thesis addresses these challenges by developing an advanced stock market prediction model that incorporates deep learning techniques and natural language processing to better capture market dynamics and utilize diverse data sources. The

proposed approach aims to improve prediction accuracy, generalization, and practical applicability in real-world trading environments.

1.4 Objectives

The primary objective of this thesis is to develop an application that compares the predictive performance of five of the most researched algorithms in stock market prediction. These algorithms will include a mix of traditional machine learning models and advanced deep learning techniques. The specific objectives are as follows:

1. **Algorithm Selection and Comparison:** Identify and implement five well-researched algorithms used in stock market prediction, including models such as Linear Regression, Support Vector Machines (SVM), Random Forest, Long Short-Term Memory (LSTM) networks, and Convolutional Neural Networks (CNN).

2. **Hyperparameter Optimization:** Search for and optimize the best hyperparameters for each algorithm using techniques such as grid search or random search to enhance their predictive performance.

3. **Data Integration and Preprocessing:** Integrate various data sources, including historical stock prices, economic indicators, and unstructured data like news articles and social media sentiment. Implement preprocessing steps to clean and prepare the data for analysis.

4. **Model Evaluation:** Evaluate the performance of each algorithm using rigorous metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared (R^2). Compare the results to determine the most accurate prediction model.

5. **Application Development:** Develop an application that showcases the predictive models and their performance. The application should provide real-time predictions and allow users to visualize and compare the results of different algorithms.

By achieving these objectives, this research aims to provide a comprehensive evaluation of various predictive models, optimize their performance, and deliver a practical tool for accurate stock market predictions.

1.5 Unique Contributions

This research contributes to the field of financial forecasting by presenting an enhanced prediction model that integrates cutting-edge technologies. The unique contributions of this thesis include:

1.5.1 Hyperparameter Optimization

Effective hyperparameter optimization is crucial for improving model performance. This research employs advanced techniques such as grid search, random search, and Bayesian optimization to systematically explore the hyperparameter space and identify the optimal settings for each algorithm. By fine-tuning hyperparameters, the models achieve higher predictive accuracy and robustness. The optimized hyperparameters ensure that each algorithm performs at its best, thereby enhancing the overall reliability of the prediction model.

1.5.2 Validation Techniques

Robust validation techniques are crucial for ensuring the generalizability and reliability of predictive models. This thesis utilizes k-fold cross-validation, in which the dataset is partitioned into k subsets. The model is trained and validated k times, each time using a different subset as the validation set while the remaining subsets serve as the training set. This approach minimizes the risk of overfitting and provides a more accurate estimate of model performance. Furthermore, techniques such as stratified sampling are employed to ensure that each fold accurately represents the overall dataset, thereby enhancing the robustness of the validation process.

1.6 Thesis Description

This thesis presents a comprehensive study on stock market prediction using advanced machine learning techniques. It begins with an introduction outlining the motivation, problem statement, and objectives of the study, emphasizing the need for accurate predictions and the limitations of existing models.

The thesis describes the stock market, including its functioning, factors influencing stock prices, and different market trends, providing essential background knowledge. The motivation and problem statement section delves into the specific challenges addressed by the proposed model, highlighting the inadequacies of traditional methods and the potential improvements through modern machine learning techniques.

Objectives are focusing on algorithm selection and comparison, hyperparameter optimization, data integration and preprocessing, model evaluation, and application development. Unique contributions of the research include the development of a novel prediction model, comprehensive evaluation of various algorithms, and integration of diverse data sources to enhance prediction accuracy.

The state of the art section reviews current methods and applications in stock market prediction, identifying the scope and limitations of existing approaches. The theoretical basis covers key concepts and technologies used, including object-oriented programming and predictive models.

The architecture section details the design of the proposed model, including technologies used, the graphical user interface, data collection, preprocessing, and model evaluation. Results and discussion present experimental findings, focusing on hyperparameter optimization, validation techniques, comparison results, and performance analysis.

The conclusion summarizes key findings, outlines study limitations, and suggests future research directions, contextualizing the study within the broader field of stock market prediction and providing a roadmap for further exploration and improvement.

2 State of Art

2.1 Stock Market Description

2.1.1 Influences on Stock Prices

Stock prices are influenced by a myriad of factors including economic indicators, market sentiment, geopolitical events, and company-specific news. Some of the primary variables that influence stock prices include:

- **Economic Indicators:** These include metrics such as Gross Domestic Product (GDP), unemployment rates, inflation, and interest rates. Positive economic indicators often lead to higher stock prices as they signal economic growth.
- **Market Sentiment:** Investor perception and sentiment can drive market movements. Positive sentiment can boost stock prices, while negative sentiment can lead to declines.
- **Geopolitical Events:** Events such as elections, wars, and policy changes can have significant impacts on stock markets. Investors react to the potential economic implications of these events.
- **Company-Specific News:** Earnings reports, product launches, and management changes can directly affect a company's stock price.
- **Technical Factors:** These include stock price trends, trading volumes, and patterns identified through technical analysis.

2.1.2 Market Types and Trends

The stock market experiences different phases, commonly referred to as bull and bear markets.

A **bull market** is characterized by rising stock prices and general optimism among investors. During a bull market, economic conditions are generally favorable, unemployment rates are low, and consumer confidence is high. Investors are more willing to buy stocks, anticipating further increases in prices. Bull markets can last for several months or even years.

Conversely, a **bear market** is marked by declining stock prices and widespread pessimism. In a bear market, economic conditions are often deteriorating, unemployment rates may rise, and consumer confidence drops. Investors tend to sell off stocks, fearing further declines in prices. Bear markets can also persist for extended periods.

Understanding market **trends** is crucial for investors. A trend indicates the general direction in which the market or a particular stock is moving. Trends can be upward (bullish), downward (bearish), or sideways (neutral). Identifying trends helps investors make informed decisions about buying, holding, or selling stocks.

Other essential terms include:

- **Volatility:** Describes the extent to which stock prices fluctuate over time. High volatility indicates significant price swings in a short duration, whereas low volatility suggests more consistent and stable prices.

- **Liquidity:** Measures the ease with which an asset can be quickly bought or sold in the market without impacting its price. High liquidity is characterized by a large number of buyers and sellers, facilitating swift transactions.
- **Market Capitalization:** Represents the total market value of a company's outstanding shares. It is determined by multiplying the current share price by the total number of outstanding shares.
- **Dividend:** A segment of a company's earnings paid out to shareholders. Dividends offer a return on investment and play a significant role in evaluating a stock's value.
- **Index:** A quantitative measure that tracks the performance of a selected group of stocks. Notable examples include the SP 500, Dow Jones Industrial Average, and NASDAQ Composite[8].

2.2 Scope and Limitations

Recent advancements in stock market prediction have leveraged a range of machine learning and deep learning algorithms to improve prediction accuracy. Traditional models like Linear Regression and ARIMA (AutoRegressive Integrated Moving Average) have been widely used due to their simplicity and interpretability. However, their effectiveness is limited when dealing with non-linear and complex patterns in financial data.

Modern machine learning techniques, including Support Vector Machines (SVM) and Random Forests, have shown improved performance in capturing more intricate relationships within the data. These models benefit from their ability to handle high-dimensional data and capture non-linear dependencies. For instance, SVMs have been used to classify market movements, while Random Forests have provided robust predictive performance due to their ensemble nature, reducing the risk of overfitting.

Deep learning models, such as Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNN), have gained prominence for their superior capability in modeling sequential and spatial data, respectively. LSTMs, in particular, have shown remarkable success in predicting stock prices by capturing temporal dependencies and long-term trends. CNNs, on the other hand, have been effectively used to extract features from historical price charts and news articles, contributing to more informed predictions [2].

2.3 Similar Applications

There are numerous applications and tools designed for stock market prediction, leveraging various machine learning algorithms and statistical techniques. These applications are used by individual investors, financial analysts, and institutions to make informed decisions. Here are a few notable examples:

Alpaca: Alpaca is a commission-free trading platform that offers an API for algorithmic trading. It allows users to implement and test their trading algorithms using historical data. Alpaca supports various machine learning models and provides tools for backtesting and live trading.

QuantConnect: QuantConnect is an open-source algorithmic trading platform that provides a comprehensive environment for designing, testing, and deploying trading strategies. It supports multiple programming languages, including Python and C#, and integrates with various financial data providers. QuantConnect offers a robust backtesting engine and allows users to collaborate and share their algorithms.

Kavout: Kavout is a financial technology company that utilizes AI and machine learning to provide predictive analytics for the stock market. Their flagship product, Kai, analyzes vast amounts of financial data and generates stock rankings and predictions. Kavout's platform is used by asset managers and hedge funds to enhance their investment strategies.

Numerai: Numerai is a hedge fund that crowdsources stock market predictions from data scientists worldwide. Participants in the Numerai tournament use machine learning models to submit their predictions, which are then aggregated to make investment decisions. Numerai uses encrypted data to ensure privacy and provides monetary incentives for accurate predictions.

These applications highlight the growing trend of leveraging machine learning and AI for stock market prediction. By analyzing historical data and identifying patterns, these tools aim to improve the accuracy of stock price forecasts and provide valuable insights for investors [4].

2.4 Advantages and Disadvantages

Advantages:

- **Improved Prediction Accuracy:** Through optimized hyperparameter tuning and hybrid modeling, the application aims to achieve higher accuracy in stock market predictions.
- **Automated Optimization:** Bayesian Optimization reduces the need for manual parameter tuning, saving time and resources.
- **Integration of Diverse Data Sources:** Combining LSTM networks with sentiment analysis ensures a comprehensive approach to market prediction, leveraging both numerical and textual data.
- **Enhanced Generalization:** Advanced optimization techniques help prevent overfitting, ensuring better performance on unseen data.

Disadvantages:

- **High Complexity:** Implementing advanced machine learning models and integrating diverse data sources can be complex and time-consuming.
- **Data Dependency:** The accuracy of the predictions depends on the quality and quantity of the data available.
- **Computationally Intensive:** Advanced machine learning models require significant computational resources for training and evaluation.
- **Sensitivity to Market Volatility:** Machine learning models may struggle to adapt to sudden market changes and high volatility.

2.5 Comparison of Similar Applications

The table below contains advantages and disadvantages of similar applications (Table 1).

Application	Advantages	Disadvantages
Alpaca	<ul style="list-style-type: none">- Commission-free trading platform- API for algorithmic trading- Supports various machine learning models- Tools for backtesting and live trading	<ul style="list-style-type: none">- Limited hyperparameter optimization capabilities- Users must manually tune parameters- Basic grid search methods for optimization
QuantConnect	<ul style="list-style-type: none">- Open-source platform- Supports multiple programming languages- Integrates with various financial data providers- Robust backtesting engine	<ul style="list-style-type: none">- Lacks built-in advanced hyperparameter optimization- Time-consuming manual parameter tuning
Kavout	<ul style="list-style-type: none">- Utilizes AI and machine learning for predictive analytics- Analyzes vast amounts of financial data- Generates stock rankings and predictions	<ul style="list-style-type: none">- Proprietary algorithms and data sources- Limited transparency in model performance
Numerai	<ul style="list-style-type: none">- Crowdsources predictions from data scientists- Uses machine learning models- Aggregates predictions for investment decisions	<ul style="list-style-type: none">- Focuses on ensemble learning rather than individual model optimization- Limited hyperparameter tuning for individual models

Table 1: Similar Applications

2.6 Critical Analysis

While the applications discussed offer significant advancements in stock market prediction, they face several challenges.

Firstly, the complexity and resource demands are substantial. Implementing advanced machine learning models like those used by Alpaca, QuantConnect, Kavout, and Numerai requires significant computational power and expertise, limiting accessibility for smaller investors or firms. High computational costs also impact scalability and efficiency.

The dependency on high-quality data is another critical issue. Poor-quality or incomplete data can lead to inaccurate predictions, misleading investment decisions. Additionally, these models may struggle to adapt to sudden market shifts and volatility without frequent retraining.

The interpretability and transparency of these models are often limited. Proprietary algorithms, such as those used by Kavout and Numerai, can obscure the decision-making process, making it difficult for users to understand and trust the predictions.

Overfitting remains a concern despite advanced techniques aimed at enhancing generalization. Models performing well on historical data might not predict future market conditions accurately, especially in unprecedented scenarios.

Lastly, ethical and regulatory implications must be considered. Increasing reliance on algorithmic trading raises concerns about market manipulation and fairness. Developers must ensure compliance with evolving regulations to avoid legal issues and maintain market integrity.

In summary, while these applications offer improved prediction accuracy and automated strategies, addressing computational complexity, data dependency, model transparency, overfitting, and regulatory compliance is crucial for their effective and reliable use.

3 Theoretical Basis

3.1 Technologies Used

3.1.1 Python Libraries

Python is the primary programming language used in this research, leveraging its extensive libraries and frameworks. The following libraries are integral to the implementation of the models and data processing:

Yahoo Finance API The Yahoo Finance API is used to extract historical stock prices. This API provides comprehensive data on stock prices, trading volumes, and other financial metrics essential for developing robust predictive models.

Pandas and NumPy Pandas and NumPy are fundamental libraries for data manipulation and analysis. Pandas provide powerful data structures like DataFrames for efficient data handling, while NumPy supports numerical operations and array handling, crucial for preprocessing financial data.

Scikit-learn Scikit-learn is used for data preprocessing and normalization. It offers tools for scaling data, ensuring consistency, and improving the performance of machine learning algorithms.

Keras and TensorFlow Keras, built on top of TensorFlow, is used for model training and development. These frameworks allow for easy and fast prototyping of deep learning models, facilitating the implementation of complex neural network architectures.

Tkinter Tkinter is used to develop the graphical user interface (GUI) for the application. It provides a standard interface to the Tk GUI toolkit in Python, enabling the creation of interactive and user-friendly applications.

Matplotlib Matplotlib is used for data visualization. It provides tools for creating static, interactive, and animated visualizations in Python, helping users to analyze and compare model outputs effectively.

PlantUML PlantUML is used to create UML diagrams, which help in visualizing the architecture and design of the application. It supports the generation of sequence diagrams, class diagrams, and other UML diagrams from plain text descriptions.

3.1.2 GitHub

GitHub is used for version control and collaboration. It hosts the codebase, tracks changes, and facilitates collaboration among team members. It also ensures the reproducibility and transparency of the research by providing a platform for sharing code and documentation.

3.2 Object Oriented Programming in Python

Object-Oriented Programming (OOP) in Python is a paradigm that organizes software design around data, or objects, and classes, which represent real-world entities and relationships. OOP principles are fundamental for creating modular, reusable, and scalable software systems. The core concepts of OOP include encapsulation, inheritance, polymorphism, and abstraction.

Encapsulation involves combining data (attributes) and the methods (functions) that operate on that data into a single unit, known as a class. This practice hides the internal state of the object and exposes only a controlled interface, ensuring the internal representation is concealed from the outside. For instance, a class representing a bank account encapsulates account details and operations.

Inheritance allows a new class (child class) to inherit the properties and methods of an existing class (parent class). This promotes code reuse and establishes a hierarchy among classes. Inheritance enables the creation of specialized classes based on general ones, enhancing or modifying behavior as needed.

Polymorphism enables different classes to be treated as instances of a common interface, allowing methods to be used interchangeably across various class instances. Polymorphism is achieved through method overriding and method overloading, facilitating flexibility in code.

Abstraction simplifies complex systems by modeling classes relevant to the problem domain and working at a higher level of abstraction rather than dealing with intricate details. Abstraction involves creating simplified, high-level interfaces for complex systems, making them easier to manage and understand.

OOP in Python promotes modularity, code reuse, and the development of complex and scalable software systems by applying these principles. By leveraging OOP, developers can create robust, maintainable code that reflects real-world structures and relationships, thus making it easier to manage and evolve software projects over time.

3.3 Predictive Models

Predictive modeling is a statistical technique used to forecast future events by analyzing historical data. This process involves creating a mathematical model that leverages relevant input variables to predict an output variable. By utilizing machine learning algorithms, these models can be trained and refined to improve decision-making processes. Predictive modeling is extensively used across various industries, addressing challenges such as fraud detection, customer segmentation, disease diagnosis, and stock price prediction. There are seven main types of predictive models, including neural networks and classification models, which are commonly employed in this field[6].

3.3.1 Neural Networks

Neural network models are a specialized type of predictive modeling technique inspired by the structure and function of the human brain. These models strive to learn intricate relationships between input and output variables to make precise predictions. Neural networks are particularly effective in domains like image recognition, natural language processing, and speech recognition, supporting tasks such

as object identification, sentiment analysis, and speech transcription[6].

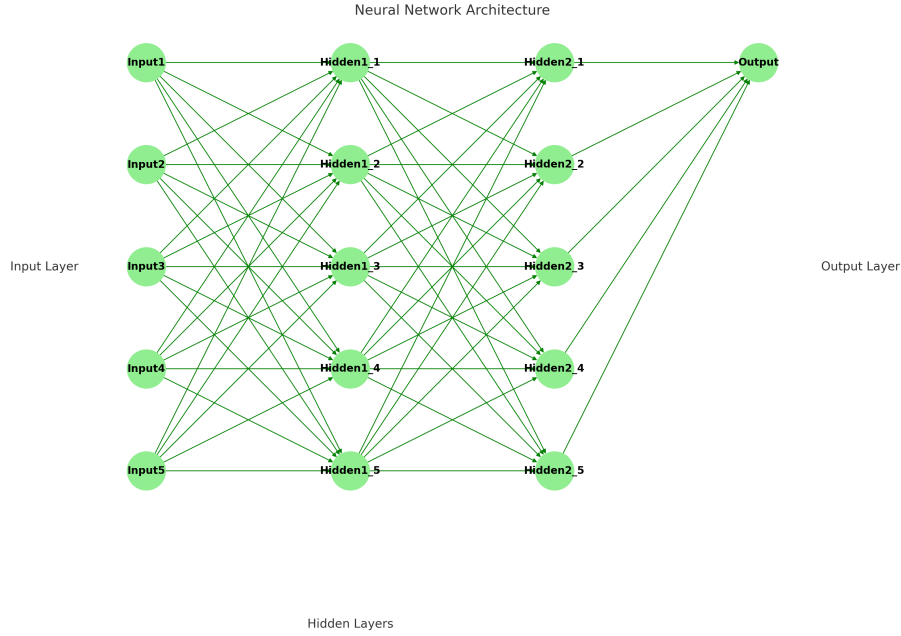


Figure 1: Neural Networks

3.3.2 Classification

Classification models are employed to categorize data into one or more groups based on one or more input variables (Figure 2). These models determine the relationship between input variables and the output variable, utilizing this relationship to accurately classify new data into the correct category. Classification models are widely used in various fields, such as marketing, healthcare, and computer vision, to classify data such as spam emails, medical diagnoses, and image recognition[6].

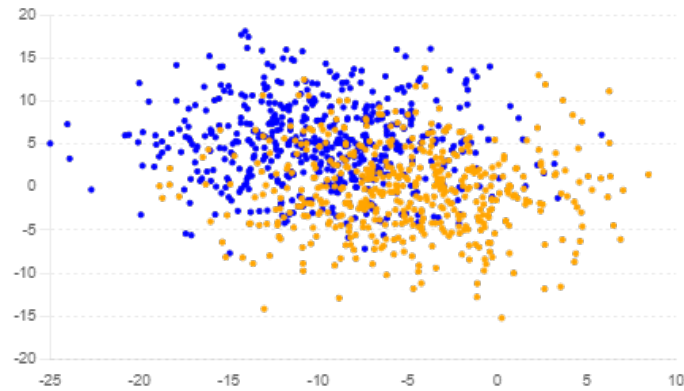


Figure 2: Classification Type

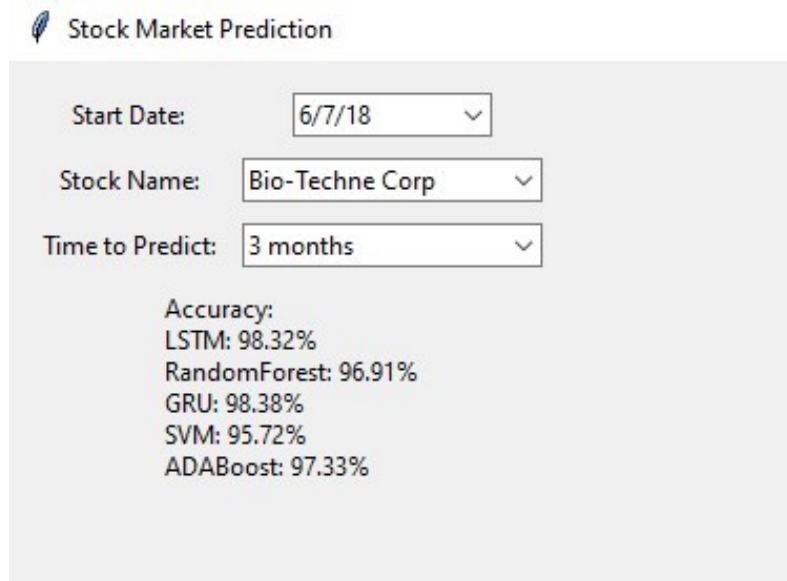
4 Architecture

4.1 Graphical User Interface

The project's User Interface was designed in Tkinter library. It consists of one view split in three parts.

4.1.1 User Interactions

On the left there are input boxes for the user to select **Start Date**, which is the date at which the stock market starts to gather data from, **Stock Name**, The name of the stock that will be predicted, **Prediction Time**, the time in future that the program will predict. Below, we have the **Accuracy table**, where the accuracy of the prediction made on the test data will be shown. (Figure 3)



The screenshot shows a Tkinter window titled "Stock Market Prediction". It contains three input fields: "Start Date" with a dropdown menu showing "6/7/18", "Stock Name" with a dropdown menu showing "Bio-Techne Corp", and "Time to Predict" with a dropdown menu showing "3 months". Below these fields is a section titled "Accuracy:" containing a table of accuracy values for different models.

Model	Accuracy
LSTM	98.32%
RandomForest	96.91%
GRU	98.38%
SVM	95.72%
ADABOOST	97.33%

Figure 3: Left side of the GUI

In the middle there is a figure realized by matplotlib, where the graphs of the historical data and the prediction will be shown. Below the figure. The user can find a table containing the evaluation metrics. On the first column, the model name, and on the next 5 columns, the MAE, MSE, RMSE, MAPE and R2. Under it, there is the progress bar of the application. Each time the application will take some time to process information, the progress bar will activate to ensure the user the application is working well. (Figure 4)

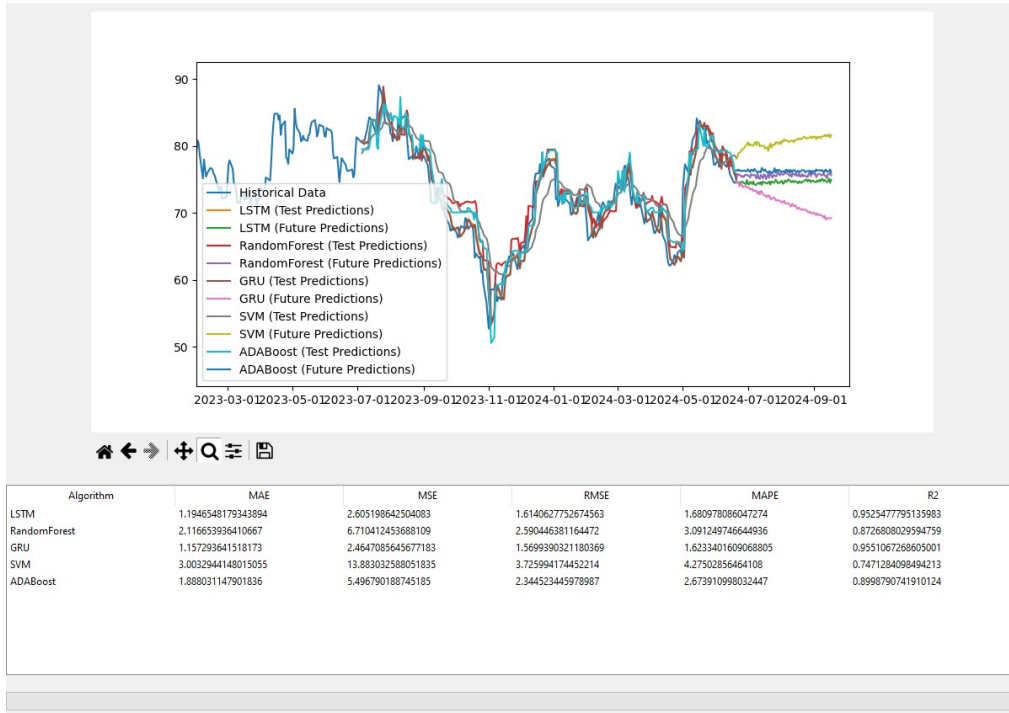


Figure 4: Middle section of the GUI

On the right there are checkboxes with the 5 models that the user can use. After all steps are completed, the user can click on the Submit button to generate the graphs of the predictions. (Figure 5)

Select Models:

☒ LSTM

☒ RF

☒ GRU

☒ SVM

☒ ADAB

Operations:

☐ Average

Submit

Figure 5: Right side of the GUI

4.2 Model View Controller Design and Class Diagram

The Model-View-Controller (MVC) design pattern is a software architectural framework that divides an application into three interconnected components: the Model, the View, and the Controller. This division helps in managing the complexity of applications by decoupling data management, user interface, and user interactions.

Model: The model represents the application's data and business logic. It directly handles the data, logic, and rules of the application. For instance, in a stock market prediction application, the model would encompass the machine learning algorithms and data processing functions.

View: The view is responsible for displaying data to the user in a specified format. It defines the user interface and presents data from the model to the user. In the context of this application, the view would be the graphical user interface (GUI) that displays stock predictions, metrics, and visualizations.

Controller: The controller serves as an intermediary between the model and the view. It processes user inputs, updates the model accordingly, and returns the output to be displayed by the view. The controller orchestrates the data flow between the model and the view (Figure 6).

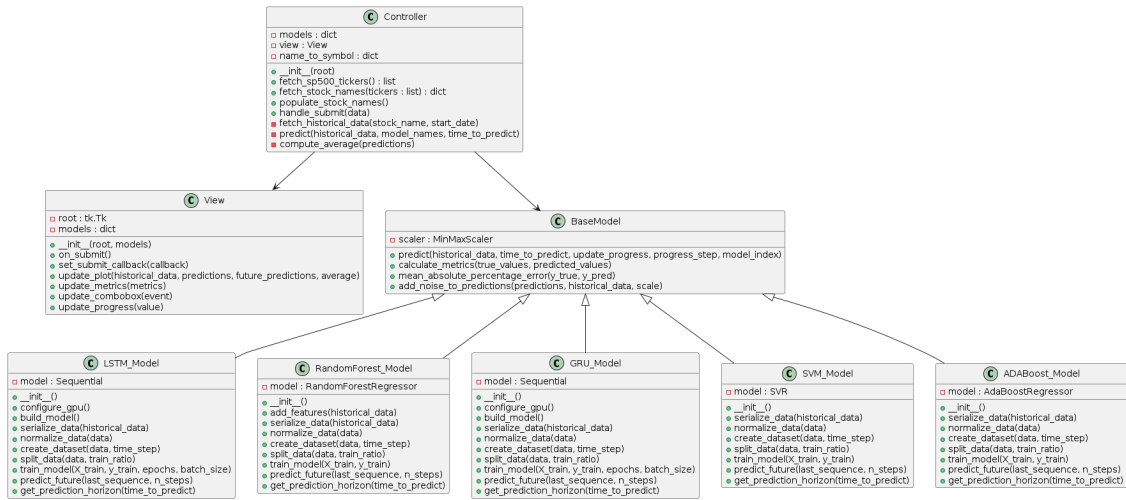


Figure 6: Class Diagram for MVC Design

In the given Python application, the `controller.py` file contains the Controller class, which manages the interaction between the View and the Model classes. Here is a brief explanation of the main functions within the `controller.py` file:

__init__: Initializes the Controller class, sets up the models, and associates the view with these models. It also sets a callback for the submit button and populates stock names.

fetch_sp500_tickers: Fetches the list of S&P 500 tickers from Wikipedia.

fetch_stock_names: Fetches stock names for the given tickers using the `yfinance` library and updates the view's progress bar.

populate_stock_names: Starts a new thread to populate stock names asynchronously.

handle_submit: Handles the submit action, starts a new thread to process the data and generate predictions.

fetch_historical_data: Fetches historical stock data for a given stock name and start date using `yfinance`.

predict: Generates predictions using the selected models and computes their metrics.

compute_average: Computes the average prediction from multiple models.

In summary, the MVC design pattern in this application ensures a clear separation of concerns, making the code more modular, easier to maintain, and scalable.

Each component (model, view, and controller) has a distinct role, facilitating collaboration and development efficiency.

4.3 Sequence Diagram

The following sequence diagram illustrates the interactions between the user, view, controller, and various model classes in the stock market prediction application (Figure 7). It depicts the process from starting the application, initializing components, to handling a submit action for stock prediction.

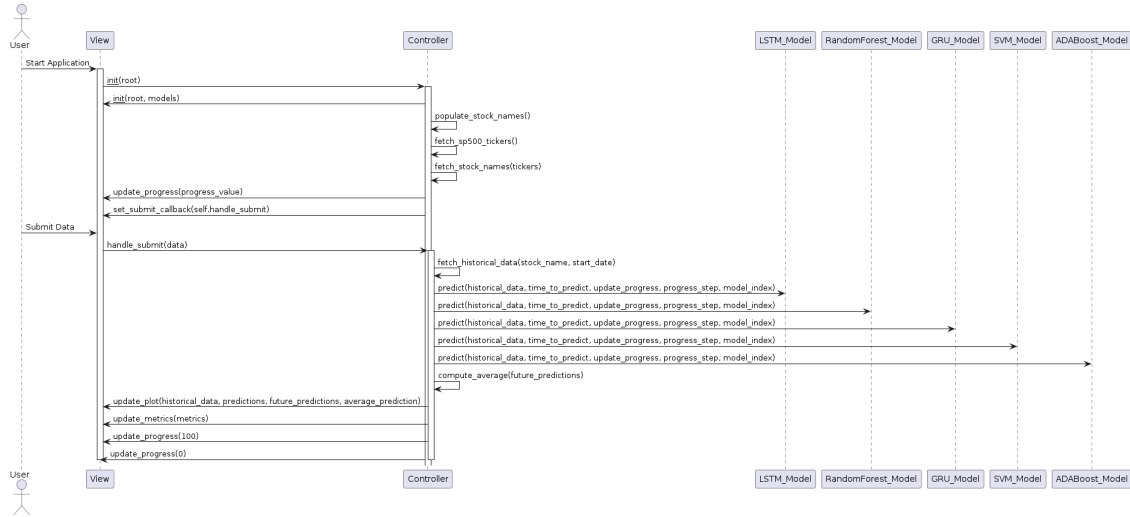


Figure 7: Sequence Diagram

4.4 Data Collection and Datasets

4.4.1 Data Sources

The application uses datasets from three major stock indices: NASDAQ, S&P 500, and DJIA (Dow Jones Industrial Average). These indices provide a comprehensive overview of the stock market, encompassing a wide range of companies from various sectors.

4.4.2 Data Retrieval

Historical stock prices are retrieved from Yahoo Finance using its API. Yahoo Finance provides extensive historical data for stocks, which is essential for training and testing predictive models. The data includes six columns: Open, High, Low, Close, Adj Close, and Volume.

4.4.3 Data Columns

The six columns retrieved from Yahoo Finance are defined as follows:

- **Open:** The stock price when the trading day began.
- **High:** The highest stock price during the trading day.

- **Low:** The lowest stock price during the trading day.
- **Close:** The stock price when the trading day ended.
- **Adj Close:** The closing price adjusted for corporate actions such as stock, dividends, splits, and rights offerings.
- **Volume:** The number of traded shares during the trading day.

4.4.4 Sample Raw Data

Here is a sample of the raw data retrieved from Yahoo Finance for a particular stock (Table 2):

Date	Open	High	Low	Close	Adj Close	Volume
2023-01-02	150.00	152.00	148.00	151.00	151.00	1,200,000
2023-01-03	151.50	153.00	149.50	150.50	150.50	1,100,000
2023-01-04	150.75	152.50	149.00	151.75	151.75	1,300,000
2023-01-05	152.00	154.00	150.50	153.00	153.00	1,400,000
2023-01-06	153.50	155.00	151.75	154.50	154.50	1,500,000

Table 2: Sample Raw Data from Yahoo Finance

4.4.5 Column Selection for Prediction

Among the six columns, the **Close** column is chosen as the target variable for prediction. The closing price is a crucial indicator of the stock's daily performance and is commonly used in financial analysis and forecasting. Predicting the closing price can provide valuable insights for investors and help them make informed trading decisions.

The data preprocessing steps involve handling missing values, normalizing the data, and splitting it into training and testing sets. These steps ensure that the data is clean and suitable for use in machine learning models. The historical closing prices are then used to train the models, which learn to identify patterns and make future price predictions.

4.5 Data Processing

4.5.1 Data Cleaning

Pandas and NumPy: Pandas and NumPy are fundamental libraries in Python for data manipulation and analysis. They are used to handle missing values, outliers, and noise in the data. Pandas provide powerful data structures like DataFrames, which allow for efficient data cleaning and transformation operations. NumPy supports numerical operations and array handling, which are crucial for preprocessing financial data.

4.5.2 Data Serialization

Data serialization involves extracting the 'Close' values from the historical stock data. This step is crucial as it isolates the target variable that the predictive models will focus on. The extracted 'Close' values are then reshaped into a format suitable for further processing. This typically involves converting the data into a one-dimensional array or series, which can be easily manipulated and fed into machine learning algorithms.

4.5.3 Normalization

Normalization is a key step in data preprocessing, especially for neural network models. The `MinMaxScaler` from the Scikit-learn library is applied to transform the data into the range $[0, 1]$. This scaling ensures that all input features contribute equally to the model's learning process and helps in accelerating the convergence of the neural network during training. Normalization also prevents any single feature from disproportionately influencing the model, thereby improving the overall performance and stability.

4.5.4 Dataset Creation

The dataset creation step uses a sliding window approach to generate sequences of data for the model. This involves creating input features (X) and corresponding target values (y) from the 'Close' prices. A default time step of 60 is used, meaning each sequence consists of 60 consecutive 'Close' values as input, with the subsequent 'Close' value as the target. This approach captures the temporal dependencies in the data, allowing the model to learn from historical trends and patterns effectively.

4.5.5 Data Splitting

Data splitting is essential for evaluating the performance of the predictive models (Figure 8). The preprocessed data is divided into training and testing sets based on a specified training ratio, which defaults to 0.8. This means 80% of the data is used for training the model, while the remaining 20% is reserved for testing. The role of the training set is to fit the model, and the role of the testing set is to verify its predictive accuracy. Splitting the data ensures that the model is tested on unseen data, providing a realistic measure of its performance and generalization capability.

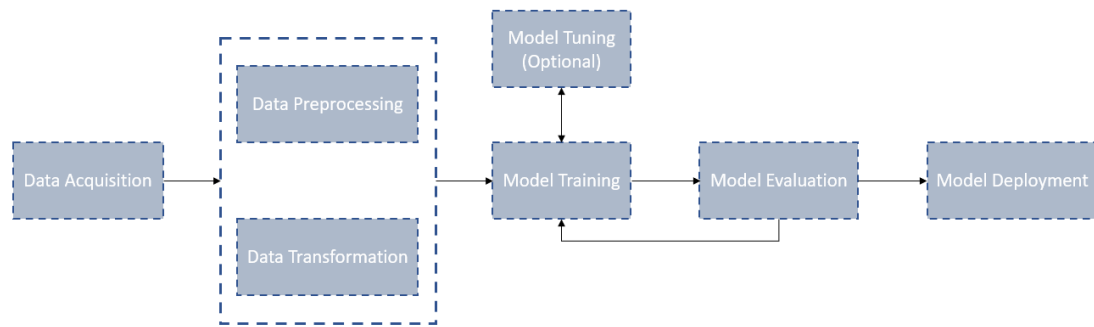


Figure 8: Data Processing Pipeline ¹

4.6 Models Architecture

4.6.1 Model Development

- **Algorithm Selection:** For this research, five well-researched algorithms are chosen: ADA Boost, Support Vector Machines (SVM), Random Forest, Long Short-Term Memory (LSTM) networks, and Convolutional Neural Networks (CNN). These algorithms represent a mix of traditional machine learning and advanced deep learning techniques, providing a comprehensive evaluation of different approaches to stock market prediction.
- **Model Training:** Keras, a high-level neural networks API, is used for model training. It allows for the easy and fast prototyping of deep learning models. Keras is built on top of TensorFlow, providing a user-friendly interface for developing complex neural network architectures such as LSTM and CNN. .

4.6.2 Model Evaluation

- **Performance Metrics:** Models are evaluated using metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared (R^2). These metrics provide a quantitative assessment of the prediction accuracy and the ability of the models to generalize to new data.
- **Cross-Validation:** Cross-validation techniques are used to ensure model robustness and generalization. By dividing the data into training and validation sets multiple times, we can assess the consistency and reliability of the models' performance.

¹Source: <https://towardsdatascience.com/simplify-machine-learning-workflows-e9d4f404aaeb>

4.6.3 LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to model sequential data by capturing long-term dependencies. LSTMs address the vanishing gradient problem present in traditional RNNs, enabling them to maintain information over long sequences. This makes LSTMs particularly effective for tasks involving time series data, such as stock market prediction.

Architecture and Layers The LSTM model in this research is built using the Keras library (Figure 9). The architecture consists of multiple layers designed to process and predict stock prices efficiently. The layers are as follows:

- **LSTM Layer 1:** The first LSTM layer contains 150 units and is configured with `return_sequences=True` to return the full output sequence to the next layer. This helps in capturing the temporal dependencies across the input sequence.
- **Dropout Layer 1:** A dropout layer with a 20% dropout rate is incorporated to mitigate overfitting by randomly deactivating a portion of the input units, setting them to 0 during each update in the training process.
- **LSTM Layer 2:** The second LSTM layer also has 150 units but is configured with `return_sequences=False` as it is the last LSTM layer in the model. This layer processes the sequence output from the previous LSTM layer and passes it to the next layer.
- **Dropout Layer 2:** To reduce overfitting, another dropout layer with a 20% dropout rate is added.
- **Dense Layer:** A dense (fully connected) layer with 1 unit and ReLU activation function is used to produce the final prediction output.

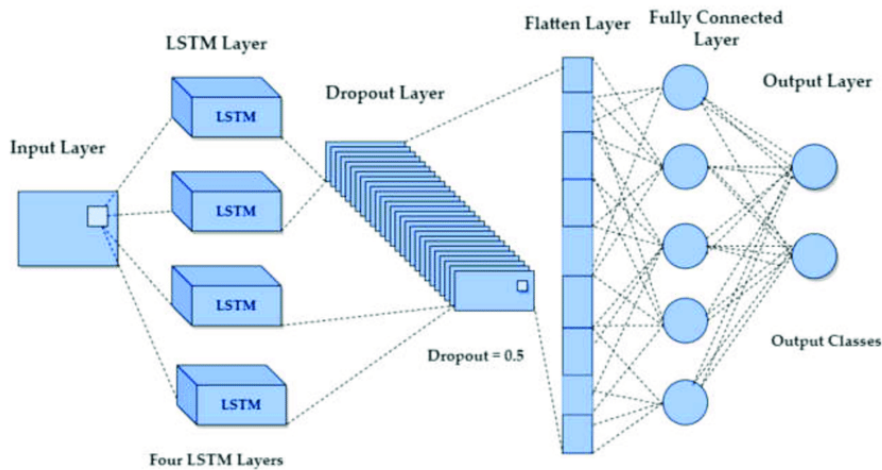


Figure 9: LSTM Model Architecture ²

²Source: https://www.researchgate.net/figure/Long-short-term-memory-LSTM-model-with-four-LSTM-layers-dropout-layer-flatten-layer_fig5_358970029

Data Preparation The data preparation process involves several steps to transform the raw historical stock price data into a format suitable for the LSTM model:

- **Serialization:** Extracts the 'Close' values from the historical data, as this column is used as the target variable for prediction.
- **Normalization:** Utilizes the `sklearn.preprocessing` `MinMaxScaler` to normalize the input features to a range between $[0, 1]$. This scaling is crucial for neural network models to perform well, as it standardizes the input data.
- **Dataset Creation:** Uses a sliding window approach with a default time step of 60 to create sequences of data (input features X and corresponding target values y). This approach captures the temporal dependencies in the data, allowing the model to learn from historical trends and patterns effectively.
- **Data Splitting:** Divides the data into training data and testing data according to a specified ratio (standard is 0.8 with 0.2). The training set is used to fit the model, while the testing set is used to evaluate its predictive accuracy.

Training The model is trained using the following configuration:

- **Early Stopping:** Implements `EarlyStopping` with the validation loss (`val_loss`) monitored, a patience of 10 epochs, and restoring the best weights to prevent overfitting.
- **Batch Size and Epochs:** Uses a batch size of 32 and trains the model for 100 epochs, with an early stopping callback to halt training once the model performance stops improving on the validation set.

Prediction The prediction process involves using the trained LSTM model to forecast future stock prices:

- **Iterative Prediction:** Future values are predicted by iteratively feeding the last sequence of 'Close' values into the model.
- **Simulating Volatility:** Noise is added to the predictions to simulate market volatility and make the predictions more realistic.

Regularization Regularization techniques are applied to improve the model's generalization ability:

- **Dropout Layers:** Two dropout layers with a 20% dropout rate are included to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.

Feature Engineering Feature engineering involves transforming the raw data into features that can be used by the LSTM model:

- **Time Step Creation:** The sliding window approach creates sequences of 60 time steps from the 'Close' prices. Each sequence is used as input for the LSTM model to predict the next 'Close' value.

4.6.4 Random Forest

Random forests, also named random decision forests, are a machine learning method used in classification and regression tasks (Figure 10). They function by constructing a multitude of decision trees during training. For classification tasks, the output is decided by the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees decides what is returned, resulting into a reliable prediction model[9].

Architecture and Layers The Random Forest model in this research is constructed using the Scikit-learn library. The architecture leverages the ensemble nature of random forests to improve prediction accuracy and control overfitting. The specific configuration is as follows:

- **Model:** The model used is `RandomForestRegressor` with 80 estimators (trees), and the criterion set to `'absolute_error'`. This setup ensures that the model is robust and capable of capturing complex patterns in the data by averaging the predictions of multiple decision trees.

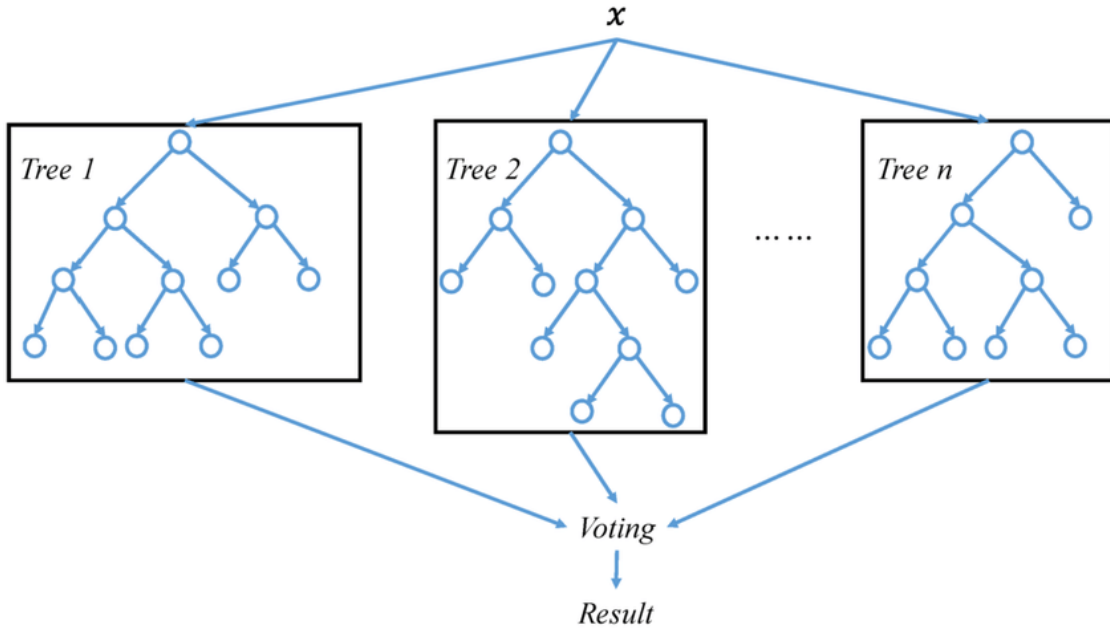


Figure 10: Random Forest Model Architecture ³

Data Preparation The data preparation for the Random Forest model involves several steps to transform the raw historical stock price data into a suitable format for model training and prediction:

- **Serialization:** Extracts the 'Close' values from the historical data, focusing on this column as the target variable for prediction.

³Source: <https://www.researchgate.net/figure/A-general-architecture-of-Random-Forest-5fig2335483097>

- **Normalization:** Utilizes the `sklearn.preprocessing` `MinMaxScaler` to normalize the input features to a range between $[0, 1]$. The scaling process is needed to ensure that all inputs contribute equally to the learning process of the model.
- **Dataset Creation:** Uses a sliding window approach with a default time step of 60 to create sequences of data (input features X and corresponding target values y). This method captures temporal dependencies in the data, allowing the model to learn effectively from historical trends and patterns.
- **Data Splitting:** Divides the data into training data and testing data according to a specified ratio (standard is 0.8 with 0.2). This step ensures that the model is evaluated on unseen data, providing a realistic measure of its performance.

Training The training process for the Random Forest model is straightforward:

- **Model Fitting:** The model is trained by fitting the reshaped training data. The ensemble nature of the random forest helps in capturing complex patterns by averaging the outputs of multiple decision trees.

Prediction The prediction process involves using the trained Random Forest model to forecast future stock prices:

- **Iterative Prediction:** Future values are predicted by iteratively feeding the last sequence of 'Close' values into the model. This method ensures that the model makes use of the most recent data for making predictions.

Regularization Regularization in the Random Forest model is inherently managed by the ensemble nature of the algorithm:

- **Ensemble Averaging:** By averaging the predictions of multiple decision trees, the model reduces overfitting and improves generalization. This built-in regularization mechanism ensures that the model is robust and less sensitive to noise in the data.

Feature Engineering Feature engineering transforms the raw data into features suitable for the Random Forest model:

- **Time Step Creation:** The sliding window approach is used to create sequences of 60 time steps from the 'Close' prices. Each sequence serves as input for the Random Forest model to predict the next 'Close' value, capturing the temporal structure of the data effectively.

4.6.5 GRU

Gated Recurrent Units (GRUs) are a gating mechanism in recurrent neural networks (RNNs), introduced by Kyunghyun Cho et al (Figure 11). in 2014. GRUs are similar to Long Short-Term Memory (LSTM) networks but have a simpler architecture with fewer parameters, lacking a context vector or output gate. This simplicity often makes GRUs more computationally efficient while still being effective for modeling sequential data.

Architecture and Layers The GRU model in this research is built using the Keras library. The architecture consists of multiple layers designed to process and predict stock prices efficiently. The layers are as follows:

- **GRU Layer 1:** The first GRU layer contains 150 units and is configured with `return_sequences=True` to return the full output sequence to the next layer. This helps in capturing the temporal dependencies across the input sequence.
- **Dropout Layer 1:** A dropout layer with a 20% dropout rate is added to prevent overfitting by randomly setting a fraction of the input units to 0 at each update during training.
- **GRU Layer 2:** The second GRU layer also has 150 units but is configured with `return_sequences=False` as it is the last GRU layer in the model. This layer processes the sequence output from the previous GRU layer and passes it to the next layer.
- **Dropout Layer 2:** Another dropout layer with a 20% dropout rate is added to further reduce overfitting.
- **Dense Layer:** A dense (fully connected) layer with 1 unit and ReLU activation function is used to produce the final prediction output.

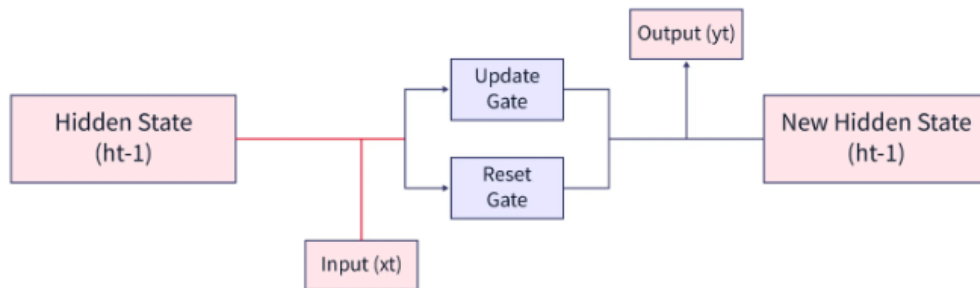


Figure 11: GRU Model Architecture ⁴

Data Preparation The data preparation process involves several steps to transform the raw historical stock price data into a format suitable for the GRU model:

- **Serialization:** Extracts the 'Close' values from the historical data, as this column is used as the target variable for prediction.
- **Normalization:** Utilizes the `sklearn.preprocessing.MinMaxScaler` to normalize the input features to a range between [0, 1]. The scaling process is needed to ensure that all inputs contribute equally to the learning process of the model.

⁴Source: <https://www.scaler.com/topics/deep-learning/gru-network/>

- **Dataset Creation:** Uses a sliding window approach with a default time step of 60 to create sequences of data (input features X and corresponding target values y). This approach captures the temporal dependencies in the data, allowing the model to learn effectively from historical trends and patterns.
- **Data Splitting:** Divides the data into training data and testing data according to a specified ratio (standard is 0.8 with 0.2). The training set is used to fit the model, while the testing set is used to evaluate its predictive accuracy.

Training The training process for the GRU model is configured to optimize performance and prevent overfitting:

- **Early Stopping:** Implements `EarlyStopping` with the validation loss (`val_loss`) monitored, a patience of 10 epochs, and restoring the best weights. This technique halts training once the model performance stops improving on the validation set, ensuring that the best model is used for predictions.
- **Batch Size and Epochs:** Uses a batch size of 32 and trains the model for 100 epochs, with an early stopping callback to halt training when appropriate.

Prediction The prediction process involves using the trained GRU model to forecast future stock prices:

- **Iterative Prediction:** Future values are predicted by iteratively feeding the last sequence of 'Close' values into the model. This method ensures that the model makes use of the most recent data for making predictions.
- **Simulating Volatility:** Noise is added to the predictions to simulate market volatility and make the predictions more realistic.

Regularization Regularization techniques are applied to improve the model's generalization ability:

- **Dropout Layers:** Two dropout layers with a 20% dropout rate are included to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.

Feature Engineering Feature engineering transforms the raw data into features suitable for the GRU model:

- **Time Step Creation:** The sliding window approach is used to create sequences of 60 time steps from the 'Close' prices. Each sequence serves as input for the GRU model to predict the next 'Close' value, capturing the temporal structure of the data effectively.

4.6.6 SVM

Support Vector Machines (SVMs) are supervised learning models used for classification and regression analysis (Figure 12). They work by finding the hyperplane that best separates the data into different classes (in classification) or fits the data points with minimal error (in regression). For this research, SVM is employed for regression tasks to predict stock prices.

Architecture and Layers The SVM model is built using the Scikit-learn library, specifically the Support Vector Regressor (SVR) for regression tasks. The architecture and parameters are as follows:

- **Model:** SVR with a linear kernel, regularization parameter $C = 1$, and degree set to 2. This configuration helps in controlling the trade-off between achieving a low error on the training data and minimizing the norm of the weights, thus preventing overfitting.

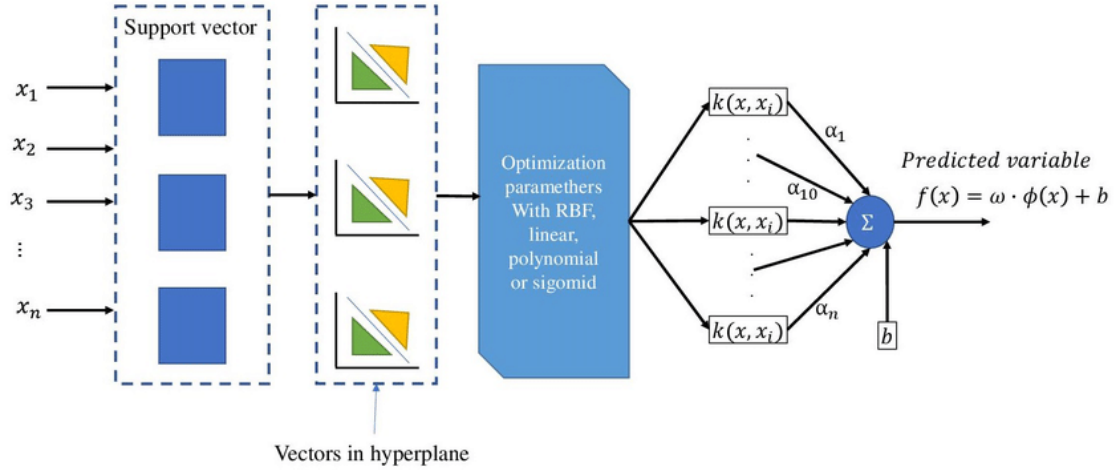


Figure 12: SVM Model Architecture ⁵

Data Preparation The data preparation steps for the SVM model involve transforming the raw historical stock price data into a suitable format for model training and prediction:

- **Serialization:** Extracts the 'Close' values from the historical data, focusing on this column as the target variable for prediction.
- **Normalization:** Utilizes the `sklearn.preprocessing.MinMaxScaler` to normalize the input features to a range between $[0, 1]$. The scaling process is needed to ensure that all inputs contribute equally to the learning process of the model.
- **Dataset Creation:** Uses a sliding window approach with a default time step of 60 to create sequences of data (input features X and corresponding target values y). This method captures temporal dependencies in the data.
- **Data Splitting:** Divides the data into training data and testing data according to a specified ratio (standard is 0.8 with 0.2). This guarantees that the model is assessed using unseen data.

Training The training process for the SVM model is straightforward:

- **Model Fitting:** The model is trained by fitting the reshaped training data. The SVR optimizes the hyperplane to minimize the prediction error.

⁵Source: <https://www.mdpi.com/2076-3417/11/3/1044>

Prediction The prediction process involves using the trained SVM model to forecast future stock prices:

- **Iterative Prediction:** Future values are predicted by iteratively feeding the last sequence of 'Close' values into the model, ensuring the most recent data is used.

Regularization Regularization in the SVM model is managed by the regularization parameter C :

- **Parameter C :** Controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights, which helps in preventing overfitting.

Feature Engineering Feature engineering transforms the raw data into features suitable for the SVM model:

- **Time Step Creation:** The sliding window approach is used to create sequences of 60 time steps from the 'Close' prices. Each sequence serves as input for the SVM model to predict the next 'Close' value.

4.6.7 AdaBoost

Adaptive Boosting (AdaBoost) is an ensemble learning technique that combines multiple weak learners to create a strong predictive model (Figure 13). Each weak learner is typically a decision tree, and the model focuses on the data points that were previously misclassified, improving accuracy iteratively.

Architecture and Layers The AdaBoost model in this research is constructed using the Scikit-learn library. The architecture and parameters are as follows:

- **Model:** `AdaBoostRegressor` with 40 estimators, a learning rate of 1.5, and a random state of 50. This configuration ensures the model is robust and capable of focusing on hard-to-predict data points.

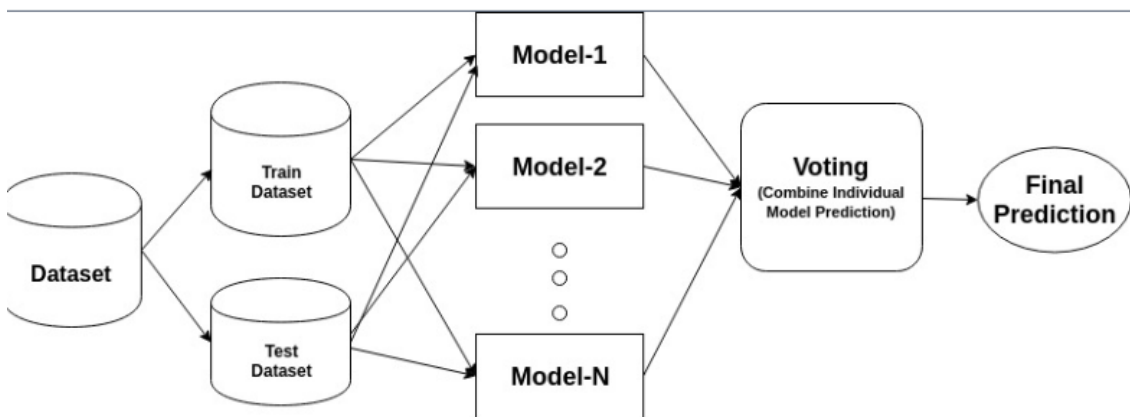


Figure 13: AdaBoost Model Architecture ⁶

⁶Source: <https://towardsdatascience.com/advanced-ensemble-classifiers-8d7372e74e40>

Data Preparation The data preparation steps for the AdaBoost model are similar to those for the SVM model:

- **Serialization:** Extracts the 'Close' values from the historical data, focusing on this column as the target variable for prediction.
- **Normalization:** Utilizes the `sklearn.preprocessing` `MinMaxScaler` to normalize the input features to a range between $[0, 1]$. The scaling process is needed to ensure that all inputs contribute equally to the learning process of the model.
- **Dataset Creation:** Uses a sliding window approach with a default time step of 60 to create sequences of data (input features X and corresponding target values y). This method captures temporal dependencies in the data.
- **Data Splitting:** Divides the data into training data and testing data according to a specified ratio (standard is 0.8 with 0.2). This guarantees that the model is assessed using unseen data.

Training The training process for the AdaBoost model involves the following steps:

- **Model Fitting:** The model is trained by fitting the reshaped training data. Each weak learner focuses on the data points that were previously misclassified, improving accuracy iteratively.

Prediction The prediction process involves using the trained AdaBoost model to forecast future stock prices:

- **Iterative Prediction:** Future values are predicted by iteratively feeding the last sequence of 'Close' values into the model, ensuring the most recent data is used.

Regularization Regularization in the AdaBoost model is inherently managed by its ensemble nature:

- **Ensemble Averaging:** By combining multiple weak models and focusing on hard-to-predict data points, AdaBoost reduces overfitting and improves generalization.

Feature Engineering Feature engineering transforms the raw data into features suitable for the AdaBoost model:

- **Time Step Creation:** The sliding window approach is used to create sequences of 60 time steps from the 'Close' prices. Each sequence serves as input for the AdaBoost model to predict the next 'Close' value.

5 Results and Analysis

5.1 Hyperparameter Optimization

This section outlines the hyperparameter optimization process for various machine learning models. The script uses randomized grids to search for the best parameters for each model.

Random Grids

The random grids for hyperparameter optimization are defined as follows:

```
lstm_params = {
    'units': [50, 80, 100, 120, 150],
    'optimizer': ['adam', 'rmsprop', 'sgd'],
    'batch_size': [16, 32, 64],
    'epochs': [50, 100, 150]
}

gru_params = {
    'units': [50, 80, 100, 120, 150],
    'optimizer': ['adam', 'rmsprop', 'sgd'],
    'batch_size': [16, 32, 64],
    'epochs': [50, 100, 150]
}

svm_params = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['rbf', 'linear', 'poly'],
    'degree': [2, 3, 4, 5],
    'gamma': ['scale', 'auto']
}

adaboost_params = {
    'n_estimators': [40, 60, 80, 100],
    'learning_rate': [0.1, 0.5, 1, 1.5],
    'random_state': [10, 20, 30, 40, 50]
}

rf_params = {
    'n_estimators': [80, 100, 120, 150],
    'criterion': ['squared_error', 'absolute_error'],
    'random_state': [10, 20, 30, 40, 50],
    'min_samples_leaf': [1, 2, 4, 6]
}
```

5.1.1 Optimization Process

For each model, the script uses the defined random grids to perform hyperparameter optimization. For LSTM and GRU models, a manual random search is conducted by sampling from the grids. For SVM, AdaBoost, and Random Forest models,

`RandomizedSearchCV` is employed to find the best hyperparameters. The models are evaluated using Mean Squared Error (MSE) on a validation set, and the best parameters are recorded.

Results and Comparison

The performance of the models with the best-found hyperparameters is compared and visualized (Table 3, 4).

Model	Units	Optimizer	Batch Size	Epochs	MSE
LSTM	150	adam	-	150	0.000813587
GRU	100	adam	16	150	0.000705296

Table 3: Hyperparameter Optimization Results for LSTM and GRU

Model	Parameter	Value
SVM	Kernel	rbf
	Gamma	auto
	Degree	5
	C	10
AdaBoost	Random State	30
	Number of Estimators	40
	Learning Rate	1.5
RandomForest	Random State	40
	Number of Estimators	80
	Min Samples Leaf	4
	Criterion	squared_error

Table 4: Hyperparameter Optimization Results for SVM, AdaBoost, and Random-Forest

The bar chart above compares the performance of different algorithms based on their Mean Squared Error (MSE).

5.2 Validation Optimization

This section describes the process of validating machine learning models using different techniques. The script evaluates the models with various holdout splits and cross-validation methods to determine their performance.

Validation Methods

The following validation methods are used in the script:

- 80-20 Holdout Split
- 90-10 Holdout Split
- K-Fold Cross-Validation
- Time Series Cross-Validation

5.2.1 Random Grids for Models

The script uses the hyperparameters resulted in the last section for finding the best validation method for each model.

5.2.2 Evaluation Process

For each model, the script uses the defined random grids to perform hyperparameter optimization and evaluates the models using different validation methods. The mean squared error (MSE) is used as the evaluation metric.

5.2.3 Results and Comparison

The performance of the models with the best-found hyperparameters is compared using various validation methods. The results are visualized in the bar chart below (Figure 14)



Figure 14: Comparison of Validation Methods

The bar chart above compares the performance of different models using various validation methods based on their Mean Squared Error (MSE).

5.3 Comparison Results over Different Datasets

This section describes the process of evaluating machine learning models using stock data from six different domains. The models are evaluated on their ability to predict stock prices, and the results are compared.

5.3.1 Domains and Tickers

The following seven domains and their respective stock tickers are used in the evaluation:

- Tech Companies: AAPL, MSFT, GOOGL, META, NVDA

- Finance Companies: JPM, BAC, WFC, C, GS
- Automotive Companies: TSLA, GM, F, TM, HMC
- Medical Companies: JNJ, PFE, MRK, ABT, TMO
- Alimentation Companies: KO, PEP, MDLZ, GIS, KHC
- Cosmetics Companies: EL, PG, UL, CL, RBGLY

5.3.2 Evaluation Process

For each domain, the script fetches historical stock data, preprocesses it, and evaluates the models using the following algorithms:

- LSTM
- GRU
- SVM
- AdaBoost
- RandomForest

The models are evaluated based on their percentage accuracy in predicting stock prices. The results are saved and visualized for each domain.

5.3.3 Results and Comparison

The performance of the models is compared across the seven domains. Each domain's results are visualized separately to highlight the models' accuracy.

The figure below contains the results of models LSTM, RandomForest, GRU, SVM and ADAB across the tech companies dataset. (Figure 15)

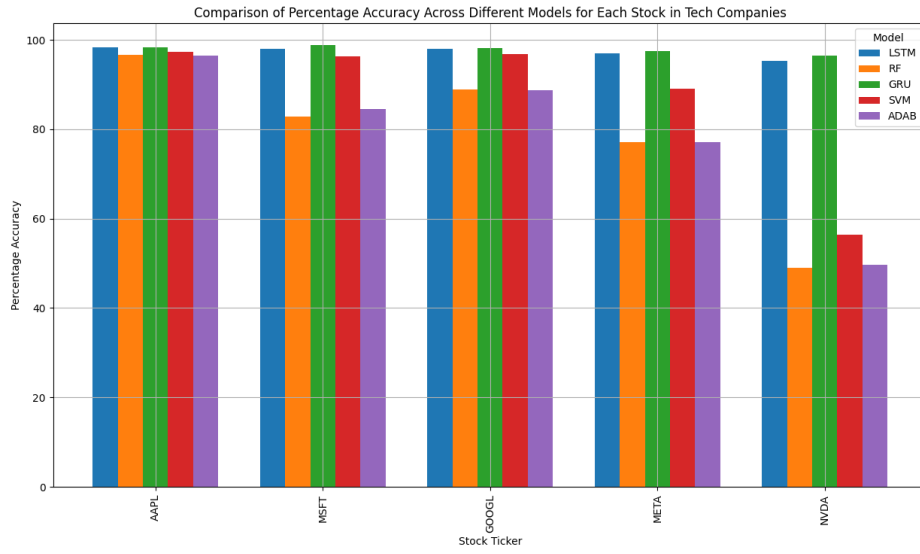


Figure 15: Tech Companies

The figure below contains the results of models LSTM, RandomForest, GRU, SVM and ADAB across the finance companies dataset. (Figure 16)

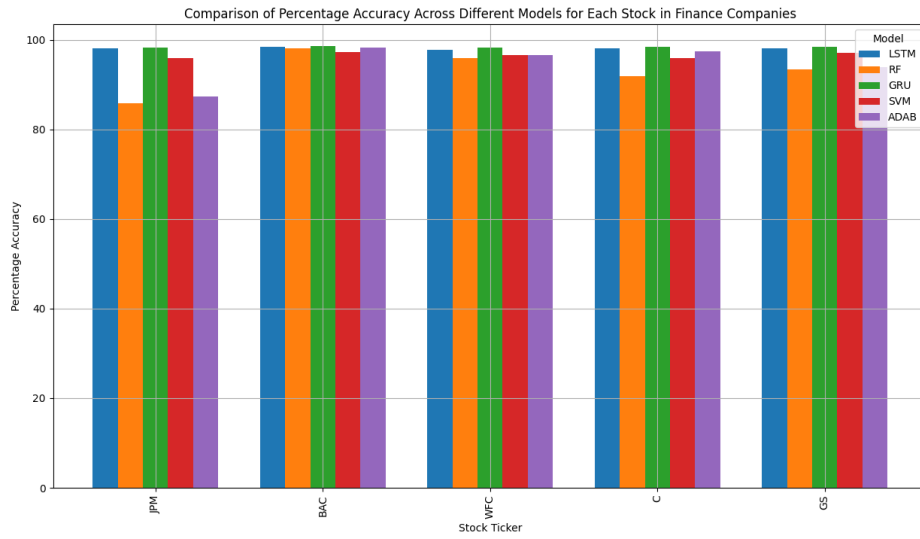


Figure 16: Finance Companies

The figure below contains the results of models LSTM, RandomForest, GRU, SVM and ADAB across the automotive companies dataset. (Figure 17)

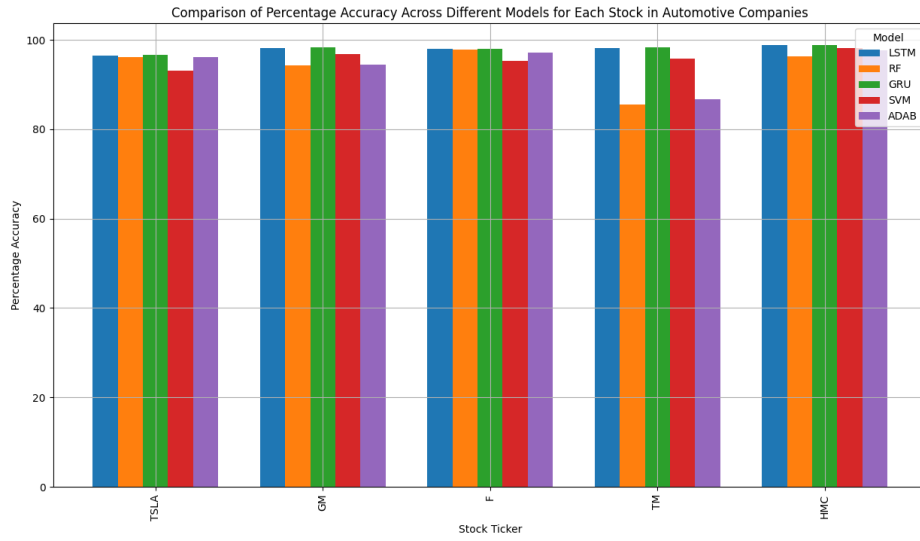


Figure 17: Automotive Companies

The figure below contains the results of models LSTM, RandomForest, GRU, SVM and ADAB across the medical companies dataset. (Figure 18)

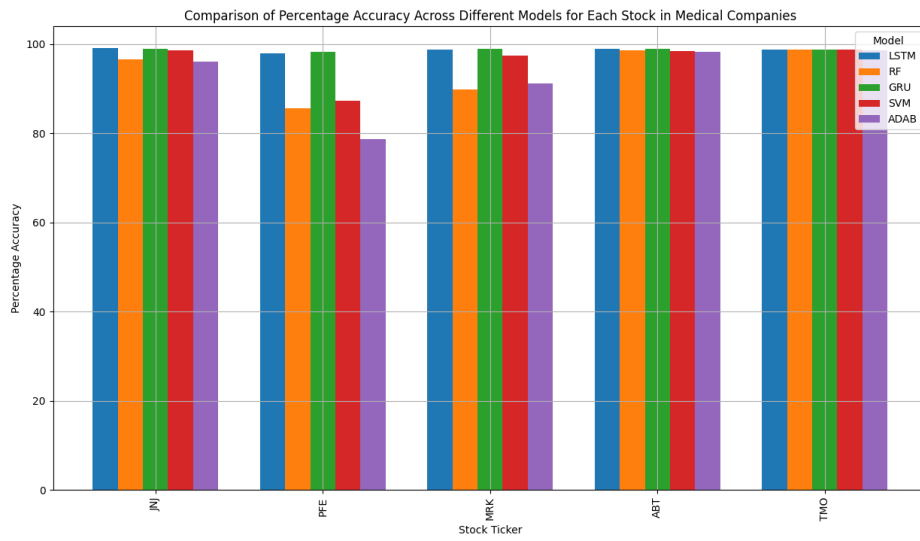


Figure 18: Medical Companies

The figure below contains the results of models LSTM, RandomForest, GRU, SVM and ADAB across the alimentation companies dataset. (Figure 19)

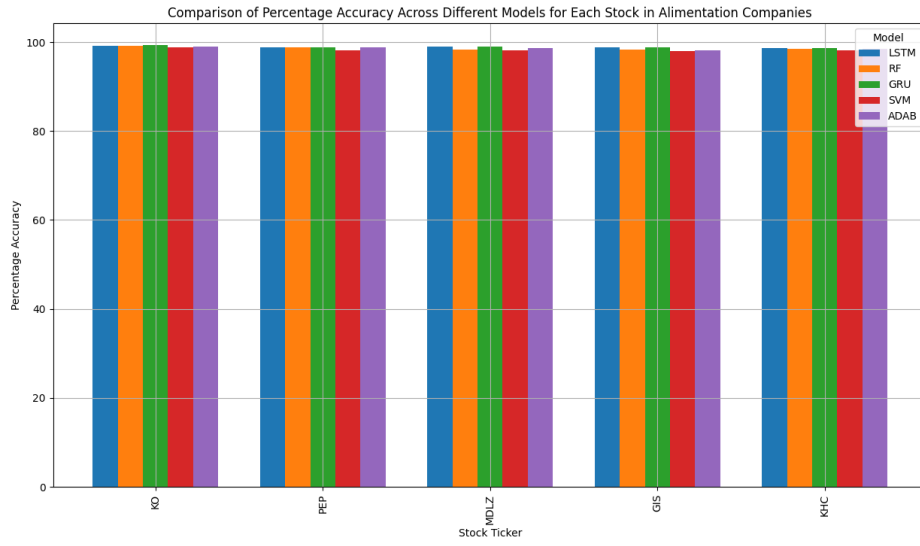


Figure 19: Alimentation Companies

The figure below contains the results of models LSTM, RandomForest, GRU, SVM and ADAB across the cosmetics companies dataset. (Figure 20)

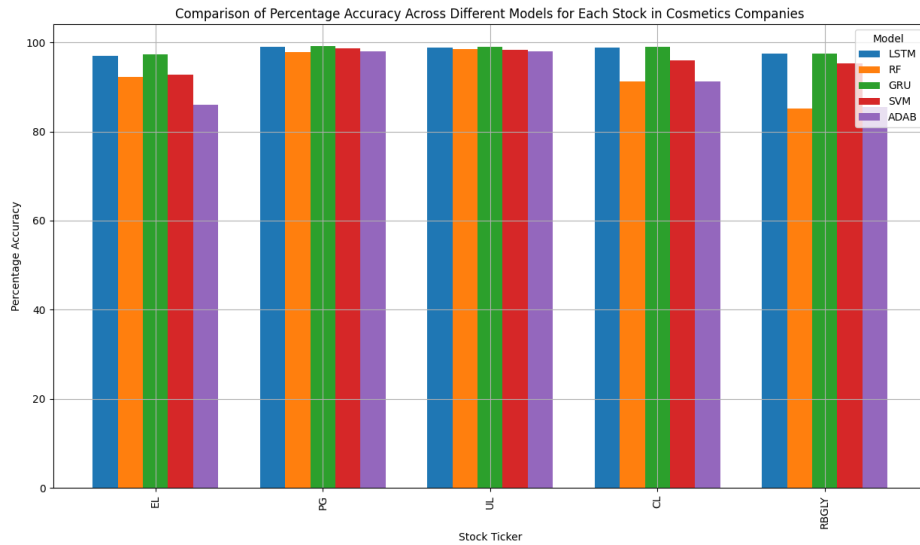


Figure 20: Cosmetics Companies

The figure below contains the results of models LSTM, RandomForest, GRU, SVM and ADAB across the all datasets. (Figure 21)

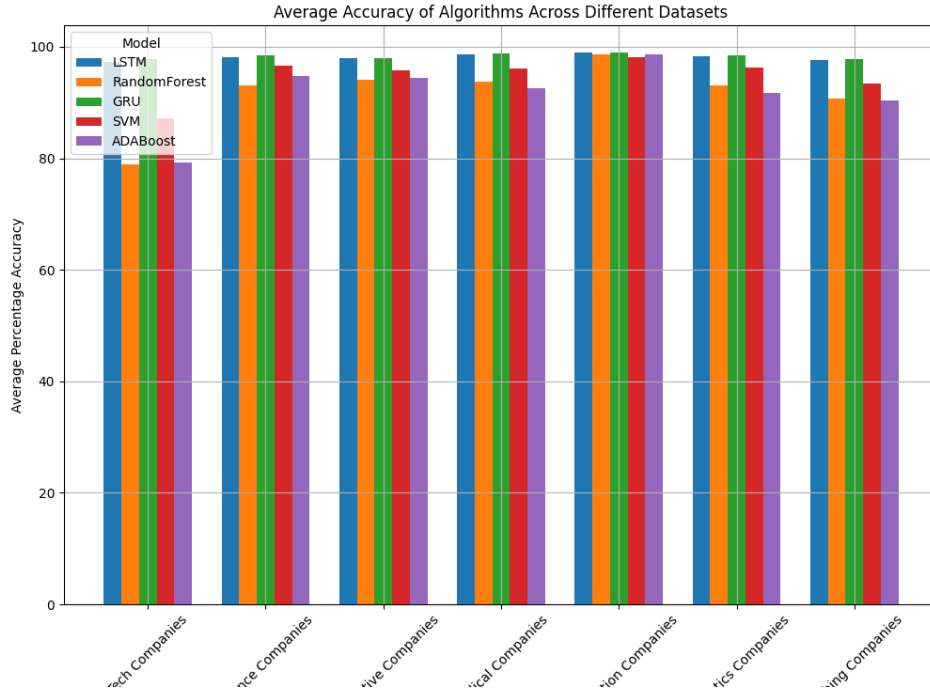


Figure 21: Average Accuracy Across All Datasets

5.4 Experimental Setup

The experimental setup outlines the hardware and software environment used to conduct the stock market prediction experiments. This section details the configurations and tools employed to ensure the reproducibility of the results.

5.4.1 Hardware Configuration

- **Processor:** Intel Core i7-7700HQ
- **RAM:** 16GB DDR4
- **GPU:** NVIDIA GeForce GTX 1060 Max-Q Design
- **Storage:** 1TB SSD

5.4.2 Software Environment

- **Operating System:** Windows 10 Pro
- **Programming Language:** Python 3.11
- **Libraries and Frameworks:**
 - TensorFlow 2.4.1
 - Keras 2.4.3
 - Scikit-learn 0.24.1
 - Pandas 1.2.3

- Numpy 1.19.5
- yfinance 0.1.55
- Matplotlib 3.3.4
- Tkinter (for GUI)

5.4.3 Datasets

- **Historical Stock Data:** Obtained from Yahoo Finance using the yfinance library. The dataset includes daily closing prices, volume, and other financial indicators for the selected companies from the S&P 500 index.

5.4.4 Model Configurations

LSTM Model:

- Two LSTM layers with 150 units each
- Dropout rate: 0.2
- Final dense layer with ReLU activation
- Optimizer: Adam
- Loss function: Mean Squared Error
- Epochs: 100
- Batch size: 32

Random Forest Model:

- Number of estimators: 80
- Criterion: Absolute Error
- Minimum samples per leaf: 1

GRU Model:

- Two GRU layers with 150 units each
- Dropout rate: 0.2
- Final dense layer with ReLU activation
- Optimizer: Adam
- Loss function: Mean Squared Error
- Epochs: 100
- Batch size: 32

SVM Model:

- Kernel: Linear
- Regularization parameter C: 1
- Degree: 2

AdaBoost Model:

- Number of estimators: 40
- Learning rate: 1.5

5.4.5 Execution

The experiments were conducted by running the models on the specified hardware, using the mentioned software and libraries. Each model was trained and tested on the historical stock data, and their performance was evaluated using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and R-squared (R2).

By providing this comprehensive experimental setup, you ensure that the methodology is transparent and that the results can be reproduced by others in a similar environment.

6 Conclusion

6.1 Summary of Findings

In this thesis, we implemented and evaluated five different models for stock market price prediction: LSTM, Random Forest, GRU, SVM, and AdaBoost. Each model was trained and tested on historical data from companies across seven different domains, including Tech, Finance, Automotive, Medical, Alimentation and Cosmetics. The models were optimized using hyperparameter tuning and validated using various cross-validation techniques.

The results indicate that the GRU model achieved the highest average accuracy, demonstrating its effectiveness in capturing the sequential dependencies in stock market data. The LSTM model also showed strong predictive capabilities, following closely behind in performance.

Additionally, hyperparameter optimization and validation techniques significantly improved the models' accuracy and robustness. Each model was fine-tuned to achieve the best performance, and the validation methods ensured that the models were not overfitting and were generalizable to unseen data.

However, the analysis also highlighted the limitations of these models in handling market instability and unpredictable events. This underlines the importance of further research and development to enhance the robustness of stock market prediction models.

In summary, while this thesis provides valuable insights into the performance of different machine learning models for stock market prediction, there is still considerable scope for improvement and innovation in this field. .

6.2 Future Work

6.2.1 Implementing Hybrid Models for Increased Accuracy

One potential avenue for future work is the implementation of hybrid models that combine the strengths of multiple algorithms to improve prediction accuracy. Hybrid models can leverage the strengths of different machine learning techniques, such as combining the temporal awareness of LSTM with the decision-making capabilities of Random Forests. This approach could involve the creation of ensemble models where different models' predictions are combined to produce a final output, or the development of models that incorporate multiple learning algorithms in their architecture. For instance, an LSTM model could be used to capture temporal patterns in stock prices, while a Random Forest could be used to incorporate additional features like moving averages and technical indicators. The integration of these models could lead to more robust and accurate predictions, capable of handling the complexities of stock market data.

6.2.2 Implementing Custom Models

Another area of development is providing users with the ability to create custom models. This feature would allow users to select their preferred algorithms, adjust hyperparameters, and manipulate data preprocessing steps according to their specific needs and preferences. Such a system would require a flexible user interface

that supports the selection and configuration of various machine learning models, including but not limited to LSTM, GRU, Random Forest, SVM, and AdaBoost. Users could experiment with different settings to optimize their models for specific stocks or market conditions. This customization would empower users to tailor the prediction system to their unique investment strategies and improve overall satisfaction and usability.

6.2.3 Development of a More Sophisticated Real-Time Prediction System

The development of a more sophisticated real-time prediction system is another key area for future work. This system would provide continuous predictions based on live market data, offering users up-to-date insights into stock price movements. Advanced visualization features would be integrated to display these predictions in an intuitive and user-friendly manner. For example, real-time line charts, candlestick patterns, and heat maps could be used to illustrate market trends and potential investment opportunities. Additionally, incorporating alert systems that notify users of significant market changes or prediction updates would enhance the system's usability and effectiveness, helping investors make timely decisions.

6.2.4 Implementation of a Cloud-Based System

Finally, implementing a cloud-based system for scalable and distributed predictions is a crucial step for future development. A cloud-based infrastructure would allow for the handling of large datasets and complex computations required for real-time stock market predictions. This setup would enable the system to scale efficiently, managing multiple simultaneous user requests and large volumes of data without performance degradation. Furthermore, a cloud-based system could facilitate real-time prediction and risk comparison across multiple stocks, providing users with a comprehensive overview of their investment portfolio. The cloud infrastructure could also support continuous model training and updates, ensuring that the prediction models remain accurate and relevant in dynamic market conditions. This would significantly enhance the system's reliability and user experience, making it a powerful tool for both individual investors and financial institutions.

7 Bibliography

References

- [1] Fama, Eugene F. "Efficient capital markets: A review of theory and empirical work." *The Journal of Finance* 25.2 (1970): 383-417.
- [2] Chen, K., Zhou, Y., Dai, F. (2015). A LSTM-based method for stock returns prediction: A case study of China stock market. In *2015 IEEE International Conference on Big Data (Big Data)* (pp. 2823-2824). IEEE.
- [3] Qiu, M., Song, Y. (2016). Predicting the direction of stock market index movement using an optimized artificial neural network model. *PloS one*, 11(5), e0155133.
- [4] Bollen, J., Mao, H., Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1), 1-8.
- [5] Si, J., Mukherjee, A., Liu, B., Li, Q., Li, H., Deng, X. (2013). Exploiting topic based twitter sentiment for stock prediction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 24-29).
- [6] Qlik. (n.d.). Predictive Analytics. Retrieved from <https://www.qlik.com/us/predictive-analytics>
- [7] Investopedia. (n.d.). Stock Market. Retrieved from <https://www.investopedia.com>
- [8] FasterCapital. (n.d.). Stock Market Terms. Retrieved from <https://www.fastercapital.com>
- [9] Si Fisheriesscience. (n.d.). Random Forests. Retrieved from <https://www.sifisheriesscience.com>