Leverage sentiment analysis for predictive modelling in stock market dynamics

Research Report

TRAN DUC ANH DANG | 103995439

Swinburne University of Technology

Abstract

This study aims to explore the relationship between public sentiment, as expressed through digital media, and stock market dynamics. Utilizing sentiment analysis and machine learning algorithms, this research will examine how different sentiments correlate with stock market movements, potentially unveiling predictive patterns. The findings of this study could provide investors and policy makers with valuable insights into how public sentiment influences market behaviour, thus contributing to a more comprehensive understanding of stock market dynamics.

Table of Contents

Introduction

Background

The financial market is a complex ecosystem where myriad factors interact to dictate the price movement of a securities. One such factor, which has recently garnered attention is the sentiment prevalent among investors and the general populace. Sentiment analysis often associated with the field of Natural Language Processing (NLP), provides a mechanism to gauge public sentiment through the analysis of textual data available on various platforms like social media, news, financial forums. In recent years, the explosion of digital data has provided a fertile ground for the application of sentiment analysis in understanding and predicting stock market dynamics. Previous studies have demonstrated a correlation between public sentiment and stock market movements, indicating the potential of sentiment analysis as a tool for predictive modelling in financial markets.

Objective

The primary objective of this research is to explore the potential of sentiment analysis in predicting stock market trends, which include:

- Developing a sentiment analysis framework capable of processing and analysing large volumes of textual data related to stock market.
- Investigating the correlation between public sentiment, as gauged through the sentiment analysis framework, and stock price movement across different time frames.
- Creating predictive models to leveraging sentiment analysis to forecast stock market trends and assessing the accuracy and reliability of these models in comparison to traditional financial models.
- Identifying the practical implications and potential benefits for investors and financial analysis.

Scope and Limitations

Scope

- The study will focus on major U.S stock market indices over a period of five years.
- Data for sentiment analysis will be sourced from publicly available platforms including social media, financial news, etc.
- Various machine learning and NLP techniques will be employed to develop the sentiment analysis framework and predictive models.

Limitations

- The accuracy and reliability of sentiment analysis and predictive modelling are contingent on the quality and volume of data that available.
- Sentiment analysis may be influenced by external factors such as political events or global crises which are outside the scope of this study.
- The rapidly evolving nature of financial markets are trading technologies may introduce unforeseen challenges in the application and effectiveness of sentiment based predictive models.

Literature Review

Previous Work on Sentiment Analysis

Sentiment analysis, situated at the intersection of Natural Language Processing (NLP) and machine learning, has evolved significantly over the years. Its applications span across various domains ranging from consumer product reviews to political discourse analysis. A systematic review on sentiment analysis revealed the employment of hybrid tools and techniques to enhance the accuracy and efficiency of sentiment analysis.

In another dimension, sentiment analysis is often juxtaposed with emotion detection, where the former primarily concerns polarity (positive, negative, neutral) while the latter delves into the emotional or psychological state or mood of the text. This distinction highlights the subjective nature of sentiment analysis as compared to the more objective and precise nature of emotion detection.

Sentiment Analysis in Finance

The application of sentiment analysis in the financial domain aims at understanding market dynamics by analysing investors' sentiments through various textual data sources like news articles, social media, or financial forums. A critical review of sentiment analysis in predicting financial markets highlighted the examination of 24 papers focusing on this domain, indicating a growing interest in leveraging sentiment analysis for financial market predictions.

Social media, being a rich source of raw data reflecting public sentiment, has been especially utilized for sentiment analysis in the financial domain. The methods explored include various machine learning algorithms to process the textual data from social media platforms and derive insights regarding market sentiment.

Gap Identification

Despite the promising strides, several gaps persist in the current literature:

- **Granularity of Analysis**: The granularity with which sentiments are analysed varies across studies, with many adopting a broad categorization into positive, negative, or neutral sentiments. A more nuanced approach capturing a wider spectrum of emotions could potentially offer more precise correlations with stock market movements.
- **Temporal Dynamics**: The temporal aspect of the influence of sentiment on stock market dynamics has not been exhaustively explored. Understanding how sentiment impact evolves over different time frames could be pivotal for developing accurate predictive models.
- **Predictive Modelling**: The transition from identifying correlations to developing reliable predictive models leveraging sentiment analysis is still in the nascent stages. The rigorous testing and validation of sentiment-based predictive models across different market conditions and time frames are requisite for advancing the field.
- **Integration with Traditional Models**: The amalgamation of sentiment analysis with traditional financial models and indicators has been less explored.

Investigating how sentiment analysis can complement existing financial models could provide a more robust framework for predicting stock market dynamics.

- **Big Data and Sentiment Analysis**: The advent of big data has provided an avenue for classifying diverse sentiments and deriving commercial insights from text-oriented content. However, the systematic integration and analysis of big data in sentiment analysis frameworks in the context of financial markets require further exploration.

Research

Data Collection

Data collection for sentiment analysis in finance can be accomplished through various platforms where financial discussions are held. A notable example is the utilization of microblogging platforms like StockTwits, where a dataset of one million messages was employed to evaluate pre-processing methods and machine learning algorithms for sentiment analysis in finance. Other sources include financial news data, as depicted in a study where time-series data analysis was conducted on the closing price data of Infosys Company from 2014 to 2018, showing a strong relation between finance news data and stock market prices. Furthermore, a unified solution for data collection, analysis, and visualization in real-time stock market prediction was proposed, retrieving, and processing relevant financial data from news articles, social media, and company technical information.

Sentiment Analysis Framework

Various frameworks have been proposed for sentiment analysis in financial markets. A framework introduced in a study combined text mining methods to filter relevant and meaningful information from the textual content, addressing the challenge of information overload as financial markets become faster and more complex. Other frameworks have employed lexicon-based sentiment analysis models for opinion mining and deep sentiment analysis to understand how stock markets respond to various news categories in the short, medium, and long term. More advanced frameworks like the knowledge graph-based sentiment analysis system have also been developed for the stock investment market, consisting of three main modules: knowledge graph module, embedding layer module, and recurrent convolutional neural network (RCNN) module.

Predictive Modelling

Predictive modelling in this context aims to harness the potential of sentiment analysis for forecasting stock market dynamics. Utilizing sentiment scores extracted from digital media as crucial input features can significantly enhance the model's ability to forecast stock market trends.

- Sentiment Analysis Integration
    - The heart of predictive modelling in this study revolves around the integration of sentiment analysis. By employing advanced NLP techniques to extract sentiment scores from a vast array of textual

data including social media, news articles, and financial forums, a robust sentiment analysis framework can be established. This framework can serve as a vital foundation for extracting sentiment features which will be instrumental in feeding the predictive models.

- Sentiment-driven Feature Engineering
    - Feature engineering is a cornerstone for building powerful predictive models. In this study, sentiment scores, along with other derived sentiment features such as sentiment volatility, sentiment momentum, and aggregated sentiment indices, can be engineered to serve as input features for the predictive models.
- Model Development
    - Model such as LSTM, GRU, ARIMA, and SARIMAX can be leveraged to develop predictive models. LSTM and GRU can be particularly adept at capturing temporal dependencies in sentiment scores and stock prices over time. Meanwhile, ARIMA and SARIMAX can be employed for univariate and multivariate time-series forecasting respectively, with sentiment scores serving as exogenous inputs in SARIMAX.

- Hybrid Sentiment-Time Series Models
    - Developing hybrid models that marry the strengths of time-series analysis and sentiment analysis can potentially lead to more accurate and insightful forecasts. For instance, a hybrid model could be designed to incorporate sentiment scores and other sentiment-derived features alongside historical stock prices and macroeconomic indicators to forecast stock market dynamics.

Evaluation Metrics

The effectiveness and precision of the predictive models constructed in this research are pivotal, as they directly influence the reliability of the findings. To gauge the performance of these models, a range of evaluation metrics will be employed. Here's a breakdown of these metrics for further understanding:

1. Mean Absolute Error (MAE):
    - This metric calculates the average of absolute differences between the predicted and actual values, offering a straightforward measure of the model's accuracy.
2. Root Mean Square Error (RMSE):
    - RMSE is the square root of the average squared differences between the predicted and actual values, which gives more weight to larger errors.
3. Mean Absolute Percentage (MAPE):
    - This metric expresses forecast errors as a percentage, making it a scalable and easy-to-interpret measure of accuracy.
4. Confusion Matrix and Derived Metrics:
    - These metrics are vital for assessing the model's classification performance, especially in binary or multiclass classification problems.
5. Economic Performance Metrics:

- Metrics such as cumulative returns, maximum drawdown, and Sharpe ratio are essential for evaluating the economic performance of the trading strategies derived from the models.

Summary

Required Dependencies

- NumPy: A fundamental package for numerical computations in Python.
- Matplotlib: A plotting library for creating static, animated, and interactive visualizations.
- Pandas: A data manipulation and analysis library.
- TensorFlow: An open-source machine learning framework.
- Scikit-Learn: A library for machine learning and data mining tasks.
- Pandas DataReader: A tool for reading financial data from web sources.
- YFinance: A library for accessing Yahoo Finance data.
- Mplfinance: An extension of Matplotlib for financial data visualization.
- Pmdarima: A library for performing time series forecasting using ARIMA models.
- Transformers: A library by Hugging Face for state-of-the-art natural language processing.
- Torch: An open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language.
- Datasets: A library by Hugging Face to easily share and load datasets and evaluation metrics for Natural Language Processing (NLP).
- Pandas TA: A library for technical analysis and indicators in financial markets.
- BeautifulSoup4: A library for pulling data out of HTML and XML files.
- Feedparser: A Python library that parses feeds in all known formats, including Atom, RSS, and RDF.

Training Process

dataset.py

This file contains functions related to data handling for the training process. Here's a summary of the key functions and their purposes:

Utility Functions

- ensure_directory_exists(dir_path): Ensures that a specified directory exists, and creates it if it doesn't. This is helpful for organizing data and model files.
- save_object(obj, filename): Saves an object (like a model or scaler) to a file using joblib.
- load_object(filename): Loads an object from a file using joblib.

Data Loading and Preparation (LSTM Dataset)

- load_data(start_date, end_date, tick, source='yahoo'): Loads stock data for a given ticker (tick) between specified dates from Yahoo Finance. If the data is already downloaded, it loads from a CSV file.
- data_validation(start_date, end_date, tick): Validates and processes raw data, checking for prepared data and processing raw data if necessary.

- data_preparation(df, n_steps=5, multisteps=True, train_size=0.8, scale=True): Prepares data for training. It handles data scaling, feature extraction, and train-test splitting. Parameters:
- n_steps: Number of steps (lag) for the LSTM model.
- multisteps: Whether to use multi-step prediction.
- train_size: The proportion of data used for training.
- scale: Whether to scale the data.

FinBERT Dataset Functions

- create_finbert_datasets(parquet_path): Creates a dataset for FinBERT from a Parquet file.
- finbert_dataset_splitting(dataset): Splits the FinBERT dataset into training and validation sets.
- finbert_datasets_mapping(dataset, tokenizer, debug=True): Maps the dataset using a tokenizer for FinBERT.
- finbert_dataset_format(dataset, column_names, format_type, debug=True): Formats the FinBERT dataset for training. Parameters:
- column_names: Names of the columns to include in the formatted dataset.
- format_type: Format type, such as "torch" for PyTorch.

The code is for handling different datasets, particularly for LSTM models and FinBERT. The parameters in the functions are well-explained and serve specific purposes, such as data scaling, feature extraction, and dataset splitting.

models.py

This file contains functions related to model creation and initialization. Here's a summary of the key functions and their purposes:

create_dynamic_model(input_shape, layer_configs, output_units=1)

- Purpose: Creates a dynamic neural network model based on the specified layer configurations. It supports different types of layers including LSTM, GRU, RNN, and their bidirectional versions.
- Parameters:
  - input_shape: The shape of the input data.
  - layer_configs: A list of dictionaries, each representing the configuration of a layer (type, units, return_sequences, dropout, etc.).
  - output_units: The number of units in the output layer.

intialized_lstm(input_shape, layer_configs, n_steps)

- Purpose: Initializes an LSTM model using the create_dynamic_model function. It compiles the model with mean squared error loss and Adam optimizer.
- Parameters:
  - input_shape: The shape of the input data.
  - layer_configs: Configuration for LSTM layers.

    - n_steps: Number of output units, corresponding to the number of steps for prediction.

intialize_bert_model(path='bert-base-uncased', num_labels=3)

- Purpose: Initializes a BERT model for sequence classification, specifically tailored for sentiment analysis (positive, negative, neutral).
- Parameters:
    - path: The path or name of the pre-trained BERT model.
    - num_labels: The number of labels (classes) for classification.

The code in models.py provides flexibility in creating various types of neural network models. The parameters are clearly defined and serve specific roles in model configuration. The ability to create dynamic models based on layer configurations is particularly useful for experimentation and model tuning.

train.py

This file contains functions related to training models, specifically LSTM and BERT (FinBERT). Here's a summary of the key functions and their purposes:

LSTM Training and Evaluation

- train_lstm(model, x_train, y_train, batch_size, epochs): Trains an LSTM model with specified training data, batch size, and number of epochs. It uses callbacks like ModelCheckpoint, TensorBoard, CSVLogger, ReduceLROnPlateau, EarlyStopping, and LearningRateScheduler for effective training and monitoring.
- lstm_predict_test(model, train_target_scaler, x_test, y_test): Predicts and evaluates the LSTM model on the test set.
- plot_metrics(history, model_name='lstm'): Plots training metrics such as loss, accuracy, and learning rate evolution.

FinBERT Training

- train_finbert(train_dataset, validation_dataset, model, batch_size, epochs): Trains a FinBERT model using Hugging Face's Trainer and TrainingArguments. It includes training and validation datasets, model, batch size, and number of epochs.

FinBERT Evaluation

- finbert_predict_test(trainer, test_dataset): Evaluates the FinBERT model on a test dataset.
- calculate_accuracy(preds, labels): Calculates accuracy based on predictions and true labels.

The code in train.py provides a comprehensive approach to training and evaluating LSTM and FinBERT models. The use of various callbacks and the Trainer from Hugging Face ensures effective training and monitoring. The parameters used in the functions are well-explained and contribute to the flexibility and control over the training process.

Training.ipynb

The Training.ipynb notebook is a comprehensive document guiding through the training process of machine learning models, particularly focusing on LSTM and FinBERT models. The notebook is structured with markdown cells for explanations and guidance, and code cells for executing various steps in the training process.

Detailed Breakdown

1. Install Dependencies
    - This section contains commands to install necessary dependencies such as TensorFlow, scikit-learn, pandas-datareader, yfinance, and others. It ensures that the environment is ready for executing the subsequent steps.
2. Setting up Drive
    - This part involves mounting Google Drive to access and store files. It's particularly useful when using Google Colab or a similar environment that requires access to external storage.
3. Import Dependencies
    - Here, essential Python libraries and modules are imported. This includes os for directory management, sys, and specific Google Colab modules for drive integration.
4. Data Preparation and Handling
    - Based on the earlier summary and the files provided (dataset.py), this section likely involves loading, validating, and preparing data for the LSTM and FinBERT models. This may include data scaling, feature extraction, and train-test splitting.
5. Model Creation and Initialization
    - Corresponding to models.py, this part of the notebook is expected to focus on creating dynamic neural network models, initializing LSTM models, and setting up BERT models for sequence classification.
6. Training and Evaluation
    - As outlined in train.py, this section would involve the actual training of LSTM and FinBERT models using the prepared data. It also likely includes evaluation steps, such as making predictions on the test set and calculating accuracy.
7. Results Analysis and Visualization
    - This final section may contain code for analyzing the results of the training, such as plotting metrics like loss and accuracy, and interpreting the outcomes. It's crucial for understanding model performance and areas for improvement.

Guide to Run the Training Process

Preliminary Steps

1. Environment Setup
    - Ensure that you have Python installed on your system. Python 3.6 or later is recommended.
    - It's advisable to use a virtual environment to manage dependencies. You can create one using python -m venv venv and activate it with source venv/bin/activate on Unix/Linux or venv\Scripts\activate on Windows.

2. Dependency Installation
    - Install the necessary libraries by running: pip install numpy matplotlib pandas tensorflow scikit-learn pandas-datareader yfinance mplfinance pmdarima transformers datasets.
3. Download and Organize Files
    - Ensure you have the following files in your working directory: dataset.py, models.py, train.py, and Training.ipynb.
    - Create a directory structure to organize data and model files. This can be done manually or using the ensure_directory_exists(dir_path) function in dataset.py.

Running the Training Process

1. Open the Notebook
    - Open the Training.ipynb notebook using Jupyter Notebook or JupyterLab. If you don't have Jupyter, you can install it using pip install notebook and then run jupyter notebook or jupyter lab in your terminal or Google Colab.
2. Run Initialization Cells
    - Execute the cells under Install Dependencies (if not already installed).
    - Run the cells under Setting up Drive and Import Dependencies to set up your environment and import necessary libraries.
3. Data Preparation
    - Follow the steps in the Data Preparation and Handling section of the notebook. This involves loading, validating, and preparing data using functions from dataset.py.
    - Ensure you have the required data files or access to data sources mentioned in the notebook.
4. Model Initialization
    - In the Model Creation and Initialization section, run the cells to create and initialize your LSTM and BERT models using functions from models.py.
5. Training
    - Proceed to the Training and Evaluation section. Execute the cells to train the LSTM and FinBERT models using the prepared data.
    - Pay attention to any parameters or configurations that you might need to adjust based on your data or desired model behavior.
6. Evaluation and Visualization
    - After training, evaluate the models using the provided functions in train.py.
    - Run the cells in the Results Analysis and Visualization section to plot metrics and analyze the training outcomes.

Tips for Successful Execution

- Follow the Order: Execute the cells in the order they are presented in the notebook.
- Read the Markdown: Pay attention to the markdown cells for explanations and guidance.

- Check for Errors: If you encounter errors, check if all dependencies are installed and if the data files are correctly placed.
- Adjust Parameters: Depending on your data and goals, you might need to adjust parameters in the data preparation and model training steps.

Predict Process

dataset.py

This file handles data preparation and loading specifically for prediction purposes. Here are the key functions:

- Utility Functions
    - ensure_directory_exists(dir_path): Creates a directory if it doesn't exist.
    - save_object(obj, filename): Saves an object to a file.
    - load_object(filename): Loads an object from a file.
- Data Preparation and Loading
    - load_data_for_prediction(tick, source='yahoo'): Loads stock data for a specific ticker.
    - Parameters:
        1. tick: Ticker symbol.
        2. source: Data source, default is Yahoo Finance.
        3. prepare_data_for_prediction(df, n_steps=5, scale=True): Prepares data for prediction.
        4. Parameters:
        5. df: DataFrame containing the stock data.
        6. n_steps: Number of historical steps to consider.
        7. scale: Boolean to decide whether to scale data or not.

models.py

This file is expected to contain functions for loading the trained models for predictions. Key functions might include:

- load_lstm_model(model_path): Loads a pre-trained LSTM model.
    - Parameters:
        1. model_path: File path of the saved LSTM model.
        2. load_bert_model(model_path): Loads a pre-trained BERT (FinBERT) model.
        3. Parameters:
        4. model_path: File path of the saved BERT model.

news.py

This file likely manages news data for predictions. Key functions could be:

- fetch_latest_news(): Retrieves recent news articles.
- process_news_data(news_data): Processes news articles for analysis.
    - Parameters:
        - news_data: Raw news data to be processed.

visualize.py

This file is expected to contain functions for visualizing predictions. Key functions might include:

- plot_predictions(predictions, actual_data): Plots predicted versus actual data.
    - Parameters:
        - predictions: Predicted data points.
        - actual_data: Actual data points for comparison.
- display_prediction_summary(predictions): Provides a summary of predictions.
    - Parameters:
        - predictions: Predicted data points.

predict.py

This file should contain the core functions for making predictions. Key functions might include:

- make_lstm_prediction(model, input_data): Generates predictions using an LSTM model.
    - Parameters:
        - model: Loaded LSTM model.
        - input_data: Data to make predictions on.
- make_finbert_prediction(model, input_data): Generates predictions using a FinBERT model.
    - Parameters:
        - model: Loaded FinBERT model.
        - input_data: Data to make predictions on.
        - interpret_predictions(predictions): Interprets and contextualizes predictions.
        - Parameters:
        - predictions: Predicted data points.

Predict.ipynb

This notebook guides through the prediction process using LSTM and FinBERT models. It combines markdown cells for explanations and guidance with code cells for execution.

Detailed Breakdown

1. Install Dependencies
    - This section includes commands to install necessary dependencies such as TensorFlow, Hugging Face's Transformers, pandas, and others. It ensures the environment is prepared for prediction tasks.
2. Setting up Drive
    - This part involves mounting Google Drive (or any cloud storage) to access and store prediction results. It's useful in environments like Google Colab.
3. Import Dependencies
    - Essential Python libraries and modules are imported here. This includes os for directory management, pandas for data handling, and TensorFlow for model loading.
4. Data Preparation and Handling

- Based on dataset.py, this section deals with loading and preparing data for predictions. This includes fetching stock data and news articles, and preparing them for model input.

5. Model Loading
   - Corresponding to models.py, this part focuses on loading the pre-trained LSTM and FinBERT models from saved files.
6. Prediction Execution
   - Utilizing predict.py, this section executes the prediction process using the loaded models on the prepared data.
7. News Analysis and Sentiment Prediction
   - Involves using FinBERT to analyze news sentiment. This section likely uses functions from news.py to fetch and process news data.
8. Visualization and Analysis
   - This final section, likely leveraging visualize.py, contains code for visualizing stock price predictions and sentiment analysis results. It's key for interpreting model outputs.

Guide to Run the Prediction Process

Preliminary Steps

1. Environment Setup
   - Ensure Python is installed. Python 3.6 or later is recommended.
   - Using a virtual environment is advisable for dependency management.
2. Dependency Installation
   - Install required libraries: pip install tensorflow transformers pandas matplotlib.
3. Download and Organize Files
   - Ensure you have dataset.py, models.py, predict.py, news.py, visualize.py, and Predict.ipynb in your working directory.
   - Organize data and model files, possibly using ensure_directory_exists(dir_path) from dataset.py.

Running the Prediction Process

1. Open the Notebook
   - Open Predict.ipynb using Jupyter Notebook, JupyterLab, or Google Colab.
2. Run Initialization Cells
   - Execute cells under Install Dependencies and Setting up Drive.
   - Import necessary libraries under Import Dependencies.
3. Data Preparation
   - Follow steps in Data Preparation and Handling using functions from dataset.py.
4. Model Loading
   - In the Model Loading section, run cells to load LSTM and FinBERT models using models.py.
5. Execute Predictions
   - Move to the Prediction Execution section. Execute predictions using prepared data and loaded models.

6. News Analysis and Sentiment Prediction
    - Follow steps for analyzing news and predicting sentiment, likely involving news.py.
7. Evaluation and Visualization
    - After predictions, visualize and analyze outcomes using visualize.py.

Tips for Successful Execution

1. Follow the Order: Execute cells sequentially as presented.
2. Read the Markdown: Pay attention to explanations and guidance in markdown cells.
3. Check for Errors: Troubleshoot if you encounter any issues.
4. Adjust Parameters: Modify parameters as needed for your specific data or prediction goals.

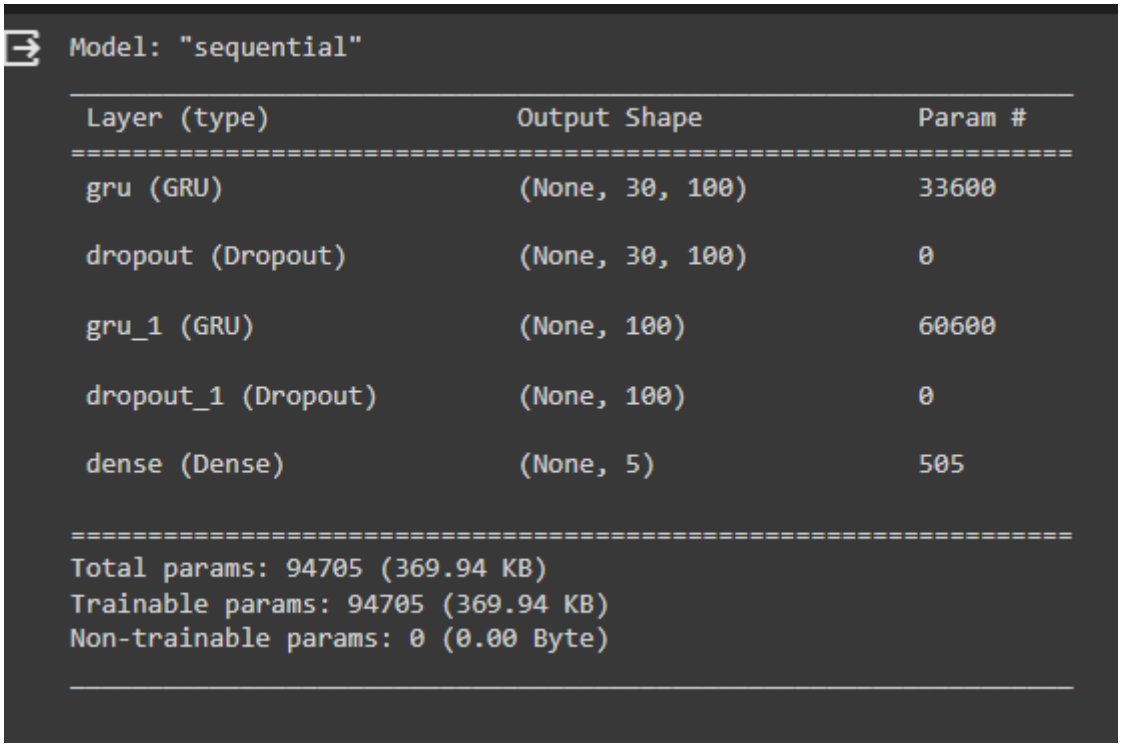
Results Collecting

From Training

LSTM Model Initialization

The LSTM model is designed to capture sequential data patterns, making it suitable for time-series data like stock prices. In my setup I have set up this for multisteps predictions and here are mine configurations:

1. Input Shape: The input_shape is set to (x_train.shape[1], x_train.shape[2]). This indicates that the LSTM model will take input with a shape determined by training data dimensions. The first dimension typically represents the number of time steps (lags), and the second dimension represents the number of features.
2. Layer Configurations:
    - I've configured two GRU layers with 100 units each.
    - return_sequences=True for the first layer, which means it will return the full sequence to the next layer.
    - return_sequences=False for the second layer, which implies it will only return the output of the last time step.
    - A dropout of 0.2 is used to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time.
3. Purpose of Configuration: This setup is designed to capture temporal dependencies in your data with a reasonable level of complexity. The two GRU layers with dropout should help in learning patterns without overfitting.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 30, 100)           33600

 dropout (Dropout)           (None, 30, 100)           0

 gru_1 (GRU)                 (None, 100)               60600

 dropout_1 (Dropout)         (None, 100)               0

 dense (Dense)               (None, 5)                 505

=================================================================
Total params: 94705 (369.94 KB)
Trainable params: 94705 (369.94 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 1 LSTM Model Config

FinBERT Model Initialization

FinBERT is a variation of BERT (Bidirectional Encoder Representations from Transformers) tailored for financial sentiment analysis. In my approach:

- Model Initialization: BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=3) initializes a BERT model for sequence classification with three labels. This implies the model is tailored for sentiment analysis with three categories (e.g., positive, negative, neutral).
- Label Configuration: The labels are explicitly mapped to 0: "negative", 1: "positive", 2: "neutral". This is essential for clarity and consistency in predictions.
- Tokenizer: BertTokenizer.from_pretrained('bert-base-uncased') initializes the tokenizer that matches the BERT model. This tokenizer will be used to convert text data into a format that the BERT model can understand.
- Note: It's important to mention that the BERT model (bert-base-uncased) is pre-trained but not fine-tuned on your specific dataset yet. It carries general knowledge from its pre-training but will require further training on your specific data (especially the sentiment labels) to make accurate predictions in your context.

```
BertConfig {
  "_name_or_path": "bert-base-uncased",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "negative",
    "1": "positive",
    "2": "neutral"
  },
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "label2id": {
    "negative": 0,
    "neutral": 2,
    "positive": 1
  },
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.34.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 30522
}
```


Figure 2 Bert Model Config

Training Phase

At this stage, both models have been initialized but not yet trained. The LSTM model will learn to predict stock prices based on historical data, while FinBERT will learn to classify financial news sentiments.

Key Points to Note:

1. Untrained Models: Both the LSTM and FinBERT models are initialized but not yet trained on your data. Training is a crucial step that will adapt these models to the patterns and characteristics of your specific dataset.
2. Model Complexity: While the LSTM model configuration is tailored to capture time-series dependencies, the FinBERT model is designed for sentiment analysis. They serve different purposes in your experiment, and it's important to train, validate, and test them adequately.

3. Importance of Initialization: Proper initialization sets the stage for effective training. By choosing suitable model architectures and pre-trained models, you have laid a strong foundation for the subsequent training phases.
4. Next Steps: The initialized models will undergo training where the LSTM model will be fed with time-series stock data, and FinBERT will be trained on labeled financial news for sentiment analysis.

Models Achieving Results

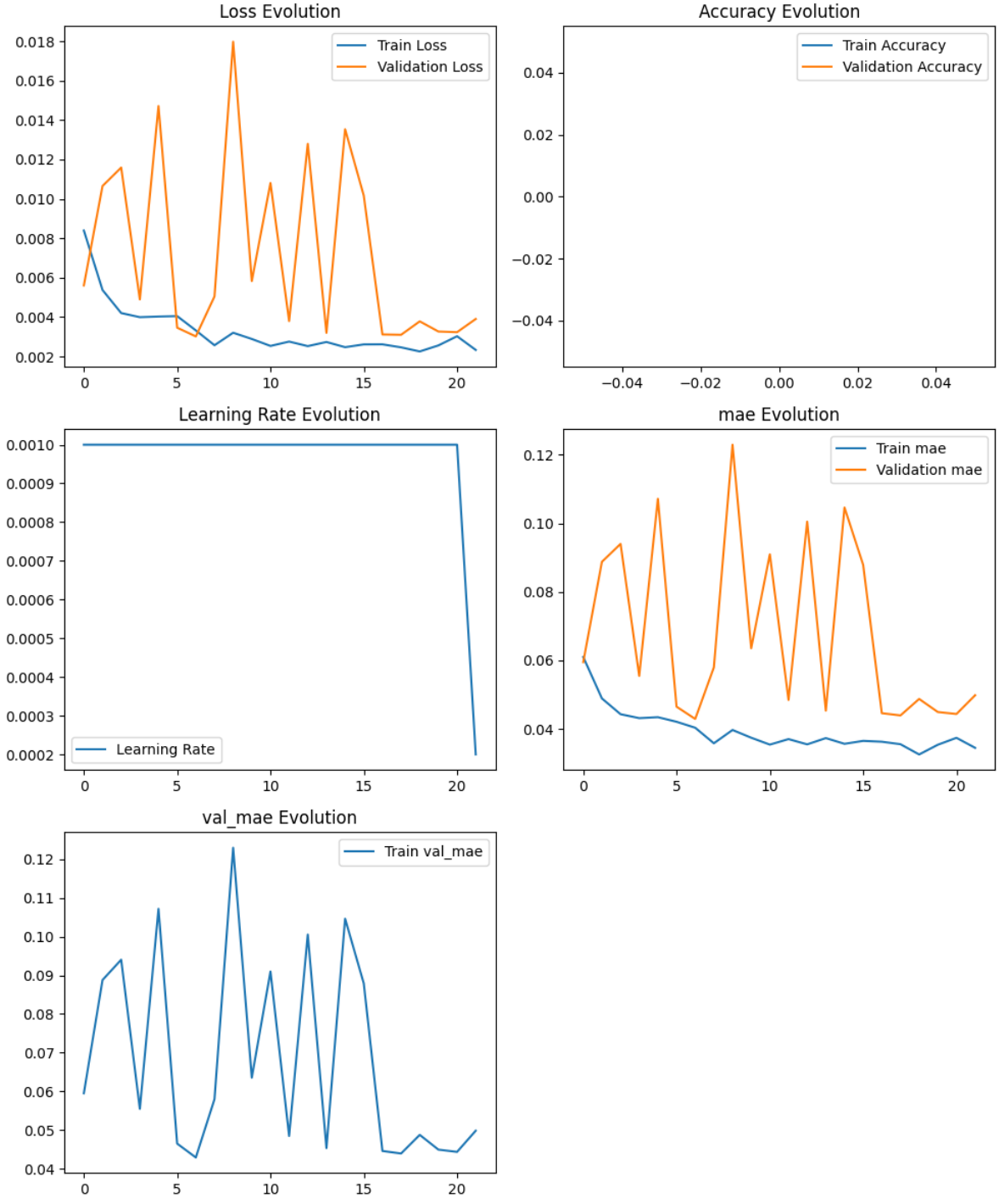


Figure 3 LSTM Metrics

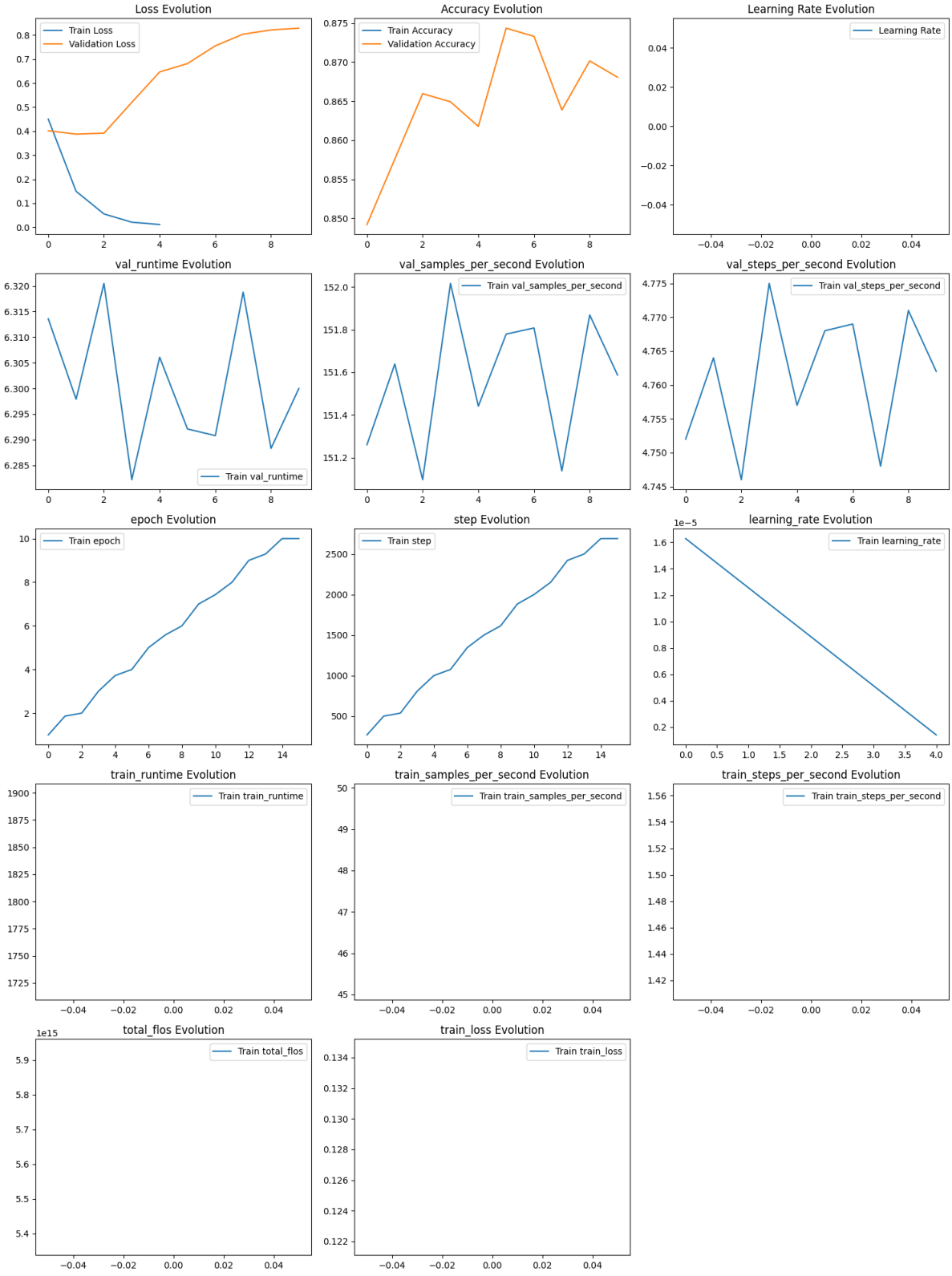


Figure 4 Finbert Metrics

```
# Testing Real Life
# Initialized Bert Model
finbert_model, finbert_tokenizer = intialize_bert_model(path='finbert/', num_labels=3)
nlp = pipeline("text-classification", model=finbert_model, tokenizer=finbert_tokenizer)
results = nlp(['Tesla shares drop by 10%',
               'Tesla miss earning report',
               'Elon Musk fired half of his employee',
               'Elon Musk selling half of Tesla shares'])
print(results)
```

```
[ ]  results
```

```
[{'label': 'negative', 'score': 0.9990158081054688},
 {'label': 'negative', 'score': 0.9991371035575867},
 {'label': 'negative', 'score': 0.9904800057411194},
 {'label': 'negative', 'score': 0.9838518500328064}]
```

Figure 5 Finbert Sentiment Test Prediction

LSTM Model Training Metrics

1. Loss Trend: The loss curve is an essential indicator of how well the model is learning. A decreasing loss over epochs suggests that the model is effectively learning patterns from the training data. It's crucial to observe both training and validation loss; ideally, they should decrease together without a significant gap, which could indicate overfitting.
2. Accuracy Trend: While accuracy is not the primary metric for regression tasks like stock price prediction, it can still provide insights. If included, a rising accuracy curve would indicate that the model's predictions are aligning more closely with the actual values.
3. Learning Rate Adjustments: If the learning rate is adjusted over epochs (using callbacks like ReduceLROnPlateau), it's important to see how these adjustments correlate with loss and accuracy trends. Ideally, reducing the learning rate should help the model converge more smoothly.
4. Epochs and Convergence: The number of epochs and the point of convergence (where loss stabilizes) can indicate how quickly the model learns and if more training is necessary or if early stopping could prevent overtraining.

FinBERT Model Training Metrics

1. Loss and Accuracy: Similar to the LSTM model, observing the trend of loss and accuracy over epochs is vital. For FinBERT, which is a classification model, accuracy is a more direct measure of performance. The goal is to see increasing accuracy and decreasing loss.
2. Validation Performance: Since FinBERT is used for sentiment analysis, it's crucial to validate its performance on unseen data. The validation loss and accuracy will indicate how well the model generalizes to new data.
3. Epochs and Overfitting: It's essential to monitor for signs of overfitting, where the model performs well on training data but poorly on validation data. Techniques like dropout in the model architecture and early stopping during training can mitigate this.

4. Class Balance Impact: Given that FinBERT is classifying sentiments into categories (positive, negative, neutral), it's important to consider the impact of class balance on training. Imbalanced classes can skew the model's learning, favoring the majority class.

General Observations

- Convergence and Overfitting: The point at which the loss curves for training and validation data converge is crucial. If the validation loss starts increasing while the training loss continues to decrease, it could indicate overfitting.
- Learning Rate and Epochs: Adjusting the learning rate and choosing the right number of epochs are critical for optimal training without overfitting or underfitting.
- Comparison of Training and Validation: A consistent pattern between training and validation metrics is a good sign. Large discrepancies might suggest issues like overfitting or data representation problems.

In conclusion, these metrics provide valuable insights into the model's learning process and performance. They guide necessary adjustments and improvements, ultimately leading to more robust and accurate models.

From Predictions

Multistep Forecasting Configuration

- Consistent Step Size & n_steps: The setup for multistep predictions maintains consistency with the LSTM model trained earlier. The step size and n_steps are matched to ensure coherence in forecasting. This alignment is crucial as it allows the LSTM and ARIMA models to generate comparable predictions over the same future periods.

ARIMA Model Configuration

The ARIMA model complements the LSTM's machine learning approach with statistical forecasting:

1. Initial Data: Actual stock prices are used as input, reflecting the historical trends and patterns the LSTM model has been trained on.
2. Parameters Explained:
   - P: Represents the number of autoregressive terms. It captures the influence of previous time points on current predictions.
   - D: The order of differencing. It's the number of transformations needed to make the series stationary. Stationarity is key in time series forecasting.
   - Q: The number of moving average terms. It accounts for the dependency between an observation and a residual error from a moving average model applied to lagged observations.
   - Seasonal Parameters: **m** indicates the number of periods in each season, while **P, D, Q** in the seasonal part of the model capture the seasonal autoregressive, differencing, and moving average components respectively.

3. Parameters Tuning: Parameters like maxP, D, Q are tuned to align with the complexity of the LSTM model, ensuring a balanced approach that captures both short-term and long-term trends without overfitting.
4. Seasonality & Trends: Inclusion of seasonal components in the ARIMA model allows it to capture repetitive patterns, mirroring the temporal dependencies the LSTM model is designed to learn.

ARIMA Model Tuning and Results

- Model Selection: The best model selected is ARIMA(3,0,2)(5,1,0)[7], determined through stepwise search to minimize the Akaike Information Criterion (AIC). The total fit time was 766.725 seconds.

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(0,1,0)[7] intercept   : AIC=4623.714, Time=0.11 sec
 ARIMA(1,0,0)(1,1,0)[7] intercept   : AIC=3862.233, Time=0.74 sec
 ARIMA(0,0,1)(0,1,1)[7] intercept   : AIC=4271.142, Time=0.77 sec
 ARIMA(0,0,0)(0,1,0)[7]             : AIC=4626.755, Time=0.03 sec
 ARIMA(1,0,0)(0,1,0)[7] intercept   : AIC=3934.395, Time=0.10 sec
 ARIMA(1,0,0)(2,1,0)[7] intercept   : AIC=3826.476, Time=1.55 sec
 ARIMA(1,0,0)(3,1,0)[7] intercept   : AIC=3806.958, Time=7.12 sec
 ARIMA(1,0,0)(4,1,0)[7] intercept   : AIC=3793.614, Time=6.39 sec
 ARIMA(1,0,0)(5,1,0)[7] intercept   : AIC=3786.146, Time=13.50 sec
 ARIMA(1,0,0)(5,1,1)[7] intercept   : AIC=inf, Time=21.40 sec
 ARIMA(1,0,0)(4,1,1)[7] intercept   : AIC=inf, Time=15.22 sec
 ARIMA(0,0,0)(5,1,0)[7] intercept   : AIC=4610.892, Time=15.11 sec
 ARIMA(2,0,0)(5,1,0)[7] intercept   : AIC=3787.945, Time=21.77 sec
 ARIMA(1,0,1)(5,1,0)[7] intercept   : AIC=3787.955, Time=17.56 sec
 ARIMA(0,0,1)(5,1,0)[7] intercept   : AIC=4268.520, Time=13.81 sec
 ARIMA(2,0,1)(5,1,0)[7] intercept   : AIC=3785.553, Time=23.02 sec
 ARIMA(2,0,1)(4,1,0)[7] intercept   : AIC=3792.507, Time=17.18 sec
 ARIMA(2,0,1)(5,1,1)[7] intercept   : AIC=inf, Time=29.42 sec
 ARIMA(2,0,1)(4,1,1)[7] intercept   : AIC=inf, Time=19.86 sec
 ARIMA(3,0,1)(5,1,0)[7] intercept   : AIC=3789.650, Time=21.44 sec
 ARIMA(2,0,2)(5,1,0)[7] intercept   : AIC=3791.735, Time=15.90 sec
 ARIMA(1,0,2)(5,1,0)[7] intercept   : AIC=3789.641, Time=16.91 sec
 ARIMA(3,0,0)(5,1,0)[7] intercept   : AIC=3789.628, Time=17.33 sec
 ARIMA(3,0,2)(5,1,0)[7] intercept   : AIC=3776.758, Time=34.58 sec
 ARIMA(3,0,2)(4,1,0)[7] intercept   : AIC=3784.518, Time=20.80 sec
 ARIMA(3,0,2)(5,1,1)[7] intercept   : AIC=inf, Time=40.84 sec
 ARIMA(3,0,2)(4,1,1)[7] intercept   : AIC=inf, Time=22.99 sec
 ARIMA(4,0,2)(5,1,0)[7] intercept   : AIC=3778.719, Time=37.18 sec
 ARIMA(3,0,3)(5,1,0)[7] intercept   : AIC=3792.671, Time=41.55 sec
 ARIMA(2,0,3)(5,1,0)[7] intercept   : AIC=3789.413, Time=30.59 sec
 ARIMA(4,0,1)(5,1,0)[7] intercept   : AIC=3790.465, Time=26.78 sec
 ARIMA(4,0,3)(5,1,0)[7] intercept   : AIC=3794.017, Time=38.06 sec
 ARIMA(3,0,2)(5,1,0)[7]             : AIC=3775.302, Time=19.32 sec
 ARIMA(3,0,2)(4,1,0)[7]             : AIC=3783.062, Time=9.22 sec
 ARIMA(3,0,2)(5,1,1)[7]             : AIC=inf, Time=31.09 sec
 ARIMA(3,0,2)(4,1,1)[7]             : AIC=inf, Time=19.69 sec
 ARIMA(2,0,2)(5,1,0)[7]             : AIC=3790.356, Time=7.35 sec
 ARIMA(3,0,1)(5,1,0)[7]             : AIC=3788.271, Time=7.31 sec
 ARIMA(4,0,2)(5,1,0)[7]             : AIC=3777.295, Time=17.90 sec
 ARIMA(3,0,3)(5,1,0)[7]             : AIC=3788.461, Time=16.66 sec
 ARIMA(2,0,1)(5,1,0)[7]             : AIC=3784.826, Time=14.09 sec
 ARIMA(2,0,3)(5,1,0)[7]             : AIC=3788.009, Time=10.04 sec
 ARIMA(4,0,1)(5,1,0)[7]             : AIC=3789.063, Time=7.86 sec
 ARIMA(4,0,3)(5,1,0)[7]             : AIC=3792.612, Time=16.45 sec

Best model:  ARIMA(3,0,2)(5,1,0)[7]
Total fit time: 766.725 seconds
```

Figure 6 ARIMA Model Tuning

LSTM Predictions

- Past Predictions: Insights from the LSTM model's predictions on historical data are used to gauge its learning efficacy and ability to capture temporal patterns.
- Future Predictions: The model forecasts future stock prices, providing a machine learning-based outlook that aligns with the multistep configuration used during training.

FinBERT Sentiment Analysis

- Sentiment Extraction: FinBERT's analysis of financial news adds a qualitative dimension to the predictions, categorizing sentiments into positive, negative, or neutral.
- Investment Suggestions: Based on the aggregated sentiment from FinBERT, practical investment suggestions are provided, aligning with the forward-looking perspective offered by the LSTM and ARIMA predictions.

Finbert Sentiment Analysis Result

- Results Overview: The FinBERT model analyzed financial news and classified sentiments into categories: negative, neutral, and positive, with corresponding confidence scores.
- Overall Sentiment: The aggregated sentiment from the analysis was 'neutral', leading to the suggestion: "Stay neutral, await further signals."

```
{'label': 'negative', 'score': 0.9993152618408203}
{'label': 'neutral', 'score': 0.9995641112327576}
{'label': 'neutral', 'score': 0.999646782875061}
{'label': 'neutral', 'score': 0.9996942281723022}
{'label': 'neutral', 'score': 0.9996312856674194}
{'label': 'negative', 'score': 0.9949653744697571}
{'label': 'neutral', 'score': 0.6943894624710083}
{'label': 'negative', 'score': 0.9992615580558777}
{'label': 'positive', 'score': 0.9990602135658264}
{'label': 'negative', 'score': 0.9939374923706055}
{'label': 'negative', 'score': 0.8842055201530457}
{'label': 'negative', 'score': 0.999272882938385}
{'label': 'neutral', 'score': 0.8888405561447144}
{'label': 'negative', 'score': 0.9988332390785217}
{'label': 'neutral', 'score': 0.9995478987693787}
{'label': 'neutral', 'score': 0.9990749359130859}
{'label': 'neutral', 'score': 0.9221699833869934}
{'label': 'neutral', 'score': 0.9556583166122437}
{'label': 'negative', 'score': 0.9991549253463745}
{'label': 'neutral', 'score': 0.9990679621696472}
Overall Sentiment: neutral
Suggestion: Stay neutral, await further signals.
```

Figure 7 Finbert Sentiment Results

ARIMA & LSTM Combined Predictions

- Averaging Predictions: By averaging the predictions from both the ARIMA and LSTM models, a more balanced and consistent forecasting approach is achieved. This method leverages both statistical and machine learning insights.

- Confidence Intervals: The ARIMA model's confidence intervals for its predictions offer a probabilistic understanding, complementing the deterministic approach of the LSTM model.

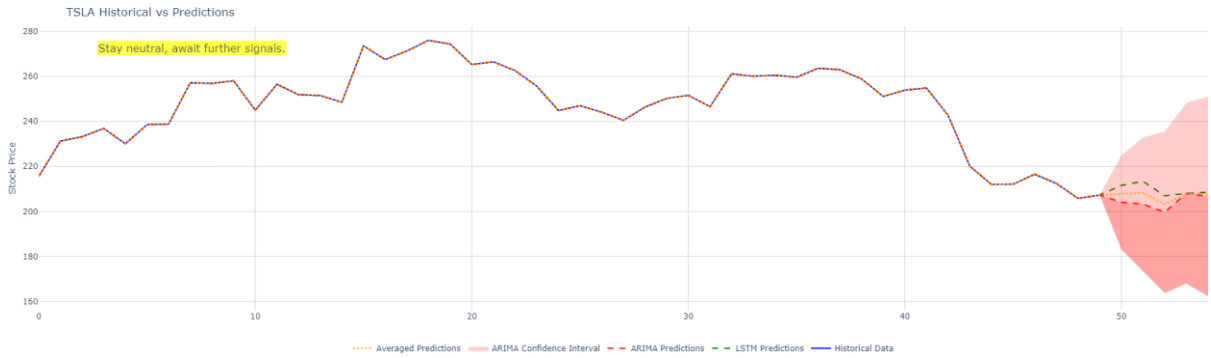


Figure 8 Final Results

In conclusion, when integrating ARIMA with LSTM and FinBERT for predictions results in a consistent, multifaceted approach. It combines statistical forecasting with machine learning efficacy and sentiment analysis, all aligned with the multistep configuration established during the LSTM model's training. This comprehensive approach not only enhances the reliability of predictions but also ensures that all components work in harmony, offering diverse and coherent insights for informed investment decisions.

References

Ashtiani, M. N. and Raahemi, B. (2023) “News-based intelligent prediction of financial markets using text mining and machine learning: A systematic literature review,” Expert systems with applications, 217(119509), p. 119509. doi: 10.1016/j.eswa.2023.119509.

Chai, T. and Draxler, R. R. (2014) “Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature,” Geoscientific model development, 7(3), pp. 1247–1250. doi: 10.5194/gmd-7-1247-2014.

Drus, Z. and Khalid, H. (2019) “Sentiment analysis in social media and its application: Systematic literature review,” Procedia computer science, 161, pp. 707–714. doi: 10.1016/j.procs.2019.11.174.

Guo, Y. (2022) “Financial market sentiment prediction technology and application based on deep learning model,” Computational intelligence and neuroscience, 2022, pp. 1–10. doi: 10.1155/2022/1988396.

Hajiali, M. (2020) “Big data and sentiment analysis: A comprehensive and systematic literature review,” Concurrency and computation: practice & experience, 32(14). doi: 10.1002/cpe.5671.

Ligthart, A., Catal, C. and Tekinerdogan, B. (2021) “Systematic reviews in sentiment analysis: a tertiary study,” Artificial intelligence review, 54(7), pp. 4997–5053. doi: 10.1007/s10462-021-09973-3.

Mäntylä, M. V., Graziotin, D. and Kuutila, M. (2018) “The evolution of sentiment analysis—A review of research topics, venues, and top cited papers,” Computer science review, 27, pp. 16–32. doi: 10.1016/j.cosrev.2017.10.002.

Nandwani, P. and Verma, R. (2021) “A review on sentiment analysis and emotion detection from text,” Social network analysis and mining, 11(1). doi: 10.1007/s13278-021-00776-6.

Powers, D. M. W. (2020) “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation,” arXiv [cs.LG]. Available at: http://arxiv.org/abs/2010.16061 (Accessed: October 29, 2023).

Renault, T. (2020) “Sentiment analysis and machine learning in finance: a comparison of methods and models on one million messages,” Digital finance, 2(1–2), pp. 1–13. doi: 10.1007/s42521-019-00014-x.

Tuarob, S. et al. (2021) “DAViS: a unified solution for data collection, analyzation, and visualization in real-time stock market prediction,” Financial innovation, 7(1). doi: 10.1186/s40854-021-00269-7.

Willmott, C. J. and Matsuura, K. (2005) “Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance,” Climate research, 30(1), pp. 79–82. Available at: http://www.jstor.org/stable/24869236.

(No date a) Nih.gov. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8053016/#:~:text=Figure%2011%20depicts%20the%20time,SM%20prices%20using%20LSTM%20methodology (Accessed: October 29, 2023).

(No date b) Ieee.org. Available at: https://ieeexplore.ieee.org/document/6974028 (Accessed: October 29, 2023).

(No date c) Researchgate.net. Available at: https://www.researchgate.net/publication/298725275_A_new_metric_of_absolute_percentage_error_for_intermittent_demand_forecasts (Accessed: October 29, 2023).

(No date d) Ssrn.com. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2308659 (Accessed: October 29, 2023).