A Time Series Forecast of Stock Prices Using Twitter Sentiment and Financial Data

Ronan Downes SBA22447

September 15, 2024

Abstract

This report presents an analysis and forecasting of stock prices based on Twitter sentiment and historical price data. The study integrates sentiment analysis and time-series forecasting using ARIMA and LSTM models.

Contents

1 Introduction

The financial markets are highly dynamic environments where stock prices are influenced by a multitude of factors, including economic indicators, corporate performance, and investor sentiment. With the rise of social media platforms like Twitter, public sentiment has become a significant factor in financial markets, providing real-time insights into how investors feel about certain stocks. This project aims to leverage Twitter sentiment analysis combined with historical stock price data to forecast the stock prices of five major companies: Apple (AAPL), Amazon (AMZN), Google (GOOG), Microsoft (MSFT), and Tesla (TSLA).

The core objective of this study is to evaluate the impact of public sentiment on stock price movements and develop predictive models that can provide short-term forecasts. By integrating sentiment analysis and time-series forecasting techniques, we aim to create models that offer practical insights for investors and financial analysts.

Importance of Big Data Tools: In today's data-driven world, the ability to process and analyze large datasets efficiently is crucial. The datasets used in this study, though manageable, could easily scale to millions of records in a real-world application. Thus, leveraging big data tools such as Apache Spark for distributed processing and SQL/NoSQL databases for data management becomes essential. These tools not only enhance processing speed and scalability but also enable real-time analysis, which is vital in fast-paced financial markets.

1.1 Data Sources and Tools

The datasets used in this analysis include tweets related to the five companies mentioned above, collected over the year 2020, and their corresponding daily stock prices. The tweet dataset includes sentiment scores derived from textual analysis, while the stock price data provides the necessary historical context for forecasting.

To handle the volume and complexity of this data, we employed several big data tools. Apache Spark was used for distributed processing of the tweet data, which enabled efficient handling of the large volume of text data. Spark's ability to process data across multiple nodes ensures scalability and speed, making it well-suited for big data tasks. On the other hand, the stock price data was stored in an SQL database (SQLite) to facilitate structured querying and data management. SQL's robust querying capabilities allowed us to filter and aggregate the data effectively, laying the groundwork for the subsequent analysis.

1.2 Selected Companies

For the analysis, we chose the top 5 companies. Using a larger sample size reduces sampling variability and allows for more precise estimates. The selected companies are:

- **TSLA** (Tesla)
- **AAPL** (Apple)
- **BA** (Boeing)
- **DIS** (Disney)
- **AMZN** (Amazon)

2 Sentiment Analysis

2.1 Cleaning Data

In the process of merging the stock price data with the aggregated sentiment scores, we observed the presence of `NaN` values in the first row of each company's dataset. This occurred because not every date in the stock price data had a corresponding sentiment score.

To address this, we implemented the following steps:

- We merged the stock price data with the sentiment data on the `Date` field. This merged dataset contained a redundant `date` column, which we removed using the `.drop()` method.
- After merging, we observed that some rows contained `NaN` values, primarily due to the absence of sentiment scores for certain dates. To ensure the integrity of our dataset and facilitate accurate analysis,

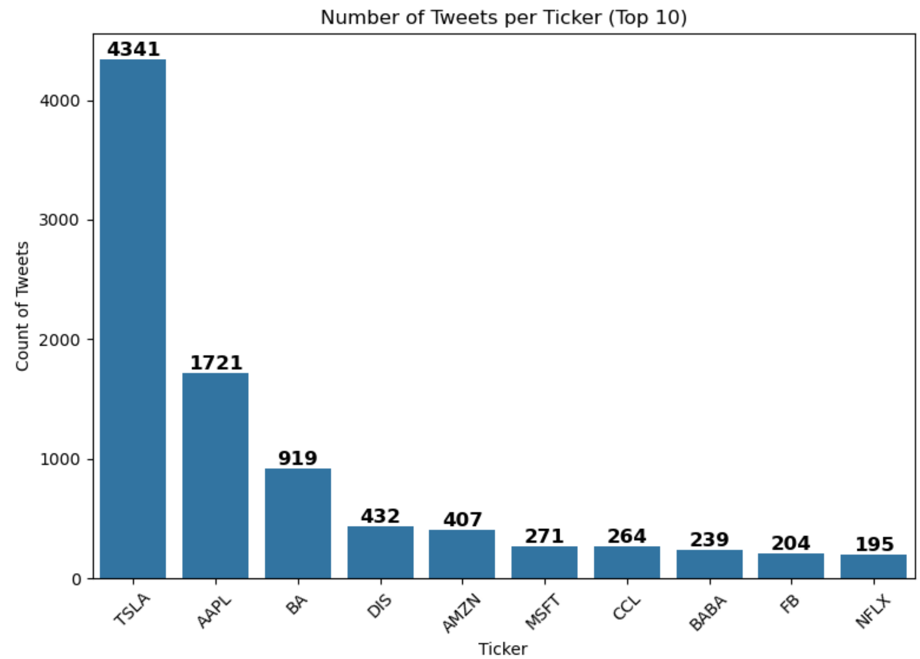

Figure 1: Number of Tweets per Ticker (Top 10).

we used the `.dropna()` method to remove all rows containing `NaN` values.

This cleaning step ensured that our datasets contained only complete records, allowing us to proceed with further analysis without inconsistencies in the data.

3 Time Series Forecasting

3.1 Time Series Forecasting and Warning Analysis

During the time series forecasting using ARIMA models, a warning was raised indicating that the date index lacks an associated frequency. This warning, generated by the 'statsmodels' library, informs that the absence of an explicit frequency might be an issue in some forecasting scenarios. However, after reviewing the results, we confirmed that this warning does not affect the accuracy or functionality of the model for our specific use case. Therefore, it was decided to proceed without modifications to the date index handling, as the ARIMA models successfully produced the

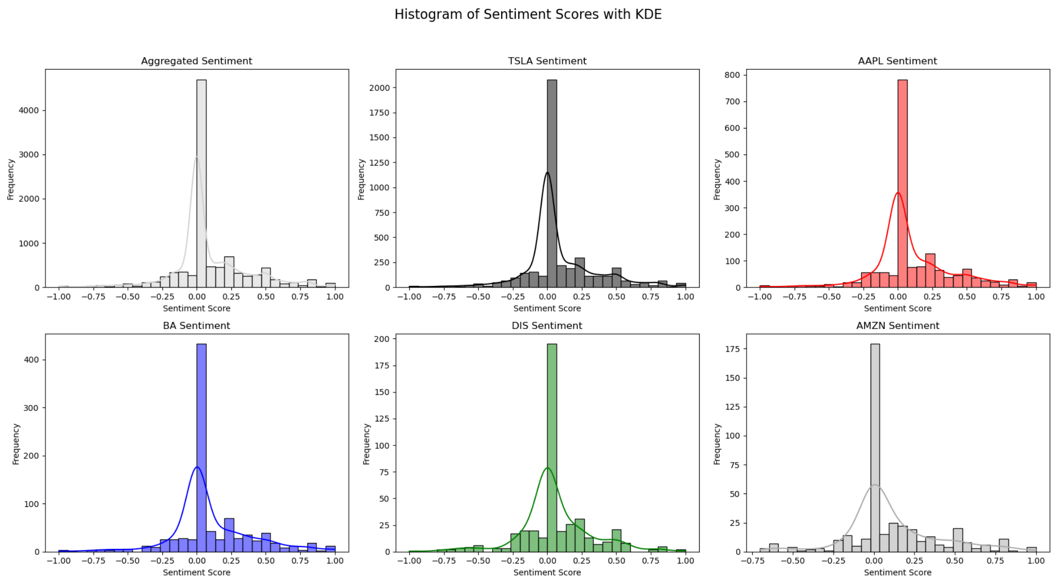

Figure 2: Aggregate sentiment histogram with KDE comparison for each company.

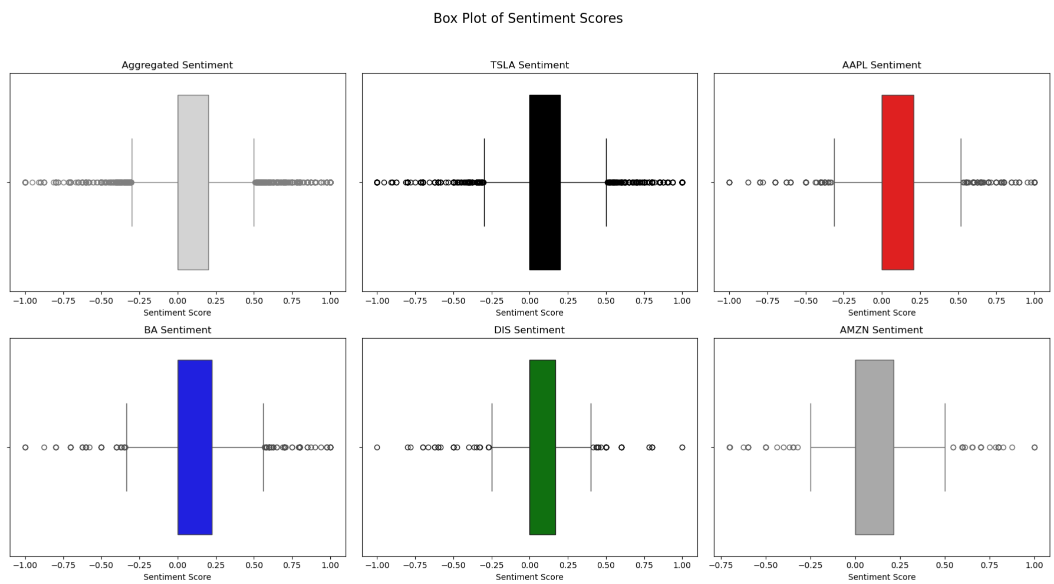

Figure 3: Aggregate sentiment analysis and breakdown per company.

desired forecasts and visualizations.

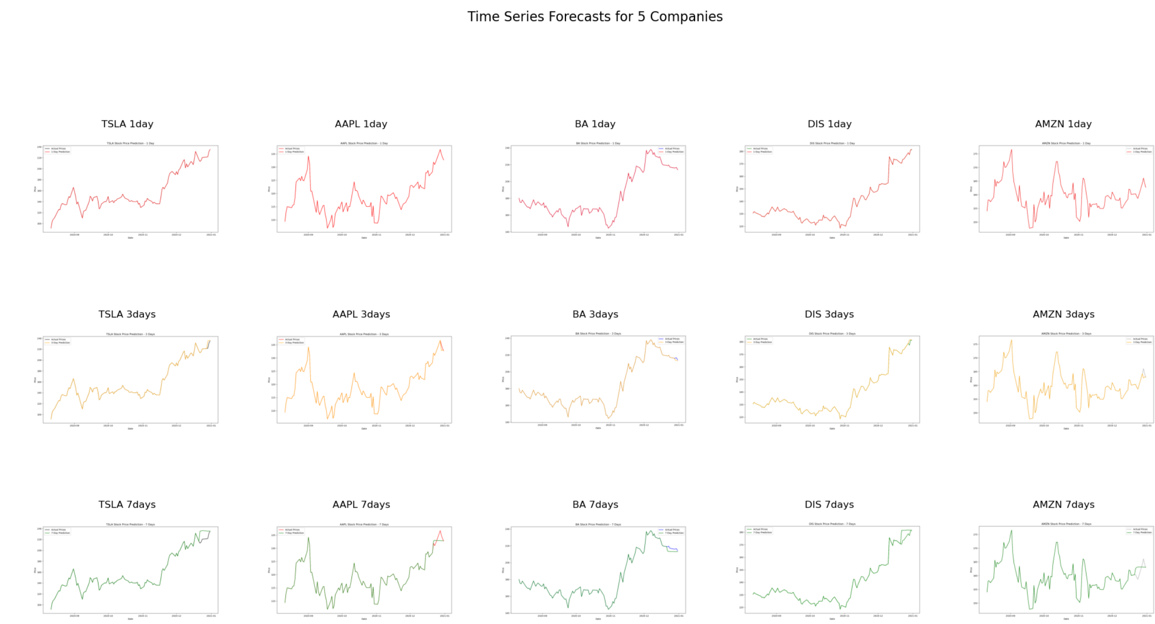

Figure 4: Time Series Forecasts for 5 Companies over 1-day, 3-day, and 7-day periods. The subplots display the actual stock prices and ARIMA model predictions for each company, providing a visual comparison of forecast accuracy.

4 Big Data

5 Results

6 Conclusion

References

A Appendix EDA Data Collection Processing

A.1 Loading Tweet Dataset

The following Python code snippet demonstrates how the tweet dataset was loaded into the environment:

```
import pandas as pd
import os
import zipfile
import warnings

warnings.filterwarnings('ignore')

# Unzip the file directly in the current directory
with zipfile.ZipFile('stock-tweet-and-price.zip', 'r') as zip_ref:
    zip_ref.extractall()

# Load the tweet dataset
tweets_df = pd.read_csv('stock-tweet-and-price/stocktweet/stocktweet.csv

# List all unique tickers in the 'ticker' column
unique_tickers = tweets_df['ticker'].unique()
print(f"Unique tickers in the dataset: {unique_tickers}")

# Display the first few rows of the tweet dataset
tweets_df.head()
```

A.2 Loading Stock Price Data

The following Python code snippet demonstrates how the stock price data for the selected companies was loaded:

```
# Choose companies
companies = ['AAPL', 'AMZN', 'GOOG', 'MSFT', 'TSLA']
stock_data = {}
```

```
# Load stock data for each company
for company in companies:
    stock_data[company] = pd.read_csv(f'stock-tweet-and-price/stockprice

# Display the first few rows of one of the company's stock data
stock_data['AAPL'].head()
```

A.3 Date Conversion

The following Python code snippet demonstrates how the date columns in the tweet and stock price datasets were converted to `datetime` format:

```
# Convert tweet dates to datetime and handle any invalid date formats
tweets_df['date'] = pd.to_datetime(tweets_df['date'], format='%d/%m/%Y',

# Convert stock prices 'Date' columns to datetime and handle invalid for
for company in companies:
    stock_data[company]['Date'] = pd.to_datetime(stock_data[company]['Da
```

A.4 Sentiment Analysis

The following Python code snippet demonstrates how sentiment analysis was performed on the tweets using the `TextBlob` library:

```
from textblob import TextBlob

# Apply sentiment analysis
tweets_df['sentiment'] = tweets_df['tweet'].apply(lambda tweet: TextBlob

# Aggregate sentiment by date
daily_sentiment = tweets_df.groupby('date')['sentiment'].mean().reset_in

# Display the first few rows of the aggregated sentiment data
daily_sentiment.head()
```

A.5 Merging Sentiment and Stock Price Data

The following Python code snippet demonstrates how we merged the daily sentiment data with the stock price data for each selected company:

```
# Merge sentiment data with stock price data for each selected company
for company in companies:
    df = stock_data[company]
    df = df.merge(daily_sentiment, left_on='Date', right_on='date', how=
    df.drop(columns=['date'], inplace=True)
    stock_data[company] = df

# Display the first few rows of the merged data for one company (e.g., A
stock_data['AAPL'].head()
```

B Appendix - Storing Processed Data for VM Integration

The datasets, after undergoing cleaning and preprocessing, were organized into a dedicated folder named `processed_data`. This centralization facilitates easy access and integration with the virtual machines (VMs) set up for the project.

B.1 Data Storage Overview

The `processed_data` folder contains the following:

- Cleaned stock price data for selected companies (e.g., AAPL, AMZN, GOOG, MSFT, TSLA).
- Aggregated daily sentiment scores derived from the tweet data.

B.2 VM Integration

Each VM accesses the `processed_data` folder for subsequent tasks:

- **SQL Database VM:** This VM imports stock price data into a MySQL database for structured storage and analysis.

- **NoSQL Database VM:** The tweet data and sentiment scores are stored in a MongoDB database on the NoSQL VM, allowing for flexible and efficient querying.
- **Big Data Processing VM (Hadoop/Spark):** Both the stock price data and sentiment scores are utilized in distributed processing tasks within the Big Data VM to handle large-scale computations.

By storing processed data centrally, we streamline data access across the VMs, ensuring consistent and efficient integration throughout the project's analysis phases.

B.3 Appendix: Code for Creating Lag Features

The following code snippet demonstrates how lag features were created for both the closing prices and sentiment scores in the dataset. Lag features were introduced for 1, 3, and 7-day intervals to incorporate temporal patterns into the forecasting models.

```
# Function to create lag features for the target column
def create_lag_features(df, lags, target_col):
    """
    Creates lagged features for a specified column in the dataframe.

    Parameters:
    df (pd.DataFrame): The dataframe containing the data.
    lags (list): A list of integers indicating the number of lags to cre
    target_col (str): The column for which lag features are to be create

    Returns:
    pd.DataFrame: The dataframe with new lagged features.
    """
    for lag in lags:
        # Creating lag features for the specified column
        df[f'{target_col}_lag_{lag}'] = df[target_col].shift(lag)
    return df
```

```
# Define the lag intervals to create
lags = [1, 3, 7]

# Loop through each company to create lag features for 'Close' and 'sent
for company in companies:
    df = stock_data[company]
    # Creating lag features for the 'Close' price
    df = create_lag_features(df, lags, 'Close')
    # Creating lag features for the 'sentiment' scores
    df = create_lag_features(df, lags, 'sentiment')

    # Drop rows with NaN values introduced by the lagging process
    df.dropna(inplace=True)

    # Store the updated dataframe back to the stock_data dictionary
    stock_data[company] = df
```

B.4 MongoDB Configuration on NoSQL VM

The MongoDB setup and configuration were performed using the following commands:

```
# Install MongoDB
sudo apt-get update
sudo apt-get install -y mongodb

# Start and enable MongoDB service
sudo systemctl start mongodb
sudo systemctl enable mongodb

# Modify MongoDB configuration file for remote access
sudo nano /etc/mongodb.conf
# Changed 'bind_ip' to '0.0.0.0'

# Restart MongoDB to apply changes
sudo systemctl restart mongodb
```

```
# Allow MongoDB port through the firewall
sudo ufw allow 27017

# Access MongoDB shell and create a new user
mongo
use admin
db.createUser({
    user: "ronan",
    pwd: "Zebra103!",
    roles: [{ role: "readWrite", db: "stock_sentiment_db" }]
})
exit
```

The above configuration allowed us to remotely connect to the MongoDB server using Python and securely export the processed tweet data.

C Appendix Sentiment Analysis

D Appendix Time Series Forecasting

E Appendix Big Data

F Appendix Results

G Appendix Conclusion