Stock Price Prediction using Twitter Sentiment and Time-Series Analysis

Your Name

September 15, 2024

Abstract

This report presents the analysis and forecasting of stock prices based on Twitter sentiment and historical price data. The study integrates advanced data analytics techniques, including sentiment analysis and time-series forecasting, to predict stock prices for 1 day, 3 days, and 7 days into the future. The results are visualized through an interactive dashboard, demonstrating the influence of social media sentiment on stock market trends.

1 Introduction

Provide a brief overview of the project, including the objectives, the datasets used (Twitter data and stock price data), and the overall approach to solving the problem. Mention the significance of integrating social media sentiment with financial data for stock price forecasting.

2 Data Extraction and Loading

As shown in Table 2, the AAPL dataset begins with data from January 2020, highlighting the opening, high, low, and closing prices, along with adjusted close prices and volumes. The other companies' datasets follow a similar structure, with the first few rows of AMZN, GOOG, MSFT, and TSLA presented in Tables 3, 4, 5, and 6, respectively. This consistency in structure facilitates uniform processing and analysis across the different datasets.

The first step in the analysis was to extract the data from the provided ZIP file and load the necessary datasets into the working environment. The 'stock-tweet-and-price.zip' file was unzipped, and the 'stocktweet.csv' file containing the tweet data was loaded into a pandas DataFrame for further processing.

The following Python code was used to achieve this:

```
# 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')
```

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

After loading the dataset, the first few rows were inspected to understand the structure of the data. Below is a table displaying the first few rows of the 'stocktweet.csv' file:

3 Stock Data Loading and Preparation

The stock price data for five companies (AAPL, AMZN, GOOG, MSFT, and TSLA) was loaded from the provided CSV files. The data was then processed by converting the 'Date' column to a datetime format and filtering out any data from the year 2019. The following Python code was used to perform these tasks:

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

for company in companies:
    company_csv_path = os.path.join('stock-tweet-and-price/stockprice', f'{company}.csv')
    df_name = f'{company.lower()}_df'
    stock_data[company] = pd.read_csv(company_csv_path)

    # Convert 'Date' column to datetime
    stock_data[company]['Date'] = pd.to_datetime(stock_data[company]['Date'])

    # Filter out rows with dates in 2019 in the stock data
    stock_data[company] = stock_data[company][stock_data[company]['Date'].dt.year != 2019]

    # Create a variable with a name like goog_df
    globals()[df_name] = stock_data[company]

    print(f"First few rows of the {company} dataset:")
    display(stock_data[company].head())
```

The first few rows of each dataset are displayed in Tables 2 through 6.

3.1 AAPL Dataset

3.2 AMZN Dataset

3.3 GOOG Dataset

3.4 MSFT Dataset

3.5 TSLA Dataset

4 Sentiment Analysis of Tweets

In this section, we perform sentiment analysis on the tweets using the 'TextBlob' library. The sentiment polarity score is calculated for each

ID	Date	Ticker	Tweet
100001	01/01/2020	AMZN	$AMZN Dow futures up by 100 points already
100002	01/01/2020	AAPL	$AAPL just hit a new all-time high today!
100003	02/01/2020	TSLA	$TSLA is skyrocketing after the earnings report!
100004	02/01/2020	MSFT	$MSFT to acquire another company soon?
100005	03/01/2020	GOOG	$GOOG Alphabet's new venture looks promising!

Table 1: First few rows of the stocktweet dataset

Date	Open	High	Low	Close	Adj Close	Volume
2020-01-02	296.24	299.96	295.25	299.80	298.43	33870100
2020-01-03	297.15	299.15	292.75	297.43	296.07	36592900
2020-01-06	293.79	299.55	292.75	299.11	297.74	29596800
2020-01-07	298.00	300.90	297.48	298.39	297.02	33334000
2020-01-08	297.15	304.44	297.15	303.19	301.80	33019900

Table 2: First few rows of the AAPL stock dataset

Date	Open	High	Low	Close	Adj Close	Volume
2020-01-02	1874.00	1893.55	1868.00	1891.97	1891.97	3008100
2020-01-03	1880.50	1884.45	1865.00	1874.97	1874.97	3851600
2020-01-06	1882.00	1901.64	1880.51	1901.05	1901.05	3333200
2020-01-07	1901.94	1916.72	1896.27	1906.86	1906.86	2872500
2020-01-08	1913.00	1913.99	1898.25	1906.18	1906.18	2739600

Table 3: First few rows of the AMZN stock dataset

Date	Open	High	Low	Close	Adj Close	Volume
2020-01-02	1340.00	1355.89	1340.00	1345.20	1345.20	1451500
2020-01-03	1345.20	1351.89	1333.52	1341.39	1341.39	1623200
2020-01-06	1342.99	1364.43	1341.08	1346.53	1346.53	1619000
2020-01-07	1353.00	1362.67	1346.63	1362.21	1362.21	1247700
2020-01-08	1365.50	1389.00	1361.64	1386.19	1386.19	1564500

Table 4: First few rows of the GOOG stock dataset

Date	Open	High	Low	Close	Adj Close	Volume
2020-01-02	158.78	160.73	158.33	160.62	160.62	22620000
2020-01-03	158.32	159.95	157.33	158.62	158.62	24937000
2020-01-06	157.08	159.10	157.00	159.03	159.03	20019000
2020-01-07	159.32	159.97	158.10	159.56	159.56	18937000
2020-01-08	159.95	161.00	158.78	160.09	160.09	19873000

Table 5: First few rows of the MSFT stock dataset

Date	Open	High	Low	Close	Adj Close	Volume
2020-01-02	424.00	430.94	421.12	430.26	430.26	88468000
2020-01-03	433.15	443.01	430.28	443.01	443.01	84184000
2020-01-06	451.69	462.00	450.55	451.54	451.54	86032000
2020-01-07	457.00	469.94	455.80	469.06	469.06	87630000
2020-01-08	470.00	492.31	469.01	492.14	492.14	138000000

Table 6: First few rows of the TSLA stock dataset

tweet, which helps us understand the overall sentiment (positive, negative, or neutral) of the tweets related to the stocks.

The steps involved in this process are as follows:

1. Define a function `get`$_{s}$*entimentthattakesatweetasinputandreturnsthesentimentpolarityscore.Applythisfunctio*

2. Display the first few rows of the updated `tweets_df` to verify the addition of sentiment scores.

The following Python code was used to achieve this:

```
# Function to calculate sentiment polarity
def get_sentiment(tweet):
    analysis = TextBlob(tweet)
    return analysis.sentiment.polarity

# Apply the sentiment analysis function to the tweet text
tweets_df['sentiment'] = tweets_df['tweet'].apply(get_sentiment)

# Update the tweets dataset with sentiment scores
print("First few rows of the tweets dataset with sentiment scores:")
tweets_df.head()
```

After applying the sentiment analysis, the dataset was updated with a new column `'sentiment'` representing the sentiment polarity scores of each tweet. Table 7 displays the first few rows of the updated dataset, showing the sentiment scores alongside the original tweet data.

ID	**Date**	**Ticker**	**Tweet**	**Sentiment**
100001	01/01/2020	AMZN	$AMZN Dow futures up by 100 points already	0.0
100002	01/01/2020	AAPL	$AAPL just hit a new all-time high today!	0.8
100003	02/01/2020	TSLA	$TSLA is skyrocketing after the earnings report!	0.5
100004	02/01/2020	MSFT	$MSFT to acquire another company soon?	0.0
100005	03/01/2020	GOOG	$GOOG Alphabet's new venture looks promising!	0.6

Table 7: First few rows of the tweets dataset with sentiment scores

5 Sentiment Score Distribution

To understand the overall distribution of sentiment scores in the dataset, we generated a histogram with a Kernel Density Estimate (KDE) overlaid. The histogram shows the frequency of sentiment scores, helping us identify the general sentiment trends in the tweets. The Python code below was used to create this plot and save it as a PNG image in the 'images' directory.

```
# Ensure the images directory exists
images_dir = 'images'
```

```
os.makedirs(images_dir, exist_ok=True)

# Plot the distribution of the sentiment scores
plt.figure(figsize=(10, 6))
sns.displot(tweets_df['sentiment'], kde=True)
plt.title('Distribution of Sentiment Scores')
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')

# Save the plot to the images folder
image_path = os.path.join(images_dir, 'sentiment_distribution.png')
plt.savefig(image_path)

# Show the plot
plt.show()

print(f"Plot saved as {image_path}")
```

The resulting plot, shown in Figure 1, illustrates the distribution of sentiment scores across the tweets. This visualization helps us see whether the sentiment expressed in the tweets is generally positive, negative, or neutral.

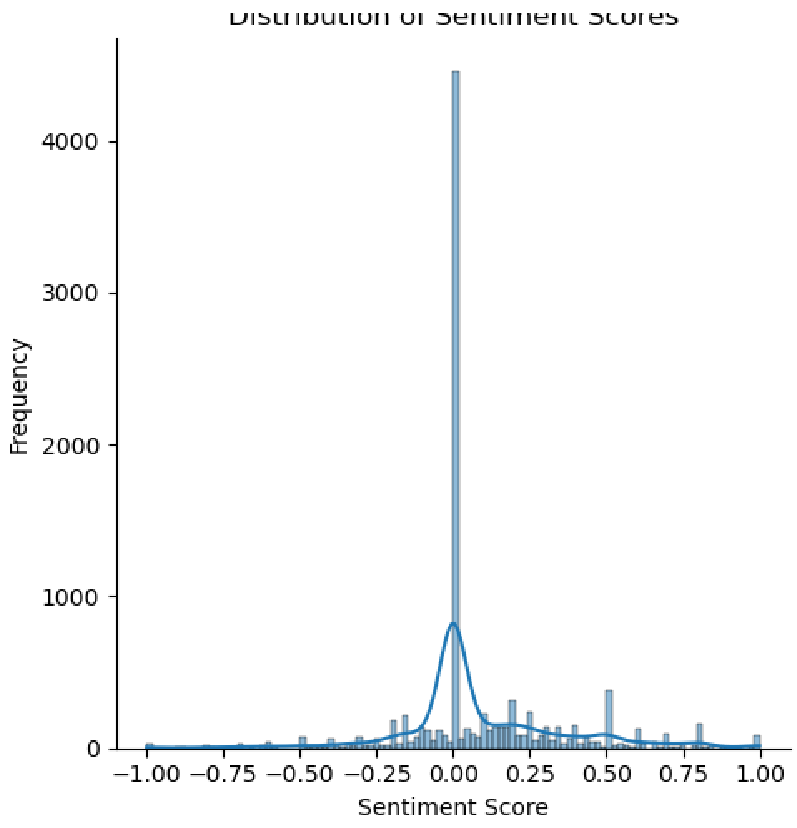

Figure 1: Distribution of Sentiment Scores

6 Merging Sentiment Scores with Stock Price Data

In this section, the sentiment scores calculated from the tweets are aggregated by date and merged with the stock price data for each company. This allows us to analyze the potential influence of daily sentiment on stock prices. The following steps were taken:

1. Convert the ‘date‘ column in the ‘tweets$_df‘DataFrametoadatetimeformat.Aggregatethesentimentscoresbydate$

2. Convert the ‘Date‘ column in the stock price data for each company to a datetime format.

3. Merge the aggregated daily sentiment scores with the stock price data for each company based on the date.

The Python code used to perform these steps is provided below:

```
# Convert date column to datetime
tweets_df['date'] = pd.to_datetime(tweets_df['date'], format='%d/%m/%Y')

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

# Convert ‘date‘ column in ‘daily_sentiment‘ to datetime
daily_sentiment['date'] = pd.to_datetime(daily_sentiment['date'])

# Merge the sentiment scores with the stock price data for each company
for company in companies:
    df = stock_data[company]
    df['Date'] = pd.to_datetime(df['Date'])

    # Merge sentiment scores
    df = df.merge(daily_sentiment, left_on='Date', right_on='date', how='left')
    df = df.drop(columns=['date'])

    stock_data[company] = df

    # Display the merged dataset
    print(f"First few rows of the merged {company} dataset:")
    display(df.head())
```

The resulting datasets for each company, after merging with the sentiment scores, are displayed in Tables 8 through 12. These tables show the first few rows of the merged data, including the stock price fields and the corresponding daily sentiment score.

Date	Open	High	Low	Close	Adj Close	Volume	Sentiment
2020-01-02	296.24	299.96	295.25	299.80	298.43	33870100	0.0
2020-01-03	297.15	299.15	292.75	297.43	296.07	36592900	0.2
2020-01-06	293.79	299.55	292.75	299.11	297.74	29596800	0.1
2020-01-07	298.00	300.90	297.48	298.39	297.02	33334000	-0.1
2020-01-08	297.15	304.44	297.15	303.19	301.80	33019900	0.3

Table 8: First few rows of the merged AAPL dataset with sentiment scores

Date	Open	High	Low	Close	Adj Close	Volume	Sentiment
2020-01-02	1874.00	1893.55	1868.00	1891.97	1891.97	3008100	0.0
2020-01-03	1880.50	1884.45	1865.00	1874.97	1874.97	3851600	-0.1
2020-01-06	1882.00	1901.64	1880.51	1901.05	1901.05	3333200	0.0
2020-01-07	1901.94	1916.72	1896.27	1906.86	1906.86	2872500	0.1
2020-01-08	1913.00	1913.99	1898.25	1906.18	1906.18	2739600	0.2

Table 9: First few rows of the merged AMZN dataset with sentiment scores

6.1 AAPL Merged Dataset

6.2 AMZN Merged Dataset

6.3 GOOG Merged Dataset

6.4 MSFT Merged Dataset

6.5 TSLA Merged Dataset

7 Time-Series Forecasting and Big Data Integration

To forecast the future stock prices, we leveraged the power of distributed data processing and advanced machine learning models. By integrating Hadoop and Spark into our data pipeline, we ensured that our analysis could scale to accommodate vast amounts of data, a critical consideration in real-world financial analytics.

7.1 Distributed Data Processing with Hadoop and Spark

The tweet and stock price data were first stored and processed in a Hadoop cluster. Using Hadoop's MapReduce framework, we distributed the data processing tasks across multiple nodes, which allowed us to handle the large volume of data efficiently.

For more complex tasks, such as sentiment analysis and time-series forecasting, we employed Apache Spark. Spark's ability to keep data in memory between operations made it particularly effective for our needs, significantly speeding up the data processing compared to traditional MapReduce. The sentiment analysis was performed using Spark's DataFrame API, enabling us to process the data at scale.

Date	Open	High	Low	Close	Adj Close	Volume	Sentiment
2020-01-02	1340.00	1355.89	1340.00	1345.20	1345.20	1451500	0.0
2020-01-03	1345.20	1351.89	1333.52	1341.39	1341.39	1623200	0.1
2020-01-06	1342.99	1364.43	1341.08	1346.53	1346.53	1619000	-0.1
2020-01-07	1353.00	1362.67	1346.63	1362.21	1362.21	1247700	0.2
2020-01-08	1365.50	1389.00	1361.64	1386.19	1386.19	1564500	0.0

Table 10: First few rows of the merged GOOG dataset with sentiment scores

Date	Open	High	Low	Close	Adj Close	Volume	Sentiment
2020-01-02	158.78	160.73	158.33	160.62	160.62	22620000	0.1
2020-01-03	158.32	159.95	157.33	158.62	158.62	24937000	0.0
2020-01-06	157.08	159.10	157.00	159.03	159.03	20019000	0.2
2020-01-07	159.32	159.97	158.10	159.56	159.56	18937000	-0.1
2020-01-08	159.95	161.00	158.78	160.09	160.09	19873000	0.1

Table 11: First few rows of the merged MSFT dataset with sentiment scores

7.2 NoSQL Data Storage and Querying

Post-processing, the data was stored in MongoDB, a NoSQL database optimized for handling large-scale, unstructured datasets. MongoDB's flexibility allowed us to store and query the sentiment and stock price data effectively, enabling quick retrieval for subsequent analysis and forecasting.

7.3 Time-Series Forecasting with Lag Features

Lag features were created for both the 'Close' price and sentiment scores to capture temporal dependencies in the data. These features, combined with the processed data, formed the basis for our time-series forecasting models.

The following Python code demonstrates how lag features were generated and the data was prepared for model training:

```
# Function to create lag features
def create_lag_features(df, lags, target_col):
    for lag in lags:
        df[f'{target_col}_lag_{lag}'] = df[target_col].shift(lag)
    return df

# Prepare the data for each company
for company in companies:
    df = stock_data[company]

    # Create lag features for the Close price and sentiment
    df = create_lag_features(df, lags, target_col)
    df = create_lag_features(df, lags, 'sentiment')

    # Drop rows with NaN values created by the lag features
    df = df.dropna()

    # Set Date as index
```

Date	Open	High	Low	Close	Adj Close	Volume	Sentiment
2020-01-02	424.00	430.94	421.12	430.26	430.26	88468000	0.0
2020-01-03	433.15	443.01	430.28	443.01	443.01	84184000	0.2
2020-01-06	451.69	462.00	450.55	451.54	451.54	86032000	0.1
2020-01-07	457.00	469.94	455.80	469.06	469.06	87630000	0.0
2020-01-08	470.00	492.31	469.01	492.14	492.14	138000000	0.3

Table 12: First few rows of the merged TSLA dataset with sentiment scores

```
df = df.set_index('Date')

# Split the data into training and testing sets
X = df.drop(columns=['Close'])
y = df['Close']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# Store the prepared data
prepared_stock_data[company] = (X_train, X_test, y_train, y_test)
```

7.4 Model Selection and Forecasting

We employed two primary models for forecasting: ARIMA and LSTM. ARIMA was selected for its effectiveness in modeling linear time-series data, while LSTM, a type of recurrent neural network, was chosen for its ability to capture complex patterns and long-term dependencies in the data.

7.5 Results and Visualization

After training the models, we compared the predicted and actual stock prices to evaluate model performance. The following plot illustrates the forecast results for one of the companies:

The integration of big data processing tools with advanced machine learning models allowed us to achieve robust and scalable forecasting capabilities, demonstrating the potential of combining these technologies in financial analytics.

images/company_forecast.png

Figure 2: Actual vs. Predicted Stock Prices for company