Stock Movement Prediction Using Data Scraping and Machine Learning

BY

SRI SARU KUMAR S

Abstract

This project focuses on predicting stock price movements using a combination of data scraping, feature engineering, and machine learning models. The primary objective was to create a predictive model capable of forecasting stock trends based on historical price data and external features such as financial news sentiment. Data was collected through web scraping from reputable financial news sources and stock market APIs, providing a rich dataset for training and evaluation.

The project leveraged several machine learning algorithms, including a Random Forest Classifier, Gradient Boosting Classifier, and an LSTM (Long Short-Term Memory) model tailored for time-series data. The models were trained using a combination of hyperparameter tuning and cross-validation to optimize performance. Evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC were employed to assess the models' effectiveness.

Key findings highlighted the performance of the Gradient Boosting Classifier, which outperformed the Random Forest and LSTM models in terms of accuracy and robustness. Challenges encountered included handling imbalanced data and ensuring data consistency during the scraping process. Despite these challenges, the project achieved a viable predictive model, providing insights into potential future improvements, such as incorporating real-time data feeds and additional market indicators for enhanced prediction accuracy.

Introduction:

Background:

Stock market prediction is a critical task in financial markets, as it helps investors and traders make informed decisions regarding their investments. Accurate predictions of stock price movements can provide a significant competitive edge, enabling stakeholders to maximize returns and manage risks more effectively. With the advent of data science and machine learning, there has been a growing interest in using advanced analytical techniques to forecast stock trends based on historical data, sentiment analysis, and other external economic indicators.

The volatility of financial markets, driven by a myriad of factors such as economic data, geopolitical events, and investor sentiment, makes stock movement prediction both complex and essential. As such, financial analysts and data scientists strive to build robust models capable of extracting meaningful patterns from historical prices and supplementary data to forecast future market behavior.

Problem Statement:

The main problem addressed in this project is to predict the direction of stock price movements (up or down) based on a combination of historical stock price data and external features, such as sentiment analysis from financial news articles. Traditional methods of stock prediction often rely solely on historical data, but incorporating additional external factors can potentially improve prediction accuracy and provide more comprehensive insights.

The project aims to create and evaluate machine learning models capable of classifying stock price movements, helping investors make data-driven decisions.

Project Objectives:

The main objectives of this project are:

Data Collection: Develop a web scraper to collect historical stock price data and sentiment information from financial news sources.

Data Preprocessing: Clean and preprocess the data to ensure consistency and prepare it for model training, including feature engineering and scaling.

Model Development: Implement machine learning models, including Random Forest, Gradient Boosting, and an LSTM neural network, to predict stock price movements.

Model Evaluation: Assess the performance of the models using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

Comparison and Insights: Compare the performance of the different models to determine the most effective approach and highlight areas for potential improvement.

Future Enhancements: Suggest potential directions for future work, such as integrating real-time data streams and exploring additional external features.

Data Scraping Process

Data Source

The data for this project was sourced from multiple platforms to capture both historical stock prices and sentiment data from financial news. The primary sources included:

- **Financial Market Data**: Stock price data was collected using APIs like Alpha Vantage and Yahoo Finance.
- **News and Sentiment Data**: News articles related to the financial markets were scraped from websites such as [news site name] and [other relevant sources].

These sources provided the raw data required for training and evaluating predictive models by capturing stock prices, trading volumes, and sentiment indicators.

Methodology

The data collection was done using Python libraries to automate the process efficiently. Key tools used were:

- **BeautifulSoup**: Used for parsing and scraping web data from HTML pages.
- **Selenium**: Used for interacting with dynamically loaded content that requires JavaScript to render.
- **Pandas**: Utilized for data manipulation and storage.
- **Requests**: Used for making HTTP requests to retrieve data from APIs.

Code Snippet:

```
import praw
import re
import nltk
import csv
from nltk.corpus import stopwords
import pandas as pd  # For saving to CSV
import time

# Download NLTK stopwords (run once)
nltk.download('stopwords')

# Initialize stopwords
stop_words = set(stopwords.words('english'))

# Configure PRAW with your Reddit API credentials
reddit = praw.Reddit(
    client_id='dGZIjyPDfm0AGILS_Cnmmw',
    client_secret='G9x6B53pR-qyMiQABLm-U1CnJO3Spg',
    user_agent='python:stock_scraper:1.0 (by u/MetalMaleficent6947)'
)

# Function to scrape data from a subreddit with pagination support
```

```
def scrape_reddit(subreddit_name, limit=1000):
    subreddit = reddit.subreddit(subreddit_name)
    posts = []
    count = 0

    # Scrape posts in batches, adding a time delay to prevent hitting rate limits
    for post in subreddit.new(limit=None):  # Fetches all available posts
        if count >= limit:
            break
        posts.append({
            'title': post.title,
            'selftext': post.selftext,
            'score': post.score,
            'created_utc': post.created_utc
        })
        count += 1

        # Delay to prevent rate limiting
        time.sleep(1)  # Adjust the delay as needed (e.g., 1 second)

    return posts

# Text preprocessing function
def preprocess_text(text):
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Remove special characters and numbers
    text = re.sub(r'\W+|\d+', ' ', text)
    # Convert to lowercase
    text = text.lower()
    # Remove stopwords
    text_tokens = text.split()
    text_tokens = [word for word in text_tokens if word not in stop_words]
    # Rejoin tokens
    return ' '.join(text_tokens)

# Save data to a CSV file
def save_to_csv(data, filename):
    # Convert list of dictionaries to DataFrame
    df = pd.DataFrame(data)
    # Save DataFrame to CSV
    df.to_csv(filename, index=False)
    print(f"Data saved to {filename}")

# Example usage
if __name__ == "__main__":
    # Scrape a larger number of posts from a subreddit (e.g., r/wallstreetbets)
    scraped_data = scrape_reddit(subreddit_name="wallstreetbets", limit=1000)

    # Preprocess the title and selftext of each post
    for post in scraped_data:
        post['cleaned_title'] = preprocess_text(post['title'])
        post['cleaned_selftext'] = preprocess_text(post['selftext'])

    # Save the processed data to a CSV file
    save_to_csv(scraped_data, "/content/drive/MyDrive/Intern/reddit_scraped_data.csv")
```

Challenges Encountered

Several challenges arose during the data scraping process:

1. **Rate-Limiting**: Many APIs imposed rate limits on the number of requests that could be made in a specific time period, which hindered continuous data collection.

2. **Dynamic Content**: Some websites required interaction with JavaScript to display relevant content, making it difficult to scrape data using standard methods.
3. **Missing or Inconsistent Data**: Financial data often had missing values or formatting inconsistencies that needed to be addressed during preprocessing.
4. **Data Overload**: Large datasets could lead to performance issues, especially when working with real-time data streaming.

Solutions

To overcome the challenges, the following techniques were employed:

- **Rate-Limiting Management**: Implemented a retry mechanism with a time delay to ensure that requests stayed within API limits and avoided bans. For example, the time.sleep() function was used to introduce delays between API calls.
- **Handling Dynamic Content**: Utilized **Selenium** to interact with web pages that loaded data dynamically via JavaScript. This allowed for more flexibility in navigating through pages and capturing the content.
- **Data Imputation and Preprocessing**: Applied data cleaning techniques such as forward filling and back filling to address missing values, ensuring the model could be trained without issues.
- **Parallel Processing**: Used the concurrent.futures library to parallelize data collection tasks, optimizing the time required to scrape and download large datasets.