WiDS Project: Market Mood and Moves: Sentiment-Driven Stock Prediction

Name- Vihaan Vajifdar
Mentors- Meet & Sarthak

In this project, I am building a sentiment-driven stock prediction engine. My core objective is to move beyond traditional financial models that view markets as purely rational. Instead, I am adopting a **Behavioral Finance** approach, which treats the market as a "voting machine" in the short term, driven heavily by human psychology and news sentiment.

Over the past three weeks, I have progressed from setting up basic data pipelines to designing complex neural network architectures. This report details my journey through Data Engineering, Natural Language Processing (NLP), and Sequence Modeling.

Introduction & Problem Statement

1.1 The Behavioral Finance Thesis

In standard financial theory, markets are often assumed to be rational. However, reality demonstrates that investors often act on emotion, leading to phenomena like the GameStop mania where prices decouple from business fundamentals due to "hype". My project is grounded in the philosophy that while the market is a weighing machine in the long run, it is a voting machine in the short run. By quantifying this "voting" process (sentiment), we can identify profitable divergences.

1.2 The Sequence Problem

A core technical challenge in this domain is that financial data violates the standard machine learning assumption that samples are Independent and Identically Distributed (I.I.D.). Stock prices are sequential; a price of $150 means something very different if yesterday was $100 (uptrend) versus $200 (crash). Therefore, simple regression models are insufficient, necessitating the use of sequence models that possess "memory."

2. Phase 1: Data Engineering & Market Microstructure

2.1 System Architecture

I have designed a pipeline that acts as an assembly line: global news (via NewsAPI) and stock prices (via yFinance) enter the system, get synchronized, and are processed into signals.

2.2 Handling Market Microstructure

To ensure the system is production-ready, I had to account for "friction" in trading. Real-world trading involves **Slippage**, where the execution price differs from the quoted price due to market volatility. In my data preparation, I am modeling a static slippage penalty of 0.05% on trades to avoid overestimating returns.

2.3 Challenge Solved: Timezone Alignment

One of the most critical engineering hurdles I resolved was the mismatch between global news cycles and Indian market hours.

- **The Problem:** The Indian market (NSE) operates from 09:15 to 15:30 IST. However, news occurs 24/7. A significant news event occurring at 8:00 PM IST cannot affect *today's* close; it affects *tomorrow's* open.
- **The Solution:** I implemented a `DataAligner` logic that shifts any post-market news to the next trading day.

2.4 Challenge Solved: Professional Data Storage

Initially, I used CSV files, but I found they were inefficient for incremental updates and lost data type information (e.g., timestamps turning into strings). I migrated the storage backend to **SQLite**, which allows for efficient querying and preserves data integrity.

3. Phase 2: Natural Language Processing (NLP)

3.1 Limitations of Static Embeddings

In Week 2, I learned that traditional models like Word2Vec fail in finance due to **Polysemy**—the capacity for a word to have multiple meanings.

- *Example:* The word "Bank" could mean a financial institution or a river edge. Static models average these meanings into a single vector, creating a "smeared" representation. To solve this, I adopted **BERT**, which generates dynamic embeddings where the representation of a token is a function of its context.

3.2 FinBERT & Domain Adaptation

While BERT is powerful, it is trained on general Wikipedia text. In finance, words like "Liability" or "Share" have specific, often neutral meanings that general models misclassify as negative or generic. I am utilizing **FinBERT**, which undergoes a rigorous 3-stage training pipeline:

1. **Stage 1:** General pre-training on Wikipedia (Google BERT).
2. **Stage 2 (Domain Adaptation):** Further pre-training on the **TRC2-Financial** dataset (46,000 financial articles).
3. **Stage 3 (Fine-Tuning):** Supervised learning on the Financial PhraseBank.

This domain adaptation allows the model to achieve ~97% accuracy on financial text, a massive improvement over the 80% baseline of standard BERT.

3.3 Challenge Solved: Named Entity Recognition (NER)

A major issue in automating news analysis is noise. Searching for "Amazon" often retrieves articles about the rainforest. To filter this, I implemented an NER pipeline using **spaCy**.

4. Phase 3: Sequence Modeling & Time Series

4.1 The Stationarity Problem

Neural networks struggle to model non-stationary data (data with trends) because the statistical properties (mean, variance) change over time Before training, I validated my data using the **Augmented Dickey-Fuller (ADF)** test.

- **Process:** I applied a Log Return transformation $R_t = \ln(Close_t / Close_{t-1})$ to convert the raw prices into a stationary series, ensuring the ADF p-value was < 0.05.

4.2 LSTM Architecture

To model the time series, I chose **Long Short-Term Memory (LSTM)** networks over standard RNNs. Standard RNNs suffer from the **Vanishing Gradient Problem**, where gradients decay exponentially as they propagate back through time, causing the model to forget early history.

The LSTM solves this by separating the **Cell State (C_t)** (long-term memory) from the **Hidden State (H_t)** (short-term output) using three specific gates:

1. **Input Gate (I_t):** Decides what new market information is relevant enough to store.
2. **Forget Gate (F_t):** Crucial for adapting to market regime changes, deciding what old history to discard.
3. **Output Gate (O_t):** Filters the cell state to produce the prediction for the next step.

Code Implementation:

I implemented the SentimentLSTM class in PyTorch, utilizing a sliding window of 60 days.

5. Strategy & Evaluation Metrics

5.1 Sentiment-Weighted Momentum Strategy

The final output of the system is not just a raw number but a trading decision. I have designed a **"Confirmation Filter"** strategy:

- **Logic:** We only trade when the technical prediction (LSTM) and the fundamental sentiment (FinBERT) agree.
- **Signal:**
  - **BUY:** LSTM predicts UP (>0.001) **AND** Sentiment is Positive (>0.2).
  - **SELL:** LSTM predicts DOWN (<-0.001) **AND** Sentiment is Negative (<-0.2).

    - **HOLD:** If signals diverge (e.g., Price Up but Sentiment Negative), we stay out to manage risk.

5.2 Performance Metrics

To evaluate the success of this strategy, I will not rely solely on accuracy. I will compute:

- **Sharpe Ratio:** To measure the excess return per unit of volatility.
- **Calmar Ratio:** To measure the return relative to the Maximum Drawdown (MDD), ensuring the strategy survives market crashes.

---

6. Conclusion & Next Steps

I have successfully built the "Time Series Engine" and the "NLP Engine." The data pipeline is robust, handling timezone mismatches and noise filtering via NER. The LSTM model has been implemented and is ready for training.

Next Steps:

1. Train the LSTM model on 1 year of historical data for AAPL and MSFT.
2. Run the backtesting engine to calculate the Sharpe and Calmar ratios.
3. Visualize the predictions against actual prices to check for "lag" (a common failure mode where the model simply predicts $P_t \approx P_{t-1}$).