Submitted by: Krishna Bansal

Market Mood & Moves

A Multimodal Deep Learning Approach to Sentiment-Driven Stock Prediction

Project Tenure Report: WiDS 5.0

Mentors: Meet & Sarthak

Abstract

This report documents the development of a sentiment-aware predictive engine for Apple Inc. (AAPL). The central thesis of this work challenges the traditional Efficient Market Hypothesis (EMH) by positing that market inefficiencies are frequently driven by latent human sentiment and psychological biases. By fusing high-frequency news sentiment—quantified via the domain-specific **FinBERT** transformer—with quantitative price and momentum indicators, we develop a multimodal forecasting framework. The core of this system is an **Attention-augmented Long Short-Term Memory (LSTM)** network designed to capture non-linear temporal dependencies across 50-day sequence windows. This document provides a rigorous academic walkthrough of the data engineering pipeline, the transition from Recurrent Neural Networks to gated memory cells, and the final implementation of a filtered trading strategy. We analyze the efficacy of domain-adapted embeddings and the significance of temporal attention in reducing forecast error during high-volatility market regimes.

Contents

1 Python for Quantitative Finance: Pandas & NumPy

The foundation of high-stakes financial modeling is a robust data infrastructure. Financial time-series are unique in their irregularity and high noise-to-signal ratio. Mastering the PyData stack was the critical first step in building a system capable of handling thousands of price points and news headlines without computational lag.

1.1 Numerical Efficiency and Vectorization

Standard Python 'for' loops are avoided in finance due to the Global Interpreter Lock (GIL). By using NumPy *ndarrays*, I implemented vectorized log-return calculations. Log returns are statistically superior to simple returns because they are time-additive and symmetric:

$$r_t = \ln(P_t) - \ln(P_{t-1})$$

This transformation ensures that a 10% increase followed by a 10% decrease results in a return that accurately reflects the price movement, a property critical for stabilizing neural network gradients. Furthermore, I leveraged NumPy's broadcasting to compute rolling standard deviations and Z-scores across years of historical data, establishing a baseline for market volatility that adjusts to changing market regimes.

1.2 Pandas: Time-Series Integrity and Alignment

Using Pandas, I addressed the "Alignment Problem" where news sentiment (unstructured) and price (structured) arrive at different frequencies. I developed a `DataAligner` function that maps sentiment to the correct trading session. Using `pd.tseries.offsets.BusinessDay`, I ensured that weekend news was programmatically shifted to Monday's opening bell, eliminating "Look-ahead Bias."

1.3 Handling the Timezone Challenge

A specific implementation challenge involved the conversion of news timestamps (often in UTC) to Indian Standard Time (IST) while mapping them to the US Market hours (EST). I utilized `tz_convert` to align these disparate data sources, ensuring that a news event breaking after the 15:30 IST close was correctly categorized as a signal for the *following* day's open. This rigorous causal mapping prevents the model from "cheating" by using future information to predict past prices.

2 Introduction & Behavioral Finance Thesis

2.1 Beyond the Efficient Market Hypothesis

The **Efficient Market Hypothesis (EMH)** assumes that prices reflect all available information instantaneously. However, this project follows the **Behavioral Finance** paradigm, which argues that market participants are prone to cognitive biases. We treat the market as a "voting machine" in the short run, where sentiment drives localized price dislocations before fundamentals catch up.

2.2 The Role of Noise Traders and Herding Behavior

In this report, we explicitly model the impact of "Noise Traders"—actors who trade on sentiment rather than intrinsic value. Sentiment-driven volatility often creates a self-fulfilling prophecy; high negative news leads to panic selling, which triggers technical indicators (like RSI), fueling further sell-offs. By quantifying this cycle using FinBERT, we move from observing "what the price is" to understanding "why the crowd is moving."

2.3 Information Asymmetry and Multimodal Signals

This psychological overlay allows for the identification of overreactions, where market sentiment diverges too far from the price action, creating mean-reversion opportunities. We posit that news sentiment acts as a "leading indicator" that precedes liquidity shocks. By capturing the "fear" or "greed" index through NLP, the model anticipates momentum shifts that purely price-based models often miss.

2.4 System Architecture Overview

The system architecture follows a hierarchical flow:

1. **Data Acquisition:** Automated scraping of business headlines via NewsAPI.
2. **Semantic Transformation:** Mapping text to a continuous $[-1, 1]$ sentiment space.
3. **Temporal Aggregation:** Consolidating 50 days of hybrid features into a single input tensor for the sequence engine.

3 NLP Foundations & Preprocessing Pipeline

Raw financial text is notoriously messy, filled with ticker symbols, HTML artifacts, and specialized jargon. I developed a cleaning pipeline that transforms unstructured strings into "Clean Semantic Tensors."

3.1 Linguistic Normalization and Tokenization

- **WordPiece Tokenization:** I transitioned to WordPiece tokenization (used by BERT). This handles complex jargon like "over-leveraged" by breaking words into ["over", "##leveraged"], preventing "Out-of-Vocabulary" (OOV) errors.
- **Stop-Word Filtering:** Using NLTK, I filtered 179 common English words that add zero sentiment value, focusing the model's attention on semantic "impact" words like "slashed," "surged," or "missed."
- **Lemmatization:** I used **WordNet Lemmatizer** to reduce words like "rallying" and "rallied" to "rally." Unlike stemming, lemmatization preserves the semantic base, ensuring consistent weights across different sentence structures.

3.2 The Lexicon Baseline: VADER

Initial benchmarks were established using **VADER**. While useful for social media, its lack of financial context was evident. For example, VADER might view "debt" as negative, but in a corporate context, "debt reduction" is a strong positive. This context-dependency necessitated the move to deep contextual embeddings.

3.3 Named Entity Recognition (NER) Filtering

A "Pro Challenge" I tackled was using spaCy to implement **NER filtering**. Financial news often mentions multiple companies (e.g., "Apple beats Samsung in sales"). A generic sentiment score would be diluted; by filtering headlines for the specific "ORG" entity (AAPL), I ensured that the sentiment fed into the model was strictly relevant to our target asset, drastically improving the signal-to-noise ratio in the training data.

4 Deep Dive into BERT & FinBERT

4.1 The Transformer Revolution

The core innovation of our sentiment engine is the **Transformer Encoder**. Unlike previous models that processed words sequentially, Transformers utilize **Self-Attention** to calculate the relevance of every word in a headline relative to every other word simultaneously.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

4.2 GELU: The Modern Activation Function

In our FinBERT implementation, we moved away from ReLU to **GELU (Gaussian Error Linear Unit)**. GELU weights inputs by their percentile, providing a smoother gradient flow for the small, nuanced changes in financial sentiment probabilities.

4.3 Domain Adaptation and Training Stages

FinBERT undergoes a 3-stage training pipeline:

1. **General Pre-training:** BERT learns English grammar from Wikipedia.
2. **Domain Adaptation:** The model is exposed to the **TRC2-financial** corpus (29M words), learning that "bullish" is a market term, not an animal description.
3. **Fine-tuning:** Trained on the **Financial PhraseBank** to classify specifically into "Positive," "Negative," and "Neutral" categories.

4.4 Chunking for Long Documents

BERT's self-attention has an $O(N^2)$ complexity, limiting input length to 512 tokens. For long-form earnings reports, I implemented a sliding-window chunking strategy with a 10% overlap. This ensures that sentiment "spills" across windows are captured, and the final sentiment is calculated via a **Softmax Mean** of all chunks.

5 Sequence Modeling: The RNN Engine

Stock prices are not independent data points; they are a sequence where the state at time t is conditioned on the history $(x_1, \ldots, x_{t-1})$. Using the **Probability Chain Rule**, we model the market as:

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_1, \ldots, x_{t-1})$$

5.1 Recurrent Neural Networks (RNNs)

Traditional Neural Networks lack a temporal dimension. I implemented a **Recurrent Neural Network** where the model maintains a **Hidden State (h_t)**—a latent representation of the past history.

$$h_t = \phi(W_{xh} x_t + W_{hh} h_{t-1} + b_h)$$

By feeding h_{t-1} back into the calculation for h_t, the RNN creates a "recurrent" loop that theoretically allows it to remember the entire history.

5.2 The Vanishing Gradient Problem

During training, I observed the **Vanishing Gradient** crisis. When backpropagating through time (BPTT) over a 50-day window, the gradients are multiplied repeatedly by the weight matrix W. If $W < 1$, the gradient shrinks exponentially, causing the model to "forget" what happened at the start of the month.

5.3 Exploding Gradients and Clipping

To mitigate sudden spikes in gradient values during volatile market regimes, I implemented **Gradient Clipping**. By scaling the gradients whenever their norm exceeded a threshold of 1.0, I prevented the "exploding gradient" problem, which often causes weights to become NaNs during training on high-volatility datasets like AAPL during earnings season.

6 Advanced Memory: The LSTM Architecture

To resolve the gradient issues of RNNs, I transitioned the project to the **Long Short-Term Memory (LSTM)** network. The key innovation is the **Cell State (C_t)**, which acts as a "long-term memory conveyor belt" protected by three regulatory gates.

6.1 The Gating Logic Derivations

I implemented the following gate logic within my `SentimentLSTM` class to control information flow:

- **Forget Gate (f_t):** Decides what historical info is irrelevant noise.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- **Input Gate (i_t):** Decides which new sentiment signals to store in memory.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

- **Output Gate (o_t):** Filters the long-term memory to produce the hidden state h_t.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

6.2 The "Constant Error Carrousel"

The beauty of the LSTM is the linear flow of the cell state. Because the gradient can flow through C_t without being repeatedly multiplied by a weight matrix (unless the forget gate is closed), the "Constant Error Carrousel" ensures that the signal from a major sentiment event 50 days ago reaches the final output with its semantic integrity intact. This allows the model to identify "long-memory" patterns, such as a multi-month accumulation phase following a positive news catalyst.

7 Attention Mechanism: Learning What Matters

7.1 The "Recency Bias" Problem

Even with LSTMs, the model naturally prioritizes the most recent days in a sequence. However, in trading, an earnings call 30 days ago is often more predictive than a quiet trading day yesterday. To solve this, I implemented a custom **Temporal Attention Layer**.

7.2 Implementing Temporal Attention

This layer calculates an **Alignment Score** (e_i) for every day in the sequence.

$$\alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^{50} \exp(e_j)} \tag{1}$$

The final "Context Vector" is a weighted sum of all 50 days of hidden states. If Day 15 had a massive spike in negative sentiment, the Attention mechanism "shines a spotlight" on that day.

7.3 Interpretable AI: Visualizing Weights

By extracting the attention weights, I created "Attention Heatmaps." These maps revealed that during periods of high price volatility, the model's focus shifts significantly toward news sentiment features, while during "flat" markets, it prioritizes technical indicators like RSI. This interpretability allows us to understand the "conviction" behind a model's prediction, distinguishing between a move driven by news and a move driven by technical momentum.

7.4 Softmax vs. Sparsemax

I experimented with **Sparsemax** as an alternative to Softmax for attention. While Softmax assigns at least some weight to every day, Sparsemax can assign exactly zero weight to irrelevant days, effectively pruning "noise days" from the model's consideration and focusing solely on high-impact catalysts.

8 Experimental Results & Model Evaluation

8.1 Training Setup and Hyperparameters

The model was trained for 100 epochs using the **Adam Optimizer**. I utilized a **Hybrid Scaling** strategy:

- **Standardization (Z-score):** Applied to Returns and Sentiment to center them at zero.
- **Min-Max Scaling:** Applied to technical indicators (RSI, MACD) to keep them in $[0, 1]$.

8.2 Stationarity Testing: The ADF Filter

Before training, I implemented the **Augmented Dickey-Fuller (ADF)** test. Raw stock prices are "non-stationary" (random walks), which leads to spurious regressions. By converting prices to Log-Returns and performing the ADF test, I achieved a p-value < 0.01, ensuring the data was stationary and suitable for gradient-based learning.

8.3 Directional Accuracy and Metrics

Result: The model achieved a **58.4% Directional Accuracy** on the AAPL test set. In the context of equity markets, any hit rate above 55% is considered a significant statistical edge.

Model Variant	Accuracy	RMSE	F1-Score
Baseline (LSTM only)	52.1%	0.045	0.51
Multimodal (FinBERT + LSTM)	56.2%	0.038	0.55
Final (Attention + Multimodal)	**58.4%**	**0.031**	**0.59**

Table 1: Comparative performance metrics across architectural iterations.

9 Trading Strategy & Conclusion

9.1 The Decision Engine: Signal Agreement

The final layer is the translation of probabilities into a **Buy/Sell Decision Engine**. I implemented a logic that requires "Agreement" between the modalities:

```
def get_trade_signal(lstm_pred, sentiment_score):
    if lstm_pred > 0.001 and sentiment_score > 0.2:
        return 'BUY'
    elif lstm_pred < -0.001 and sentiment_score < -0.2:
        return 'SELL'
    else:
        return 'HOLD'
```

This dual-confirmation strategy reduces "False Positives" by ensuring that technical momentum is supported by fundamental news sentiment.

9.2 Execution Workflow and Risk Management

A production-ready workflow was designed to minimize execution risk:

- **Pre-Market Scrape:** Analyze news from the last 24h to set the sentiment bias.
- **Model Inference:** Feed the sequence window to get the $T + 1$ probability.
- **Risk Controls:** Implementation of a -2.0% hard stop-loss to protect against extreme tail-risk events.

9.3 Conclusion & Future Scope

This 12-week tenure provided a deep dive into the fusion of AI and Finance. We demonstrated that market "Mood" is a quantifiable leading indicator. **Future Directions:**

- **Reinforcement Learning:** Transitioning to PPO (Proximal Policy Optimization) for dynamic position sizing.
- **Real-Time Integration:** Migrating from static CSVs to live Alpaca Trade streams for paper trading.
- **Cross-Asset Analysis:** Analyzing how sentiment in correlated sectors (e.g., Semiconductors) influences AAPL price dynamics.