# What's the Vibe: Can Financial News Predict the Stock Market? - A ML Case Study

Mihaela Goga

November 12, 2025

## Contents

## 1 Background

Predicting the direction of major stock market indices, such as the S&P 500, is a notoriously difficult task. Traditional quantitative methods, which rely on historical price data (technical analysis) or firm fundamentals (fundamental analysis), often fail to capture the bigger picture of market movements.[1] Behind market movement, there is a decisive component: human psychology that encompasses a range of emotions, including the collective sentiment, fear, and optimism of investors. More often than not, financial news largely influences this.

This has lead to a new major area of research that focuses on improving market predictions by using two kinds of data derived from news text. These hybrid models fuse quantitative data (numbers) with human sentiment gathered from qualitative data (news articles).[1] Studies findings show that incorporating sentiment analyses using transformers models like BERT and FinBERT can make forecasting models more accurate.[2] This research has explored various architectures, from combining FinBERT with LSTMs to hybrid approaches that add sentiment to GRU-based models.[1]

This project is a contribution to this sub-field and my primary hypothesis is that *topic-specific* sentiment is more useful than a broad and generic sentiment score for the whole market. As a relevant example, a negative sentiment score specifically for 'inflation' (articles mentioning 'cpi') [1] provides more information than a general 'negative' score from an article on a single company's earnings. My idea is based on Aspect-Based Sentiment Analysis (ABSA), a very intersting area of research which has been shown to improve prediction by differentiating between what people are discussing and how they feel about each specific topic. [3]

Therefore, this report explains the design, implementation, and evaluation of a new, multi-modal predictive framework.

## 2 Data and Resources

The project integrated two distinct categories of data to build the final prediction model. The data acquisition process included a data engineering pipeline for raw financial news articles, and a collection of historical time-series data for quantitative modeling.

### 2.1 Raw News Article Data

The primary dataset consists of financial news articles fetched in real-time. To achieve this a python script was written, packaged as a Docker container to fetch and process the data. The process was automated as a serverless CI/CD pipeline using GCP services, mainly Cloud Build (container creation from Git tags), Artifact registry (container storage) and Cloud Run jobs (execution of the script). A Cloud Scheduler function was also integrated to fetch the news articles once at a 24 hours time frame. Instead of one generic search, I decided to create a queries dictionary to gather articles which include four distinctive categories keywords: broad market concepts, major indexes, key financial companies and tech giants. I restricted my queries to high-quality domains like `bloomberg.com`, `wsj.com`, `reuters.com`, `cnbc.com`, `businessinsider.com`.

However, I discovered that the API only provided teasers, not the full article. My Cloud Function failed to scrape the full content from these sites. The main reasons is tied to anti-scraping policies and paywalls. I searched for other alternatives (specialized APIs and data sources) but soon i had to abandon the idea because it was too expensive. My original plan was to gather at least two to three years worth of historical data but had to restrict it to two months. I found a manual work around by copying and pasting the full content into Firestore myself.

I obtained a free API key from `NewsAPI.org` and created a custom serializer to handle the data time objects and a function to download all the articles collection from Firestore Database (saved in `articles_export.json`). The file contains the source name and link, publication timestamps, full article content and id. The processed articles are stored in the Firestore Database as a query dataset. The date of the collected articles was set between 20-08-2025 and 29-10-2025. Before moving to the next step I implemented a N-Gram analysis on the collected articles to find the most discussed topics. This was done by preprocessing all article text, counting the frequency of all 1-grams, 2-grams, and 3-grams, and selecting them based on number of occurences and relevance.

## 2.2 Quantitative Time-Series Data

The second data pipeline collected structured, numerical data for the Aspect Based Sentiment analyses and S&P 500 prediction model. This tabular data was loaded into Google BigQuery. The script leverages three key sources:

- **Yahoo Finance:** Collected market data such as the daily S&P 500 prices (`ĜSPC`) main prediction target, and the CBOE Volatility Index (`V̂IX`), the key predictive feature, ”fear index” measuring the market volatility. It has an inverse relationship with S&P 500 making it great for understanding the investors sentiment.
- **FRED (Federal Reserve Economic Data):** Represents the macroeconomic indicators, such as the Federal Funds interest rate (`FEDFUNDS`), great for understanding the economic environment and 10-year treasury yields (`DGS10`), a key indicator of long-term investor confidence.
- **Google Trends:** This source was used to capture public sentiment by tracking search interest for seven keywords, including ”inflation,” ”recession,” and ”unemployment.”

### Data Processing and Normalization

Data from these sources required pre-processing. A custom upload function was developed to prevent data duplication on subsequent runs. A `DELETE/INSERT` pattern was used to mitigate that.

Processing the search interest keywords had been a challenge as well because Google Trends API limits comparisons to a maximum of five keywords at a time. To solve this, an anchor method was implemented. The seven keywords were split into two batches and ”recession” was used as an anchor term in both. Data was fetched for both batches and a scaling ratio was performed to get the median score. This ratio helped to scale data for the second batch,making the scores comparable to the first. Data was combined into a single dataset.

# 3 Methods

The project’s methodology is structured as a sequential pipeline where raw data is transformed in predictions. The process involved the following three main steps: training a domain-specific sentiment classifier, applying this classifier to generate a daily topic-sentiment score, and using the score as a feature in the final S&P 500 prediction model.

## 3.1 Model 1: Financial Sentiment Classifier

To build a sentiment classifier model I needed to make a custom one because the pre-trained models lack the specific nuance of financial terms (e.g. ”bearish” or ”volatile” have their own specific meanings).

### Model Training

The model is a **transfer learning** implementation based on the DistilBERT architecture. The model was fine-tuned on the *FinanceMTEB/financial_phrasebank* dataset, a collection of financial sentences hand-labeled as positive, negative, or neutral.

One of the main challenges in this dataset was the significant class imbalance (seems to be the biggest problems in a lot of tasks these days) with a large majority of neutral sentences. To solve this, I used **class weights** which forced the model to pay more attention to the rare ”positive” and ”negative” examples. A custom PyTorch model class was implemented to load the pre-trained *distilbert-base-uncased* model. The core layers were frozen and I added a new, unfrozen classification head and the calculated class weights were passed directly into the *CrossEntropyLoss* function. The *TrainingArguments* were configured to optimize for the **f1_macro** score, as I considered this to be a far more reliable metric than accuracy.

## 3.2 Model 1 Application: Topic-Based Sentiment (simplified ABSA)

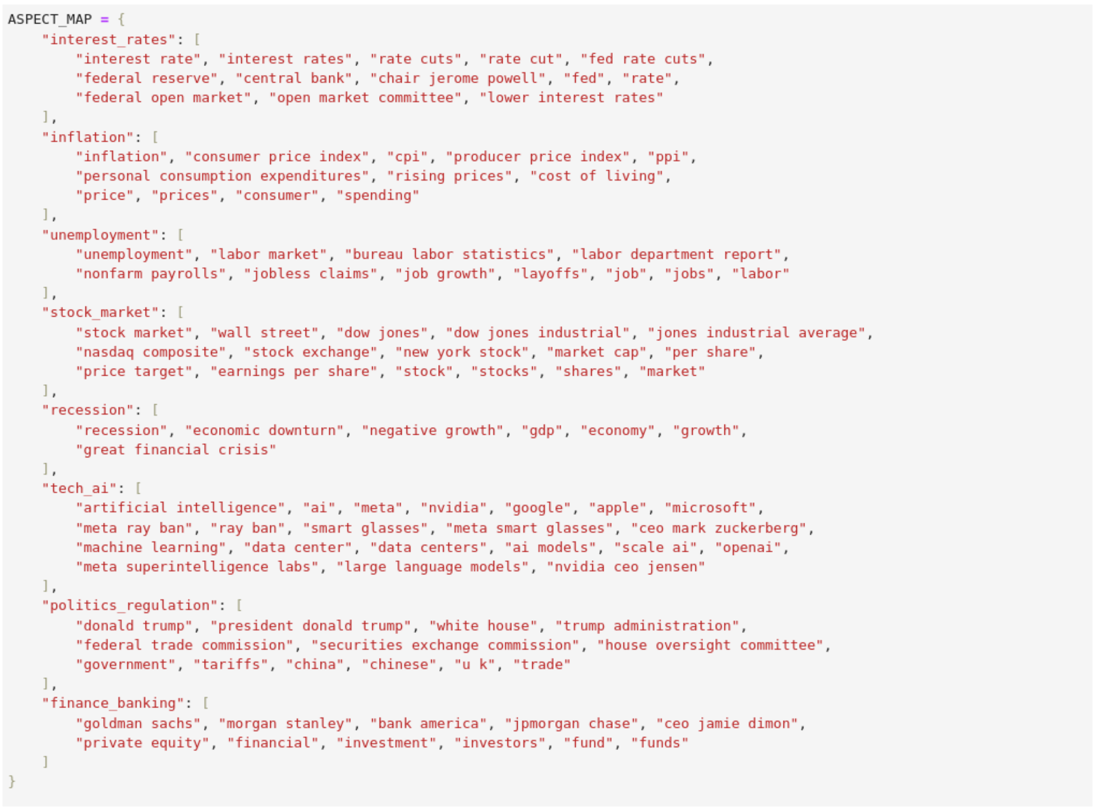
Once the custom classifier was trained, it was time to analyze the collected news articles. Since the scope of this project was limited (time-constraints and complexity) a true Aspect-Based Sentiment Analysis (ABSA) was not possible. Therefore, I implemented a middle-ground method that was developed to generate topic-specific sentiment features. This process involved three steps described below:

1. **Article Scoring:** A function was created to load the trained model. For each of the 379 articles, this function processed the *entire* article content and generated a single, sentiment score. This score was calculated by applying a `softmax` function to the model’s output logits followed by computing.

   $$Score = P(\text{positive}) - P(\text{negative})$$

   This result was a float between -1.0 and 1.0.

2. **Keyword Mapping:** A Python dictionary named `ASPECT_MAP` was created. This map assigns financial topics (e.g., `inflation`, `tech_ai`) to a diverse and curated list of relevant keywords.

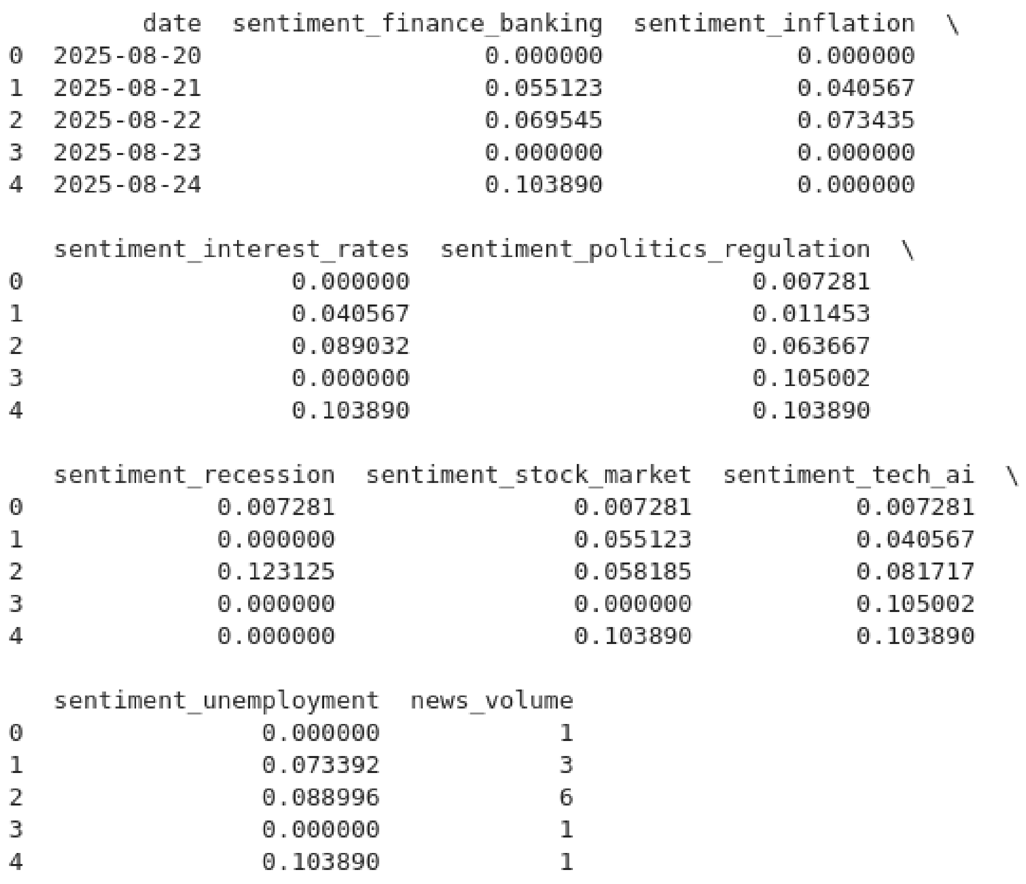


```python
ASPECT_MAP = {
    "interest_rates": [
        "interest rate", "interest rates", "rate cuts", "rate cut", "fed rate cuts",
        "federal reserve", "central bank", "chair jerome powell", "fed", "rate",
        "federal open market", "open market committee", "lower interest rates"
    ],
    "inflation": [
        "inflation", "consumer price index", "cpi", "producer price index", "ppi",
        "personal consumption expenditures", "rising prices", "cost of living",
        "price", "prices", "consumer", "spending"
    ],
    "unemployment": [
        "unemployment", "labor market", "bureau labor statistics", "labor department report",
        "nonfarm payrolls", "jobless claims", "job growth", "layoffs", "job", "jobs", "labor"
    ],
    "stock_market": [
        "stock market", "wall street", "dow jones", "dow jones industrial", "jones industrial average",
        "nasdaq composite", "stock exchange", "new york stock", "market cap", "per share",
        "price target", "earnings per share", "stock", "stocks", "shares", "market"
    ],
    "recession": [
        "recession", "economic downturn", "negative growth", "gdp", "economy", "growth",
        "great financial crisis"
    ],
    "tech_ai": [
        "artificial intelligence", "ai", "meta", "nvidia", "google", "apple", "microsoft",
        "meta ray ban", "ray ban", "smart glasses", "meta smart glasses", "ceo mark zuckerberg",
        "machine learning", "data center", "data centers", "ai models", "scale ai", "openai",
        "meta superintelligence labs", "large language models", "nvidia ceo jensen"
    ],
    "politics_regulation": [
        "donald trump", "president donald trump", "white house", "trump administration",
        "federal trade commission", "securities exchange commission", "house oversight committee",
        "government", "tariffs", "china", "chinese", "u k", "trade"
    ],
    "finance_banking": [
        "goldman sachs", "morgan stanley", "bank america", "jpmorgan chase", "ceo jamie dimon",
        "private equity", "financial", "investment", "investors", "fund", "funds"
    ]
}
```

Figure 1: The Python dictionary structure of the `ASPECT_MAP`.

3. **Assignment and Aggregation:** The pipeline iterated through every article. For each article, a single score (e.g. -0.75) was retrieved. The given article's text was checked for keywords from the `ASPECT_MAP`. If the article contained keywords for ”inflation,” the -0.75 score was then assigned to the ”inflation” topic for that specific day. This process was repeated for each topic found in the map.

Finally, all scores for each topic were averanged by category and day (see Figure 2) and the final feature table was saved to BigQuery.



```
         date  sentiment_finance_banking  sentiment_inflation  \
0  2025-08-20                   0.000000             0.000000   
1  2025-08-21                   0.055123             0.040567   
2  2025-08-22                   0.069545             0.073435   
3  2025-08-23                   0.000000             0.000000   
4  2025-08-24                   0.103890             0.000000   

   sentiment_interest_rates  sentiment_politics_regulation  \
0                  0.000000                       0.007281   
1                  0.040567                       0.011453   
2                  0.089032                       0.063667   
3                  0.000000                       0.105002   
4                  0.103890                       0.103890   

   sentiment_recession  sentiment_stock_market  sentiment_tech_ai  \
0             0.007281                0.007281           0.007281   
1             0.000000                0.055123           0.040567   
2             0.123125                0.058185           0.081717   
3             0.000000                0.000000           0.105002   
4             0.000000                0.103890           0.103890   

   sentiment_unemployment  news_volume  
0                0.000000            1  
1                0.073392            3  
2                0.088996            6  
3                0.000000            1  
4                0.103890            1  
```

Figure 2: Example of the final aggregated daily sentiment scores by topic.

## 3.3 Model 2: S&P 500 Prediction

The last step was to train and evaluate the final model in order to predict the next-day direction of the S&P 500 using all the prepared features.

### Data Preprocessing and Feature Engineering

A master dataset was created by merging all sources from BigQuery: market data, economic indicators, Google Trends data, and the aggregated daily sentiment scores from Model 1. The two steps are described below:

**1. Target Variable Engineering:** I created a binary target variable. The main objective was not to predict the exact price of the S&P 500, which would be very hard but its **direction**. A new binary column was created.

- A `reference_price` was set for each day using a forward-fill. It included the last valid S&P 500 closing price. Thus, Friday's close was applied to Saturday and Sunday.
- A `target_price` was set by backward-filling, which assigned the next valid closing price (e.g. Monday's close) to the preceding days (e.g. Friday, Saturday, and Sunday).
- The target was assigned **1** (Buy) if the *target_price* was greater than or equal to the *reference_price*, and **0** (Sell) if its lower.

**2. Feature and Data Splitting:** To prevent data leakage,the dataset was split chronologically by calendar week rather than randomly. The split ratios were set to **80% for**

**training, 15% for testing, and 5% for validation**. Following the split, the features were processed using a *ColumnTransformer* pipeline. This pipeline applied *StandardScaler* to all numerical data and used *OneHotEncoder* to convert the categorical feature (*day_of_week*) into a numerical format.

### Model Comparison

I compared and evaluated three different models to find the optimal one.

- **Logistic Regression:** A simple baseline model implemented in Scikit-learn pipeline. To manage the imbalanced data, the `class_weight='balanced'` parameter was used.
- **XGBoost:** A powerful tree-based model. Same as the Logistic Regression model a `scale_pos_weight` parameter was used to handle imbalanced data, calculated as the ratio negative-to-positive samples found in the training set.
- **GRU (Gated Recurrent Unit):** A Recurrent Neural Network (RNN) very similar to an LSTM but it has fewer parameter. If i had more data i would have probably implemented an LSTM but since we are working with very little data GRU works better in this case.It was built in PyTorch. Before traing,the 2D preprocessed data was converted into 3D sequences with a lookback period of 5 days. The model was trained using an Adam optimizer and `nn.BCEWithLogitsLoss` loss function configured with a positive weight to handle class imbalance. To prevent overfitting, I implemented an early stopping at 5 epochs. The model with the best validation loss was saved.

The models were trained on the 29-sample `train` set. The `test` set was the development set for performance evaluation and represented the early stopping for the GRU. The final performance was measured using the true, held-out `validation` set.

## 4 Results

The project's results are evaluated in two parts: the output of the topic-sentiment pipeline (Model 1) and the performance of the predictive classifiers (Model 2).

### 4.1 Model 1: Topic-Based Sentiment Results

The implementation of the simplified ABSA method successfully generated a daily time-series of sentiment scores for each of the pre-defined topics. As shown in Figure 1, the sentiment for all topics fluctuated around a near-neutral baseline.

Overall, the average sentiment across all topics was slightly positive (0.025). The analysis identified **TECH_AI** as the "Most Positive Topic" (Avg: 0.034), indicating a generally optimistic tone in news articles mentioning AI and major tech companies. Conversely, **UNEMPLOYMENT** was identified as the "Most Volatile Topic" (Std. Dev: 0.052), suggesting significant disagreement or rapid changes in tone in articles related to the labor market. These daily topic scores served as the primary qualitative features for the predictive model.

## 4.2 Model 2: Predictive Model Performance

The three predictive models were trained on the training set and evaluated on the held-out test set. The performance metrics are summarized in Table 1 and are also displayed in the confusion matrices in Figure 3.

Table 1: Model Performance Metrics on Test Set (Buy/Sell Classification)

| Model | Accuracy (%) | Precision (Buy) | Recall (Buy) | F1-Score (Buy) |
|---|---|---|---|---|
| Logistic Regression | 82.14 | 0.857 | 0.857 | 0.857 |
| XGBoost | 80.36 | 0.875 | 0.800 | 0.836 |
| GRU Network | 33.33 | 0.500 | 0.147 | 0.227 |

### Analysis of Results

At first glance, the Logistic Regression and XGBoost models appear highly successful, with accuracies of 82.14% and 80.36%, respectively. However, given the extremely small size of the test set (n=56), these high accuracy figures are not reliable and are **a strong indicator of overfitting**. The models have likely memorized the noise and specific patterns present in the small training set, and their performance would not generalize to new, unseen data.

A more critical analysis comes from the confusion matrices. The GRU model's performance was notably poor, with an accuracy of 33.33%, which is worse than a random guess. The matrix for the GRU reveals a major failure in **Recall (0.147)** for the "Buy" class. It correctly identified only 5 "Buy" signals while misclassifying 29 of them as "Sell". This indicates the model almost completely failed to learn the "Buy" pattern. This failure is almost certainly due to the dataset being far too small. Deep learning models like GRUs require vast amounts of data to learn complex time patterns, and the 5-day lookback on a 29-sample training set provided insufficient data for meaningful learning.

Ultimately, while the Logistic Regression and XGBoost models appear to function on this dataset, the results are likely just a coincidence and a consequence of overfitting. The GRU model's failure confirms that the dataset, in its current form, is insufficient for complex time-series modeling.

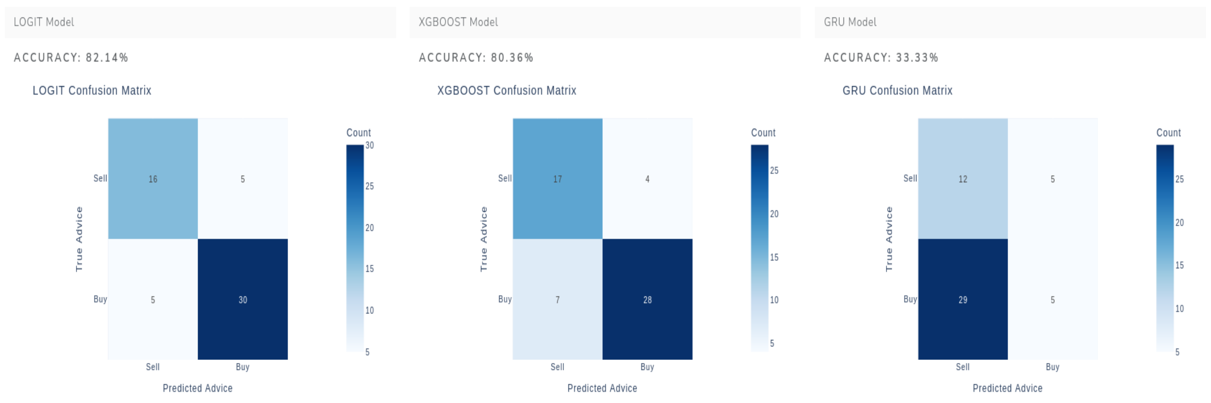
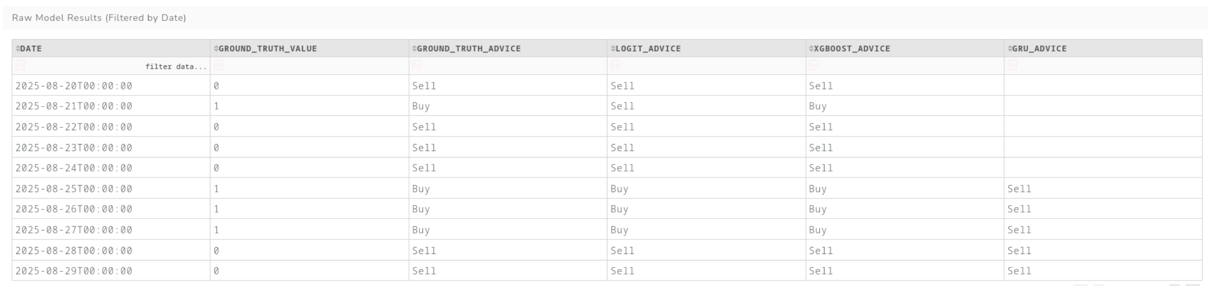


Figure 3: Confusion Matrices for (a) Logistic Regression, (b) XGBoost, and (c) GRU models.


Raw Model Results (Filtered by Date)

| DATE | GROUND_TRUTH_VALUE | GROUND_TRUTH_ADVICE | LOGIT_ADVICE | XGBOOST_ADVICE | GRU_ADVICE |
|---|---|---|---|---|---|
| filter data... | | | | | |
| 2025-08-20T00:00:00 | 0 | Sell | Sell | Sell | |
| 2025-08-21T00:00:00 | 1 | Buy | Sell | Buy | |
| 2025-08-22T00:00:00 | 0 | Sell | Sell | Sell | |
| 2025-08-23T00:00:00 | 0 | Sell | Sell | Sell | |
| 2025-08-24T00:00:00 | 0 | Sell | Sell | Sell | |
| 2025-08-25T00:00:00 | 1 | Buy | Buy | Buy | Sell |
| 2025-08-26T00:00:00 | 1 | Buy | Buy | Buy | Sell |
| 2025-08-27T00:00:00 | 1 | Buy | Buy | Buy | Sell |
| 2025-08-28T00:00:00 | 0 | Sell | Sell | Sell | Sell |
| 2025-08-29T00:00:00 | 0 | Sell | Sell | Sell | Sell |

Figure 4: Raw model results showing predictions vs. ground truth.

## 5 Analysis and Discussion

The results of this project highlight a critical disconnect between a sound architecture and the practical limitations of data acquisition. The high-accuracy scores of the Logistic Regression and XGBoost models (82.14% and 80.36%, respectively) are the strongest evidence of the project's primary limitation: severe data scarcity. A model trained on only 29 samples and tested on 56 is not learning generalizable market patterns. In practice, such a model is overfitting to the noise specific to that short time-period. The failure of the GRU model (33.33% accuracy) confirms this conclusion.

The project's core hypothesis, that topic-specific sentiment is a superior predictor to generic sentiment remains unproven. While the background research supports this approach [3], the model's overfitting makes it impossible to determine if it learned a genuine correlation or simply memorized a random one.

## 6 Conclusion

This project successfully designed and implemented a complete, end-to-end, multi-modal pipeline for financial market prediction. It demonstrated the integration of a cloud-native data collection system (GCP, BigQuery, Firestore), the development of a domain-specific sentiment classifier via transfer learning, and the implementation of a ”simplified ABSA” method to generate topic-specific features.

The final predictive results, however, were inconclusive. The primary challenge, which became the project's defining limitation, was the inability to acquire a sufficiently large dataset of historical news articles due to API restrictions and paywalls. The resulting dataset was far too small for effective model training. This led to classical overfitting in the simpler models (Logistic Regression, XGBoost) and complete model failure in the complex model (GRU). Therefore, the project's core hypothesis that topic-specific sentiment features can improve stock market prediction remains untested.

The primary takeaway is that while the project's idea has potential and serves as a proof-of-concept, the findings are useless. Future work must focus exclusively on solving this data bottleneck.

## References

[1] M. Singh, H. Tunga, and S. Kar, *Sentivol-ga: A volatility-scaled genetic fusion of predictive models and financial sentiment for adaptive stock forecasting*, ResearchGate Publication: 396396986, Preprint, 2025.

[2] T. Jiang et al., “Financial sentiment analysis using finbert with application in predicting stock movement,” *arXiv preprint arXiv:2306.02136*, 2023. arXiv: `2306.02136`.

[3] K. N. Dang, K. C. Nguyen, L. V. Truong, K.-D. Cao-Phan, and T. M. Pham, *Aspect-based sentiment analysis for stock price movement prediction*, ResearchGate Publication: 397036378, Preprint, 2024.