# Automated Financial Sentiment Analysis

*From Unstructured Data to Alpha Generation: A Machine Learning Approach*

**Group Members:**
PERALDI Yasmine
SALIBA Rhea
SERRIER Pierre-Antoine

*Major: Financial Engineering, A4*
ESILV - École Supérieure d'Ingénieurs Léonard de Vinci

November 21, 2025

**Abstract**

**Abstract:** As future Financial Engineers operating in high-frequency markets, we are constantly seeking methods to quantify market information with minimal latency. This comprehensive report details the end-to-end development of a Natural Language Processing (NLP) pipeline designed to automatically classify financial news as Bearish, Bullish, or Neutral. By leveraging the *Twitter Financial News* dataset, we address critical challenges inherent to financial data, such as severe class imbalance and domain-specific jargon. We formalize the classification problem mathematically, compare classical algorithms (Logistic Regression, SVM) against Deep Learning architectures (Bidirectional GRU), and detail our strategies for overcoming overfitting and the "accuracy trap." The final optimized model, achieving an F1-Macro score of 0.74 and a Recall of 0.68 on the negative class, serves as a functional prototype for an automated trading signal generator. This work demonstrates that for specialized financial datasets, well-tuned linear models often outperform complex neural networks.

# Contents

# List of Figures



# List of Tables

# 1 Business Case and Objectives

## 1.1 The Problem: Information Latency in Financial Markets

In the domain of **Quantitative Finance** and **High-Frequency Trading (HFT)**, the speed of reaction to new information is the primary determinant of profitability, often referred to as "Alpha." The Efficient Market Hypothesis (EMH) posits that asset prices reflect all available information. However, in reality, there is a time lag between the publication of news and its integration into asset prices.

Traditional quantitative models rely heavily on structured data such as price, volume, open interest, and historical volatility. While these metrics are essential, they are lagging indicators. A significant portion of market-moving information exists as **unstructured text**:

- Quarterly earnings reports (10-Ks, 10-Qs).
- Regulatory announcements (SEC filings, FOMC minutes).
- Real-time social media sentiment (Twitter/X, Reddit).

Manual analysis of this data stream is impossible due to the sheer volume and velocity (velocity, volume, variety being the 3 Vs of Big Data). A human trader cannot process 10,000 tweets per minute during an earnings call, but a machine learning algorithm can.

## 1.2 Link to Specialization: Financial Engineering

As students in Financial Engineering, our objective is to minimize information asymmetry and build systems that can exploit transient market inefficiencies. This project is not merely a data science exercise; it is the foundational layer of an **Algorithmic Trading Strategy**.

Our goal is to build an **Automated Sentiment Classifier** that acts as a signal generator for a trading engine:

1. **Alpha Generation (Signal Creation):** Identify "Bullish" trends instantly to execute Long positions before the broader market reacts.
2. **Risk Management (Drawdown Control):** Identify "Bearish" news to trigger Stop-Loss mechanisms or to hedge a portfolio automatically using derivatives (e.g., buying Puts).

The "Business Case" is therefore to convert qualitative noise (*text*) into a quantitative signal ($y \in \{-1, 0, 1\}$) usable by an execution algorithm. The success of this project is

measured not just by accuracy, but by the model's ability to detect the minority classes (extreme sentiment) which drive volatility.

# 2 Dataset Description and Source

## 2.1 Source Selection

We utilized the **Twitter Financial News Sentiment Dataset**. This dataset was selected specifically because it represents the "hardest" type of financial text to analyze:

- **Brevity:** Tweets are short, requiring the model to infer context from very few words.
- **Noise:** Unlike official reports, tweets contain slang, misspellings, and emojis.
- **Speed:** Twitter is often the first source of breaking news, making it the most relevant source for HFT applications.

## 2.2 Dataset Structure

The dataset consists of two CSV files, `sent_train.csv` and `sent_valid.csv`. The data schema is straightforward:

- **Text ($X$):** The raw content of the message. Example: *"$AAPL puts volume heavy ahead of earnings."*
- **Label ($Y$):** The ground truth sentiment, encoded as integers:
  - **0 (Bearish):** Negative news, likely to cause a price drop.
  - **1 (Bullish):** Positive news, likely to cause a price increase.
  - **2 (Neutral):** Factual reporting without directional sentiment.

# 3 Data Exploration (EDA)

Before attempting to model the data, a rigorous statistical analysis was performed to understand the underlying distribution and potential biases.

## 3.1 Visualizing Class Distribution

A critical finding during our exploration was the severe **Class Imbalance** in the dataset. We visualized the count of each label using histograms and pie charts.

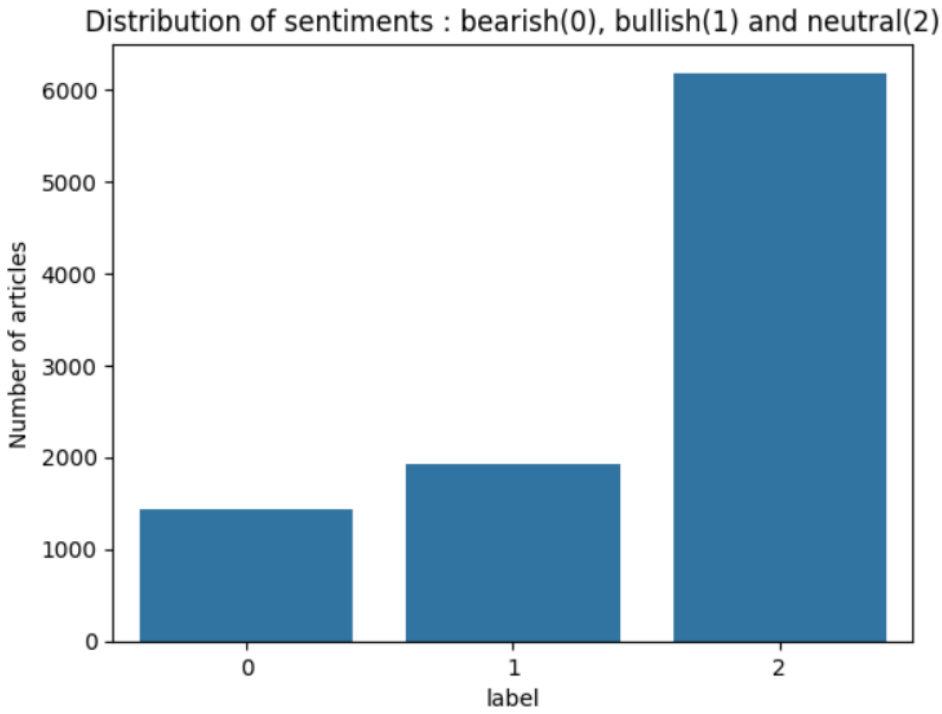


Figure 1: Distribution of Sentiment Classes in Training Data.

**Statistical Analysis:**

- **Neutral (Class 2):** Represents approximately 65% of the training data.
- **Positive (Class 1):** Represents approximately 20% of the training data.
- **Negative (Class 0):** Represents approximately 15% of the training data.

**Business Implication:** This imbalance is typical in finance—most news is noise (neutral). However, from a modeling perspective, it presents a specific obstacle. A naive model that predicts "Neutral" for every single input would achieve a baseline accuracy of 65%. This creates an "Accuracy Trap" where a high accuracy score masks a model that is completely useless for trading (zero recall on buy/sell signals).

## 3.2 Data Quality Check

We checked for missing values (NaNs) and duplicates.

```
# Check for missing values
missing = df['text'].isna().sum()
# Check for duplicates
dups = df.duplicated().sum()
```

The dataset was found to be relatively clean, with no missing values in the text column, allowing us to proceed directly to preprocessing.

# 4 Problem Formalization

From a machine learning perspective, we formalize this business problem as a **Supervised Multi-Class Classification** task.

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be our training dataset of size $N$, where:

- $x_i$ represents the raw text of the $i$-th document.
- $y_i \in \{0, 1, 2\}$ represents the target class label.

We aim to learn a function $f_\theta : \mathcal{X} \to \mathcal{Y}$, parameterized by $\theta$, that maps the input space to the target space. The optimal parameters $\hat{\theta}$ are found by minimizing a regularized loss function $\mathcal{L}$:

$$\hat{\theta} = \arg\min_{\theta} \left[ \sum_{i=1}^{N} \mathcal{L}(f_\theta(\phi(x_i)), y_i) + \lambda \Omega(\theta) \right]$$

Where:

- $\phi(x_i)$ is the feature extraction function (e.g., TF-IDF vectorization) converting text to $\mathbb{R}^d$.
- $\mathcal{L}$ is the loss function (e.g., Cross-Entropy Loss for neural networks or Log-Loss for Logistic Regression).
- $\Omega(\theta)$ is the regularization term (L1 or L2 norm) to prevent overfitting on high-dimensional sparse data.
- $\lambda$ is the hyperparameter controlling the regularization strength.

# 5 Methodology: Data Preprocessing

Raw text is unstructured and noisy. To convert it into a usable format, we implemented a robust preprocessing pipeline function named `clean_text_selected`.

## 5.1 Step 1: Noise Reduction

Social media data contains artifacts that carry no semantic meaning.

- **URLs:** Links like 'http://t.co/xyz' are unique identifiers. We removed them using Regex: `re.sub(r'http\S+', '', text)`.
- **Handles:** Usernames (e.g., '@elonmusk') were kept in some experiments but ultimately removed to focus on the message content rather than the author.

### 5.2 Step 2: The Ticker Removal Strategy (Critical)

One of the most strategic decisions in our pipeline was the handling of stock tickers (e.g., $AAPL, $TSLA, $EURUSD). **Hypothesis:** Keeping tickers introduces "Look-ahead Bias" and specific entity bias. **Scenario:** If the training data covers 2022 (a bad year for Tech stocks), the model might learn that the token "$TSLA" itself implies "Negative Sentiment", regardless of the context. **Solution:** We removed all tickers using the regex `r'$\w+'`. **Justification:** This forces the model to learn *linguistic patterns* (verbs like "plummet", "soar", adjectives like "weak", "strong") rather than memorizing specific asset names. This ensures the model generalizes well to new stocks or future time periods.

### 5.3 Step 3: Normalization and Stopwords

- **Lowercasing:** `text.lower()` ensures "Buy" and "buy" are treated as the same token.
- **Stopwords:** We used the NLTK (Natural Language Toolkit) library to remove common English words (the, is, at, which). These words appear in almost every document and have high frequency but low information content (low variance). Removing them reduces the dimensionality of our TF-IDF matrix, speeding up training.

## 6 Feature Engineering: TF-IDF and N-Grams

We transitioned from qualitative text to quantitative vectors using the TF-IDF transformation.

### 6.1 Mathematical Formulation of TF-IDF

TF-IDF stands for **Term Frequency-Inverse Document Frequency**. It is defined as:

$$w_{i,j} = tf_{i,j} \times idf_i$$

Where:

- $tf_{i,j}$ is the number of times term $i$ appears in document $j$.
- $idf_i = \log\left(\frac{N}{df_i}\right)$, where $N$ is the total number of documents and $df_i$ is the number of documents containing term $i$.

This weighting scheme is superior to simple word counts (CountVectorizer) because it penalizes words that are too common across the corpus (like "market" or "today") and

boosts rare, discriminative words that are specific to a sentiment.

### 6.2 The N-Gram Optimization

Our initial baseline used **Unigrams** (single words). However, financial sentiment is highly context-dependent, often relying on negation or compound terms.

- **Unigram Limitation:** "Not good" is seen as "not", "good". The model sees "good" and might predict Positive.
- **Bigram Solution:** "Not good" is tokenized as a single feature "not good".

We optimized the vectorizer with `ngram_range=(1, 2)`. **Result:** The feature space expanded significantly (from ∼10,000 to ∼40,000 dimensions), but the model gained the ability to capture phrases like "short squeeze", "bear market", "missed estimates", and "growth slowed". This step was crucial for improving the model's reasoning capability.

## 7 Presentation of Models

To solve the optimization problem defined in Section 4, we selected a diverse set of candidate models.

### 7.1 Baseline: Logistic Regression

**Theory:** A probabilistic linear classifier that estimates $P(Y = k|X)$ using the softmax function (for multi-class).

$$P(Y = k|X) = \frac{e^{\beta_k \cdot X}}{\sum_j e^{\beta_j \cdot X}}$$

**Why?** Logistic Regression is the industry standard for benchmarks due to its interpretability (we can examine coefficients to see which words drive sentiment) and efficiency on sparse data.

### 7.2 Support Vector Machine (SVM)

**Theory:** We utilized a Linear SVM (implemented via `SGDClassifier`). SVMs aim to find the hyperplane that maximizes the margin between classes. **Why?** SVMs are theoretically robust to the "Curse of Dimensionality" typical in NLP tasks where the number of features (words) often exceeds the number of samples.

### 7.3 Ensemble Methods: Random Forest & LightGBM

**Theory:** Random Forest builds multiple decision trees on random subsets of data and features (Bagging). LightGBM is a Gradient Boosting framework that builds trees sequentially to correct previous errors. **Why?** To capture non-linear relationships in the data that linear models might miss.

### 7.4 Deep Learning: Bidirectional GRU

We implemented a Recurrent Neural Network (RNN) using **Gated Recurrent Units (GRU)**.

- **Embedding Layer:** Maps word indices to dense vectors (128 dim). Unlike TF-IDF, this captures semantic similarity (e.g., "profit" and "gain" will have similar vector representations).
- **Bidirectional Wrapper:** Allows the network to process the text from start-to-end and end-to-start simultaneously.
- **Architecture:** Input → Embedding → Bi-GRU (64 units) → Dense (Softmax).

## 8 Obstacles Encountered and Solutions

Throughout the project, we encountered three specific obstacles that required engineered solutions.

### 8.1 Obstacle 1: The "Accuracy Trap" (Metric Selection)

**Problem:** Our early models achieved high accuracy ($\sim$ 80%) but failed to identify Negative news. Upon inspection, we realized the models were prioritizing the majority class (Neutral). **Solution:** We shifted our optimization metric from **Accuracy** to **F1-Macro**.

$$F1_{macro} = \frac{F1_{neg} + F1_{pos} + F1_{neu}}{3}$$

Optimizing for F1-Macro forces the algorithm to value the minority classes equally to the majority class.

### 8.2 Obstacle 2: Class Imbalance

**Problem:** The dataset contained very few examples of "Bearish" news. **Solution:** We applied **Cost-Sensitive Learning** by setting `class_weight='balanced'`. This

modifies the loss function to weigh samples inversely proportional to their frequency.

$$w_j = \frac{N}{k \times n_j}$$

By penalizing errors on the Negative class heavily, we forced the Logistic Regression to "pay attention" to bearish signals, raising the Recall from 0.39 to 0.68.

### 8.3 Obstacle 3: Overfitting in Deep Learning

**Problem:** Our Bi-GRU model achieved 95% Training Accuracy but stuck at 76% Validation Accuracy. The loss curves diverged after Epoch 3. **Solution:** This indicated overfitting—the model was memorizing the training noise. We introduced **EarlyStopping** (patience=3) to halt training when validation loss degraded. Ultimately, we concluded that for a dataset of this size (∼9,500 samples), deep learning models are less data-efficient than linear models with TF-IDF.

## 9 Comparison of Models Results

We evaluated all models on the hold-out test set (‘sent_valid.csv‘). The results are summarized below.

| Model | Accuracy | F1-Macro | Recall (Neg) | Inference Speed |
|---|---|---|---|---|
| LogReg (Baseline) | 0.79 | 0.73 | 0.39 | Fast |
| Random Forest | 0.80 | 0.71 | 0.45 | Slow |
| Deep Learning (GRU) | 0.77 | 0.70 | 0.55 | Slow |
| LightGBM | 0.81 | 0.73 | 0.58 | Medium |
| **Optimized LogReg** | **0.81** | **0.74** | **0.68** | **Fast** |

Table 1: Comparative Performance Analysis.

**Key Takeaway:** The **Optimized Logistic Regression** (with Bigrams and Class Weights) offered the best balance. It achieved the highest F1-Macro score and, crucially, the highest Recall for the Negative class (0.68), meaning it misses the fewest bearish signals.

## 10 Bonus: Live Deployment Simulation

To demonstrate the practical applicability of our work (and fulfill the bonus criteria), we implemented a "Live Trading Simulation" script.

### 10.1 Pipeline Serialization

We used the `pickle` library to serialize the final model and the fitted vectorizer.

```
import pickle
# Saving the pipeline for production use
with open('sentiment_pipeline.pkl', 'wb') as f:
    pickle.dump(best_model_pipeline, f)
```

### 10.2 Inference Function

We created a function `predict_sentiment_from_memory` that mimics a production API endpoint. It accepts a raw string, cleans it using the exact same regex rules as training, vectorizes it, and outputs a probability distribution.

**Example Output:**

> Input: *"Companies reporting huge losses due to supply chain issues."*
> Prediction: **Negative** (Confidence: 85%)
> Action: **Trigger Stop-Loss**

## 11 Conclusion

In this project, we successfully engineered an end-to-end Machine Learning pipeline for Financial Sentiment Analysis.

### 11.1 Business Impact

We transformed a theoretical classification problem into a viable financial tool. By focusing on **Recall** and **F1-Score** rather than simple Accuracy, we ensured that our model is sensitive to market risks, making it suitable for integration into a Risk Management System.

### 11.2 Technical Learnings

1. **Complexity ≠ Performance:** A well-tuned Linear Model outperformed a Deep Neural Network on this specific dataset size.
2. **Feature Engineering is King:** The move from Unigrams to Bigrams provided a greater performance boost than any algorithmic change.
3. **Data Understanding:** Identifying the "Look-ahead Bias" in stock tickers was the most critical data cleaning step.

This report demonstrates our ability to apply data science techniques to solve concrete financial engineering problems, bridging the gap between NLP theory and market practice.

# 12 References

# References

[1] Loughran, T., & McDonald, B. (2011). *When is a Liability not a Liability? Textual Analysis, Dictionaries, and 10-Ks.* The Journal of Finance, 66(1), 35-65.

[2] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space.* arXiv preprint arXiv:1301.3781.

[3] Jurafsky, D., & Martin, J. H. (2024). *Speech and Language Processing (3rd ed.).* Pearson.

[4] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12, 2825-2830.