School of
Computing Science

Adapting Natural Language Processing and Deep Learning Tools to an end-to-end Day Trading System

Enrique Alejandro González Amador

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8RZ

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

5th September 2025

Abstract

A concise summary of your project, including problem statement, methodology, key results, and conclusion.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

(Please note that you are under no obligation to sign this declaration, but doing so would help future students.)

Name:

Signature:

Acknowledgements

My sincere gratitude to my supervisor, Joemon Jose, for their guidance and support throughout this project.

Thanks also to the Ministry of Science, Humanities, Technology and Innovation of Mexico, which continues to create opportunities for Mexicans to go abroad to the top academic circles worldwide. Without their funding, none of this would have been possible.

To the School of Computing Science and the department in charge of the University's cluster, for graciously allowing me to use the hardware and infrastructure which allowed me to produce the results showcased on this report.

Lastly, to my friends, family and loved ones, who always cheer me on to keep going.

Contents

List of Figures

List of Tables

Listings

1. Introduction

Trading strategies have evolved alongside the stock market since its inception. As new technologies and understandings emerge, both investors and researchers have tried to put the latest thing to the test to either turn a profit or deepen our understanding of both the markets and the tools we use to do so. As the stock market is used by investors to trade stocks in hopes of making a profit, there is a constant incentive to obtain an edge, whether it be in sophistication of approach, timing or otherwise, to maximize the winning trades and minimize the loosing ones.

- Context and motivation
- Problem definition
- Research objectives/questions
- achievements
- Structure of the report

2. Background

This chapter will go over the multidisciplinary approach of this project. The following sections will cover fundamental knowledge required to understand the core technological and theoretical components . These components are drawn from the fields of deep learning, natural language processing, quantitative finance, and statistical inference.

After laying the groundwork on which this project is built, the later sections will explain the main objectives and the research questions answered throughout the implementation and experimentation of this dissertation.

2.1 Deep Learning: long short-term memory (LSTM)

Time series prediction is a challenging area of Machine Learning and Deep Learning ([Lai et al., 2018]; [Lim and Zohren, 2021]). Traditional recurrent neural networks (RNNs) can theoretically keep track of dependencies and relationships during training trough the use of backpropagation, where the RNN calculates the gradient of the loss function with respect to its weights one layer at a time, iterating backwards till the calculation reaches the first layer; the RNN would use this calculation to further adjust its weights and achieve a lower loss. By understanding 'learning' as an optimization problem ([Rumelhart et al., 1986]), backpropagation became a commonly used tool for traditional RNNs in deep learning.

However, as neural networks grew in size and depth, the nature of the backpropagation process introduced a significant hurdle when training recurrent neural networks([Hochreiter, 1991]). In the vanishing gradient problem, small number multiplication makes the gradients in the backpropagation calculations increasingly small, exponentially approaching zero, causing the training to slow down or even halt completely. This problem of RNNs became a research area that led to several advancements([Hochreiter and Schmidhuber, 1997]; [Kolen and Kremer, 2001]).

Introduced in 1997 ([Hochreiter and Schmidhuber, 1997]), long short-term memory (LSTM) neural networks were proposed as a solution to the vanishing gradient problem. By employing a sophisticated system of cell states and gates, LSTM networks can keep track of dependencies and relationships in long sequences of data without running into the vanishing gradient problem, which makes them effective for time series forecasting and prediction.

LSTM neural networks are at the core of the system developed for this project. The prediction models used in both the testing and the final simulation pipeline are used to run either classification or regression models. The models are trained on a large section of the composite dataset and finetuned to create predictions for the remaining of the data, which will be used for evaluation of performance and the investment simulations.

2.2 Natural Language Processing: BERT

Since this project involves the analysis of large datasets containing text messages (the tweet dataset), there is a need to understand how the text classification tools that will be used on this

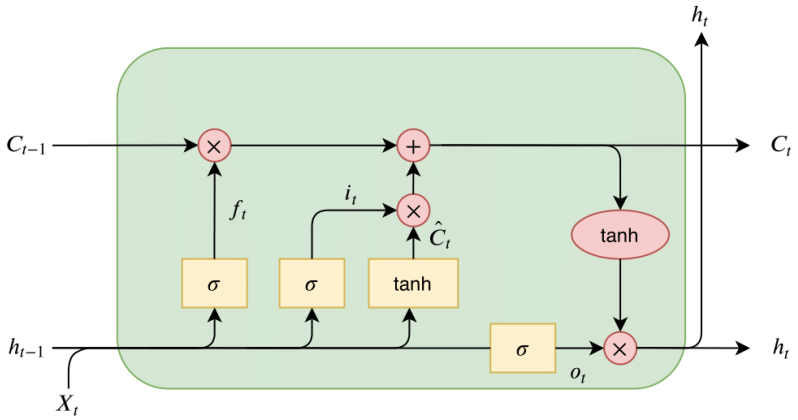

Figure 1: LSTM Cell Diagram

project came about, and how the field of Natural Language Processing (NLP) has progressed up to this point.

Early work in the language processing field was heavily based on pattern and rule matching, trying to encode linguistic rules into early computing systems. Works in machine intelligence ([TURING, 1950]) and linguistics ([Chomsky, 1957]) defined the approaches taken to try an emulate language tasks such as conversational agents and machine translation. An example that displays the culmination of this approaches into a computer chat-bot is ELIZA ([Weizenbaum, 1966]), groundbreaking for the time but is considered rudimentary by today's standards.

When the limitations of rule based systems became apparent, the field moved on into statistical approaches([Brown et al., 1990]; [Manning and Schütze, 1999]). N-gram and Hidden Markov models (HMVs) became the state of the art techniques for tasks such as part-of-speech tagging and machine translation tasks.

As the field continued to advance the introduction of early deep learning techniques allowed for further advancement on the field of NLP. The introduction of the first neural language models ([Bengio et al., 2003]), the introduction and popularization of word embeddings ([Mikolov et al., 2013]) and the usage of RNN and LSTM for tasks such as machine translation ([Sutskever et al., 2014]) as it became the state-of-the-art before the introduction of transformers.

This historical overview leads to the introduction of BERT. Shortly after the introduction of the transformer architecture in 2017([Vaswani et al., 2017]), a new model that built upon this architecture was introduced. BERT, which stands for Bidirectional Encoder Representations from Transformers, significantly advanced the ability of computers to process human language and generate text. Introduced in 2018 ([Devlin et al., 2018]), BERT brought about significant improvements in a variety of NLP tasks such as text classification. As a pre-trained model, it has already been trained on a large corpus of text. This allows the model to be used directly or to be fine-tuned on smaller, task-specific datasets.

To capture semantic and contextual information from tweets, the system employs several pre-trained BERT-based language models. Each model is fine-tuned for text classification (e.g., sentiment, stance, or emotion). The resulting classification outputs (such as probabilities or embeddings) are then used as features that serve as inputs to subsequent deep learning tasks.

This approach leverages the linguistic knowledge already encoded in large pre-trained models, while transforming raw text into structured, high-level representations suitable for downstream models. Combining multiple BERT variants enables the generation of diverse feature sets, which can improve robustness and performance compared to relying on a single model.

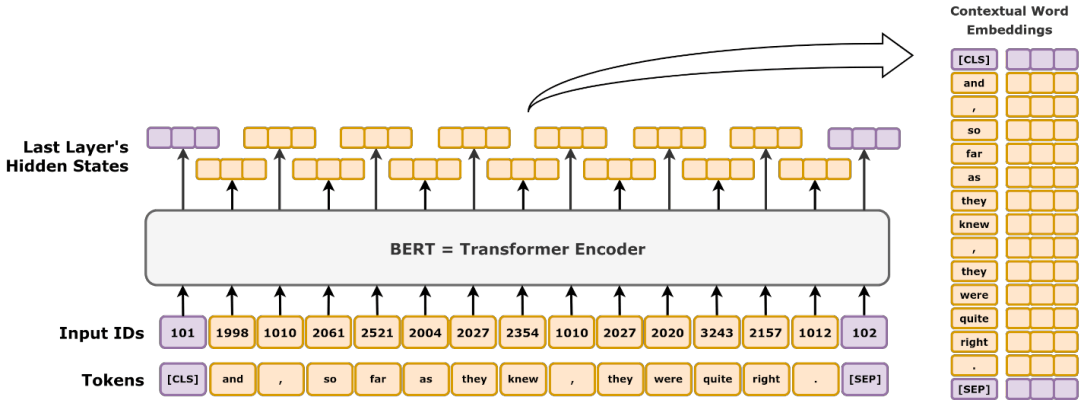

Figure 2: High-level schematic diagram of BERT.

2.3 Staking Strategies: Kelly Criterion

For any investor or gambler, determining the optimal amount of capital to allocate to an opportunity is a fundamental aspect of a successful strategy. Go too small, and potential returns are needlessly diminished; risk too much, and the threat of ruin looms large. The Kelly Criterion offers a mathematical solution to this dilemma, providing a formula for sizing positions to maximize the long-term growth rate of assets. Developed by John L. Kelly Jr. in 1956, the criterion provides a disciplined framework that balances the probability of success with the potential returns, a tool that has been embraced in fields ranging from sports betting to sophisticated investment management.

Developed originally as a way to analyze long-distance telephone signal noise ([Kelly, 1956]), the Kelly Criterion's potential applications for gambling and investment were quickly recognized. The general form for the Kelly Criterion, assuming a loss of the full wager with every bet lost, is defined as:

$$f^* = p - \frac{q}{b} = p - \frac{1-p}{b} \tag{1}$$

where:

- f^* is the fraction of the current bankroll to wager.
- p is the probability of a win.
- $q = 1 - p$ is the probability of a loss.
- b is the proportion of the bet gained with a win.

This general form of the formula is intuitive to understand and was rigorously proven in its original paper. However, the original formulation did not account for the specifics of investment returns; to address this, different formulations were made to account for continuous returns and other factors ([Thorp, 2008]; [Rotando and Thorp, 1992]). For the purpose of investment, the formula for the Kelly Criterion can be adapted as:

$$f^* = \frac{p}{l} - \frac{q}{g} \tag{2}$$

where:

- f^* is the fraction of the assets to apply to the stock or security.
- p is the probability that the investment increases in value.
- q is the probability that the investment decreases in value.
- g is the fraction that is gained in a positive outcome.

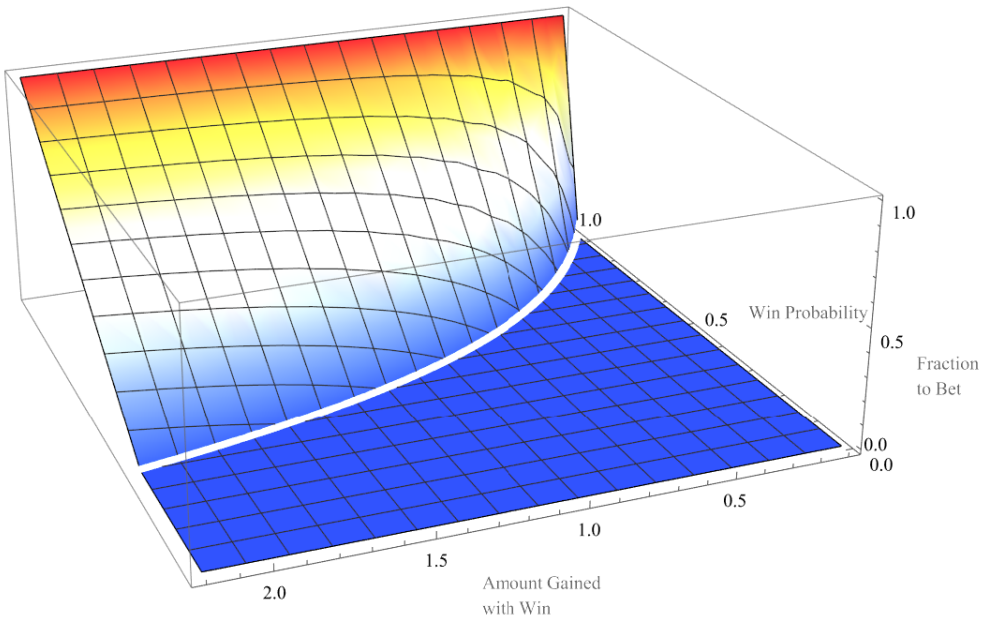

Figure 3: 3D figure representing the optimal Kelly bet size (vertical axis).

There are two main problems with using the Kelly Criterion for investments with probabilities informed by a deep learning model. First, even if we tune our LSTM network to output a value between zero and one (which would mimic traditional probability), this is not an actual probability, just the confidence estimation from the model of wether of not the stock will go up or down based on the patterns learned from training data and the input of today. Second, for the Kelly Criterion to hold true in investment scenarios, we must know how much the stock being traded will go up or down. This information is almost always unknown and it can only be estimated based on historical data. In later sections, the strategies and tools chosen to mitigate this limitations will be further explained.

2.4 Statistical Inference: Isotonic Regression

As mentioned in the deep learning overview of this report, even if we tune a deep learning model to output results of model confidence as a number between zero and one in accordance with Kolmogorov Axioms of probability ([von Plato, 1933]), we are not actually getting the probability of the predicted stock to go up or down, the model is simply giving its degree of confidence based on its training data and the learned patterns encoded in its weights. This becomes problematic since the Kelly Criterion requires the probability to be known in order to maximize the expected returns in the long run. Even assuming a clean and sufficiently large dataset and an optimal model, it is unrealistic to expect its predictions to be analogous to true probability.

In order to address this problem, this project makes use of isotonic regression as an intermediate step between the classifier score and the staking calculation. Isotonic Regression was introduced alongside its cornerstone algorithm, the Pool Adjacent Violators Algorithm (PAVA). Both introduced in the same paper ([Ayer et al., 1955]), they became a tool used in many fields for their flexibility.

Functionally, isotonic regression fits a line into a sequence of data points such that the fitted line is non-decreasing (or non-increasing) at all points, while remaining as close to the observations as possible ([Fielding, 1974]). The strength of the technique lies in its ability to correct monotonic distortions as well as its non-linearity.

In 2002, it was first shown that probability calibration using isotonic regression could be used

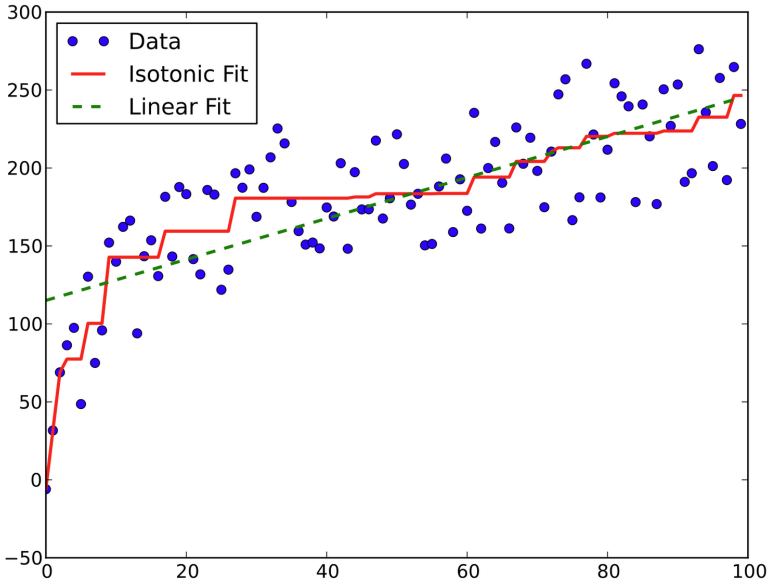

Figure 4: Isotonic regression (solid red line) compared to linear regression on the same data, both fit to minimize the mean squared error.

for classification problems in the context of machine learning with satisfactory results ([Zadrozny and Elkan, 2002]). The paper clearly articulates how isotonic regression works well for both binary and multiclass classification problems as well as showing several advantages compared to other calibration methods, particularly its non-parametric nature. The experimental results ultimately showed the benefits of using calibrated probabilities instead of taking the classifiers scores at face value, since these are more often than not, uncalibrated.

2.5 Return on Investment

2.6 Project Aims

2.6.1 Research Objectives

- Objective 1: Stocks and tweets feature engineering
  - Extract technical indicators (e.g., moving averages, RSI, MACD, volatility) from stock price data.
  - Perform sentiment analysis, stance detection and emotion analysis on tweets related to the stocks.
  - Align and aggregate stock features with tweet-derived features on a common time scale.
  - Run tests using combinations of features to understand the impact of the engineered features
- Objective 2: Model and features testing and evaluation
  - Design an environment to test different deep learning architectures.
  - Compare results to baselines such as ARIMA and previous other relevant stock prediction papers ([Xu and Cohen, 2018]; [Hu et al., 2019])
  - Evaluate predictive performance using metrics such as accuracy, F1-score, RMSE, and Matthews coefficient.
- Objective 3: Creation of an investment simulation pipeline

  - Design a framework to simulate trading strategies based on model predictions.
  - Implement a staking strategy using the Kelly Criterion and other risk management techniques.
  - Assess profitability and risk through performance metrics (e.g., cumulative returns, Sharpe ratio).

2.6.2 Research Questions

- What is the overall profitability of a day trading strategy that combines LSTM-based stock price predictions with real-time sentiment analysis from Twitter?
- To what extent does the inclusion of text classification of tweets improve the predictive accuracy of the LSTM model for short-term stock price movements?
- Can a trading model that integrates both historical price data and social media sentiment consistently outperform a baseline buy-and-hold strategy in a simulated environment?
- What is the simulated risk-adjusted return (e.g., Sharpe ratio) of the proposed trading strategy, and how does it compare to traditional market benchmarks?
- How do different transaction cost assumptions impact the overall return on investment of the simulated trading strategy?
- What is the maximum drawdown of the trading strategy, and what factors (e.g., high market volatility, conflicting signals from LSTM and tweet analysis) contribute to periods of significant loss?
- How does the strategy perform during different market conditions, such as bull markets, bear markets, and periods of high volatility?
- What is the isolated contribution of the LSTM model to the profitability of the trading strategy when sentiment analysis is excluded?
- Conversely, what is the profitability of a strategy guided solely by tweet sentiment analysis, and how does it compare to the integrated model?
- How does the choice of sentiment classification (e.g., positive, negative, neutral vs. more nuanced emotions like fear or excitement) affect the trading outcomes?
- How does the performance of the LSTM model compare to other time-series forecasting models (e.g., ARIMA, GARCH) in the context of this day trading simulation?
- What is the optimal look-back period for historical stock data to train the LSTM model for maximizing predictive accuracy in day trading?
- How does the model's performance vary when applied to stocks from different sectors or with different market capitalizations?
- What is the ideal threshold for sentiment scores from tweets to trigger a buy or sell signal in order to maximize the return on investment?2

3. Methodology, Design & Implementation

3.1 Architecture

The entire project is written in Python. A complete list of the packages utilized is found in the next section.

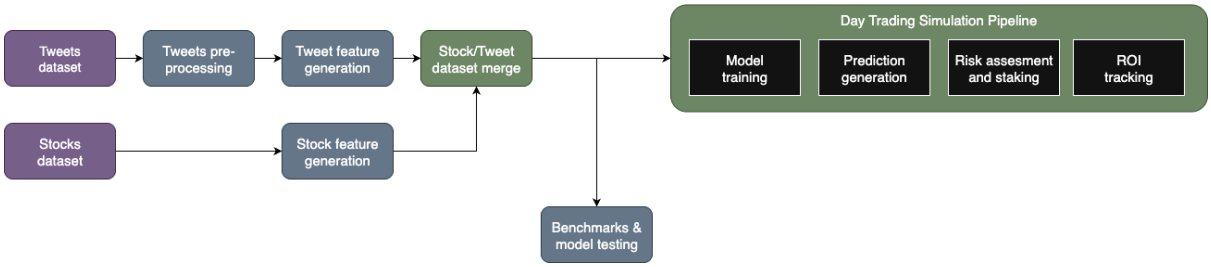

Figure 5: High level logic of the System's architecture.

As seen in Figure 1, the actual system (the returns simulation pipeline) is a separate entity from the pre-processing and testing/benchmarking steps as the computational work required to run a full simulation for each possible configuration of features and parameters makes it prohibitive in both time and resources availability.

All the source code lives on standalone Jupyter notebooks (.ipynb files) and can be run independently provided that all the necessary files and packages exist on the environment. If the system were to be deployed for real trading operations, the notebooks could be triggered via a bash script or using an automation package such as papermill.

3.2 Libraries and Modules by Functionality

- **Data Handling**
  - **pandas (pd)** : Manage stock price series and tweet-derived tabular data.
  - **numpy (np)** : Perform numerical operations and efficient array computations.
  - **json** : Parse tweet datasets and save model configurations.
  - **os** : Manage file paths and environment variables (e.g., suppress TensorFlow logs).
  - **site** : Minimal usage for environment path management.
- **Feature Engineering**
  - **pandas_ta (ta)** : Generate technical indicators (RSI, MACD, etc.) from stock price data.
  - **re** : Clean and preprocess tweets using regular expressions.
- **Preprocessing and Calibration (sklearn)**

  - **preprocessing** : - `StandardScaler, MinMaxScaler, RobustScaler` : Normalize features before training.
    - `IsotonicRegression` : Calibrate predicted probabilities.
  - **utils** : - `compute_class_weight` : Handle class imbalance in financial classification tasks.
- **Model Training and Architecture**
  - **tensorflow (tf)**
    * **keras.models** : `Sequential, load_model` — Build and reload deep sequential models.
    * **keras.layers** : `LSTM, Dense, Dropout` — Core RNN and fully-connected layers.
    * **keras.callbacks** : `EarlyStopping, ReduceLROnPlateau` — Stabilize training and prevent overfitting.
    * **keras.regularizers** : `l2` — Apply weight penalties for generalization.
  - **torch** : Backend engine for HuggingFace Transformers; enables GPU-accelerated training.
- **Model Evaluation (sklearn)**
  - **metrics** : - `precision_score, recall_score, f1_score, matthews_corrcoef` : Classification evaluation.
    - `mean_squared_error, mean_absolute_error, r2_score` : Regression evaluation.
    - `confusion_matrix` : Assess classification performance visually.
  - **model_selection** : - `train_test_split` : Split data into training and testing sets.
    - `TimeSeriesSplit` : Ensure chronological integrity in time series validation.
- **NLP and Transformers (HuggingFace)**
  - `AutoTokenizer, AutoModelForSequenceClassification` : General-purpose pre-trained models for tweet classification.
  - `RobertaForSequenceClassification, RobertaTokenizer` : Fine-tuned RoBERTa for financial sentiment analysis.
  - `pipeline` : Rapid prototyping of classification workflows.
- **Visualization**
  - **matplotlib.pyplot (plt)** : Create plots for exploratory analysis and results.
  - **seaborn (sns)** : Generate statistical plots such as heatmaps for feature correlations.
  - **wordcloud (WordCloud)** : Visualize word frequency distributions in tweets.
- **Utilities**
  - **joblib** : Save/load trained preprocessing objects like scalers.
  - **tqdm** : Track progress in training loops and data preprocessing.
  - **warnings** : `filterwarnings('ignore')` — Keep experiment logs clean by suppressing irrelevant warnings.

3.2 Dataset selection and Data preparation

All the data used in both the evaluation and simulation stages of the project comes from the Stocknet dataset, originally introduced in the work of Xu and Cohen [2018].

The Stocknet dataset provides a comprehensive benchmark for the task of stock movement prediction by integrating both financial market data and social media signals. It covers a two-year period from January 1, 2014 to January 1, 2016, and includes a total of 88 target stocks. The selection procedure follows the approach outlined by Xu and Cohen [2018]: all eight stocks in the Conglomerates sector are included, while in each of the remaining eight sectors the ten largest firms by market capitalization are selected. This ensures both sectoral diversity and sufficient representation of highly traded firms. The complete list of stocks is reported in the appendix of the original paper.

The dataset consists of two main components: *tweet data*, collected from Twitter, and *price*

data, retrieved from Yahoo Finance. For both components, the repository provides raw and preprocessed versions, organized by stock ticker.

- **Tweet data:**
  - Raw format: JSON objects following the standard Twitter API structure.
  - Preprocessed format: JSON files containing the keys `text`, `user_id_str`, and `created_at`.
- **Price data:**
  - Raw format: CSV files with daily entries including date, open, high, low, close, adjusted close prices, and volume.
  - Preprocessed format: TXT files with daily entries containing the date, percentage movement, normalized open, high, low, and close prices, as well as volume.

By combining market signals with large-scale social media data, the Stocknet dataset enables research on multimodal approaches to financial prediction. Its design facilitates the study of temporally-dependent relationships between unstructured text and structured price series, making it particularly suitable for deep learning models that exploit both modalities.

3.3 Dataset Pre-processing

The initial Stocknet dataset, while comprehensive, requires significant pre-processing to transform its raw components into a format suitable for machine learning models. This phase involves loading heterogeneous data sources, cleaning textual data to remove noise, and structuring the final datasets for subsequent feature engineering and model training stages. The process is executed across several sequential notebooks.

First, the raw price and tweet data are loaded and consolidated. The price data, originally stored in separate CSV files for each of the 88 tickers, is read into memory, combined into a single tidy pandas DataFrame, and then saved in the efficient Parquet format as `stock_prices.parquet`. Similarly, the tweet data, distributed across a nested directory structure of JSON files, is parsed to extract key fields—namely the ticker, tweet text, creation timestamp, and user ID. This consolidated tweet data is then saved as `stock_tweets.parquet`.

The core of the pre-processing work focuses on sanitizing the tweet text. An initial cleaning function is applied to each tweet to eliminate common social media artifacts that provide little semantic value for sentiment analysis. This step removes URLs, user mentions (e.g., `@username`), hashtags, stock cashtags (e.g., `$TICKER`), and various punctuation marks. The regular expression pattern used for this initial sanitization is shown below.

Listing 1: Initial Regex Pattern for Tweet Sanitization

```
char_patterns = re.compile(
    'http[s]?://(?:[a-zA-Z]|[0-9]|[[a-zA-Z]+|
    @[a-zA-Z]+|[,.^_$*%-;!?:]'
    )

for i in range(len(tweet_df["text"])):
    tweet_df["text"][i] = char_patterns.sub('', tweet_df["text"][i])
```

Following the initial cleaning, two distinct data structures are prepared for different modeling approaches. The first approach involves creating a daily summary of market sentiment. To achieve this, all individual tweets for a given stock on a specific day are aggregated by concatenating their text into a single document. This produces a DataFrame where each row corresponds to one stock on one trading day, containing all associated tweet text.

A second, more refined cleaning step is then applied to both the aggregated and non-aggregated tweet data. This step removes words and abbreviations that could act as trivial predictors or introduce noise, such as the stock's own ticker symbol (to prevent the model from

simply learning a correlation with the ticker's presence), retweet indicators ('rt'), and common filler words like 'inc' and 'ie'. This ensures that the model focuses on the semantic content of the tweets rather than superficial patterns.

Listing 2: Refined Regex for Removing Tickers and Noise

```
for i in range(len(tweet_df["text"])):
    # The ticker is dynamically inserted into the pattern
    ticker = tweet_df["ticker"][i]

    more_patterns = re.compile(
        r'(\s*)({})(\s*)|(\s*)ATUSER(\s*)|(\s*)AT_USER(\s*)|
        (\s*)URL(\s*)|(\s*)rt(\s*)|(\s*)inc(\s*)|
        (\s*)ie(\s*)'.format(ticker), flags=re.IGNORECASE
        )

    tweet_df["text"][i] = more_patterns.sub('', tweet_df["text"][i])
```

This multi-stage pre-processing pipeline results in two final, cleaned datasets ready for analysis:

- `merged_stock_tweet.parquet`: Contains tweets aggregated on a daily per-stock basis.
- `stock_tweet_no_merge.parquet`: Contains individual tweets, cleaned but not aggregated.

These clean and structured datasets serve as the foundation for all subsequent feature extraction and model training tasks.

3.4 Tweet Feature Engineering

After pre-processing, the raw text of the tweets is transformed into a set of quantitative features suitable for machine learning. This process leverages several state-of-the-art pre-trained transformer models from the HuggingFace library to extract nuanced signals related to sentiment, emotion, and financial stance. Each feature is calculated on a per-tweet basis before being aggregated to a daily frequency.

First, a general sentiment score is assigned to each tweet using the `nlptown/bert-base-multilingual-uncased-sentiment` model. This model classifies text on an ordinal scale from 1 (very negative) to 5 (very positive). The score is determined by taking the argmax of the model's output logits, which provides a single integer representing the overall sentiment polarity.

Listing 3: Function for Calculating Sentiment Score

```
def cal_sentiment_score(text):
    # Tokenize and move tensors to GPU
    inputs = tokenizer.encode(text, return_tensors='pt',
       truncation=True, max_length=512).to(device)

    with torch.no_grad():
        outputs = bert_model(inputs)

    # The sentiment is the index of the highest logit, plus one
    sentiment = int(torch.argmax(outputs.logits)) + 1
    return sentiment
```

To capture a more detailed emotional context, a second model, `j-hartmann/emotion-english-distilroberta-base`, is employed. This classifier analyzes each tweet and returns a probability distribution across seven distinct emotional categories: anger, disgust, fear, joy, neutral, sadness, and surprise. This process generates seven new feature columns, each containing the confidence score for the corresponding emotion.

Finally, to incorporate a domain-specific signal, a financial stance detection model is used. The chosen model, `zhayunduo/roberta-base-stocktwits-finetuned`, has been specifically fine-tuned on data from StockTwits, a social network for investors and traders. It classifies each tweet as either 'Positive' (bullish) or 'Negative' (bearish) and provides a confidence score for this classification. This is crucial for distinguishing general sentiment from a direct opinion on a stock's future performance.

Listing 4: Function for Financial Stance Detection

```
def stance_score(text):
    # The pipeline handles tokenization and model inference
    result = nlp(text)
    label = result[0]['label']
    score = result[0]['score']
    return label, score
```

Since the predictive models operate on a daily timescale, these tweet-level features must be aggregated for each stock on each trading day. The aggregation is performed as follows:

- **Sentiment Score**: The final daily sentiment is the **average** of all individual tweet sentiment scores for that day.
- **Emotion Scores**: For each of the seven emotions, the daily value is the **sum** of all corresponding emotion scores from tweets on that day.
- **Stance**: The stance is captured in two columns representing the total daily count of tweets classified as 'Positive' and 'Negative', respectively.

This multi-faceted feature engineering approach produces a rich daily summary of social media signals, which serves as a primary input for the stock movement prediction models. The final enriched dataset is saved as `stock_tweets_withsentiment_withemotion_withstance_nomerge.parquet`.

3.5 Stocks Feature Engineering

In addition to social media signals, a comprehensive set of technical indicators is engineered from the historical price data to capture market dynamics such as trend, momentum, volatility, and volume. These features are widely used in quantitative trading and provide the models with a quantitative basis for understanding price action.

The process leverages the `pandas_ta` library, a powerful tool that integrates seamlessly with pandas DataFrames to calculate a wide array of technical indicators. To ensure the chronological integrity required for these calculations, the master dataset is first sorted by ticker and date. Then, the data is grouped by each unique ticker, and the feature engineering function is applied independently to each stock's time series. This prevents data leakage across different assets.

The selection of indicators is designed to provide a multi-faceted view of the market:

- **Trend Indicators:** Exponential Moving Averages (EMAs) with periods of 12, 26, and 50 days are calculated to identify short, medium, and long-term trends. The Moving Average Convergence Divergence (MACD) indicator is also included, providing the MACD line, signal line, and histogram.
- **Momentum Indicators:** The Relative Strength Index (RSI) with a 14-day period is used to measure the speed and change of price movements, helping to identify overbought or oversold conditions. The Stochastic RSI is also computed to provide a more sensitive momentum reading.
- **Volatility Indicators:** The Average True Range (ATR) measures market volatility, while Bollinger Bands (with a 20-day period and 2 standard deviations) create a dynamic envelope around the price, indicating periods of high or low volatility.

- **Volume Indicators:** On-Balance Volume (OBV) is used to relate price and volume, providing insights into the strength of a price trend based on trading volume.

The core logic for this process is encapsulated in a single function that is applied to each stock's data, as shown below.

Listing 5: Function for Calculating Technical Indicators

```
def apply_ta_indicators(df_group):
    df_group.set_index(pd.DatetimeIndex(df_group['date']),
       inplace=True)
    # Trend indicators
    df_group.ta.ema(length=12, append=True)
    df_group.ta.ema(length=26, append=True)
    df_group.ta.ema(length=50, append=True)
    df_group.ta.macd(fast=12, slow=26, signal=9, append=True)


    # Momentum indicators
    df_group.ta.rsi(length=14, append=True)
    df_group.ta.stochrsi(length=14, append=True)

    # Volatility indicators
    df_group.ta.atr(length=14, append=True)
    bb = ta.bbands(df_group['close'], length=20, std=2)
    df_group['BB_upper'] = bb['BBU_20_2.0']
    df_group['BB_middle'] = bb['BBM_20_2.0']
    df_group['BB_lower'] = bb['BBL_20_2.0']

    # Volume indicators
    df_group.ta.obv(append=True)
    return df_group.reset_index(drop=True)

master_df = master_df.groupby('ticker').apply(apply_ta_indicators)
```

This systematic approach ensures that each stock's features are calculated based solely on its own historical data, resulting in a robust feature set ready for model training.

3.6 Final Dataset

The final step in the data preparation pipeline is the construction of a master DataFrame that consolidates all engineered features from the different data sources: historical stock prices, aggregated daily tweet metrics, and company metadata. This unified dataset serves as the single source of truth for training and evaluating all predictive models.

The process begins by merging the daily stock price data with the daily aggregated tweet features. A left merge is performed on the stock price table, ensuring that all trading days are preserved, even those for which no tweet data is available. For such days, the tweet-derived features (e.g., sentiment, emotion scores, stance counts) are imputed with a value of zero, representing a neutral or non-existent social media signal. Subsequently, static company information, such as sector and company name, is merged in based on the stock ticker.

The resulting master DataFrame is a rich, high-dimensional dataset where each row corresponds to a single trading day for a specific stock. It contains all the necessary information for building multimodal prediction models, combining fundamental company attributes, market-based technical indicators, and social media sentiment.

The columns of the final master DataFrame are described below:

`date`: The date of the trading session.

`open, high, low, close, adj_close`: Standard daily Open, High, Low, Close, and Adjusted Close prices.

`volume`: The number of shares traded on that day.

`ticker`: The stock's ticker symbol.

`company`: The stock's company name.

`sector`: The industrial sector to which the company belongs.

`stance_positive, stance_negative`: The total daily count of tweets classified as bullish and bearish, respectively.

`sentiment`: The average sentiment score (from 1 to 5) of all tweets for that day.

`emotion_`: Seven columns (`emotion_anger`, `emotion_disgust`, etc.) containing the summed daily scores for each emotion.

`MACD, MACDh*, MACDs*`: The MACD line, histogram, and signal line.

`RSI_14`: The 14-day Relative Strength Index.

`STOCHRSIk*, STOCHRSId*`: The K and D lines of the Stochastic RSI.

`ATRr_14`: The 14-day Average True Range.

`BB_upper, BB_middle, BB_lower`: The upper, middle, and lower Bollinger Bands.

`OBV`: The On-Balance Volume indicator.

`EMA_12, EMA_26, EMA_50`: Exponential Moving Averages for 12, 26, and 50-day periods.

3.7 Model Evaluation and Metric Tracking

To facilitate a rigorous and reproducible evaluation of predictive models, a custom Python class, StockPredictionPipeline, was created. This class serves as a comprehensive framework that encapsulates the entire experimental workflow, from data processing to model training, evaluation, and results aggregation. Its primary design goal is to enable the systematic benchmarking of different model architectures and problem formulations across a diverse portfolio of stocks.

The pipeline's flexibility is centered around its initialization, where the user defines the core parameters of the experiment. These inputs include the dataset, the feature set, the model architecture (e.g., LSTM, BiLSTM), the time-series sequence length, and, most importantly, the problem_type. This parameter explicitly configures the pipeline to perform either regression (predicting the magnitude of the next day's price change) or classification (predicting the direction of the next day's price change).

Listing 6: Feature Set and Pipeline Initialization

```
feature_columns = [
        'open', 'high', 'low', 'close', 'volume',
        'stance_positive', 'stance_negative',
        'EMA_12', 'EMA_26', 'EMA_50', 'MACD_12_26_9',
            'MACDh_12_26_9',
        'MACDs_12_26_9', 'RSI_14', 'ATRr_14', 'STOCHRSIk_14_14_3_3',
        'STOCHRSId_14_14_3_3', 'BB_upper', 'BB_middle', 'BB_lower',
            'OBV'
        ]
```

```
8
9  #The pipeline can be initialized for either regression or
       classification
10
11 pipeline = StockPredictionPipeline(
12                                     df=master_df,
13                                     feature_columns=feature_columns,
14                                     model_type='BiLSTM',
15                                     sequence_length=12,
16                                     problem_type='regression' # or
                                           'classification'
17                                     )
```

A distinct model is trained for each company, allowing the network to capture the unique dynamics of individual stocks. The model architecture is constructed dynamically based on the chosen problem_type. While the core recurrent layers (e.g., BiLSTM) and regularization layers (Batch Normalization, Dropout) remain consistent, the output layer and compilation settings are tailored specifically to the task.

For regression, the model employs a final Dense layer with a linear activation function and is compiled with the Huber loss function to robustly predict continuous log returns.

For classification, the model uses a Dense layer with a sigmoid activation function to output a probability and is compiled with binary cross-entropy loss for the directional prediction task.

Listing 7: Task-Specific Model Architecture

```
1  def build_model(self, input_shape):
2      # ... (Shared RNN, Batch Norm, and Dense layers) ...
3
4      # Problem-specific output layer and compilation
5      if self.problem_type == 'regression':
6          model.add(layers.Dense(1, activation='linear'))
7          model.compile(
8              optimizer=keras.optimizers.Adam(learning_rate=0.001),
9              loss='huber',
10             metrics=['mae', 'mse']
11             )
12     else:  # classification
13         model.add(layers.Dense(1, activation='sigmoid'))
14         model.compile(
15             optimizer=keras.optimizers.Adam(learning_rate=0.001),
16             loss='binary_crossentropy',
17             metrics=['accuracy', 'precision', 'recall']
18             )
19 return model
```

The evaluation logic is equally task-aware. The pipeline calculates a comprehensive suite of metrics appropriate for the chosen problem type, ensuring a thorough assessment of performance.

When a regression model is trained, it calculates standard regression metrics (MSE, MAE, R^2). Crucially, it also evaluates the model's directional capability by converting the predicted log returns into binary up/down signals and computing corresponding classification metrics (Accuracy, MCC, F1 Score).

When a classification model is trained, it directly calculates the full set of classification metrics from the model's probabilistic output. In this mode, the regression-specific metrics are recorded as not applicable (NaN).

This dual-metric system ensures that regardless of the primary task, the model's ability to predict market direction—a key indicator of practical utility—is always evaluated. All perfor-

mance metrics, along with metadata like company, sector, and sample sizes, are captured for each individual stock.

Listing 8: Comprehensive Metric Collection

```
result = {
        'company': company_name,
        'sector': sector,
        'model_type': self.model_type,
        'problem_type': self.problem_type,
        # Regression metrics (NaN if classification)
        'mse': mse,
        'mae': mae,
        'r2': r2,
        # Classification / Directional metrics
        'mcc': mcc,
        'f1': f1,
        'precision': precision,
        'recall': recall,
        'directional_accuracy': directional_accuracy,
        # Data info
        'n_samples': len(X),
        'test_samples': len(X_test)
        }
```

Upon processing all companies, the pipeline aggregates the individual results into a final Pandas DataFrame. An analysis function then computes overall performance statistics (mean and standard deviation) for each metric, provides performance breakdowns by market sector, and identifies top-performing models. This aggregated data is saved to a timestamped CSV file, creating a persistent and detailed record of the experiment. This structured methodology allows for a direct and fair comparison between the two distinct approaches: predicting the magnitude of price changes versus predicting the direction.

3.8 Investment Simulation Pipeline

The core of this project is a sophisticated, event-driven backtesting system designed to simulate a realistic trading environment. This pipeline moves forward day-by-day, making dynamic decisions about model training, stock selection, and capital allocation based on historical data available up to that point. This approach avoids lookahead bias and provides a robust evaluation of the strategy's real-world viability. The entire system is governed by a set of key configurable parameters that define its behavior.

Listing 9: Global Configuration for the Simulation

```
MODEL_SAVE_PATH = "trained_models/"
INITIAL_TRAINING_DAYS = 1100 # Days of data before simulation begins
KELLY_FRACTION = 0.05 # Fraction of the Kelly bet to take
SECTOR_CONFIDENCE_THRESHOLD = 0.40 # Min avg sector probability to
    consider
RETRAIN_INTERVAL = 200 # Days before a model is flagged for
    retraining
MAX_DAY_GAP = 5 # Max gap to consider a data period contiguous
```

The pipeline is architected around several key functions that handle data preparation, model training, prediction, and portfolio management.

3.8.1 Dynamic Data Handling and Robustness

To handle the non-continuous nature of real-world financial data (due to weekends, holidays, or data gaps), the system first identifies contiguous trading periods for each stock. This ensures that the time-series sequences fed into the models represent uninterrupted data, preventing the models from learning from artificial jumps. Furthermore, the sequence length for each model is not fixed but is calculated dynamically based on the amount of historical data available for each specific company. This adaptive approach allows the system to build more robust models for stocks with longer histories while still being able to model those with less available data.

Listing 10: Handling of Contiguous Data Periods

```
def create_contiguous_sequences(data: np.ndarray,
                                targets: np.ndarray,
                                contiguous_periods: list,
                                sequence_length: int):

    X, y = [], []

    for start_idx, end_idx in contiguous_periods:
        # ... logic to skip periods shorter than sequence_length ...

        for i in range(start_idx + sequence_length, end_idx + 1):
            X.append(data[i-sequence_length:i])
            y.append(targets[i])

    return np.array(X), np.array(y)
```

3.8.2 Per-Asset Modeling and Probability Calibration

A separate Bidirectional LSTM (BiLSTM) model is trained for each individual stock. The task is framed as a binary classification problem: predicting whether the next day's closing price will be higher than the current day's close. A critical step in the training process is the calibration of model outputs. The raw output of a neural network's sigmoid function, while bounded between 0 and 1, does not represent a true, reliable probability. To address this, an Isotonic Regression model is fitted on the validation set predictions. This post-processing step calibrates the model's output, transforming it into more accurate probabilities that can be confidently used for quantitative decision-making, such as in the Kelly Criterion.

For each asset, the system saves four essential artifacts: the trained Keras model, the fitted data scaler, the Isotonic Regression calibrator, and the dynamically calculated sequence length.

Listing 11: Model Training and Isotonic Calibration

```
def train_company_models(company_data_df: pd.DataFrame,
                         ticker: str, ...):

    # ... data splitting and scaling ...

    # Define and train the LSTM model
    model = keras.Sequential([
        layers.Input(shape=(X_train_scaled.shape[1],
            X_train_scaled.shape[2])),
        layers.Bidirectional(layers.LSTM(128,
            return_sequences=True, ...)),
        layers.Bidirectional(layers.LSTM(64,
            return_sequences=False, ...)),
        layers.BatchNormalization(),
```

```
        layers.Dense(32, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(1, activation='sigmoid')
        ])

    model.compile(loss='binary_crossentropy', ...)
    model.fit(X_train_scaled, y_train, ...)

    # Train the Isotonic Regression calibrator
    validation_predictions = model.predict(X_val_scaled).flatten()
    calibrator = IsotonicRegression(y_min=0.0, y_max=1.0,
        out_of_bounds='clip')
    calibrator.fit(validation_predictions, y_val)

    # Save all components
    model.save(f"{ticker}_lstm.keras")
    joblib.dump(calibrator, f"{ticker}_calibrator.pkl")
    joblib.dump(scaler, f"{ticker}_scaler.pkl")
    joblib.dump(sequence_length, f"{ticker}_seq_length.pkl")
```

3.8.3 Portfolio Construction and Capital Allocation

Each day within the simulation, the system generates a calibrated probability prediction for every stock using the trained model. These predictions are then fed into a portfolio construction module that implements a sector-based investment strategy combined with the Fractional Kelly Criterion for position sizing.

The process is as follows:

1. **Sector-Level Filtering:** Stocks are grouped by their respective sectors. If the average calibrated probability for a sector falls below the parameter `SECTOR_CONFIDENCE_THRESHOLD`, the entire sector is disregarded for that day.

2. **Best-in-Sector Selection:** For each qualifying sector, the single stock with the highest calibrated probability is selected as the investment candidate.

3. **Fractional Kelly Sizing:** The investment amount for each selected stock is determined using the Fractional Kelly Criterion. The optimal fraction of capital, f, is given by:

$$f = k \cdot \left(p - \frac{1-p}{b} \right),$$

   where

   - p is the calibrated probability of the stock's price increasing,
   - b is the historical average payout ratio (win size divided by loss size), calculated once from the initial training data,
   - k is the `KELLY_FRACTION`, a conservative scaling factor used to reduce risk.

This disciplined approach ensures diversification across sectors and employs a mathematically grounded method for both risk and capital management.

Listing 12: Portfolio Selection and Sizing Logic

```
def select_and_size_portfolio(daily_predictions_df,
                              payout_map, ...):

    investment_decisions = []
    for sector, group in daily_predictions_df.groupby('sector'):
```

```
        # 1. Sector-Level Filtering
        if group['calibrated_prediction'].mean() < sector_threshold:
            continue


        # 2. Best-in-Sector Selection
        best_stock =
            group.loc[group['calibrated_prediction'].idxmax()]
        p = best_stock['calibrated_prediction']
        b = payout_map.get(best_stock['ticker'], 0)

        if b <= 0:
            continue

        # 3. Fractional Kelly Sizing
        kelly_percentage = p - ((1 - p) / b)
        if kelly_percentage > 0:
            investment_fraction = kelly_percentage * kelly_fraction
            investment_amount = total_capital * investment_fraction
            investment_decisions.append(...)

    return pd.DataFrame(investment_decisions)
```

3.8.4 Simulation Loop and Performance Tracking

The main simulation function orchestrates the entire process over a long historical period. It begins by training models on an initial `INITIAL_TRAINING_DAYS` of data. Then, it iterates through the remaining dates one by one. On each day, it:

1. Identifies models that require training or retraining (based on the parameter `RETRAIN_INTERVAL`).
2. Generates predictions for all available assets using the latest historical data.
3. Constructs the daily portfolio and determines investment sizes.
4. Calculates the profit or loss (PnL) from the previous day's investments based on the actual market outcome.
5. Updates the total capital.
6. Logs all daily activity, including capital levels, PnL, and specific trades made.

Finally, upon completion of the simulation, key performance metrics such as the Total Return on Investment (ROI) and the annualized Sharpe Ratio are calculated from the log to provide a comprehensive summary of the strategy's performance.

4. Experiments

- Experimental setup
- Evaluation metrics
- Results and analysis
- Comparisons with baselines or existing methods

4.1 Objectives/research questions

4.2 data

4.3 measures metrics

4.4 Results

Table 1: Performance Metrics of Different Models

Model	Avg. Accuracy	Avg. MCC
HedgeFundAnalyst, Xu and Cohen [2018]	0.582	0.080
BERT Only	0.48	-0.041
LSTM (Only Stock)	**0.752**	**0.506**
LSTM (Stock + sentiment)	0.679	0.360
LSTM (Stock + emotion)	0.674	0.349
LSTM (Stock + stance)	**0.736**	**0.475**
LSTM (Stock + stance + emotion)	0.648	0.292
LSTM (Stock + stance + sentiment)	0.660	0.325
LSTM (Stock + stance + sentiment + emotion)	0.606	0.215

LSTM (stock) , this contains all data, I think, did you check those stocks with tweets only as well, (first line could be the baseline results as reported in the original paper for reference); stance improves the stocks (assuming we are comparing only stocks with tweets and if correct will be an interesting result; did you consider emotion distribution instead of average;

0.0.1 Result Analysis

Table 2: Top and Worst 5 Performing Tickers by MCC (Stock Data Only)

Ticker	Company	Sector	MCC	Count
Top 5 Performing Tickers				
BP	BP p.l.c.	Basic Materials	0.910	266
GD	General Dynamics Corporation	Industrial Goods	0.866	153
MA	Mastercard Incorporated	Financial	0.780	276
NVS	Novartis AG	Healthcare	0.775	170
D	Dominion Energy Inc.	Utilities	0.768	439
Worst 5 Performing Tickers				
BHP	BHP Billiton Limited	Basic Materials	0.045	195
UNH	United Health Group Inc.	Healthcare	0.045	253
JNJ	Johnson & Johnson	Healthcare	0.162	368
FB	Facebook Inc.	Technology	0.186	475
GE	General Electric Company	Industrial Goods	0.211	413

Table 3: Top and Worst 5 Performing Tickers by MCC (Stock data + Stance)

Ticker	Company	Sector	MCC	Count
Top 5 Performing Tickers				
GD	General Dynamics Corporation	Industrial Goods	0.866	153
SLB	Schlumberger Limited	Basic Materials	0.798	216
MA	Mastercard Incorporated	Financial	0.780	276
VZ	Verizon Communications Inc.	Technology	0.770	361
BA	The Boeing Company	Industrial Goods	0.753	327
Worst 5 Performing Tickers				
UNH	UnitedHealth Group Inc.	Healthcare	-0.199	253
BHP	BHP Billiton Limited	Basic Materials	0.150	195
HD	The Home Depot	Services	0.186	307
AMZN	Amazon.com Inc.	Services	0.188	470
WMT	Wal-Mart Stores Inc.	Services	0.198	358

Table 4: Top and Worst 5 Performing Tickers by MCC (All features)

Ticker	Company	Sector	MCC	Count
Top 5 Performing Tickers				
BP	BP p.l.c.	Basic Materials	0.656	266
SO	The Southern Company	Utilities	0.630	171
MRK	Merck & Co.	Healthcare	0.619	323
UPS	United Parcel Service Inc.	Services	0.596	195
XOM	Exxon Mobil Corporation	Basic Materials	0.579	388
Worst 5 Performing Tickers				
UNH	UnitedHealth Group Inc.	Healthcare	-0.282	253
GD	General Dynamics Corporation	Industrial Goods	-0.189	153
PEP	Pepsico Inc.	Consumer Goods	-0.137	234
DIS	The Walt Disney	Services	-0.077	377
MO	Altria Group Inc.	Consumer Goods	-0.045	237

5. Conclusions

- Summary of findings
- Contributions of the work
- Limitations
- Future work

Bibliography

Miriam Ayer, H. D. Brunk, G. M. Ewing, W. T. Reid, and Edward Silverman. An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics*, 26(4):641–647, 1955. ISSN 00034851. URL http://www.jstor.org/stable/2236377.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003. ISSN 1532-4435.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, 1990. URL https://aclanthology.org/J90-2002/.

Noam Chomsky. *Syntactic Structures*. De Gruyter Mouton, Berlin, Boston, 1957. ISBN 9783112316009. doi: doi:10.1515/9783112316009. URL https://doi.org/10.1515/9783112316009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

A. Fielding. Statistical inference under order restrictions. the theory and application of isotonic regression. *Journal of the Royal Statistical Society. Series A (General)*, 137(1):92–93, 1974. ISSN 00359238. URL http://www.jstor.org/stable/2345150.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. 04 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735.

Ziniu Hu, Weiqing Liu, Jiang Bian, Xuanzhe Liu, and Tie-Yan Liu. Listening to chaotic whispers: A deep learning framework for news-oriented stock trend prediction, 2019. URL https://arxiv.org/abs/1712.02136.

J. L. Kelly. A new interpretation of information rate. *The Bell System Technical Journal*, 35 (4):917–926, 1956. doi: 10.1002/j.1538-7305.1956.tb03809.x.

John F. Kolen and Stefan C. Kremer. *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pages 237–243. 2001. doi: 10.1109/9780470544037.ch14.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks, 2018. URL https://arxiv.org/abs/1703.07015.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200209, February 2021. ISSN 1471-2962. doi: 10.1098/rsta.2020.0209. URL http://dx.doi.org/10.1098/rsta.2020.0209.

Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0262133601.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL https://arxiv.org/abs/1301.3781.

Louis M. Rotando and Edward O. Thorp. The kelly criterion and the stock market. *Am. Math. Monthly*, 99(10):922–931, December 1992. ISSN 0002-9890. doi: 10.2307/2324484. URL https://doi.org/10.2307/2324484.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL https://api.semanticscholar.org/CorpusID:205001834.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014. URL https://arxiv.org/abs/1409.3215.

Edward O. Thorp. Chaper 9 - the kelly criterion in blackjack sports betting, and the stock market*. In S.A. Zenios and W.T. Ziemba, editors, *Handbook of Asset and Liability Management*, pages 385–428. North-Holland, San Diego, 2008. ISBN 978-0-444-53248-0. doi: https://doi.org/10.1016/B978-044453248-0.50015-0. URL https://www.sciencedirect.com/science/article/pii/B9780444532480500150.

A. M. TURING. I.—computing machinery and intelligence. *Mind*, LIX(236):433–460, 10 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL https://doi.org/10.1093/mind/LIX.236.433.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

Jan von Plato. Chapter 75 - a.n. kolmogorov, grundbegriffe der wahrscheinlichkeitsrechnung (1933). In I. Grattan-Guinness, Roger Cooke, Leo Corry, Pierre Crépel, and Niccolo Guicciardini, editors, *Landmark Writings in Western Mathematics 1640-1940*, pages 960–969. Elsevier Science, Amsterdam, 1933. ISBN 978-0-444-50871-3. doi: https://doi.org/10.1016/B978-044450871-3/50156-X. URL https://www.sciencedirect.com/science/article/pii/B978044450871350156X.

Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966. ISSN 0001-0782. doi: 10.1145/365153.365168. URL https://doi.org/10.1145/365153.365168.

Yumo Xu and Shay B. Cohen. Stock movement prediction from tweets and historical prices. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1970–1979, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1183. URL https://aclanthology.org/P18-1183/.

Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, page 694–699, New York, NY, USA,

2002. Association for Computing Machinery. ISBN 158113567X. doi: 10.1145/775047.775151. URL https://doi.org/10.1145/775047.775151.

Appendix A: Supplementary Material

- Code snippets
- Extra tables or figures
- Detailed mathematical derivations

Table 5: Top and Worst 5 Performing Tickers by MCC (Stock data + emotion)

Ticker	Company	Sector	MCC	Count
Top 5 Performing Tickers				
SLB	Schlumberger Limited	Basic Materials	0.798	216
MRK	Merck & Co.	Healthcare	0.702	323
VZ	Verizon Communications Inc.	Technology	0.655	361
XOM	Exxon Mobil Corporation	Basic Materials	0.640	388
MA	Mastercard Incorporated	Financial	0.632	276
Worst 5 Performing Tickers				
MCD	McDonald's Corporation	Services	-0.386	363
UNH	UnitedHealth Group Inc.	Healthcare	-0.277	253
BHP	BHP Billiton Limited	Basic Materials	-0.156	195
MSFT	Microsoft Corporation	Technology	-0.024	459
AMGN	Amgen Inc.	Healthcare	0.000	305

Table 6: Top and Worst 5 Performing Tickers by MCC (Stock data + sentiment)

Ticker	Company	Sector	MCC	Count
Top 5 Performing Tickers				
XOM	Exxon Mobil Corporation	Basic Materials	0.841	388
SLB	Schlumberger Limited	Basic Materials	0.798	216
GD	General Dynamics Corporation	Industrial Goods	0.756	153
BP	BP p.l.c.	Basic Materials	0.709	266
UPS	United Parcel Service Inc.	Services	0.685	195
Worst 5 Performing Tickers				
UNH	UnitedHealth Group Inc.	Healthcare	-0.359	253
AAPL	Apple Inc.	Consumer Goods	-0.028	480
PG	The Procter & Gamble	Consumer Goods	-0.024	300
LMT	Lockheed Martin Corporation	Industrial Goods	0.010	205
WMT	Wal-Mart Stores Inc.	Services	0.033	358

Table 7: Top and Worst 5 Performing Tickers by MCC (Stock data + stance + emotion)

Ticker	Company	Sector	MCC	Count
Top 5 Performing Tickers				
XOM	Exxon Mobil Corporation	Basic Materials	0.746	388
VZ	Verizon Communications Inc.	Technology	0.655	361
UTX	United Technologies Corporation	Industrial Goods	0.633	217
T	AT&T Inc.	Technology	0.623	472
KO	The Coca-Cola Company	Consumer Goods	0.589	356
Worst 5 Performing Tickers				
UNH	UnitedHealth Group Inc.	Healthcare	-0.439	253
CAT	Caterpillar Inc.	Industrial Goods	-0.126	322
AMGN	Amgen Inc.	Healthcare	-0.071	305
GD	General Dynamics Corporation	Industrial Goods	0.000	153
FB	Facebook Inc.	Technology	0.015	475

Table 8: Top and Worst 5 Performing Tickers by MCC (Stock data + stance + sentiment)

Ticker	Company	Sector	MCC	Count
Top 5 Performing Tickers				
UPS	United Parcel Service Inc.	Services	0.809	195
GD	General Dynamics Corporation	Industrial Goods	0.756	153
UTX	United Technologies Corporation	Industrial Goods	0.633	217
MRK	Merck & Co.	Healthcare	0.618	323
SLB	Schlumberger Limited	Basic Materials	0.596	216
Worst 5 Performing Tickers				
UNH	UnitedHealth Group Inc.	Healthcare	-0.519	253
MSFT	Microsoft Corporation	Technology	-0.065	459
FB	Facebook Inc.	Technology	-0.031	475
AAPL	Apple Inc.	Consumer Goods	-0.028	480
WMT	Wal-Mart Stores Inc.	Services	0.017	358