

Uber Reviews Sentiment Analysis Using Forecasting

202418009¹, 202418011², 202418048³

202418009@daiict.ac.in
202418011@daiict.ac.in
202418048@daiict.ac.in

Project Definition

Forecast Sentiment Based on Textual Review Content

Understanding sentiment trends helps businesses across industries—from gauging campaign impact to improving customer experience. Analyzing early feedback offers insights into audience behavior, while forecasting future opinions enables companies to proactively meet customer needs and boost satisfaction. The methods we can use are sentiment analysis with time series forecasting, by doing so we can find the existing patterns in the data and how the user sentiments evolve. Textual sentiment can reveal nuanced user feelings (e.g., "Very convenient" vs. "Good") that raw scores might miss, helping organizations understand specific pain points or strengths.

Abstract

The goal of this research is to Forecast Sentiment Based on Textual Review Content. Objective of this model is to Predict future sentiment trends by analyzing the evolution of textual sentiment in the content column over time. Textual sentiment can reveal nuanced user feelings (e.g., "Very convenient" vs. "Good") that raw scores might miss, helping Uber understand specific pain points or strengths. This paper uses Time series analysis with deep learning methods to predict the sentiment over time.

Literature Survey

Sentiment analysis: Sentiment Analysis: Review of Methods and Models

Sentiment analysis, or opinion mining, is a natural language processing (NLP) subfield that identifies the emotional tone in text, classifying it as positive, negative, or neutral. It has become vital for analyzing user-generated content from social media, e-commerce, and reviews, aiding businesses, researchers, and policymakers in understanding public opinion. This review covers the evolution of sentiment classification methods, from rule-based to deep learning approaches.

1. Rule-Based and Lexicon-Based Methods

Early sentiment analysis used rule-based and lexicon-based methods, relying on predefined word lists to assign senti-

ment scores (e.g., "happy" as positive, "sad" as negative). Techniques like tokenization and negation handling improve accuracy, with VADER being a key example tailored for social media, including emoticons and slang. These methods are simple but struggle with context, sarcasm, and domain-specific terms.

2. Machine Learning-Based Methods

Machine learning (ML) transformed sentiment classification by learning from labeled data. Key algorithms include:

- Naive Bayes: A probabilistic model efficient for short texts like tweets, though less effective with nuanced sentiments.
- Support Vector Machines (SVM): Robust for text classification, especially with TF-IDF, excelling in review analysis.
- Logistic Regression: Simple and interpretable, it performs well on balanced datasets.
- K-Nearest Neighbors (KNN): Effective for small datasets but scales poorly with larger ones.

Feature engineering (e.g., bag-of-words, n-grams) is essential, though ML methods may miss semantic relationships and rely on data quality.

3. Deep Learning-Based Models

Deep learning advanced sentiment classification with neural networks capturing complex patterns. Key models include:

- Recurrent Neural Networks (RNNs): Suitable for sequential text, with LSTMs and GRUs improving long-term dependency capture, ideal for sentence-level analysis.
- Convolutional Neural Networks (CNNs): Adapted from image processing, they efficiently extract features from short texts like posts.
- Transformers and Attention Mechanisms: BERT and similar models use bidirectional context and attention to excel in tasks like aspect-based analysis, outperforming earlier methods on datasets like IMDB reviews.
- Pre-trained Language Models: RoBERTa, GPT, and XLNet, fine-tuned from large corpora, deliver top performance but demand significant computational power.

Time Series Analysis:

1. Traditional Time Series Models

Traditional models lay the groundwork for forecasting and have been adapted for sentiment data:

- Autoregressive Integrated Moving Average (ARIMA): ARIMA forecasts using past data and errors, applied to Weibo sentiment with LDA clustering to model sentiment shifts, effectively capturing trends.
- Exponential Smoothing (ETS): ETS prioritizes recent data, used with news sentiment to forecast stock prices, showing slight accuracy gains tied to immediate behavioral shifts.
- Seasonal Decomposition: Breaks time series into components, applied to Twitter sentiment to detect recurring opinion patterns, aiding behavioral cycle analysis.

These models are efficient and interpretable but falter with non-linear or complex sentiment-behavior dynamics.

2. Machine Learning-Based Models

Machine learning (ML) improves forecasting by handling non-linearities and multivariate data, ideal for sentiment:

- Support Vector Regression (SVR): SVR enhances stock price predictions using Twitter sentiment, modeling investor behavior.
- Random Forests: Forecasts cryptocurrency prices with social media sentiment, revealing collective emotional impacts on markets.
- Gradient Boosting: XGBoost, applied to e-commerce trends in Marketing 5.0, boosts consumer behavior predictions by combining sentiment and transaction data.

3. Deep Learning-Based Models

Deep learning transforms forecasting for sentiment and behavior with complex dependency modeling:

- Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM): LSTMs with news and Twitter sentiment predict stock volatility (e.g., -0.7 correlation) and cryptocurrency trends, capturing investor behavior.
- Convolutional Neural Networks (CNNs): CNNs with news sentiment improve short-term financial forecasts by detecting behavioral shifts like panic selling.
- Transformers: Outperform LSTMs in cryptocurrency forecasting using tweet sentiment, leveraging attention for volatile behavior prediction; BERT enhances aspect-based analysis.
- Generative Adversarial Networks (GANs): GANs generate synthetic data, showing potential for sentiment evolution modeling, though accuracy gains are dataset-specific.

Gaps in Current Research

Sentiment analysis integrated with time series analysis has advanced forecasting in domains like finance and public opinion, yet significant gaps persist. Current methods struggle with contextual nuances like sarcasm, often relying on static lexicons or models that miss evolving language patterns, while temporal dependency modeling remains limited, with

traditional approaches like ARIMA unable to capture non-linear sentiment shifts and even advanced deep learning models lacking intraday granularity. Multimodal integration is underdeveloped, ignoring non-textual cues like images or audio that could enrich time series predictions. Scalability is a challenge, with deep learning's computational demands clashing with real-time needs, and simpler models lacking sophistication.

Links Used for This Research

- A Survey on Sentiment Analysis Methods, Applications, and Challenges - Springer: <https://link.springer.com/article/10.1007/s10462-022-10166-2>
- Sentiment Analysis Algorithms and Applications: A Survey - ScienceDirect: <https://www.sciencedirect.com/science/article/pii/S2090447914000550>
- A Comprehensive Survey on Sentiment Analysis Techniques - IJTech: <https://ijtech.eng.ui.ac.id/article/view/11111>
- A Multi-Method Survey on the Use of Sentiment Analysis in Multivariate Financial Time Series Forecasting - MDPI: <https://www.mdpi.com/2076-3417/13/3/1447>
- Time Series Forecasting Using Transformers with Sentiment Analysis on Financial Data - Archivo Digital UPM: <https://oa.upm.es/78566/>

Dataset Selection

Identification:

Uber Customer Reviews Dataset: <https://www.kaggle.com/datasets/kanchana1990/uber-customer-reviews-dataset-2024>

Contains:

- Username: The reviewer's username is anonymized for privacy.
- Content: Captures the detailed review text where users share their experiences and opinions.
- Score: A numerical rating (1–5) indicating the user's satisfaction level with the product or service.
- ThumbsUpCount: Represents the number of likes or upvotes a review has received, showing how helpful it is to others.
- ReviewCreatedVersion: Specifies the app version when the review was written, helping track feedback across updates.
- At: A timestamp indicating the exact date and time when the review was posted.
- ReplyContent: Contains the developer's response to the review, addressing user concerns or feedback.
- RepliedAt: Records the date and time when the developer replied, useful for analyzing response times.
- AppVersion: The app version string associated with the review, helping track sentiment changes across different releases.

Justification:

This dataset contains over 12,000 customer reviews of the Uber app collected from the Google Play Store. The reviews provide insights into user experiences, including ratings, feedback on services, and developer responses. The data is cleaned and anonymized to ensure privacy compliance and ethical usage. It serves as a valuable resource for sentiment analysis, natural language processing (NLP) with time series.

Preprocessing:

- **Text normalization:** Convert all text to lowercase for consistency and remove extra spaces, special characters, emojis.
- **Feature Extraction:** Extract year, month, and day from At to analyze time-based trends.
- **Remove Unnecessary Data:** Exclude user images as they are not relevant to the analysis.
- **Sentiment Analysis:** Use NLP tools like VADER or TextBlob to extract sentiment from review text.
- **Categorization:** Optionally, classify reviews based on app version for deeper insights.

Methodology

The task of this research is to forecast sentiment based on textual reviews and how they change over time. Sentiment analysis is a potent tool with varied applications across industries. It is helpful for social media and brand monitoring, customer support and feedback analysis, market research, etc. By performing sentiment analysis on initial customer feedback, we can identify a new product's target audience or demographics and evaluate the success of a marketing campaign. The reviews of customers often changes based on the service they get. Forecasting the future sentiments could help organizations to give customers more satisfaction.

1.1 Data: The data collected for this task was taken from Kaggle. The data was appropriate for this task because it was having mixed customer reviews and also it was having time stamps on hour basis, which is the most suitable to forecast sentiments. The dataset contains reviews, timestamp with date, score of review. The method used to extract sentiment for the task was Vader/ Bert.

1.2 Model: The model used for this was GRU. GRUs (Gated Recurrent Units) are a form of recurrent neural network (RNN) that can efficiently model sequences by capturing dependencies between sequential data types. Traditional RNNs struggle with learning long-term dependencies due to issues like vanishing gradients. To address this, researchers proposed more advanced architectures, notably GRU and LSTM. GRU networks simplify the LSTM design by combining the forget and input gates into a single update gate, while still effectively learning temporal patterns in data. They offer comparable performance to LSTM with fewer parameters and faster training times.

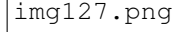


Figure 1: GRU model architecture used for sentiment forecasting.

Model Design and Architecture: GRU

2.1. Explanation of Architecture:

Gated Recurrent Unit (GRU) networks are a type of recurrent neural network (RNN) architecture designed to model sequences and capture dependencies across time. Developed as a simplified alternative to LSTMs, GRUs are particularly effective in tasks where context over time is important, such as language modeling, time series forecasting, and sentiment analysis. Below, we explain the core components and functioning of the GRU architecture.

2.2. Core Components of GRU

A GRU network is composed of units that regulate information flow using gates, allowing it to capture dependencies over sequences without the complexity of LSTMs. The units consist of:

2.2.1. Update Gate The update gate controls how much of the past information needs to be passed along to the future. It combines the functionality of LSTM's forget and input gates:

$$zt = \sigma(Wz \cdot [ht - 1, xt]) \quad (1)$$

2.2.2. Reset Gate The reset gate determines how much of the past information to forget:

$$rt = \sigma(Wr \cdot [ht - 1, xt]) \quad (2)$$

2.2.3. Candidate Activation and Final Memory The candidate activation combines the current input with the reset-modified previous state to produce a candidate hidden state:

$$\tilde{ht} = \tanh(W \cdot [rt * ht - 1, xt]) \quad (3)$$

The final hidden state is then computed as a linear interpolation between the previous state and the candidate:

$$ht = (1 - zt) * ht - 1 + zt * \tilde{ht} \quad (4)$$

2.3. Working Mechanism of GRU

Each GRU unit processes information as follows:

- **Update Gate Decision:** Determines how much of the past state to carry forward.
- **Reset Gate Decision:** Decides how much past information to discard before computing the candidate activation.
- **Candidate Activation:** Computes a candidate memory content based on the reset gate's output.
- **Final State Computation:** Merges the previous state and candidate activation based on the update gate.

2.4. Advantages of GRU

- **Simpler Architecture:** GRUs use fewer gates and parameters than LSTMs, making them faster to train.
- **Efficient for Small Datasets:** Performs well on smaller datasets where overfitting is a risk.
- **Competitive Performance:** Despite their simplicity, GRUs often match or exceed LSTM performance in tasks like sentiment analysis, language modeling, and sequence prediction.

Training Setup/Strategy

The model training followed a structured pipeline including data preprocessing, feature engineering, data scaling, sequence creation, and model training with early stopping to prevent overfitting. The detailed approach is given below:

1. **Feature Engineering:** A 7-day rolling average of the sentiment compound score was created to smooth out short-term fluctuations.
2. **Data Scaling:** StandardScaler was used to normalize the rolling sentiment values, which helps stabilize the learning process.
3. **Data Splitting:** The dataset was split into 80% training and 20% testing sets.
4. **Model Training:** A GRU-based deep learning model was used, consisting of 4 stacked GRU layers (each with 100 units) and Dropout layers (0.3) to reduce overfitting. The model used a linear activation in the final Dense layer to predict sentiment scores.

Pseudocode:

Algorithm 1 Train GRU Model for Sentiment Forecasting

```
1: Input: CSV file with sentiment scores
2: Load data from reviews.csv
3: Compute 7-day rolling mean of compound scores
4: Fill missing values with mean of rolling scores
5: Scale the rolling scores using StandardScaler
6: Create sequences of length 7 for GRU input
7: Split sequences into 80% train and 20% test sets
8: Initialize Sequential GRU model
9: for 4 GRU layers do
10:   Add GRU layer with 100 units
11:   if not last layer then
12:     Set return=True
13:   else
14:     Set return=False
15:   end if
16:   Add Dropout layer with rate 0.3
17: end for
18: Add final Dense output layer with 1 unit (linear activation)
19: Compile model with Adam optimizer and MSE loss
20: Setup early stopping (monitor="loss", patience=10)
21: Train model for 100 epochs, batch size 64
```

Loss function: Mean Squared Error

The mean squared error (MSE) is a crucial concept in the field of statistics and machine learning. It serves as a measure to evaluate the accuracy of a model by quantifying the difference between observed and predicted values. Mean Squared Error is a metric used to assess the quality of a predictive model. It is the average of the squares of the errors, where the error is the difference between the actual and predicted values. The formula for MSE is:

$$\text{MSE}_{\text{test}} = \frac{1}{m} \left(\hat{y}_i^{(\text{test})} - y_i^{(\text{test})} \right)^2 \quad (5)$$

MSE is widely used because it penalizes larger errors more heavily due to the squaring of the differences. This means that it effectively highlights models that make significant prediction errors. Here are some reasons MSE is favoured:

- 1. **Sensitivity:** MSE is sensitive to large errors, making it useful in applications where bigger mistakes are costly.
 - 2. **Differentiability:** The function is differentiable, which is helpful in optimization and helps in gradient-based learning algorithms.
 - 3. **Interpretability:** Provides a clear measure of model performance, with lower MSE values indicating better model precision.
- 4.2. **Alternatives to Mean Squared Error :-** Depending on the application, other metrics may be more suitable:
- * **Mean absolute error (MAE):** Measures the average magnitude of errors, providing a more interpretable metric when units are important.
 - * **Root Mean Squared Error (RMSE):** Takes the square root of MSE, bringing the error metric back to the same unit as the output variable.
 - * **R-squared:** Provides a relative measure of how well the model explains the variability of the outcome.

The end-to-end architecture of the model and design of BERT

The implementation of an end-to-end architecture for a sentiment analysis model using the BERT (Bidirectional Encoder Representations from Transformers) model. It processes Uber reviews to classify them into three categories: negative, neutral, and positive. Below is a breakdown of the implementation.

5.1.1 **Configuration Setup** The configuration section defines the essential parameters needed for the model:

- **DATA:** Path to the CSV file containing the Uber reviews dataset.
- **MODEL:** Directory where the trained model and tokenizer will be saved for future use.
- **BATCH:** Number of samples processed in one iteration during training (affects training speed and memory).
- **MAX:** Maximum token length allowed for each input text after BERT tokenization.

- **NUM:** Number of complete passes through the training dataset.
- **LEARNING:** Step size the optimizer uses to update weights during training.
- **RANDOM:** Fixed seed value to ensure reproducible results across training runs.

5.1.2. Dataset Architecture

We design a custom `ReviewDataset` class to efficiently prepare our data for training and evaluation with BERT. This class takes in raw text data and corresponding labels, and uses the `BertTokenizer` to tokenize the input texts, converting them into a format suitable for the BERT model. Specifically, it encodes each input into tensors including input-ids, attention-mask, and labels, which are required by the model during training. Once the data is processed and properly formatted, this dataset class can be seamlessly used with a `DataLoader` to handle batching, shuffling, and feeding the data into the model during training or inference.

5.1.3. Data Preprocessing

We start by reading the `uber.csv` file to load the dataset containing text reviews and their corresponding sentiment labels. To prepare the data for machine learning, we map the sentiment strings to numerical labels using a dictionary: negative: 0, neutral: 1, positive: 2. This numerical format is essential for training classification models. Next, we filter out any invalid or missing data entries to ensure data quality and consistency. Once the dataset is cleaned and labeled, we split it into training and test sets using stratification. Stratification ensures that the proportion of each sentiment class is preserved in both sets, which helps maintain balanced performance across all classes during model evaluation.

5.1.4. Tokenization & BERT Model Setup

We begin by loading the pre-trained `bert-base-uncased` tokenizer and model using `BertTokenizer.from_pretrained("bert-base-uncased")` and `BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=3)`. The model is configured for a three-class classification task, corresponding to negative, neutral, and positive sentiments. To fine-tune the BERT model effectively, we use the AdamW optimizer with a low learning rate of $2e-5$, which is a standard and stable choice for adapting pre-trained transformers. Next, we prepare our datasets by tokenizing the review texts using the same tokenizer and creating instances of our custom `review Dataset` class for both training and test sets. These datasets are built from the content and label columns of the respective `DataFrames` and are capped with a maximum token length (`MAXLENGTH`). Finally, we use `DataLoader` to batch and shuffle the training data and to prepare the test data for evaluation, making the data pipeline ready for model training.

5.1.5. Model Training Loop

The model is trained over several epochs, where in each epoch it loops through batches of data from the training set. For every batch, the input is passed through the model to compute predictions, and the loss is calculated by comparing these predictions with the actual labels. This loss is then backpropagated to update the model's weights using the optimizer, gradually improving the model's performance. Throughout the epoch, the training loss for each batch is accumulated, and the average training loss is tracked and reported at the end of the epoch to monitor the model's learning progress.

5.1.6. Evaluation

After training, the model is set to evaluation mode to disable dropout and other training-specific behaviors, ensuring consistent predictions. It then iterates through the test data without updating weights, using the trained model to predict sentiment labels for each batch. During this process, it collects both the predicted labels and the true labels. Once all predictions are gathered, a classification report is generated, which provides detailed performance metrics including precision, recall, and F1-score for each class—negative, neutral, and positive—giving a clear picture of how well the model performs across different sentiment categories.

5.1.7. Saving the Model

Once training and evaluation are complete, the trained model and tokenizer are saved to a local directory named `'bert-sentiment-model'`. This is done to preserve the model's learned parameters and the tokenizer configuration, allowing for future use without retraining. By saving them with the `'save_pretrained()'` method, they can be easily reloaded later using `'from_pretrained()'` for deployment or inference tasks, making the sentiment analysis model readily accessible for real-world applications or further experimentation.

Result of the BERT model

Label	Precision	Recall	F1-score	Support
Negative	0.91	0.95	0.93	332
Neutral	0.99	0.94	0.96	506
Positive	0.98	0.99	0.98	1562
Accuracy			0.97	2400
Macro Avg	0.96	0.96	0.96	2400
Weighted Avg	0.97	0.97	0.97	2400

Table 1: Test Classification Report

Test Classification Report This is a classification report summarizing the performance of a sentiment analysis model on a test dataset with three classes: negative, neutral, and positive. Here's a breakdown of what each metric means and how your model performed:

Precision measures how accurate the model's positive predictions are—it tells us, out of all the times the model predicted a certain class, how often it was actually correct. Recall, on the other hand, measures how well the model captures all real instances of a class—it shows how many actual examples of that class the model managed to identify. The F1-score combines both precision and recall into a single metric using their harmonic mean, making it especially useful when the class distribution is imbalanced. Support simply refers to how many true samples of each class are present in the dataset, helping provide context for the other metrics.

Interpretation: The model shows its strongest performance on positive reviews, which also happen to be the most frequent class in the dataset—likely giving the model more examples to learn from. Its performance on negative reviews is slightly weaker, particularly in terms of precision, meaning it sometimes incorrectly labels other sentiments as negative (false positives). Interestingly, the neutral class is handled with high accuracy, showing near-perfect precision, which indicates the model is very good at correctly identifying neutral reviews without confusing them with others.

5.2. The End-to-End Architecture of the Model and Design of GRU

The implementation of a complete end-to-end architecture for time series forecasting of sentiment evolution using a GRU-based neural network, applied to Uber review data. Here's a structured explanation of the implementation and design of your model architecture:

5.2.1. Data Loading & Sentiment Annotation

We start with a raw CSV file that contains Uber reviews along with their timestamps. To analyze sentiment, you use a pre-trained transformer model, specifically 'cardiffnlp/twitter-roberta-base-sentiment', which processes each review and assigns a sentiment label - Positive, Negative, or Neutral - together with confidence scores for each class. These confidence scores are then extracted and converted into three separate numerical features for each review: 'scorepositive', 'scorenegative', and 'scoreneutral'. This enriches the dataset with structured sentiment information, making it suitable for further analysis or modeling.

5.2.2. Daily Aggregation

The data is grouped by day to compute daily averages of the sentiment scores, resulting in a structured time series. For each day, the average values of `score_positive`, `score_negative`, and `score_neutral` are calculated, creating a snapshot of sentiment trends over time. The outcome is a multivariate time series where each time point represents a day, and the three sentiment scores for that day serve as the variables. This format allows for analyzing sentiment trends over time and can be used for forecasting or detecting shifts in public opinion or sentiment.

5.2.3. Sequence Preparation for Supervised Learning

You define a sliding window approach to create input-output pairs for forecasting sentiment scores over time. The sequence length is set to 15, meaning each input will consist of sentiment data from the past 15 days. The forecast length is set to 7, indicating that the model will predict sentiment scores for the next 7 days. To ensure stable training and improve model performance, the data is scaled using a `MinMaxScaler`, which normalizes the sentiment scores to a consistent range. The input shape is `(batch_size, 15, 3)`, where each sample contains sentiment scores for 15 days, with 3 features per day (positive, negative, and neutral scores). The output shape is `(batch_size, 7 × 3)`, which reshapes the forecasted data to match the target format, where each sample predicts the sentiment scores for the next 7 days, with 3 features per day. This setup allows the model to learn temporal patterns and make accurate future predictions.

5.2.4. GRU Model Design

The GRU model is implemented using PyTorch's `nn.Module`. It consists of a 2-layer GRU network with a dropout layer (rate 0.3) for regularization. The model takes sequences of sentiment data as input, processes them through the GRU layer, and uses the final hidden state to predict sentiment scores for a 7-day forecast. A fully connected layer maps the GRU output to the forecasted values. The model is trained with Mean Squared Error (MSE) loss and optimized using the Adam optimizer with a learning rate of 0.001. The architecture is designed for predicting future sentiment trends based on past data.

5.2.5. Training with Validation

We train the model using Mean Squared Error (MSE) Loss and the Adam optimizer, which helps the model minimize the error between its predictions and the true sentiment values. Throughout the training process, we track and plot both training and validation loss for each epoch, allowing us to monitor potential overfitting and ensure the model generalizes well to unseen data. We utilize a 'DataLoader' for mini-batch training, which automatically handles shuffling of the data during each epoch, promoting better generalization and reducing the risk of the model memorizing the training set.

5.2.6. Forecasting Sentiment Evolution

After training, the model uses the most recent 15 days of sentiment data as input to predict the sentiment scores for the next 7 days. Once the model generates its predictions, the output, which is scaled, is rescaled back to the original score range using the `inverse_transform` method of the `MinMaxScaler`. This ensures that the predicted values are in the same range as the original sentiment scores. Finally, these rescaled predicted scores are mapped back to sentiment labels—Positive, Negative, or Neutral—based on predefined thresholds, allowing you to interpret the

model’s forecasts as actionable sentiment labels for the next week.

6. Experiment Set-up:

To evaluate the performance of the GRU-based sentiment forecasting model, the following experimental configuration was used:

6.1. Model Architecture Configuration

The model is designed to observe patterns over a fixed time window and forecast future sentiment trends. It uses an input sequence length of 15 time steps (**SEQ_LENGTH = 15**), allowing it to analyze recent sentiment patterns. The forecast horizon is set to 7 days (**FORECAST_DAYS = 7**), meaning the model predicts sentiment scores for the next week. Each time step includes three features — positive, neutral, and negative sentiment scores — making the input dimensionality **input_dim = 3**. The GRU-based architecture consists of two layers (**num_layers = 2**), each with a hidden size of 64 units (**hidden_dim = 64**), enabling it to capture complex temporal dependencies. A dropout rate of 0.3 (**dropout = 0.3**) is applied for regularization to mitigate overfitting during training.

6.2. Training Configuration

- **Learning Rate (lr):** 0.001
Used with the Adam optimizer for stable convergence.
- **Epochs (EPOCHS):** 25
Total number of training iterations over the dataset.
- **Batch Size (BATCH):** 16
Mini-batch size used during training.

6.3. Data Splitting & Loading

- **Validation Split (test):** 20%
The dataset is split into 80% training and 20% validation sets.
- **Shuffling:** Enabled during training for robustness, although not typically recommended for time series unless explicitly modeling it as shuffled.

6.4. Clarification on Parameters

- input is derived from the data (three sentiment scores) and not a tunable hyperparameter.
- device=0 is related to GPU usage and not part of training logic.
- shuffle=False in DataLoader aids general training but may not align with strict temporal modeling; its impact was not a focus of this experiment.

7. Choosing Proper BaseModel

7.1. For Sentiment Analysis: Vader and TextBlob

We chose Vader and TextBlob as the base models because their losses are higher as compared to others.

Classification Reports for VADER and TextBlob

Label	Precision	Recall	F1-score	Support
Negative	0.98	0.96	0.97	341
Neutral	0.97	1.00	0.98	492
Positive	1.00	0.99	0.99	1566
Accuracy			0.99	2399
Macro Avg	0.98	0.98	0.98	2399
Weighted Avg	0.99	0.99	0.99	2399

Table 2: VADER Classification Report

VADER Classification Report

Label	Precision	Recall	F1-score	Support
Negative	0.86	0.50	0.63	341
Neutral	0.51	0.82	0.63	492
Positive	0.93	0.83	0.88	1566
Accuracy			0.78	2399
Macro Avg	0.77	0.72	0.71	2399
Weighted Avg	0.83	0.78	0.79	2399

Table 3: TextBlob Classification Report

TextBlob Classification Report

7.2. For Forecasting: LSTM

We chose LSTM as the base model because its losses are higher as compared to others.

MSE	0.0001
MAE	0.0074

8. Compare the model’s performance GRU with SOTA architectures

Validation Metrics Comparison

Metric	GRU	Transformer
MSE	0.0002	0.0005
MAE	0.0106	0.0093

Mean Squared Error (MSE):

GRU wins with a lower MSE (0.0002 vs 0.0005). This indicates that the GRU has smaller squared deviations, i.e., it is better at penalizing large errors.

Mean Absolute Error (MAE):

Transformer wins with a lower MAE (0.0093 vs 0.0106). This indicates that the Transformer has an overall lower average error, possibly making it more stable for small fluctuations.