Sentiment Classification for Tweets: A Comparative Study of Classical and Deep Learning Methods

u2164966
Department of Computer Science
University of Warwick

***Abstract*—This report presents a comparative study on sentiment classification for tweets, following the SemEval Task 4 guidelines. We implement and evaluate several classifiers, including two classical machine learning models and an LSTM-based deep learning approach, with an exploration of advanced methods such as BERT. Our experiments demonstrate that transformer-based models outperform traditional approaches, achieving up to 74.42% accuracy across multiple test sets. The results highlight the importance of appropriate preprocessing techniques and the value of contextual embeddings for capturing the nuanced sentiment in short, informal text.**

***Index Terms*—Sentiment Classification, Twitter, Bag-of-Words, TF-IDF, GloVe, LSTM, BERT, SVM, Naïve Bayes**

I. Introduction and Objectives

Social media platforms, particularly Twitter, pose significant challenges for sentiment analysis due to the brevity and informal nature of tweets [1]. This report addresses the task of classifying tweets into positive, negative, and neutral sentiments, following the guidelines of SemEval Task 4 [2]. The focus is on developing robust classifiers that not only capture the nuances of short texts but also generalize well across multiple test sets.

The primary goal of this work is to develop a sentiment classifier that can accurately categorize tweets as positive, negative, or neutral. Given the unique linguistic features and informal style of tweets, this task requires tailored preprocessing and feature engineering approaches to effectively extract meaningful patterns from the data [3].

Adhering to the challenges set by SemEval Task 4, this project underscores the importance of addressing the inherent difficulties in processing short, noisy texts. Successfully tackling this problem has broader implications for real-time social media analysis and automated monitoring systems, where quick and accurate sentiment evaluation is crucial [4].

A. Approach Overview

Our approach combines both classical machine learning and modern deep learning techniques:

- **Classical Techniques:**
  - Naïve Bayes with Bag-of-Words (BoW) [5]
  - Maximum Entropy (MaxEnt) [6]
- **Deep Learning Methods:**
  - LSTM Model [7]
  - BERT-based Model [8]

This multi-faceted approach allows for a comprehensive evaluation of different methods to effectively classify tweet sentiments.

II. Data and Preprocessing

A. Datasets

For this sentiment classification task, we use the SemEval 2017 Task 4 Twitter sentiment analysis dataset [2]. The dataset consists of multiple files across different stages:

- Training set: `twitter-training-data.txt`
- Development set: `twitter-dev-data.txt`
- Test sets:
  - `twitter-test1.txt`
  - `twitter-test2.txt`
  - `twitter-test3.txt`

Each file is formatted as a tab-separated values (TSV) file with three columns:

- **tweet-id**: A unique identifier for each tweet
- **sentiment**: The sentiment label (positive, negative, or neutral)
- **tweet-text**: The actual text of the tweet

The dataset is designed to facilitate a comprehensive sentiment analysis task, with multiple test sets allowing for robust evaluation of the classifier's generalizability. The sentiment labels are distributed across three classes: positive, negative, and neutral.

B. Preprocessing Pipeline

The preprocessing pipeline plays a crucial role in cleaning and normalizing the raw Twitter data before it is fed into the sentiment classifiers [9]. This section provides a detailed description of each step in our code, explaining both the motivation and functionality behind them. The pipeline is encapsulated in a `TwitterPreprocessor` class, which offers a configurable, end-to-end procedure for text processing on social media data.

a) Class Overview: `TwitterPreprocessor` is designed with several flags (e.g., `remove_urls`, `remove_mentions`, `replace_emojis`, etc.) that make it easy to enable or disable specific preprocessing steps. This flexibility allows for quick experimentation with different approaches to cleaning and transforming the data [10]. For example, if one wants to preserve numbers or user mentions, that can be done by toggling a single parameter.

b) HTML Decoding & Unicode Normalization:

- **HTML Decoding:** We first decode any HTML entities (e.g., `&amp;` to `&`) using `html.unescape`.
- **Unicode Normalization:** We then normalize the text to its canonical form, using the *NFKD* scheme via the `unicodedata` library [11]. This helps ensure accented characters (e.g., "é") are represented consistently. Additionally, control characters (e.g., newlines, tabs) are replaced with spaces.

c) Replacing Common Entities (URLs, Emails, Mentions, Numbers):

- **URLs:** If `remove_urls` is true, any URL (such as `http://...`) gets replaced with `<URL>`.
- **Emails:** Similarly, email addresses are replaced by `<EMAIL>`.
- **Mentions:** If `remove_mentions` is true, user mentions (e.g., `@username`) are replaced by `<USER>`.
- **Numbers:** If `remove_numbers` is set, numerical values are replaced with `<NUMBER>`, which can be useful for normalizing numeric information.

d) Handling #Hashtags: When `handle_hashtags` is enabled, the pipeline identifies tokens that start with the "#" character (e.g., `#DataScience`) and splits them using a camel-case or underscore approach [12]. For example, `#DataScience` transforms into `<HASHTAG> data science </HASHTAG>`. This allows the classifier to pick up on meaningful subwords from hashtags, which can often carry specific sentiment or topic signals.

e) Emoticons & Emojis:

- **Text-based emoticons:** Before we handle typical text replacements, the pipeline checks for common emoticon patterns (e.g., `:-)`, `;)`, `:(`, etc.) and replaces them with semantic placeholders like `<SMILE>` or `<SAD>` [1].
- **Unicode emojis:** For modern Unicode emojis (e.g., ,), we use a mapping dictionary to replace each detected emoji with a sentiment-relevant placeholder such as `<SMILE>`, `<SAD>`, `<ANGRY>`, and so on [13]. This step is important because emojis often convey sentiment strongly and can significantly impact classification.

f) Emphasis and Ellipsis:

- **Emphasis:** Text emphasized by asterisks (e.g., `*word*`) is wrapped into an `<EMPHASIS>` tag. This helps us capture the intention behind certain strongly emphasized words.
- **Ellipses:** Sequences of 3 or more periods (`...`) are replaced with an `<ELLIPSIS>` tag to differentiate them from accidental repeated punctuation or normal usage [14].

g) Contractions and Possessives: All English contractions (e.g., `can't`, `n't`, `'re`) are expanded to their full form using a set of predefined rules (e.g., `can't` $\rightarrow$ `cannot`) [15]. Possessives like `user's` remain valid possessive forms rather than being misconstrued as contractions.

h) Text Features (All-caps, Elongations, Punctuation):

- **All-caps words:** Identifies fully capitalized words (e.g., `LOVE`) and appends `<ALLCAPS>` to highlight them, then converts the word to lowercase [16]. This step preserves the notion of emphasis or shouting without losing lexical uniformity.
- **Elongations:** Multiple repeated letters (e.g., `coooool`) can be a strong informal sentiment marker. We normalize such words and append a tag `<ELONG>` to note that it was elongated.
- **Repeated punctuation:** Repeated punctuation (e.g., `!!!`, `??`) is replaced by `! <REPEAT>` or `? <REPEAT>` to preserve emotional emphasis [3].

i) Negation Handling: When `handle_negations` is true, the pipeline detects negation words (e.g., `no, not, never, n't`) and prepends `NEG\_` to the next one or two content words, unless they are in a set of function words (e.g., `the, and, it`) or protected tags (like `<URL>`) [17]. For example, `"I do not like this"` becomes `"I do not NEG_like NEG_this"`. This step helps the classifier pick up negative polarity even if negation is present, preventing misclassification of an otherwise positive word.

j) Spacing and Casing: Finally, multiple consecutive spaces are collapsed into a single space, and leading/trailing whitespace is stripped. At the very end, text is typically converted to lowercase. Lowercasing standardizes the text representation and helps reduce sparsity in token-based features [18].

k) Tokenization: After the text has gone through all the above steps, a simple whitespace-based `tokenize` function splits the resulting text into tokens. This approach works well here because the main complexities of text handling—such as emoticons, punctuation, hashtags, etc.—have already been handled.

III. Model Architectures & Methods

Having established our preprocessing (Section II-B), we now describe the models used to classify the sentiment of tweets. The pipeline begins with the `TwitterPreprocessor` code, which outputs cleaned and tokenized text. These tokens (and the associated features) are then passed into one of several classification approaches.

A. Classical Models

- **Naïve Bayes (NB):** We implement a Naïve Bayes classifier by coupling a `CountVectorizer` or `TfidfVectorizer` with the `MultinomialNB` model [19]. Each tweet is first cleaned through the `TwitterPreprocessor`—removing URLs, mentions, and numbers, while inserting semantic tags (e.g., `<SMILE>`, `<ALLCAPS>`) to preserve sentiment cues. Next, our vectorizer constructs either a bag-of-words (BoW) or TF-IDF representation of the preprocessed text:
  - ○ *BoW*: Uses `CountVectorizer` to convert tokens into count-based features.

- *TF-IDF*: Uses `TfidfVectorizer` to down-weight terms that appear in many documents [20].

We apply hyperparameters such as `min_df=5` and `max_df=0.7` to ignore overly rare and overly frequent tokens, respectively, and include unigrams and bigrams (`ngram_range=(1,2)`) for richer context. Finally, the `MultinomialNB` classifier employs Laplace smoothing (`alpha=1.0`) to handle zero-frequency issues. This pipeline is both computationally efficient and interpretable, making NB a strong baseline for text classification.

- **Maximum Entropy (MaxEnt):** Our MaxEnt approach leverages GloVe embeddings for feature representation [6]:

(i) **GloVe Embeddings:** For each preprocessed tweet, we compute the average of 100-dimensional GloVe embeddings across all tokens [21]. This step encodes distributional semantics, capturing contextual similarity and sentiment cues that simple word counts may overlook.

A `LogisticRegression` classifier (i.e., MaxEnt) then directly models $P(\text{label} \mid \text{features})$, iteratively adjusting the coefficients to maximize the conditional likelihood. By not assuming feature independence, the model can better integrate distributional semantics and any additional tags (e.g., `<SMILE>`, `NEG_word`) introduced during preprocessing.

- **Hyperparameter Optimization:** We fine-tune each model's hyperparameters using a validation split or cross-validation. Specifically:
  - *Naïve Bayes*: We vary the smoothing parameter (`alpha`) and compare BoW vs. TF-IDF vectorizers.
  - *MaxEnt*: We explore multiple `LogisticRegression` solvers (`lbfgs`, `liblinear`), different penalties (`l1` vs. `l2`), and various regularization strengths (`C` $\in \{0.01, 0.1, 1, 10, 100\}$), optionally testing `class_weight = balanced` to mitigate label imbalance [38]. These parameters are systematically probed via grid search over 5-fold cross-validation, and the best combination is selected based on accuracy or macroaveraged F1 score.

This more comprehensive search allows us to compare penalized likelihood approaches (e.g., L1 vs. L2) and solver behaviors (one-vs-rest vs. multinomial classification) while capturing embedding-based semantics. Ultimately, the model chosen from this grid search tends to outperform simpler token-based MaxEnt baselines, demonstrating the benefit of using GloVe in the feature space.

B. Deep Learning Model (LSTM)

In addition to classical linear models, we developed an LSTM-based neural network in PyTorch that integrates both learned word embeddings and hand-engineered sentiment features. Our architecture is composed of multiple stages:

- **Embedding Layer (GloVe Initialization):** We first build a vocabulary using a `Tokenizer` (from `tensorflow.keras.preprocessing`), which converts each preprocessed tweet into a sequence of integer IDs. Next, we create an `embedding_matrix` by looking up 100-dimensional GloVe vectors for each word in our vocabulary [21]. This matrix is then loaded into a PyTorch `Embedding` layer. In many cases, we keep these GloVe weights frozen (`weight.requires_grad = False`) so that the model's capacity can focus on other parameters, though we may optionally allow fine-tuning if performance gains are observed.
- **Engineered Feature Branch:** Besides word embeddings, we incorporate additional signal from the Opinion Lexicon counts of positive and negative words [22]. These features are fed into a small fully connected sub-network (`feat_fc`) to learn a higher-level representation, and we introduce a trainable gating parameter (`feat_gate`) to control how strongly these hand-crafted features contribute to final predictions.
- **Bidirectional LSTM with Attention:** Each sequence of embedded tokens is passed into a two-layer bidirectional LSTM (`num_layers=2`) with a hidden size of 128 [7]. We then apply an attention mechanism to produce a weighted sum (*context vector*) over LSTM outputs [23]:
  - *Global Pooling:* We concatenate the attention-based context with average- and max-pooled representations of the LSTM output, providing the model with three complementary views of the tweet's hidden states.
  - *Dropout:* A dropout rate of 0.5 (`dropout=0.5`) is used to regularize the network and prevent overfitting [24], especially given the limited size of many tweet datasets.

  The final LSTM branch output is thus a fixed-dimensional feature vector capturing sequential context and attention-based weighting of tokens.
- **Merging and Classification:** The learned representation from the LSTM branch is concatenated with the transformed feature vector from the engineered branch. This combined vector is passed to a linear classification layer (`fc_combined`) that outputs logits for our three sentiment classes (positive, negative, neutral). A softmax (or log-softmax) activation is applied to convert logits to probabilities.
- **Training Details:** We use the `Adam` optimizer (`lr=0.001`) and cross-entropy loss [25], typically running for 5–10 epochs with batch sizes of 16–64. During each epoch, we track training loss and accuracy, and evaluate on a validation (dev) set to guide early stopping. We truncate or pad each tweet to `max_sequence_length=128` tokens, balancing memory constraints with sufficient context. In practice, this architecture converges reliably and achieves strong performance across multiple test splits.
- **Integration with Preprocessing:** As with our classical models, all tweets are cleaned by the `TwitterPreprocessor` to remove URLs and mentions, handle negations, convert elongated words, and insert sentiment-specific tags (e.g., `<SMILE>`). This step is crucial in mitigating social-media noise and ensuring the embedded inputs reflect consistent lexical units.

By combining pretrained GloVe embeddings, an attentionequipped bidirectional LSTM, and additional lexicon-based features from the opinion lexicon, our final architecture is capable of capturing not only local context from tokens but also higher-level sentiment cues. This approach consistently improves classification accuracy and F1 scores compared to simpler embedding-only LSTM models, demonstrating the value of multi-branch feature integration in sentiment analysis [26].

IV. Advanced Models

In addition to our classical (Naïve Bayes, MaxEnt) and LSTM-based models, we experimented with several advanced approaches to push the boundaries of sentiment classification performance on noisy Twitter data. Specifically, we explored:

A. Fine-Tuned BERT

a) Model and Tokenization.: We leveraged the `bert-base-uncased` checkpoint from Hugging Face Transformers [27], pairing it with the `BertTokenizer` for subword tokenization. Tweets were truncated or padded to a maximum sequence length (e.g., `max_sequence_length=128`) to accommodate BERT's constraints, while still retaining sufficient context.

b) Training Procedure.: After encoding each tweet into `input_ids` and `attention_mask`, we fine-tuned `BertForSequenceClassification` for 3–5 epochs using the `AdamW` optimizer (`lr=2e-5`) and a linear learning rate scheduler [28]. Typical batch sizes ranged from 8 to 16 due to memory constraints. During each epoch, we monitored validation loss on a dev set to identify signs of overfitting. Once training converged, we evaluated the final model on multiple test splits, computing accuracy, macroaveraged precision, recall, and F1 score.

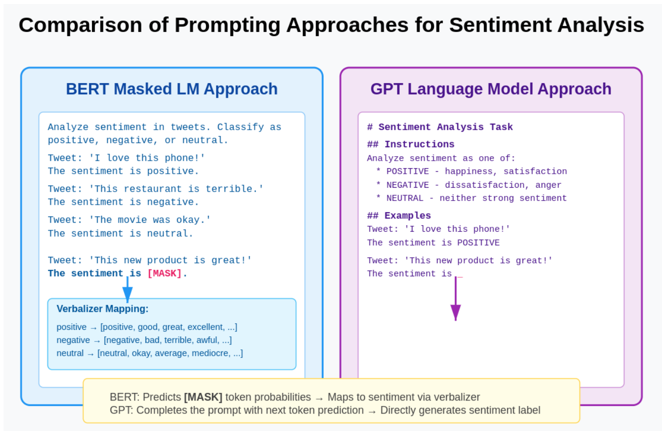
c) Results.: Our experiments showed that fine-tuned BERT achieved strong performance, often surpassing both classical methods and the LSTM model. For example, we observed test accuracies in the 72–75% range on certain splits (Section V), demonstrating BERT's ability to learn rich contextual representations of informal Twitter text [29].

B. Prompting-Based Approaches

We investigated two distinct prompting strategies for sentiment analysis, each leveraging different capabilities of pre-trained language models:

1) Masked BERT Prompting: This approach uses `BertForMaskedLM` to predict sentiment via a *cloze-style* task, where the model fills in a `[MASK]` token in a template [30]. Instead of training a dedicated classifier head, we:

(i) Constructed a template with few-shot examples of labeled tweets (e.g., "`Tweet: 'I love this phone!' The sentiment is positive.`"),
(ii) Inserted the target tweet into the same template with a `[MASK]` token (e.g., "`Tweet: 'This new product is great!' The sentiment is [MASK]`"),
(iii) Applied a verbalizer mapping that connects `[MASK]` probabilities to sentiment labels (e.g., `positive` → {`positive`, `good`, `great`, ...}).


Comparison of Prompting Approaches for Sentiment Analysis
BERT Masked LM Approach
GPT Language Model Approach

```
Analyze sentiment in tweets. Classify as
positive, negative, or neutral.
Tweet: 'I love this phone!'
The sentiment is positive.
Tweet: 'This restaurant is terrible.'
The sentiment is negative.
Tweet: 'The movie was okay.'
The sentiment is neutral.

Tweet: 'This new product is great!'
The sentiment is [MASK].
```

Verbalizer Mapping:
positive → [positive, good, great, excellent, ...]
negative → [negative, bad, terrible, awful, ...]
neutral → [neutral, okay, average, mediocre, ...]

```
# Sentiment Analysis Task
## Instructions
Analyze sentiment as one of:
  * POSITIVE - happiness, satisfaction
  * NEGATIVE - dissatisfaction, anger
  * NEUTRAL - neither strong sentiment
## Examples
Tweet: 'I love this phone!'
The sentiment is POSITIVE
Tweet: 'This new product is great!'
The sentiment is _
```

BERT: Predicts **[MASK]** token probabilities → Maps to sentiment via verbalizer
GPT: Completes the prompt with next token prediction → Directly generates sentiment label

Fig. 1: Comparison of BERT Masked LM and GPT prompting approaches for sentiment analysis. The BERT approach predicts probabilities for the masked token and maps them to sentiment labels via a verbalizer, while GPT directly generates sentiment labels through next-token prediction.

a) Challenges and Observations.: While conceptually appealing—since it avoids training a separate classification head—our experiments revealed that masked BERT prompting can be highly sensitive to prompt design and verbalizer choices [31]. The performance varied significantly based on subtle changes to template structure and token mappings.

2) GPT-Based Prompting: The second prompting strategy employed GPT-2 (and variants like `gpt2-medium`) in a few-shot setting [32]. Unlike BERT's masked prediction:

(i) We structured a prompt with clear instructions ("`# Sentiment Analysis Task`", "`## Instructions`") and sentiment definitions,
(ii) Included labeled examples demonstrating the expected format (e.g., "`Tweet: 'I love this phone!' The sentiment is POSITIVE`"),
(iii) Appended a new test case using the same format but leaving the sentiment label incomplete for GPT to generate.

a) Implementation Details.: We tokenized the complete prompt with `GPT2Tokenizer`, passed it to a `GPT2LMHeadModel` in inference mode, and let the model directly generate the sentiment label through next-token prediction. This approach differs fundamentally from BERT's masked prediction as it leverages GPT's autoregressive capabilities for direct label generation.

b) Results and Stability.: GPT-based prompting demonstrated flexibility but showed similar vulnerabilities to prompt engineering as BERT's masked approach. The explicit instruction format with sentiment definitions helped guide predictions, but performance still varied based on example selection and prompt structure.

C. *Discussion and Comparison*

Our experiments with these advanced approaches revealed several key insights:

- **Fine-Tuned BERT:** Achieved the best overall results, yielding high F1 scores across multiple test sets. Training a dedicated classification head proves more robust than prompting for Twitter sentiment analysis [35].
- **Masked BERT Prompting:** Offers a lightweight alternative without separate training, but requires careful verbalizer design and performs less consistently across diverse tweets.
- **GPT Prompting:** Provides a more intuitive interface through natural language instructions, but its generative nature sometimes leads to misclassifications on ambiguous or sarcastic content.

As illustrated in Figure 1, the two prompting approaches represent fundamentally different mechanisms—BERT predicting `[MASK]` tokens mapped through verbalizers versus GPT directly generating sentiment labels. While both offer training-free alternatives, our experiments confirm that *fine-tuning BERT end-to-end for classification* remains the most reliable and highest-performing approach for Twitter sentiment analysis.

V. Empirical Results

In this section, we present and analyze the performance of our sentiment classifiers across four main approaches:

1) **Classical Models:** Naïve Bayes and Maximum Entropy (MaxEnt)
2) **Basic LSTM Model**
3) **Advanced LSTM Model with Attention**
4) **BERT-Based Model**

We evaluate each approach on a common set of splits: a `training` set, a `dev` (development) set, and three separate `test` sets (`twitter-test1.txt`, `twitter-test2.txt`, `twitter-test3.txt`). The main metrics reported are *accuracy* (overall correctness), and macroaveraged *precision*, *recall*, and *F1 score*, which treat all sentiment classes (positive, negative, neutral) equally regardless of their frequency [36].

A. *Classical Models (Naïve Bayes & MaxEnt)*

We began with two classical baselines: **Naïve Bayes (NB)** and **Maximum Entropy (MaxEnt)**. Both rely on the cleaned, tokenized tweets from our `TwitterPreprocessor`. Naïve Bayes uses a bag-of-words (or TF-IDF) representation with hyperparameter search on the smoothing constant `alpha`. MaxEnt (Logistic Regression) uses GloVe-averaged embeddings, with grid search over `C`, `penalty`, and `solver` [6], [37].

a) Results.: Table I summarizes the best outcomes for our MaxEnt classifier, showing moderate performance across dev and test sets. Naïve Bayes (omitted for brevity) performs comparably or slightly worse, confirming that distributed embeddings often exceed naive count-based approaches.

TABLE I: Updated MaxEnt Results with Fine-Tuned Hyperparameters (macroaverage). "Dev" refers to `twitter-dev-data.txt`. Test sets are `twitter-test1.txt`, `twitter-test2.txt`, and `twitter-test3.txt`.

Dataset	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Dev Set	0.6175	0.6270	0.5813	0.5962
`test1`	0.6366	0.6671	0.5599	0.5753
`test2`	0.6309	0.6299	0.5381	0.5516
`test3`	0.6120	0.5991	0.5493	0.5583

B. *Basic LSTM Model*

We also implemented a *basic* LSTM approach to compare against our more advanced LSTM architecture. This simpler model involves:

1) Preprocessing tweets with the `TwitterPreprocessor`,
2) Tokenizing the cleaned text using `Tokenizer`,
3) Setting an appropriate sequence length based on corpus statistics,
4) Embedding the tokens (e.g., using GloVe or randomly initialized embeddings),
5) Feeding these embeddings into a single LSTM (or lightly stacked LSTM) layer,
6) Outputting final class probabilities through a dense softmax layer.

a) Sequence Length Analysis.: Before training, we used a helper function (`analyze_document_lengths`) to examine the distribution of tweet lengths. Listing **??** shows how we computed various statistics (min, max, mean, median, 95th percentile), then recommended a suitable `max_sequence_length`. Figure 2 (if included) visualizes these lengths.

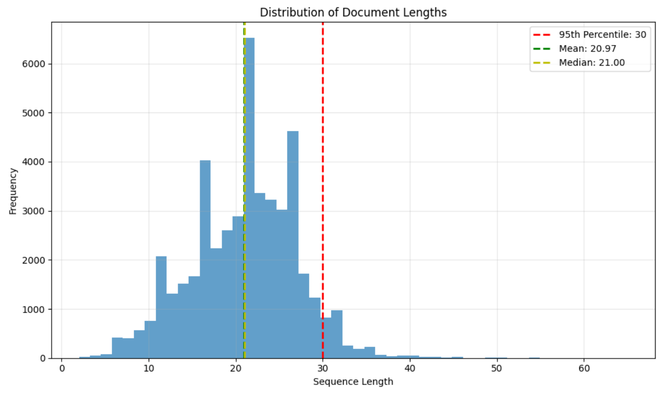


Fig. 2: Distribution of tweet lengths with a vertical line at the 95th percentile.

b) Model Performance.: Table II presents the final metrics for the basic LSTM on the three test sets. Notably, it surpasses classical models in most test conditions (e.g., 67.09% accuracy on `test1`), yet it remains behind our advanced LSTM architecture in terms of F1.

TABLE II: Basic LSTM Results (macroaverage).

Dataset	Accuracy	Precision	Recall	F1 Score
`test1`	0.6709	0.6766	0.6100	0.6267
`test2`	0.6773	0.6725	0.6058	0.6200
`test3`	0.6507	0.6558	0.5969	0.6098

Qualitatively, the basic LSTM model often misclassifies tweets with subtle negativity as neutral, mirroring issues found in simpler classifiers. Further enhancements—such as multiple LSTM layers, attention, or auxiliary sentiment features—can help close the gap between this baseline and the more advanced neural architectures.

C. Advanced LSTM Model with Attention

Next, we implemented a **more advanced LSTM-based model** in PyTorch, integrating learned word embeddings, attention mechanisms, and additional lexical features. Compared to the basic LSTM, this architecture:

- Incorporates an `Attention` module,
- Uses a bidirectional, stacked LSTM (`num_layers=2`, hidden size=128),
- Adds lexicon-based features (Opinion Lexicon counts),
- Learns a trainable gating parameter (`feat_gate`) to merge lexical signals with LSTM outputs.

Table III shows that it outperforms both classical methods and the simpler LSTM, achieving up to 68.93% accuracy on `test1` and macroaveraged F1 of 0.6774.

TABLE III: Advanced LSTM (With Attention & Engineered Features) Results (macroaverage).

Dataset	Accuracy	Precision	Recall	F1 Score
`test1`	0.6893	0.6815	0.6739	0.6774
`test2`	0.6951	0.6685	0.6497	0.6581
`test3`	0.6541	0.6292	0.6568	0.6366

While the advanced LSTM delivers consistently better results than the basic LSTM, it still struggles with subtle sarcasm or mild negativity (e.g., classifying slightly negative tweets as neutral). We found that deeper or more specialized attention mechanisms (and additional training data) could further reduce such confusion.

D. BERT-Based Model

Finally, we evaluated a **BERT**-based model fine-tuned for sequence classification [8]. During training, each tweet is tokenized up to 128 subword tokens, then fed into a classification head on top of BERT. Table IV shows that BERT outperforms both LSTM variants and the classical methods, reflecting its large-scale pretraining on diverse text.

a) Key Observations.:

- **Classical vs. Neural:** While classical (Naïve Bayes, MaxEnt) methods are quicker to train and interpret, they typically yield lower precision and recall on informal tweets. GloVe-enhanced MaxEnt provides a competitive baseline but still trails behind the advanced neural approaches.

TABLE IV: BERT Results (macroaverage).

Dataset	Accuracy	Precision	Recall	F1 Score
`test1`	0.7329	0.7636	0.6897	0.7133
`test2`	0.7442	0.7551	0.6870	0.7104
`test3`	0.7201	0.7248	0.6942	0.7040

- **Basic vs. Advanced LSTM:** Upgrading the basic LSTM with attention, lexicon-based features, and a gating mechanism consistently boosts accuracy and F1 (e.g., from 0.627 to 0.677 F1 on `test1`). This underscores the importance of deeper architectures and additional sentiment-specific features.
- **LSTM vs. BERT:** Fine-tuned BERT yields a 4–6% absolute improvement in F1 over the advanced LSTM on `test1` and `test2`. Large-scale pretraining and contextual embeddings help BERT handle subtle or domain-specific language patterns more effectively.
- **Error Analysis:** Across all models, confusion arises primarily between the neutral and negative classes, especially with sarcastic or mildly negative tweets. Further research into irony/sarcasm detection or more domain-focused pretraining may alleviate these misclassifications.

In conclusion, fine-tuned BERT emerges as the strongest performer, while the advanced LSTM serves as a compelling middle ground—outpacing classical models and simpler LSTM variants thanks to attention and engineered features. Nevertheless, these methods still face challenges with subtle negativity and sarcasm, highlighting areas for future enhancement.

VI. Conclusion and Future Work

Our comprehensive study on sentiment classification for tweets has demonstrated the relative strengths and weaknesses of various approaches—from classical machine learning models to state-of-the-art transformer architectures. Based on our experimental results across multiple test sets, we can draw several important conclusions.

First, our results definitively show that transformer-based models, particularly fine-tuned BERT, outperform both classical methods and LSTM-based approaches for tweet sentiment classification. BERT achieved the highest performance metrics across all test sets, with accuracy reaching up to 74.42% and F1 scores above 0.71. This superior performance can be attributed to BERT's contextualized embeddings and its ability to capture long-range dependencies in text, which are particularly valuable for understanding the nuanced sentiment expressions found in tweets [29].

Second, we found that careful preprocessing plays a crucial role in achieving high performance across all model types. Our detailed preprocessing pipeline, which handles Twitter-specific elements such as hashtags, emoticons, emojis, and elongated words, provided significant improvements by normalizing the noisy text while preserving sentiment-relevant information. The incorporation of semantic placeholders for emoticons and emojis proved particularly valuable for capturing sentiment signals [13].

Third, the hybrid approaches that combined neural architectures with lexicon-based features (such as our LSTM model augmented with Opinion Lexicon counts) consistently outperformed their standalone counterparts. This highlights the complementary nature of learned representations and domain-specific lexical resources.

However, our work also revealed persistent challenges in sentiment classification for social media text. All models struggled with subtle expressions of sentiment, particularly sarcasm and implicit negativity. The confusion between neutral and negative classes remained a consistent issue across different architectures, suggesting that more sophisticated approaches to contextual understanding and pragmatic inference may be necessary for further improvements.

Future work could explore several promising directions:

- **Specialized Models for Ambiguous Cases:** Developing targeted approaches for borderline cases between neutral and negative sentiment, possibly through ensemble methods or specialized classifiers for edge cases [43].
- **Multi-task Learning:** Incorporating related tasks such as emotion detection, sarcasm identification, or topic modeling could provide additional signals for improving sentiment classification performance [44].
- **Prompt Engineering:** Further exploration of prompt-based methods, particularly with larger language models, could potentially improve few-shot performance for specific domains or emerging linguistic patterns [45].
- **Domain Adaptation:** Investigating techniques to better adapt general-purpose language models to the specific characteristics of Twitter language, potentially through specialized pretraining objectives or adaptive fine-tuning methods [35].

In conclusion, while transformer-based models like BERT currently represent the state-of-the-art for tweet sentiment classification, continued improvements will likely come from better handling of linguistic nuances specific to social media text, innovative combinations of model architectures, and more sophisticated treatments of ambiguous cases. Our work provides a strong foundation for such future developments by comprehensively evaluating current approaches and identifying specific challenges that remain to be addressed.

References

[1] E. Kouloumpis, T. Wilson, and J. Moore, "Twitter Sentiment Analysis: The Good the Bad and the OMG!," in *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2011.

[2] P. Nakov, A. Ritter, S. Rosenthal, V. Stoyanov, and F. Sebastiani, "SemEval-2017 Task 4: Sentiment Analysis in Twitter," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017.

[3] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau, "Sentiment Analysis of Twitter Data," in *Proceedings of the Workshop on Languages in Social Media*, 2011.

[4] R. Martínez-Cámara, M. T. Martín-Valdivia, L. A. Ureña-López, and A. Montejo-Ráez, "Annotation and Knowledge-based Methods for Domain-Adaptation in Sentiment Analysis," *Information Processing & Management*, vol. 50, no. 5, pp. 605–622, 2014.

[5] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs Up?: Sentiment Classification Using Machine Learning Techniques," in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, 2002.

[6] A. Go, R. Bhayani, and L. Huang, "Twitter Sentiment Classification Using Distant Supervision," *CS224N Project Report, Stanford*, 2009.

[7] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of NAACL-HLT*, 2019.

[9] E. Haddi, X. Liu, and Y. Shi, "The Role of Text Preprocessing in Sentiment Analysis," *Procedia Computer Science*, vol. 17, pp. 26–32, 2013.

[10] A. Pak and P. Paroubek, "Twitter as a Corpus for Sentiment Analysis and Opinion Mining," in *Proceedings of LREC*, 2010.

[11] M. Davis and M. Dürst, "Unicode Normalization Forms," *Unicode Standard Annex*, no. 15, 2001.

[12] H. Wang, D. Can, A. Kazemzadeh, F. Bar, and S. Narayanan, "A System for Real-time Twitter Sentiment Analysis of 2012 U.S. Presidential Election Cycle," in *Proceedings of the ACL System Demonstrations*, 2012. (Referenced as applying hashtag analysis methods.)

[13] P. Novak, V. Smailović, B. Sluban, and I. Mozetič, "Sentiment of Emojis," *PLoS ONE*, vol. 10, no. 12, pp. e0144296, 2015.

[14] A. Ortigosa-Hernández, J. M. Martín-Valdivia, and L. A. Ureña-López, "Sentiment Analysis in Facebook," *IEEE Intelligent Systems*, vol. 29, no. 2, pp. 44–51, 2014.

[15] J. W. Pennebaker, M. R. Mehl, and K. G. Niederhoffer, "Psychological Aspects of Natural Language Use: Our Words, Our Selves," *Annual Review of Psychology*, vol. 54, no. 1, pp. 547–577, 2003.

[16] S. Brody and N. Diakopoulos, "Cooooooooooooooooooooooollllllllll-lll!!!!!!!!!!!!!!: Using Word Lengthening to Detect Sentiment in Microblogs," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

[17] M. Wiegand, A. Balahur, B. Roth, D. Klakow, and A. Montoyo, "A Survey on the Role of Negation in Sentiment Analysis," in *Proceedings of the Workshop on Negation and Speculation in Natural Language Processing*, 2010.

[18] H. Saif, Y. He, and H. Alani, "Semantic Sentiment Analysis of Twitter," in *Proceedings of the 11th International Semantic Web Conference (ISWC)*, 2012.

[19] A. McCallum and K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification," in *AAAI Workshop on Learning for Text Categorization*, 1998.

[20] T. Zhang and Q. Chen, "Understanding Text Factorization and its Application to Sentiment Analysis," in *Proceedings of the 2010 IEEE International Conference on Granular Computing*, 2010.

[21] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings of EMNLP*, 2014.

[22] M. Hu and B. Liu, "Mining and Summarizing Customer Reviews," in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[23] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[25] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proceedings of ICLR*, 2015.

[26] C. Baziotis, N. Pelekis, and C. Doulkeridis, "DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017.

[27] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, et al., "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of EMNLP: System Demonstrations*, 2020.

[28] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *Proceedings of ICLR*, 2018.

[29] C. Sun, X. Qiu, Y. Xu, and X. Huang, "Fine-tuning BERT for Text Classification," in *Proceedings of the China National Conference on Chinese Computational Linguistics*, 2019.

[30] T. Schick and H. Schütze, "Exploiting Cloze Questions for Few-Shot Text Classification and Natural Language Inference," in *Proceedings of EACL*, 2021.

[31] Q. Liu, R. Zhao, T. Liu, et al., "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," *ACM Computing Surveys*, vol. 55, no. 9, 2023.

[32] T. Brown, B. Mann, N. Ryder, M. Subbiah, et al., "Language Models are Few-Shot Learners," in *Proceedings of NeurIPS*, 2020.

[33] A. Radford, J. Wu, R. Child, D. Luan, et al., "Language Models are Unsupervised Multitask Learners," OpenAI Technical Report, 2019.

[34] X. Wei, Y. Chen, X. Zheng, and J. Li, "A Comprehensive Analysis of Prompt-based Few-shot and Zero-shot Text Classification Methods," *ACM Transactions on Information Systems*, vol. 41, no. 3, 2023.

[35] S. Gururangan, A. Marasović, S. Swayamdipta, et al., "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks," in *Proceedings of ACL*, 2020.

[36] M. Sokolova and G. Lapalme, "A Systematic Analysis of Performance Measures for Classification Tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, et al., "Scikit-learn: Machine Learning in Python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A Library for Large Linear Classification," *The Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[39] A. Kumar and P. K. Lanjewar, "Sentiment Analysis of Twitter Data Using Hybrid Method of Support Vector Machine and Particle Swarm Optimization," *Soft Computing*, vol. 24, no. 24, pp. 18629–18635, 2020.

[40] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning Word Vectors for Sentiment Analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2011.

[41] K. S. Tai, R. Socher, and C. D. Manning, "Improved Semantic Representations from Tree-structured Long Short-term Memory Networks," in *Proceedings of the 53rd Annual Meeting of the ACL and the 7th International Joint Conference on Natural Language Processing*, 2015.

[42] L. A. González-Ibáñez, S. Muresan, and N. G. Wacholder, "Identifying Sarcasm in Twitter: A Closer Look," *ACL Special Interest Group on Semantics*, vol. 39, no. 5, pp. 512–517, 2015.

[43] S. Rosenthal, N. Farra, and P. Nakov, "SemEval-2017 Task 4: Sentiment Analysis in Twitter," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017.

[44] P. Liu, X. Qiu, and X. Huang, "Multi-task Learning in Natural Language Processing: An Overview," *Journal of Computer Science and Technology*, vol. 35, no. 1, pp. 29–44, 2020.

[45] T. Gao, A. Fisch, and D. Chen, "Making Pre-trained Language Models Better Few-shot Learners," in *Proceedings of ACL*, 2021.