

Product Review & Sentiment Analysis Using Natural Language Processing: RSR Analyze

Rishabh Sharma, Rudra Dubey, Swatik Maurya

Under the guidance of

Dr. Ankit Kumar

School of Computer Science and Engineering, Galgotias University, Greater Noida, India

Abstract

This paper presents a sentiment analysis system we developed — RSR Analyze — that can classify product reviews as positive, negative, or neutral in real time. What makes our approach different from most existing work is that we deliberately avoided heavy transformer-based models like BERT, and instead focused on building something that is lightweight, fast, and practical to deploy. We trained our system on a large-scale Amazon review dataset and also incorporated Hinglish and code-mixed language support, since that is how a lot of real Indian users actually write reviews online. Through multiple experiments, we compared traditional models (Naive Bayes, Logistic Regression, LinearSVC) with deep learning approaches (LSTM, BERT, DistilBERT) across accuracy, F1-score, inference latency, and memory usage. Our results show that LinearSVC with TF-IDF features achieves around 88% accuracy while being significantly faster and lighter than transformer models — making it genuinely usable for real-time applications.

Keywords: Sentiment Analysis, NLP, Multilingual NLP, Hinglish, TF-IDF, LinearSVC, BERT, Edge AI, Low-Latency Systems

1. Introduction

If you have ever bought something online, you have probably spent time scrolling through reviews before making a decision. Now multiply that across millions of users and thousands of products — and you start to understand why automated sentiment analysis has become such a critical problem in e-commerce.

For our final year MCA project, we wanted to build something that goes beyond just classifying English text as positive or negative. Our goal was to create a system that could actually work in the real world — handling the messy, code-mixed, often grammatically incorrect text that users write on platforms like Amazon India, Flipkart, and Meesho.

Modern transformer-based models like BERT are impressive, but they come with a major drawback: they are expensive to run. They need GPUs, consume large amounts of memory, and introduce noticeable latency. For a startup or a mid-sized e-commerce platform that cannot afford cloud GPU instances for every API call, BERT is simply not practical.

So we asked ourselves: can we build a system that is accurate enough to be useful, but efficient enough to run on basic infrastructure? That question drove this entire project. The system we designed — RSR Analyze — uses a LinearSVC classifier with TF-IDF features, exposed through a FastAPI microservice, and it handles English, Hindi, and Hinglish inputs reasonably well.

The core research question we explored is: Can lightweight machine learning models provide competitive multilingual sentiment analysis performance under low-resource deployment constraints compared to transformer-based approaches?

2. Literature Review

2.1 Classical Approaches

Sentiment analysis as a field has been around for over two decades. The early work by Pang et al. (2002) showed that traditional machine learning techniques — Naive Bayes, SVM, Maximum Entropy — could classify movie review sentiment with reasonable accuracy using bag-of-words and TF-IDF features. When we started our project, we went back to these papers to understand why these methods are still relevant today.

TF-IDF, in particular, has held up surprisingly well over time. It is simple to implement, easy to interpret, and computationally cheap. The idea is straightforward: words that appear frequently in a document but rarely across the corpus are likely to be more meaningful for classification. That said, TF-IDF has well-known limitations:

- It treats each word independently and has no sense of word order or context
- It cannot capture semantic relationships — 'great' and 'excellent' are completely different tokens to it
- It struggles with sarcasm, negation, and nuanced expressions like 'not bad'

These limitations are real, and we saw them in our own error analysis. But for the purpose of fast inference on noisy text, TF-IDF still performs better than you might expect.

2.2 Deep Learning and Transformer-Based Models

As deep learning took over NLP, sentiment analysis models got much more powerful. LSTMs improved over basic RNNs by handling long-range dependencies in text. Then BERT arrived and essentially redefined what state-of-the-art meant — its bidirectional attention mechanism allows it to understand words in full context, not just left-to-right.

We experimented with BERT and DistilBERT ourselves, and the accuracy numbers are clearly higher. But the cost is significant:

- BERT inference on CPU takes 120ms per request in our setup — too slow for a real-time review feed
- Memory consumption is very high, often exceeding available RAM on lightweight servers
- Deploying BERT properly requires GPU acceleration, which is expensive at scale

DistilBERT reduces the model size by around 40% while retaining most of BERT's performance, which is a genuine improvement. But it still falls short of the low-latency requirement we had set for this project.

2.3 Multilingual and Hinglish NLP

This was honestly the most interesting part of the literature review for us. As students who grew up in India, we know very well that people do not write reviews in clean, standard English or Hindi. They write things like "Camera acha hai but battery bekaar hai" or "This product is total bakwas, waste of money." This is Hinglish — a spontaneous mix of Hindi and English that has no fixed grammar, uses Roman script for Hindi words, and varies wildly in spelling.

Models like IndicBERT and MuRIL were specifically designed to handle this kind of code-mixed data, and they do perform better on it. However, they are even heavier than standard BERT. For our use case, we needed a practical middle ground.

We found that combining basic Hinglish normalization (standardizing common Hindi words, mapping transliterated variants) with a strong TF-IDF pipeline gave us competitive results without the overhead of a multilingual transformer.

2.4 Research Gap

After going through the existing literature, we noticed a consistent pattern: most papers optimize for either accuracy or efficiency, rarely both. There is a lack of systems that can simultaneously handle code-mixed inputs, achieve low latency, and run on modest hardware. Our project is an attempt to address that gap — not perfectly, but practically.

3. Data Description and Preprocessing

3.1 Dataset Overview

We trained and evaluated our system on the Amazon Reviews Dataset, which is widely used in sentiment analysis research. It contains customer reviews across multiple product categories — electronics, clothing, home appliances — along with star ratings and written feedback.

To make our system relevant for Indian e-commerce, we supplemented the dataset with Hinglish and mixed-language reviews. The final composition is summarized below:

Table 1: Dataset Statistics

Language Type	Total Samples	Positive	Negative	Neutral
English	6,000	3,000	2,000	1,000
Hindi	1,500	800	500	200
Mixed	1,000	500	300	200
Total	8,500	4,300	2,800	1,400

The dataset is moderately imbalanced, with positive reviews making up the majority. This is actually expected — in real e-commerce, satisfied customers leave more reviews than dissatisfied ones. We kept this distribution rather than artificially balancing it, because we wanted the model to reflect real-world patterns.

3.2 Preprocessing Steps

Before feeding the data into any model, we ran the following preprocessing pipeline:

- Lowercasing all text for uniformity
- Removing special characters, URLs, and unnecessary symbols
- Tokenizing sentences into words
- Handling missing entries and removing duplicates
- Normalizing Hinglish text — e.g., mapping 'acha', 'accha', 'achaah' to a standard form
- Special handling of negation words like 'not', 'nahi', 'na' which flip sentiment

We tried stopwords removal but found it actually hurt performance on Hinglish text, since some common Hindi words carry sentiment weight. So we made it optional depending on the language type detected.

3.3 Dataset Challenges

Working with this dataset taught us a lot about real-world NLP problems. The most frustrating ones were:

- Sarcasm and irony — 'Great product, totally worth the repair costs in week 1' is clearly negative but hard to classify
- Mixed sentiment within a single review — 'Camera is good but battery is terrible'
- Spelling variations in Hinglish — 'bekaar', 'bekar', 'bekaarr' are all the same word
- Neutral reviews that overlap with both positive and negative tone

4. System Architecture

4.1 Overview

We designed RSR Analyze using a microservices-based architecture. The idea was to keep each component independent so it could be developed, tested, and scaled separately. The system is divided into four layers:

- Frontend — React (user interface)
- Backend — Node.js / Express (API gateway and routing)
- ML Service — Python / FastAPI (inference engine)

- Database — MySQL (persistent storage for reviews and results)

4.2 Frontend: React

The frontend lets users type in a product review and get an instant sentiment prediction. We kept the UI simple and functional — this is not the focus of the research, but it helped us demonstrate the system in real time. Key features include real-time API calls, state management for handling inputs and responses, and a clean component structure.

4.3 Backend: Node.js / Express

Node.js handles HTTP routing, forwards requests to the ML service, and manages validation. Its asynchronous, non-blocking nature means it can handle multiple concurrent requests without slowing down — which is important for a system meant to process review feeds at scale.

4.4 ML Service: Python / FastAPI

This is the heart of the system. FastAPI was chosen because it offers high throughput, automatic Swagger documentation, and integrates naturally with Python's ML ecosystem. The service loads the trained model at startup and performs inference on incoming text, returning a Positive / Negative / Neutral label along with a confidence score.

We measured inference time on a standard laptop (Intel Core i5, 8GB RAM, no GPU) and consistently got results under 5ms per request — which exceeded our target.

4.5 Database: MySQL

MySQL stores user-submitted reviews, prediction results, and system logs. This allows us to track model performance over time and retain data for potential retraining. It also enables historical analysis — something that would be useful for a business wanting to track sentiment trends across products.

4.6 Data Flow

The pipeline works as follows:

- 1. User submits a review through the React interface
- 2. Node.js backend receives the request and forwards it to FastAPI
- 3. FastAPI ML service preprocesses the text and runs inference
- 4. Prediction result is returned to the backend and then displayed on the frontend
- 5. Review and result are optionally stored in MySQL for logging

5. Methodology

5.1 Feature Engineering: Ablation Study

We ran an ablation study to understand how different feature representations affect classification accuracy. The results are in Table 2:

Table 2: Feature Engineering Ablation Study

Feature Combination	Accuracy
Unigram	81%
Bigram	83%
TF-IDF	86%
TF-IDF + Bigram	88%

The TF-IDF + Bigram combination consistently outperformed all other options. Bigrams help because they capture short phrases like 'not good' or 'very bad' that unigrams would miss entirely.

5.2 Model Comparison

We evaluated five models across four metrics. Table 3 summarizes the results:

Table 3: Model Comparison

Model	Accuracy	F1 Score	Latency	Memory
Naive Bayes	82%	0.81	2ms	Low
Logistic Regression	85%	0.84	5ms	Low
LinearSVC	88%	0.87	4ms	Low
LSTM	89%	0.88	45ms	High
BERT	92%	0.91	120ms	Very High

BERT clearly wins on accuracy, but 120ms per inference is a dealbreaker for real-time systems. LinearSVC hits 88% at just 4ms — that is a 30x speed improvement for a 4% accuracy tradeoff, which we think is a reasonable trade for most practical applications.

5.3 Final Model Selection

Based on this analysis, we selected LinearSVC with TF-IDF + Bigram features as the final model. The complete pipeline is: text preprocessing → TF-IDF + Bigram feature extraction → LinearSVC classification → sentiment label output.

6. Resource Efficiency Analysis

Table 4: Resource Usage Comparison

Model	CPU Usage	RAM	Inference Time
LinearSVC	Low	Low	4ms
LSTM	Medium	High	45ms
BERT	High	Very High	120ms

Lightweight models provide roughly 20–30x faster inference compared to BERT, with only a modest accuracy reduction. For high-throughput API services or edge deployment scenarios, this difference is massive in practical terms.

7. Multilingual Evaluation

Table 5: Performance Across Language Types

Language Type	Accuracy	F1 Score	Precision	Recall
English	89%	0.88	0.90	0.87
Hinglish	84%	0.83	0.85	0.82
Mixed	82%	0.81	0.83	0.80

Predictably, English performs best — it has clean structure, a standardized vocabulary, and better representation in the training data. Hinglish performance drops a bit, mainly because of phonetic spelling variations and transliteration inconsistencies. Mixed-language text is the hardest because sentiment can shift mid-sentence between languages.

Overall, we were fairly satisfied with these numbers. Getting 84% accuracy on Hinglish without using a multilingual transformer is genuinely non-trivial, and the normalization preprocessing we designed made a measurable difference.

8. Error Analysis

No model is perfect, and ours is no exception. Here are the main categories of errors we observed during testing:

8.1 Sarcasm and Irony

This is probably the hardest problem in sentiment analysis, and our model handles it poorly. An example from our test set: 'Great phone... died in 2 hours.' The word 'great' pushes the classifier toward positive, and it completely misses the sarcastic turn. Without contextual understanding, this is very difficult to solve with traditional ML.

8.2 Mixed Sentiment

Reviews like 'Camera acha hai but battery bekaar hai' present a real challenge — the review contains both a positive opinion and a negative one. Our system assigns a single label, which forces it to choose. In these cases, it often picks the more dominant sentiment, but it still loses nuance. Aspect-based sentiment analysis would handle this better, but that is scope for future work.

8.3 Negation Handling

'Not bad' means positive. 'Nahi acha' means not good. Getting negation right is tricky, especially in Hinglish where the negation word can be a Hindi word attached to an English adjective. We added special negation handling in preprocessing, which helped, but edge cases still slip through.

8.4 Spelling Variations and Emojis

'Gud', 'awesome', 'bekaarr' — informal spelling reduces model confidence because these variants are underrepresented in the vocabulary. Similarly, emojis can contradict the text sentiment ('Product is okay 🍌') and we do not handle that well.

9. Discussion

9.1 Strengths of the Proposed System

Looking back at what we built, there are a few things we are genuinely proud of:

- Sub-5ms inference time on standard hardware, with no GPU required
- Hinglish compatibility through a carefully designed normalization pipeline
- A clean microservices architecture that is easy to extend and maintain
- Real-world applicability — this could actually be deployed as a production API without major modifications

9.2 Limitations

We also want to be honest about what the system cannot do well:

- Accuracy falls short of transformer models by 4–8%, and this gap is noticeable on complex sentences
- Sarcasm and irony are largely unsolved at this level of model complexity

- Single-label classification cannot capture multi-aspect opinions within one review
- Noisy text degrades performance, though preprocessing helps significantly

10. Conclusion

This project started from a simple observation: most of the best NLP models in research are not actually usable by most real-world systems. Our goal was to close that gap — at least partially — by building a system that is accurate enough to be useful, fast enough to be practical, and flexible enough to handle the kind of messy multilingual text that real Indian users produce every day.

RSR Analyze demonstrates that traditional ML models, when combined with thoughtful preprocessing and feature engineering, can still be highly competitive. LinearSVC with TF-IDF + Bigram features achieves 88% accuracy at under 5ms per inference — a combination that transformers simply cannot match at that compute level.

The multilingual evaluation results also show that handling Hinglish is genuinely feasible without a heavy multilingual model, provided the preprocessing is done right. We see this as meaningful progress toward making sentiment analysis tools accessible for Indian-language e-commerce platforms.

That said, we are aware of the trade-offs. Deep context understanding, sarcasm detection, and aspect-level sentiment analysis remain open challenges. Our future work will explore hybrid architectures — combining a fast TF-IDF backbone with lightweight contextual modules — to see if we can push accuracy higher without sacrificing the efficiency that makes this system practical.

As a final thought: building this project gave us hands-on experience with the full ML pipeline, from raw data to a deployed API. That practical experience was, in many ways, the most valuable outcome.

References

- [1] Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment classification using machine learning techniques. *Proceedings of EMNLP 2002*, 79–86.
- [2] Joshi, A., Prabhu, A., Shrivastava, M., & Varma, V. (2016). Towards sub-word level compositions for sentiment analysis of Hindi-English code mixed data. *Proceedings of COLING 2016*, 2482–2491.
- [3] Yadav, S., Chakravarthi, B. R., Bansal, P., & Kumar, V. (2020). Sentiment analysis of Hindi-English code mixed data. *Proceedings of the Workshop on Tools & Resources to Empower People with Reading Difficulties (READ)*, 37–44.
- [4] Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- [5] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019*, 4171–4186.