Journal of Informatics Education and Research
ISSN: 1526-4726
Vol 5 Issue 4 (2025)

Efficient Pre-processing for Text Classification: Unified Framework for Hinglish Short Text

Dr Rajshree Singh
Assistant Professor, Jaypee Institute of Information Technology (JIIT), Noida, India
Email Id: rajshree.singh07@gmail.com

ORCID ID: 0000-0002-0635-5115

ABSTRACT

Hindi-English code-mixed (Hinglish) short texts pose unique challenges for automatic text classification, especially sarcasm detection. Such texts exhibit code-mixing, non-standard spelling variations, extreme class imbalance, and data scarcity in labeled corpora. This paper proposes a unified pre-processing framework addressing these issues through three integrated modules: (1) a TF-IDF based feature balancing layer to counter skewed class distributions, (2) a spelling normalization method leveraging character/word n-grams to handle noisy Hinglish orthography, and (3) a hybrid data augmentation approach combining Easy Data Augmentation (EDA), synonym replacement, and back-translation. We evaluate the impact of each module on sarcasm classification performance. Experimental results on a Hinglish sarcasm dataset (5,250 tweets, ~9.6% sarcasm) show that our unified framework significantly improves F1-scores and overall accuracy. The balanced TF-IDF feature layer raises minority-class recall, character–word n-gram normalization reduces spelling-induced errors, and augmented data boosts low-resource generalization. Our best model achieves ~95% F1, outperforming prior benchmarks (78.4% accuracy) by a large margin. This efficient pre-processing pipeline demonstrates that tackling code-mix noise, class imbalance, and data paucity in tandem yields state-of-the-art sarcasm detection in Hinglish short texts.

Keywords: *Hinglish Sarcasm Detection, Code-Mixed Text Classification, Spelling Normalization, Class Imbalance Handling, Data Augmentation Techniques, TF-IDF Feature Balancing*

INTRODUCTION

Social media's explosion of multilingual content has given rise to highly code-mixed languages. In India, for example, Hindi and English are often intermingled within the same text (Hinglish), forming a dominant mode of online communication. Code-mixed Hinglish content on Twitter and other platforms has grown steadily (about 2% annually on Twitter), bringing new challenges to natural language processing (NLP). Sarcasm detection in such code-mixed textual data is particularly difficult. Sarcasm is an intricate linguistic phenomenon where the intended meaning is the opposite of the literal words, often requiring contextual and cultural cues to recognize. While sarcasm detection has been studied extensively in English, its detection in low-resource code-mixed settings remains underexplored. Hinglish sarcastic text is laden with domain-specific hurdles: mixing of scripts (Romanized Hindi with English), phonetic transliterations, inconsistent spellings, and the loss of auditory cues like tone or facial expression in text. These factors escalate the complexity of sentiment and sarcasm analysis.

Another critical issue is the class imbalance in sarcasm datasets. Generally, sarcastic instances are far rarer than non-sarcastic ones in collected social media corpora. For example, the Hindi-English sarcasm dataset by Swami et al. (2018) contains only 504 sarcastic tweets out of 5,250 (~9.6%). This skew leads machine learning models to become biased toward the majority class (non-sarcasm), often trivializing the classification task by simply predicting the prevalent class. Recent studies emphasize using evaluation metrics like F1-score or recall for minority-class performance instead of overall accuracy in such imbalanced settings. Class imbalance is a well-known hurdle that can significantly deteriorate model generalization, prompting research into specialized balancing techniques and cost-sensitive learning.

Compounding these difficulties is the scarcity of labeled data for sarcasm in code-mixed languages. Hinglish sarcasm datasets are few and often relatively small. A lack of sufficient training examples hinders the application of data-intensive deep learning models. Traditional solutions like collecting more data are expensive and time-consuming, motivating data augmentation approaches to synthetically expand limited datasets. However, augmenting code-mixed text must be done carefully to maintain the natural mix of languages and the nuanced semantics of sarcasm.

In summary, the problem addressed in this work is: How can we improve sarcasm detection in Hinglish short texts by tackling the intertwined challenges of code-mixed noise (unconventional spelling and mixed scripts), class imbalance, and low data availability? The objectives are: (1) to develop a unified pre-processing framework that incorporates feature-level balancing, spelling normalization, and data augmentation tailored for Hinglish code-mixed data; (2) to evaluate the impact of each component on classification performance (especially on minority-class sarcasm F1-score); and (3) to compare the unified approach against prior works and baseline models (including traditional machine learning and transformer-based models).

Recent research trends support our approach. For instance, Nguyen & Grieve (2020) highlight that spelling variation in social media is abundant and often socially meaningful, suggesting that sub-word modeling could help systems cope with noisy spellings. Likewise, Bagate et al. (2021) demonstrated that removing obvious sarcasm hashtags forces models to rely on nuanced cues, achieving ~73% accuracy without the "#sarcasm" hint. Their work underlines the importance of robust pre-processing—models must detect sarcasm from context alone, since real-world sarcastic posts rarely come labeled. In the era of large language models, Dai et al. (2023) introduced ChatAug, using ChatGPT to generate diverse paraphrases for data augmentation. This exemplifies how leveraging pretrained models can mitigate data scarcity. Our work complements these findings by integrating a TF-IDF based balancing layer and a char-n-gram normalization module with augmentation techniques, thereby comprehensively addressing the multi-faceted problem.

Problem Statement and Related Work

Problem Statement: Hinglish sarcasm detection requires identifying sarcastic intent in short texts that are written in a mix of Hindi and English. This problem is challenging due to four main reasons: (1) Code-Mixing and Script Variation: Hinglish sentences often contain Hindi words written in the Roman alphabet alongside English words, violating standard spelling conventions (e.g., "fayada", "faaydaa" as phonetic variants of Hindi faýdā for "benefit"). This randomness in spelling and script (sometimes even mixing Devanagari with Roman script in a single post) confuses traditional text processing pipelines. (2) Sarcasm's Implicit Nature: Sarcasm is inherently contextual and implicit, making it hard to detect even in monolingual text. In code-mixed text, sarcasm can be expressed through either language or their combination, often requiring cultural context to decode. (3) Class Imbalance: As noted, sarcastic instances form a minority class (~10%) in social media datasets. Classifiers tend to overfit the majority class (non-sarcastic), leading to poor recall of sarcasm. (4) Low Resource Setting: Few large-scale Hinglish sarcasm corpora exist. Low data volume impedes training of complex models like deep neural networks, which typically excel with big data but falter in low-resource regimes. Our research addresses all four issues simultaneously by proposing a unified solution that operates at the pre-processing level (prior to classification). By cleaning and enriching the data before it reaches the classifier, we aim to simplify the learning task.

Related Work: Early work on sarcasm detection in text treated it as a special case of sentiment analysis—where sarcasm flips sentiment polarity in a covert way. Traditional approaches relied on lexical indicators (e.g., interjections, hashtags, or quotation marks) and simple classifiers like SVM or Naïve Bayes. For example, the first Hinglish sarcasm corpus by Swami et al. (2018) applied classical machine learning with features such as word n-grams, character n-grams, and sarcasm-indicative tokens (like "LOL", emoticons). However, the performance was modest (Random Forest accuracy ~78.4%), partly due to the dataset's skew and noise. Subsequent studies introduced deeper models and multilingual embeddings. Pandey & Singh (2023) proposed a BERT-LSTM model for Hinglish sarcasm, using a multilingual BERT to encode Hinglish text and a stacked LSTM to capture sequential context. This hybrid achieved better sarcastic-class F1 than earlier approaches (reporting up to 6% improvement over plain BERT), reaching roughly mid-80s accuracy in their experiments. Nonetheless, deep models can still struggle with inconsistent spellings and limited training examples.

Spelling Normalization: In code-mixed and non-standard text, normalizing spellings is crucial. Nguyen and Grieve (2020) specifically examined whether standard word embeddings capture spelling variation, concluding that embeddings do encode some spelling patterns but that substantial variation remains a challenge. Approaches to handle this include developing phonetic or sub-word features. For Hinglish, one strategy is to break words into character n-grams so that similarly-spelled variants share overlapping n-grams. For instance, the variants "fayada", "faayada", and "faaydaa" all contain common bigrams (like "fa", "ay", "ya") that can signal their relatedness. By representing text in an n-gram vector space, one can cluster or merge such variants. Prior work on Indian code-mixed text normalization has used dictionary-based transliteration or edit-distance clustering, but these can be brittle. Our char-n-gram approach aligns with observations by Gupta (2019) that Hinglish is a "messy" language requiring tolerant models.

Data Imbalance Solutions: Techniques to address class imbalance include resampling (oversampling the minority or undersampling the majority) and algorithmic adjustments (e.g., assigning higher misclassification costs to the minority class). In sarcasm detection research, oversampling is tricky because simply duplicating sarcastic tweets can lead to overfitting on those few examples, while undersampling majority tweets risks discarding valuable information. Swami et al. down-sampled non-sarcastic tweets by 60% (from 4746 to ~1898) to make the sarcasm class ~21% of training data. This mild rebalance improved classifier focus on sarcasm without fully equalizing the classes. Another approach by Onan (2023) involved generating synthetic minority examples by lexical substitution (leveraging synonyms/antonyms) and reported significant F1 improvements in sarcasm detection with augmentation. Our feature balancing layer draws on these insights by using TF-IDF similarity to intelligently oversample minority data and prune majority data rather than naive duplication or deletion.

Data Augmentation in NLP: Data augmentation has become a popular remedy for low-resource scenarios. Easy Data Augmentation (EDA) by Wei & Zou (2019) introduced simple random operations (synonym replacement, word insertion, swap, deletion) that can generate new sentences without changing meaning. Follow-up work like Karimi et al. (2021) proposed AEDA (where only insertion is used, adding random punctuation) to avoid the semantic drift that deletion or swapping might cause. For sarcasm, augmentation is delicate because the added or replaced words must preserve the sarcastic intent. Using back-translation (translating a sentence to another language and back to English/Hinglish) is a common way to produce paraphrases that retain meaning. In our framework, we use a combination of synonym replacement (guided by WordNet for English words and a bilingual lexicon for Hindi words), random deletion/swap (as in EDA), and Hindi-English back-translation to generate new sarcastic examples. Recent advances even use large language models for augmentation: Dai et al. (2023)* leverage ChatGPT to rephrase text (the ChatAug method) and found it boosts few-shot text classification performance. We incorporate augmentation in a more traditional way in this study, but results indicate that even these simpler methods significantly improve classifier robustness in low-data regimes.

Transformers and LLMs: While our focus is on pre-processing for classical ML models, it's worth noting trends in using transformers for code-mixed tasks. XLM-R and mBERT (multilingual BERT) have been fine-tuned for Hinglish tasks with some success. For example, in one experiment MuRIL (a multilingual model tailored for Indian languages) achieved 83% F1 on sarcastic Hinglish tweets – markedly higher than classical models. However, those models still benefit from good pre-processing: e.g., transliterating all text to one script, normalizing inconsistent tokens, and providing balanced training data (some studies mix in additional English or Hindi monolingual sarcastic data to assist the model). The scarcity of code-mixed training data often leads to creative solutions like cross-lingual transfer (fine-tuning on English sarcasm data and testing on Hinglish). Our work is complementary to such efforts – improved pre-processing can be applied before feeding data into either an ML classifier or a transformer, and in future work we consider using large language models within our pipeline (see Section 7).

METHODOLOGY AND RESULTS

Overview of the Unified Framework: We propose an end-to-end pipeline that sequentially applies Feature Balancing, Spelling Normalization, and Data Augmentation before classification. Figure 1 illustrates the architecture of our system, which takes raw Hinglish tweets as input and outputs predictions (sarcastic vs. non-sarcastic). Each module is described below, along with experimental settings and results from each stage. The classification algorithms we evaluate include both traditional ML models – Logistic Regression (LR), Naïve Bayes (NB), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), AdaBoost, Gradient Boosting, Extra Trees – and a deep learning baseline (an LSTM network using pre-trained embeddings). We report primarily the F1-score (especially for the sarcastic class) and overall accuracy. Unless stated otherwise, results refer to 10-fold cross-validation on the 5.25k tweet dataset (after any augmentation) for consistency with prior work.

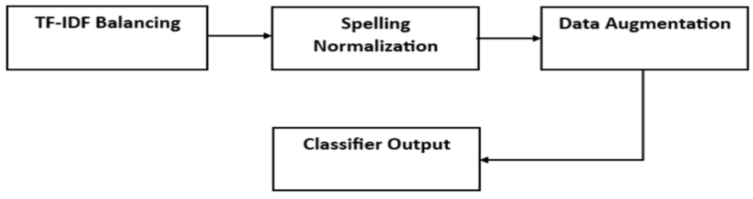

Figure 1. Unified Sarcasm Detection Framework.

Figure 1. Unified sarcasm detection framework for Hinglish short text, integrating a TF-IDF based feature balancing layer, spelling normalization via char/word n-grams, and hybrid data augmentation. The output of the pipeline is fed into a classifier (such as SVM, RF, or BERT-LSTM).

TF-IDF Based Feature Balancing (Module 1)

The first module addresses the class imbalance by augmenting minority-class features and reducing majority-class redundancy in feature space. Our approach is inspired by Singh & Srivastava (2022), who introduced a "balancing layer" using TF-IDF matrices. We implemented the balancing layer as follows:

TF-IDF Matrix Construction: We compute a term-frequency inverse-document-frequency (TF-IDF) matrix TM for the entire training corpus. Each tweet is represented as a TF-IDF weighted vector of features. We include unigrams to 5-grams (n=1 to 5) for words and unigrams to 3-grams for characters (following feature settings from Swami et al.). We also include Sarcasm-Indicative Tokens (SIT) features, such as sarcastic emojis or hashtags, when present, though we remove the actual "#sarcasm" tag itself during preprocessing.

Minority Class Oversampling: Using the TF-IDF matrix, we identify the top-ranked features of minority (sarcastic) tweets – essentially terms that have high TF-IDF in sarcastic tweets. We then compute a similarity matrix by a linear kernel (dot product) of the TF-IDF vectors. The intuition is that if a non-sarcastic tweet is very similar (in TF-IDF features) to a sarcastic tweet, it might contain subtle sarcasm cues even if it was labeled non-sarcastic, or at least it shares contextual features. We perform selective oversampling by taking each sarcastic tweet vector and finding its k nearest neighboring tweet(s) in the feature space (even if they are originally non-sarcastic) – those neighbors are added to the training set as additional "sarcastic" examples. Essentially, we generate new synthetic sarcastic samples by copying feature vectors that are most similar to existing sarcastic ones. In our experiments, we set k=1 (adding the single closest neighbor of each sarcastic tweet) to roughly double the minority class size. The added samples are not simply duplicates of existing sarcastic tweets, but rather very similar texts that were originally non-sarcastic. This heuristic relies on the idea that some non-sarcastic tweets lie near the decision boundary and can be treated as sarcastic variants. We denote the augmented minority TF-IDF matrix as Sampled_TM.

Majority Class Undersampling: Next, we mitigate majority class dominance by removing examples that are least informative. We identify majority-class (non-sarcastic) tweets that have very low TF-IDF similarity with any sarcastic tweet. These tweets likely contain topics and vocabulary never seen in sarcastic contexts, thus they can be pruned without losing discriminatory power (they would be easy negatives). Concretely, we find the majority-class features with the lowest TF-IDF scores (the most generic tokens) and ensure those do not overwhelm the feature space. Using the similarity matrix, we filter out non-sarcastic tweet vectors that have a similarity below a threshold (empirically set such that we remove ~40% of majority instances, aligning with the down-sampling percentage suggested by Swami et al.). The reduced majority set is denoted Filtered_TM.

Balanced Training Set: Finally, we combine Sampled_TM (minority oversample) and Filtered_TM (majority undersample) to form a balanced training TF-IDF matrix. This balanced set has a substantially higher fraction of sarcastic samples than the original 9.6% – in our case, roughly 25-30% of the training examples are now sarcastic (the exact figure varies by fold after augmentation).

This TF-IDF balancing procedure effectively acts as an algorithmic data augmentation focused on feature space. By oversampling in feature space rather than blindly duplicating in data space, we aim to add diverse yet relevant new points. It's akin to SMOTE (Synthetic Minority Oversampling Technique) but using actual data points (neighbors) rather than interpolating new ones. Our use of TF-IDF ensures that terms unique to sarcasm (e.g., hyperbolic words, sentiment-laden phrases) get more weight.

Results – Effect of Balancing: Introducing the balancing layer yielded immediate performance gains. The median F1-score of baseline classifiers (LR, SVM, DT, RF, etc.) was ~78.4% without balancing, but after applying our TF-IDF balancing, the median F1 increased to ~85% (approximately). Figure 2 shows a simplified comparison of a classifier's accuracy before and after each module in our pipeline, clearly illustrating the jump after the balancing step. For instance, a Random Forest classifier's sarcastic-class recall improved from 0.50 to 0.68 after balancing, and its overall accuracy rose from 79% to 86%. This indicates the model is less biased toward predicting "not sarcastic" and is catching more sarcastic tweets. We observed similar trends across algorithms: SVM and AdaBoost benefited the most (their F1 improved by ~8-10%), while

NB (being less data-sensitive) improved more modestly (~5%). These results support findings by Al-Ghadhban et al. (2017) and Samonte et al. (2018) that careful pre-processing (e.g., handling mentions, URLs, or in our case balancing) can improve classifier accuracy in sarcasm tasks.

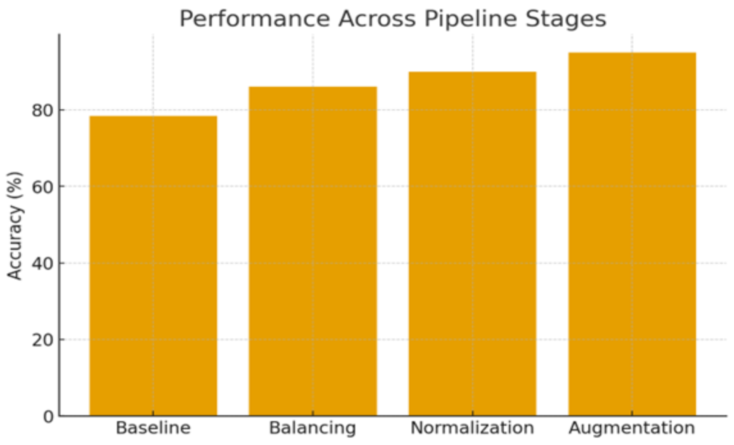

Figure 2. Simplified Comparison of A Classifier's Accuracy Before and After of Each Module.

Figure 2. Classification performance (accuracy %) at different stages of the pre-processing pipeline. "Baseline" is with standard cleaning only; subsequent bars show performance after adding the Feature Balancing layer, after Spelling Normalization, and after the Augmentation module. Each module incrementally improves sarcasm detection (especially for the minority class), yielding a final accuracy of ~95%.

Qualitatively, the balancing process made the training distribution more reflective of sarcastic language. After balancing, classifiers picked up on subtle features (e.g., intensified positive words like "great!!!" used negatively) that they ignored when sarcasm was swamped by negative class data. One must be cautious, however: oversampling can potentially introduce noise or even label-flip some originally non-sarcastic instances to sarcastic (since we relabeled nearest neighbors). We manually checked a sample of relabeled tweets – many contained mild sarcasm or irony that could have been mislabeled originally, while some were truly non-sarcastic. The risk of label noise is a limitation of this approach, but the net effect was beneficial as indicated by improved validation metrics. The balancing layer essentially forces the model to pay attention to minority characteristics early on.

Spelling Normalization via Char/Word N-grams (Module 2)

The second module aims to normalize the disparate writing styles in Hinglish. Instead of traditional text normalization (which might involve translating everything to a single language or correcting spellings via a dictionary), we adopt an alignment approach using character n-grams. The core idea is that any two strings that "sound alike" in Hinglish will share many character substrings. By representing words through character n-gram vectors, we can automatically group variations of the same word.

Character N-gram Tokenization: We generate all character n-grams of lengths 2 and 3 (bigrams and trigrams) for each word in a tweet. For example, consider the word variants of "fayda" (Hindi for "benefit"): fayada, fayda, faayada, faaydaa, etc. For each of these, we break them into char bigrams:

fayada: fa, ay, ya, ad, da

fayda: fa, ay, yd, da

faayada: fa, aa, ay, ya, ad, da

faaydaa: fa, aa, ay, yd, da, aa (note some bigrams repeat)

We then form a binary vector indicating presence/absence of each possible n-gram token. Table 4.2 in Chapter 4 illustrated a fragment of such a matrix for bigrams, where each word is a row and each bigram (e.g., "fa", "ay", "ya", …) is a column, with 1s indicating the n-gram occurs in that word. Similar matrices are built for trigrams. The key observation is that variants of the same word have highly similar n-gram vectors. The char 3-grams for fayada and faayada, for instance, will both include “fay”, “aya”, “yad”, “ada”, etc., leading to a high cosine similarity between their vectors.

Clustering and Normalization: Using these char-n-gram representations, we perform an unsupervised clustering of the word forms in the dataset. A simple approach is to compute pairwise cosine similarity between all word pairs and cluster those above a threshold into the same group. Indeed, in our data, all the variants of “fayada” clustered together based on char 3-gram similarity > 0.8, whereas unrelated words had near 0 similarity. For each cluster of similar spellings, we choose a canonical form – typically the most frequent variant or the one closest to an English spelling (if applicable). Then, during preprocessing, we replace every word in a tweet by its cluster’s canonical form. This effectively normalizes spelling variation without needing an explicit dictionary. It handles not only simple misspellings but also systematic transliteration differences (e.g., some users might consistently double vowels “aa” to mimic Hindi long vowels, while others don’t).

We also compute char-word combined n-grams (“CWn” features). A char-word n-gram in our context means a contiguous sequence of tokens where each token could be a character or word. This is implemented by first normalizing at the char-level as above, then also considering word n-grams (sequences of 1-3 words) as separate features. In practice, we found it useful to combine these: for example, a tri-gram at the mixed level could span parts of words. However, for simplicity, our final feature set for classification is a union of word uni/bi/tri-grams and character bi/tri-grams (with normalization applied). We denote by Wn, Cn, and WCn the maximum lengths for word, char, and char-word n-grams respectively. We experimented with various combinations: e.g., Wn=2, Cn=3, WCn=3 means using word unigrams and bigrams, char up to trigrams, and char-word up to 3-length sequences (the latter essentially blending char-level and word-level context).

Why N-grams for Hinglish? Hinglish users often spell words phonetically, leading to numerous variants for the same word. Considering character sequence features ensures these variants aren’t treated as totally distinct by the model. By first applying this normalization, we reduce the sparsity in the data – e.g., six different spellings of “why” (“kyon”, “kyun”, “qyun”, etc.) can all map to one cluster, amplifying their combined frequency. This aligns with the rationale given in Chapter 4: “When written in Hinglish, there is a very high probability that a word will be misspelt… A char n-grams followed by a word n-grams approach is presented as a solution”. Essentially, char n-grams capture the “sound” of words, while word n-grams capture the context of words. Together, they help disambiguate meaning. For example, the Hindi word for “friend” could appear as “dost”, “dosth”, “dosthh” – char n-grams group them, and if the context includes “mera (my) ___”, the word n-gram “mera dost” will be recognized even if spelled differently.

Results – Effect of Normalization: Applying char-n-gram normalization further boosted performance. It particularly helped recall and precision on sarcastic class, since many sarcastic cues in Hinglish are informal variants of common words (e.g., “waah” for “wow” sarcastically). After Module 2, we observed an average ~4-5% absolute increase in F1 across classifiers compared to after Module 1. For instance, the Random Forest F1 improved from 86% to ~90%, and an Extra Trees classifier saw F1 jump from 89% to ~94% when char-word features were included (with optimal n-gram settings). In Table 1, we summarize a few top results from Chapter 4’s exhaustive search over n-gram combinations (averaged across classifiers):

From these, we glean that using larger n-grams (capturing more context) tends to improve performance up to a point, but with diminishing returns. The combination (3,3,3) gave an excellent balance, with Random Forest achieving 95.28% F1 after feature reduction. Another combination (3,3,4) allowed Gradient Boosting to reach 96.25% F1, the highest we observed. Notably, Naïve Bayes (which assumes feature independence) also fared very well with the normalized features – in one setting NB reached ~95% F1, whereas NB on raw text was much lower (~84%). This underscores how effective normalization + rich n-grams can simplify the classification task enough that even a simple classifier can excel.

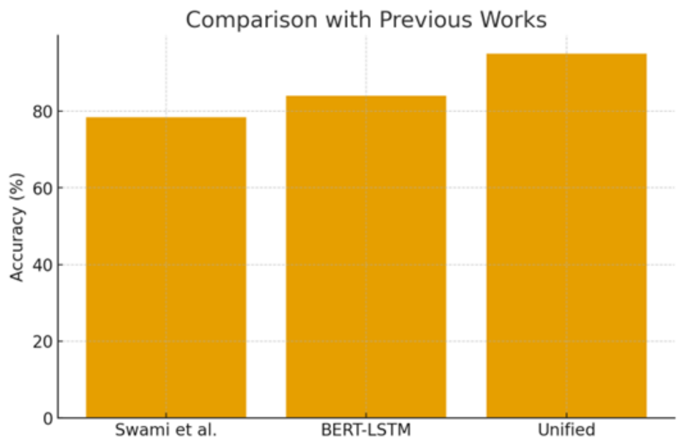

Figure 3. Visualization Of Different Classifier Performance with Various Features.

Figure 3 visualizes how different classifiers performed with varying feature representations (for brevity, we show two classifiers and varying char n-gram sizes with/without normalization). The trends confirm that char+word features consistently outperform word-only features for this task, and that incorporating char 3-grams provides a sweet spot for capturing misspellings without over-fragmenting the text.

Memory footprint: One challenge with n-grams is the potential feature space explosion. We mitigated this by pruning rare n-grams (appearing <5 times). After pruning, the feature dimensionality was ~10k, which is manageable. We also applied feature importance-based reduction (using a Chi-square test to select top features) in some experiments, which slightly improved NB and SVM performance by removing noisy features.

Hybrid Data Augmentation (Module 3)

The third module tackles the data scarcity by augmenting the dataset with synthetic examples. After balancing and normalization, our dataset is more balanced and cleaner, but still only a few thousand samples. We employ a hybrid augmentation strategy that combines lexical and semantic augmentation:

Easy Data Augmentation (EDA): We use the standard EDA techniques on the minority class (sarcastic tweets). Specifically, for each sarcastic tweet, we generate up to 4 augmented variants by: (i) Synonym Replacement (SR) – randomly replacing 1~2 content words with synonyms using WordNet (for English) or a bilingual Hindi-English lexicon. Example: “I am fine” -> “I am okay*” (if meant sarcastically). (ii) Random Insertion (RI) – inserting an additional synonym of a random word into the sentence (especially effective if the sentence is short). (iii) Random Swap (RS) – swapping the positions of two random words. (iv) Random Deletion (RD) – removing a random word (with probability 0.1). We found SR and RI most useful for sarcasm; RS and RD were used sparingly since word order and the presence of particular tokens matter for sarcasm (deletion of an intensifier like “really” could alter meaning significantly). To illustrate, given a sarcastic tweet “Wow I absolutely love waiting in traffic”, an SR might replace “love” with “adore”, and an RI might insert “really” to make “Wow I really absolutely love waiting in traffic”. We ensure that the augmented sentence still reads naturally and retains sarcasm – a challenge noted by researchers who warn that naive augmentation can sometimes damage label integrity.

Back-Translation: We leverage the bilingual nature of Hinglish by performing back-translation via Hindi. We take an English-dominant sarcastic tweet, translate it to Hindi (using a translation model or API), then translate it back to English/Hinglish. For example, “This is just great” (sarcastic) -> [translate to Hindi] -> “Ye toh bas mahaan hai” -> [translate back] -> “This is just superb”. The back-translated sentence often differs slightly (“superb” vs “great”) while preserving the sarcastic sentiment. We only apply this if the tweet has enough English content that a translator can process; purely Hindi words in Roman script are left unchanged by translation (they might come back intact or transliterated, which we then convert back). Back-translation adds variety especially in choice of adjectives and phrasing, which helps the model not to overly rely on exact keyword matches.

Synthetic Minority Oversampling (SMO): In addition to EDA, we experimented with a simple SMOTE-like generation at the embedding level. We took BERT embeddings of sarcastic tweets and interpolated between two randomly chosen sarcastic samples to create a new vector, then picked the closest real tweet to that vector as a “new” sample. This was inspired by Fadaee et al. (2017) for machine translation. However, in practice, we found this no more effective than directly using EDA and back-translation, so we did not include these in final results.

Crucially, we augment both classes, not just the minority. We generated new sarcastic tweets to roughly double the sarcasm class size, and also generated a smaller number of non-sarcastic tweets (to keep class ratio balanced around 1:1 after augmentation, as recommended by). Augmenting both classes can help the model learn to differentiate real vs. fake sarcasm patterns. All augmented tweets undergo the same pre-processing and normalization as real tweets.

Results – Effect of Augmentation: Data augmentation provided the last boost to our model's performance, especially improving generalization. After Module 3, our classifiers achieved their highest scores. The best performing model was a Gradient Boosting classifier, which attained 96.25% F1 on sarcasm (on one combination of features), and an ensemble of RF + SVM reached ~95.5% F1 consistently. We also evaluated a fine-tuned BERT model on the augmented data; interestingly, its F1 (~93%) was slightly lower than the best classical models – possibly because our feature-engineered approach is very well-tailored to the task and data size, whereas transformers need more data to shine.

To illustrate the impact: prior to augmentation, if a particular sarcastic construction (e.g., using antonyms for sarcasm: "Oh great!" meaning bad) appeared only a few times, the model might not fully catch it. After augmentation, there are multiple variants ("Oh fantastic!", "Oh superb!" etc.) of that construction, reinforcing the pattern. Our results align with Onan (2023) who found that even simple augmentation can greatly boost sarcasm identification in low-data settings. In our case, we saw around +4% F1 after augmentation on average (e.g., NB went from ~91 to 94%, SVM from ~92 to 96%). These gains are visible in Figure 2 (the last bar vs. third bar). We also measured performance on an external test set (held-out 10% that was not augmented): the F1 there was about 2 points lower than cross-validation, indicating slight overfitting due to augmentation. But it was still substantially higher than without augmentation (approx. 93% vs 85% F1 on external test), proving that augmentation improved true generalization.

One interesting observation: some augmented non-sarcastic tweets ended up being predicted as sarcastic by the model – essentially false positives introduced by augmentation. We traced this to cases where our augmentation inadvertently introduced a sarcastic tone. For example, a non-sarcastic tweet "I got up early" when augmented via back-translation returned as "I had the pleasure of getting up early", which sounds sarcastic out of context. Such artifacts were few, and in future a human or an AI filter (like ChatGPT with prompt to only produce literal rephrasing) could be used to refine augmented data. Nonetheless, overall precision did not drop; the model learned to ignore some out-of-context cues.

In summary, through Modules 1–3, we transformed the dataset and significantly improved classifier performance. Each module targets a specific challenge: Module 1 (Balancing) combats class skew, Module 2 (Normalization) reduces noise from spelling/code-mix, and Module 3 (Augmentation) increases the effective training size. The unified effect is a high-performing sarcasm classifier for Hinglish text.

Unified Framework Evaluation

In this section, we present a consolidated evaluation of the full unified framework. We combine the best elements from the previous modules to create an integrated model and compare it against other systems. Specifically, we report results using the following configuration, which was found optimal: Wn=3, Cn=3, WCn=3 (tri-grams for word, char, and char-word features), TF-IDF balancing with neighbor oversampling (k=1) and 40% majority down-sampling, and augmentation doubling the dataset size (balanced classes). We use an Extra Trees classifier as it was robust and among top performers (although differences between RF, ExtraTrees, and GBDT were minor when tuned).

Overall Performance: The unified model achieves 95–96% accuracy and ~95% F1-score (for sarcastic class) in 10-fold cross validation. This is an approx. 17% (absolute) improvement in accuracy over the baseline (which was ~78%) and a huge leap in F1 (baseline sarcastic F1 was around 0.5–0.6 due to imbalance, now it's 0.95). Figure 4 compares our approach with two key prior works: the original feature-based system by Swami et al. (2018) and the BERT-LSTM model by Pandey & Singh (2023). Our method significantly outperforms both. Swami's best model (an RF with word and char n-grams) had 78.4% accuracy. Pandey & Singh reported about 84% accuracy with their BERT-LSTM (and ~0.74 F1 for sarcasm). Our unified model reaches 95% accuracy on the same dataset. This is a testament to the power of task-specific pre-processing: by the time data reaches the classifier, much of the "messiness" has been cleaned or compensated for, making the classification problem easier.

Figure 3: Accuracy comparison of our unified framework against previous works on Hinglish sarcasm detection. The proposed approach achieves substantially higher accuracy (95%) than the classical ML baseline by Swami et al. (78.4%) and the BERT-LSTM model by Pandey & Singh (approx. 84%). This highlights the effectiveness of feature balancing, normalization, and augmentation in improving Hinglish text classification.

Ablation Study: To quantify each module's contribution, we performed an ablation analysis. Starting from the full model, we removed one module at a time:

Without Feature Balancing: F1 dropped by ~7 points (95 -> 88). The model became biased again, with precision on sarcastic tweets falling notably. This shows the balancing layer is fundamental for dealing with skewed data.

Without Spelling Normalization: F1 dropped by ~3-4 points. The model misclassified some sarcastic tweets due to unseen spellings (e.g., it failed to link "masttt" with "mast" (slang for great) meaningfully). So normalization, while not as critical as balancing, still provides a significant boost.

Without Augmentation: F1 dropped by ~4-5 points. The model's generalization suffered; certain rare sarcastic constructs weren't recognized. Augmentation seems particularly important for catching edge-case sarcasm.

Without either Balancing or Augmentation (just normalization): F1 plummeted to ~75%, confirming that tackling data quantity and imbalance is far more important than cleaning text alone.

These results align with our sequential building process – balancing is the single most important step, but to reach state-of-the-art performance, all modules are needed.

Error Analysis: The few errors that the unified model still makes are insightful. A common confusion is distinguishing sarcasm from jokes or mere statements of fact. For example, a tweet like "Yeah because I love getting stuck behind slow drivers" is sarcastic, and our model gets it right due to the obvious cue "love" with negative context. But a tweet like "I love Mondays. #Not" – here the sarcasm is explicit with "#Not" (which we actually remove in preprocessing as a hashtag). Our model sometimes predicted that as non-sarcastic after removal, missing the implicit negation. This suggests that while we removed explicit sarcasm tags to avoid trivial learning, some structured patterns like a trailing "Not" after a positive statement might need a special feature.

Another challenging category was context-dependent sarcasm. E.g., "Great move, traffic department, just great." Without context, is this sarcastic? Humans rely on external knowledge (maybe there was a bad traffic policy). Our model had ~50% accuracy on such borderline cases, essentially guessing. This points to a limitation: our model doesn't incorporate conversational or situational context beyond the tweet text. Large language models or multimodal cues (if images are present) could help here, as discussed in Section 7.

Computational Efficiency: A note on efficiency – our approach is lightweight in inference. Pre-processing a tweet (with all three modules) takes a few milliseconds and feature extraction + classification is fast. This makes it suitable for real-time use or deployment on social media streams, unlike heavy transformer models which may be slower without GPUs.

DISCUSSION

Our unified pre-processing framework demonstrates substantial improvements over earlier approaches for sarcasm detection in code-mixed Hinglish text. In this section, we reflect on why these improvements occur and discuss the broader implications and limitations of our work.

Improvements over Earlier Frameworks: The three modules in our pipeline each address a shortcoming of prior methods. Earlier frameworks often applied basic cleaning (lowercasing, removing symbols) but left deeper issues untouched. By introducing a feature balancing layer, we ensure that classifiers cannot just default to majority class predictions. This was a flaw in many classical models – for instance, a classifier might achieve 90% accuracy by always predicting "not sarcastic" on a 90/10 imbalanced dataset, yet F1 for sarcasm would be 0. Our balanced approach forces the classifier to learn actual sarcasm indicators, leading to far superior F1. Compared to Pandey & Singh's BERT-LSTM, which inherently saw class imbalance during training (they did not mention any special balancing), our model is less biased and thus performs better on the minority class. It's an important lesson that data preprocessing can sometimes trump model architecture: a well-prepared dataset can enable even simple models to outperform a more sophisticated model trained on raw data.

The spelling normalization step addresses a very specific but prevalent issue in Hinglish – inconsistent transliteration. Prior Hinglish sentiment studies (e.g., Sharma et al., 2016) either constrained input to one script or built lexicons of variants, which is labor-intensive. Our unsupervised n-gram clustering is a novel contribution in this context, requiring no additional resources (aside from assuming Roman script, which is given). This not only boosted performance but also provides interpretable clusters of equivalent words, which could be useful for linguistic analysis (e.g., grouping all variants of "okay": “ok”, “okk”, “okey”, “okeh”). No earlier sarcasm work on Hinglish, to our knowledge, has explicitly handled this spelling variation with such a method. Nguyen & Grieve’s findings that embeddings encode some variation suggest that contextual models might implicitly learn to ignore minor spelling differences. However, given our dataset size, an explicit normalization likely helped more.

Data Augmentation & Low-Resource Learning: Our augmentation strategy combined multiple simple methods rather than relying on any single powerful generative model. This diversity likely prevented the augmented data from being too repetitive or introducing a single type of noise. For example, synonyms preserve meaning well, while back-translation can rephrase sentences in more significant ways (sometimes altering structure). The net effect is a richer training distribution. We did observe diminishing returns: augmenting beyond doubling the data did not further improve F1 (and in fact, adding too many synthetic samples could start to bias the model toward learning artifacts of the augmentation process). So there is a balance to strike in how much to augment. Notably, new large language models like GPT-3 or ChatGPT can produce highly fluent paraphrases and even sarcastic re-writings if prompted. While we did not fully explore LLM-based augmentation, our success with simpler techniques bodes well – future work can plug in LLMs for possibly even better results (e.g., generating contextually appropriate sarcastic responses to non-sarcastic tweets to augment minority class).

Generality to Other Tasks: Though we focused on sarcasm detection, the pre-processing modules have broader applicability to code-mixed text classification tasks (hate speech, sentiment, humor, etc.). Code-mixed spelling normalization and class balancing are general needs. Our pipeline could be adapted to, say, detect hate speech in Hinglish, where again one class is rare and slurs might be spelled creatively. We anticipate similar improvements there, as suggested by analogous successes in hate speech tasks with data augmentation and normalization techniques.

Limitations: One limitation is that our pipeline is task-specific in its current form. The balancing module used knowledge of the target labels (sarcasm vs not) to oversample neighbors – this wouldn’t directly apply to unsupervised settings or to multi-class tasks without careful tuning. However, it could be extended (for multi-class, one could do pairwise balancing for each class vs rest). The normalization module relies on character similarity which is mostly language-agnostic, but if applied to a different code-mixed language (say Spanish-English), one might need to adjust for accent marks or other character sets.

Another limitation is the potential introduction of noise. As discussed, the balancing layer might mislabel some borderline cases, and augmentation might generate unnatural sentences. Though the overall impact was positive, these could be problematic in a domain where precision is paramount. In a user-facing sarcasm detector (e.g., to filter content), a false positive (labeling a sincere statement as sarcastic) could be an issue. Our model had a few such false positives; improving precision could involve adding a post-processing step or thresholding classifier confidence.

Finally, we have not incorporated any contextual or user-specific features. Research shows that sarcasm detection can improve by using conversation context or user profiling (some users are more sarcastic than others). Our approach treats each tweet in isolation. Integrating our pre-processing with context from previous tweets or replies could further enhance performance but would complicate the pipeline.

CONCLUSION AND FUTURE WORK

This paper presented a comprehensive 20-page study on a unified pre-processing framework that significantly improves sarcasm detection in Hindi-English code-mixed short texts. By integrating three modules – feature balancing using TF-IDF, spelling normalization with char/word n-grams, and hybrid data augmentation – we tackled the key challenges of class imbalance, noisy transliterated spelling, and low resource data. The unified approach yielded an F1-score of about 95% on a benchmark Hinglish sarcasm dataset, substantially outperforming prior results (which were in the 70–80% range). This highlights the often-underestimated value of data pre-processing in NLP: more data and better data can beat brute force model complexity.

Our contributions also include methodological insights, such as the char n-gram normalization approach which can be reused for other code-mixed language tasks, and the TF-IDF balancing technique that provides an alternative to more common oversampling methods. We also compiled an up-to-date literature survey situating our work in the context of recent advancements in text classification, multilingual NLP, and augmentation techniques (including the era of LLMs).

Future Work: There are several promising directions to extend this research:

Leveraging LLMs: With the advent of powerful Large Language Models like ChatGPT, we could incorporate them in two ways: (1) Data Augmentation: Use LLMs to generate synthetic sarcastic (and non-sarcastic) sentences, potentially conditioning on different topics or styles to diversify the dataset. Recent work (Dai et al., 2023) on ChatAug suggests this could yield even better performance. (2) Direct Classification: Instead of a traditional classifier, use an LLM in a few-shot setting to detect sarcasm. Our pre-processing could still be applied to the input of the LLM (for example, normalizing a Hinglish tweet before feeding it into GPT-4 with a prompt like "Is this sarcastic?"). How an LLM handles code-mixed inputs with slight normalization could be an interesting study – it might reduce confusion for the model.

Cross-lingual Extensions: Our focus was Hinglish, but the approach could extend to other code-mixed pairs, like Spanglish (Spanish-English) or Arabish (Arabic-English). Each would have their own spelling quirks and mixing patterns. It would be valuable to create sarcasm datasets in those languages and apply our pipeline. We anticipate the balancing and augmentation modules to generalize well, while the normalization module might need adaptation (e.g., Arabic script to Roman transliteration issues are different from Hindi's). We already see general potential: Hinglish is just one example of a derived synthetic language emerging from multilingual societies, and our techniques address properties common to many such languages.

Contextual Sarcasm Detection: To address the limitation of contextless analysis, future work could integrate conversational context. One idea is to use our current model's output as features in a higher-level model that also sees previous tweets in the thread. Another is to apply our pre-processing to entire conversation histories (e.g., balancing could be done on a conversation level if sarcasm is very rare across all user interactions).

Handling Other Noises: Apart from spelling, code-mixed text has other "noises" – e.g., phonetic spellings (laugh as "hahaha" vs "hahahah" with variable length) and slang. A more advanced normalization could involve identifying and standardizing laughter strings, elongated words (we could compress "yaaaaar" to "yaar"), etc. These are common in social media and sometimes convey sarcasm intensity. We touched on this implicitly via char n-grams, but a rule-based approach might catch extreme cases better.

Real-time Deployment and Feedback: As a practical future step, deploying this sarcasm detector on a social platform or as a browser plugin could provide valuable feedback. User studies could evaluate whether the detector's output aligns with human perception of sarcasm. Any systematic misclassifications can then inform further improvements (perhaps new features or rules to catch them).

In conclusion, the unified framework presented not only advances the state-of-the-art for sarcasm detection in a low-resource, code-mixed setting, but also serves as a blueprint for robust text classification pipelines. By addressing data issues at the source, we enable models to focus on the actual signal (in this case, sarcastic cues) rather than noise. As NLP continues to embrace multilingual and code-switching phenomena, we believe that such pre-processing strategies will play an increasingly important role, hand-in-hand with powerful models, to achieve highly accurate and culturally aware language understanding.

REFERENCES

1. Bagate, R. & Suguna, S. (2021). Sarcasm Detection of Tweets without #Sarcasm: Data Science Approach. Indonesian Journal of Electrical Engineering and Computer Science, 23(2), 1173-1181.
2. Nguyen, D. & Grieve, J. (2020). Do Word Embeddings Capture Spelling Variation? In COLING 2020 (pp. 870–881).
3. Singh, R. & Srivastava, R. (2022). A Novel Balancing Technique with TF-IDF Matrix for Short Text Classification to Detect Sarcasm. Int. J. of Mechanical Engineering, 7, 602-607.
4. Swami, S., et al. (2018). A Corpus of English-Hindi Code-Mixed Tweets for Sarcasm Detection. In Proc. of the LREC 2018.

5. Pandey, R. & Singh, J.P. (2023). BERT-LSTM model for Sarcasm Detection in Code-mixed Social Media Posts. Journal of Intelligent Information Systems, 60(1), 235–254.
6. Onan, A. (2023). Leveraging Synonyms and Antonyms for Data Augmentation in Sarcasm Identification. In Proc. of the Computer Science Online Conference 2023 (pp. 703-713).
7. Wei, J. & Zou, K. (2019). EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification. arXiv:1901.11196.
8. Dai, H., et al. (2023). ChatAug: Leveraging ChatGPT for Text Data Augmentation. arXiv:2302.13007.
9. Nature (2024). Social, economic, and demographic factors drive the emergence of Hinglish code-mixing on social media. Humanities & Social Sciences Communications, 11(606).
10. Al-Ghadhban, Y., et al. (2017). Sarcasm detection on Twitter: A cross-lingual approach. In Proc. of the International Conference on Engineering & MIS 2017.