AN EFFICIENT MACHINE LEARNING FRAMEWORK FOR ROMAN URDU SENTIMENT ANALYSIS USING HYBRID WORD–CHARACTER TF-IDF FEATURES

Abdul Hannan [1,*], Layeq Ali [2] and Rao Muhammad Asif[1]

[1]Department of Electrical Engineering, Superior University Lahore; rao.m.asif@superior.edu.pk
[2]Department of Computer Science, The University of Lahore, Lahore; layeq_1@hotmail.com
*****Correspondence: abdulhannan1757@gmail.com; Tel.: +923377482221

ABSTRACT: The problem this study focuses on is sentiment analysis on Roman Urdu text due to its low resource status coupled with high informality and inconsistencies associated with the language. Specifically, Roman Urdu lacks standardized spellings and exhibits significant use of code mixing with English. Therefore, the main goal of the study is developing a computationally affordable machine learning model for performing sentiment analysis on Roman Urdu text. The research suggests applying a hybrid strategy of feature extraction based on term frequency-inverse document frequency (TF-IDF) word-level and character-level vectors to overcome issues related to spelling inconsistency and capture semantic context simultaneously. For the purpose of building different classifiers, the data set with more than 22,000 Roman Urdu reviews was employed. Four classifiers were selected for analysis – logistic regression, linear support vector machine, naïve Bayes, and random forest. Besides, a hard voting ensemble classifier was built to utilize advantages of individual classifiers. The results revealed that logistic regression had the highest individual accuracy (78.40%) whereas the ensemble classifier had the best overall result (79.19%).

Keywords: Sentiment Analysis; Machine Learning; Term Frequency-Inverse Document Frequency (TF-IDF)

(Received 04.11.2025 Accepted 15.12.2025)

INTRODUCTION

Background and Motivation: Due to the growing popularity and widespread use of social networks, e-commerce sites, discussion groups, and other digital platforms, there is a lot of information generated in the form of text by their users. Getting useful information from such data sets poses a serious problem for both companies and researchers [1]. Sentiment Analysis can be defined as one of the important tasks of Natural Language Processing aimed at determining and classifying emotional and subjective characteristics present in any text [2]. It is applied to different types of problems including those related to consumer feedback and decision-making [3].

With advancements in machine learning technology, there have been significant improvements in the field of Sentiment Analysis in recent years [4]. The main disadvantage of lexicon-based approaches is their inability to determine the context and variations in the language [5]. Machine Learning-based approaches based on Logistic Regression, SVM, and Naïve Bayes showed good results in dealing with multi-dimensional text data [6,7].

One of the reasons for undertaking this research stems from the problems that come with analyzing languages that do not have enough resources and are not formal, such as Roman Urdu [8]. Since Roman Urdu is not a language with fixed spelling rules, grammar, and standardized vocabulary, there are certain challenges in analyzing it [8]. Additionally, the problem of code-mixing with English makes things more difficult [8].

Literature Review: Sentiment analysis has come a long way within the last ten years, shifting from conventional machine learning to advanced deep learning and transformers. A comprehensive systematic review of current research reveals that there is a clear trend towards a move away from the use of lexicons in sentiment analysis towards using artificial intelligence, especially deep learning and transformers, owing to their effectiveness in understanding the context and semantics of the text [9]. Nonetheless, machine learning remains highly applicable due to its simplicity and efficiency.

Machine learning algorithms like Support Vector Machines, Logistic Regression, Naïve Bayes, and Random Forest have been extensively employed for sentiment analysis. In particular, the combination of machine learning algorithms with solid feature engineering strategies has demonstrated high levels of efficiency. For instance, recent studies reveal that support vector machine-based models usually yield an average accuracy level of 80% or higher in sentiment classification tasks [10]. At the same time, Logistic Regression and Naïve Bayes have shown themselves as efficient approaches to text classification that are suitable for dealing with limited computational capacities [11].

The choice of appropriate features is a key factor in improving the efficiency of sentiment classification

models. One of the most common approaches to feature engineering in natural language processing is based on transforming textual information into numeric representation using techniques such as term frequency-inverse document frequency. The technique allows machine learning algorithms to efficiently distinguish relevant and irrelevant terms based on their importance within the documents and the whole dataset. In particular, recent evidence proves that the inclusion of more sophisticated features can lead to significant improvement in machine learning-based classification results [12].

At the same time, machine learning models like LSTM networks and CNNs have extensively been studied in the context of sentiment analysis. They can efficiently model sequences and contexts, resulting in better performance than other techniques. Nevertheless, the greatest advancements in the last few years were made thanks to the use of transformer-based models like BERT and RoBERTa. Such models use attention mechanisms to comprehend contextual interactions between words and outperform other architectures in different sentiment analysis tasks [13]. In addition to that, hybrid transformer-based architectures have proven their efficiency by combining contextual embeddings with advanced training methods [12].

Despite the advancements in deep learning and transformer networks, their deployment in low-resource languages and informal languages still poses difficulties. In languages like Roman Urdu, issues arise as a result of inconsistencies in spelling, grammatical structures, and code mixing with the English language. This creates a problem of noise and ambiguity in the data, resulting in poor performance by the algorithm. In this regard, feature extraction based on characters becomes an important step, as it is capable of handling spellings at a subwords level. Current research indicates that incorporating both word and character levels enhances the efficiency of the models [13].

In addition, deep learning models need larger training data sets along with heavy computational power, which cannot be achieved in most real-life scenarios. This problem makes the use of classical machine learning more appropriate when dealing with low resource conditions. Therefore, there is an increasing demand for building effective and efficient sentiment analysis models that provide high accuracy without using complicated architectures or heavy computational capabilities. This study will help fill this gap by integrating hybrid feature engineering in a machine learning-based system for Roman Urdu text.

Contribution: This study aims to provide an efficient and feasible method for conducting sentiment analysis on Roman Urdu through designing a framework based on the machine learning model along with hybrid feature extraction methods. This paper applies TF-IDF at two different levels—word- level and character-level—in order to be able to understand semantic meaning as well as variations in spellings in informal texts. Different machine learning models such as Logistic Regression, Linear SVM, Random Forest, and Naïve Bayes are applied to the dataset and compared in terms of their performance in both binary as well as multi-class classifications. It was proven that traditional machine learning methods are capable of performing competitively through careful selection of feature engineering methods. Through addressing difficulties in Roman Urdu and code-mixed texts, this study contributes to the development of sentiment analysis in multilingual and informal languages setting.

MATERIALS AND METHODS

Research Design Overview: In the present research, a supervised machine learning approach is used for analyzing the sentiment expressed in Roman Urdu consumer reviews. The entire experimentation process involves six steps, namely: (1) collection and investigation of the dataset, (2) preprocessing of text data and normalization, (3) creation of features using representation techniques based on frequency distribution, (4) training and testing of classical machine learning models, (5) optimization of hyperparameters using grid search cross validation, and (6) comparison with pre-trained mBERT model from the domain of deep learning. Two classification problems were defined and solved through experiments: one involving a three class problem (positive, negative, and neutral sentiments) and another with binary classification (positive vs. negative sentiments). Experiments were conducted in Python programming language using popular machine learning libraries such as scikit-learn and Hugging Face Transformers

Dataset

Source and Description: The dataset utilized for this project is called the Roman Urdu Reviews Dataset with English Translation, which is a publicly available dataset that includes customer reviews of products or services in Roman Urdu, which refers to the code-switched form of the Urdu language expressed through Latin script. This type of language variant is highly prevalent in South Asia's informal digital interactions, but at the same time, it is very underrepresented in current NLP benchmarks. Each data entry in the dataset has two main attributes – the first one is named 'ROMAN URDU REVIEWS' and provides the text content of the review, whereas the second attribute named 'SENTIMENT' provides the corresponding sentiment class. The dataset was read using the Python pandas module and its latin-1 character set encoding.

Table 1. Data set summary.

Attribute	Detail	Notes
Dataset Name	Roman Urdu Reviews Dataset with English Translation	CSV format, latin-1 encoding
Total Records (Raw)	~28,090	Before cleaning
Records After Cleaning	22,179	Post de-duplication and null removal
Text Column	ROMAN URDU REVIEWS	Code-switched Roman Urdu text
Label Column	SENTIMENT	Categorical: positive, negative, neutral
Positive Samples	11,296 (binary), 11,290 (cleaned binary)	—
Negative Samples	10,349 (binary)	—
Neutral Samples	6,445 (three-class only)	Excluded in binary classification
Language	Roman Urdu (transliterated)	With English-translated column

Class Distribution and Imbalance Considerations: Before any kind of preprocessing steps, the initial raw data showed an overall distribution according to the sentiment categories as follows: positive category (n = 11,296); negative category (n = 10,348); neutral category (n = 6,445); and one mislabeled instance ('neative', n = 1). The initial imbalance between the positive and the negative classes (approximately 52:48) was not a significant issue when performing binary classification; however, introducing the neutral sentiment class led to a significant imbalance between the three classes (approximately 40% for positive, 37% for negative, and 23% for neutral sentiment). In order to counteract this in the process of building the models, the Linear SVM classifier for binary classification problem was trained using the class_weight='balanced' parameter. Furthermore, the train/test split was done using stratification in order to achieve class proportionality in both splits.

Data Preprocessing and Text Normalization: Preprocessing Pipeline: A series of steps was undertaken in order to normalize the raw review text prior to extracting features from it. There are some specific problems associated with Roman Urdu text, which is different from regular English or native Urdu script, since it is non-standardized with spelling mistakes, numbers, etc. These are:

Table 2. Text Preprocessing Pipeline.

Step	Operation	Implementation Detail
1	Lowercasing	str.lower() applied to all SENTIMENT labels and review text
2	Label Standardization	Mapped 'very positive' → 'positive', 'very negative' → 'negative', 'neative' (typo) → 'negative'
3	Null Removal	dropna() on ROMAN URDU REVIEWS and SENTIMENT columns
4	Numeric Removal	re.sub(r'\d+', '', text) strips all numeric tokens
5	Punctuation Removal	re.sub(r'[^\w\s]', '', text) removes all non-alphanumeric characters
6	Whitespace Normalization	Collapses consecutive spaces and strips leading/trailing whitespace
7	Label Encoding	sklearn LabelEncoder: negative=0, neutral=1, positive=2 (three-class); negative=0, positive=1 (binary)

Label Normalization: The original sentiment column had different kinds of representation of sentiment categories such as mixed cases ('Positive', 'positive', 'POSITIVE'), long versions like ('very positive', 'very negative') and a misspelled category ('neative'). Initially, all categories were made lowercase while removing whitespaces on both sides of the category name. Later, replacement was performed for 'very positive' with 'positive', 'very negative' with 'negative' and 'neative' with 'negative'

Text Cleaning Function

- A custom Python function, clean_text(), was run on each review inthe data set. The function makes the following changes in that order:
- Turns all characters to lower case using str. lower() so that there is no duplicate token creation based on case (for example, 'Acha' and 'acha' would be considered the same token).
- All numeric sequences are stripped out using the regex \d+ because numeric tokens do not have any sentiment associated with them in this context.
- All punctuation and special characters are stripped using the regex [^\w\s] because Roman Urdu uses punctuation marks irregularly. This step leaves only word characters and spaces in the strings.
- Consecutive whitespace characters are collapsed to a single space using the regex \s+.

The result of this cleaning process was added as a new column called cleaned_text and was used for further vectorization and modeling. The result of this cleaning process was added as a new column called cleaned_text

and was used for further vectorization and modeling

Missing Value Handling: Once filtering of columns that are of no relevance to our project was done (the only remaining columns being ROMAN URDU REVIEWS and SENTIMENT), rows with any null values in these columns were deleted via the pandas.dropna() function. This process resulted in the data set's dimensions being decreased from an initial count of about 28,090 data points to 22,179 clean data points.

Feature Engineering: The appropriate numerical representation of text in Roman Urdu is among the key methodological issues involved in this study. Since there is limited availability of pre-trained word embeddings for Roman Urdu, the study adopts the approach of representing the text statistically via Term Frequency – Inverse Document Frequency (TF-IDF) vectorization. Two different TF-IDF feature spaces were developed and fused together through FeatureUnion to enable the learning of both sub- and full-lexical information at once. The TF-IDF vectorizer at the word level was tuned with the following settings:

- analyzer='word': Tokenization takes place using word boundaries.
- ngram_range=(1,2): This allows for both unigram (words) and bigram (consecutive pairs of words) combinations in the vocabulary. It accounts for context-based sentiment modification that cannot be captured using only the unigrams.
- max_features=15,000: The vocabulary size is limited to the top 15,000 informative words based on their TF-IDF scores, thus limiting the usage of memory and reducing the curse of high dimensionality.
- sublinear_tf=True: Logarithmic scaling of the term frequency component (tf = 1 + log(tf)) reduces the impact of extremely frequent words.

Character-Level TF-IDF Vectorizer:
The vectorizer for character n-grams was created alongside the configuration below:

- analyzer='char': Uses tokenization that operates on the character level.
- ngram_range=(3,5): Generates n-grams containing character sequences between the lengths of 3 and 5 characters. This range is highly appropriate for Roman Urdu since it takes into account morphemes, spelling differences, and common orthographic practices.
- max_features=8,000: Limited vocabulary of 8,000 character n-grams.
- sublinear_tf=True: Uses logarithmic TF scaling like the vectorizer for word features.

Feature Fusion via FeatureUnion: The two feature matrices based on term frequency-inverse document frequency (TF-IDF), at both word and character levels, were concatenated through scikit-learn's FeatureUnion component. In this way, two sparse matrices of features generated by the two vectorizers were horizontally concatenated together to form an input with as many as 23,000 features per document. The benefit of such a combined representation technique is that it has proved to be superior to one feature set alone, especially in the case of low-resource and agglutinative languages.

Table 3. TF-IDF Feature Engineering Configuration.

Vectorizer	Analyzer	n-gram Range	Max Features
Word TF-IDF	word	(1,2) — unigrams & bigrams	15,000
Character TF-IDF	char	(3,5) — tri- to 5- grams	8,000
Combined (Feature Union)	word + char	As above, concatenated	23,000 total

Data Partitioning: The cleaned dataset was separated into training and testing datasets through scikit-learn's train_test_split() method. Below are the parameters employed in splitting the data during both experiments:

- Percentage of Test Set: 20% test set and 80% training set.
- Random State: Random state = 42 for all splits to achieve consistency and reproducibility of results.

Stratification: stratify=df['encoded_label'] parameter ensures that both the training and test sets have the same class distributions, which is vital due to the imbalance between classes in the three-class problem.

For the three-class problem, this resulted in 22,472 training instances and 5,618 test instances. For the binary classification problem where neutral class was not considered, there were 17,316 training instances and 4,329 test instances. This result was corroborated by another diagnostic Train size: 17316 Test size: 4329.

Classification Models: Four types of classical supervised machine learning classification algorithms were used in this study, all of which were included within scikit- learn Pipeline classes for standardized data processing. The choice of these four algorithms is justified by a comprehensive approach to linear, probabilistic, tree-based, and kernel-based approaches to learning.

Logistic Regression: Logistic Regression was used as an effective linear classifier for text classification. It had been set with a maximum number of iterations equal to 2,000 (max_iter=2000). This approach suits well the problem of text classification, since it performs the learning of the linear decision surface in the TF-IDF representation space of documents, while the

regularization method by default is set to L2, which avoids overfitting. This algorithm returns class probabilities through the softmax activation in multi-class scenario

Multinomial Naive Bayes: For computational efficiency and probabilistic modeling purposes, Multinomial Naive Bayes (MultinomialNB) has been chosen. This classifier represents the conditional probability of all features (word or character n- grams) with respect to the class labels assuming independence between all of the features. However, Naive Bayes classifiers have traditionally been performing well on text classification despite the naive assumption of independence and especially when data size is small.

Linear Support Vector Machine: Linear SVC was chosen since it showed good empirical results in text classification tasks and had good computational performance on high- dimensional and sparse data sets. Linear SVC optimizes the separation margins of the class hyperplanes in the feature space. For the binary classification task, it was fitted with the class_weight = 'balanced' parameter in order to adjust for any remaining class imbalance by weighting samples based on their inverse proportion to the class label. Linear SVC was the main model used for the tuning procedure.

Random Forest: An ensemble classifier Random Forest was introduced to determine whether a nonlinear model using an ensemble of decision trees can perform better than linear classifiers when it comes to Roman Urdu text representation. It was implemented with n_estimators=400 and random_state=42 for consistency. An ensemble classifier like Random Forest is essentially a bagging algorithm that uses a voting mechanism to produce a collective prediction of multiple independent decision trees. Although Random Forest is less frequently used in text classification problems than linear classifiers, it helps set a benchmark for tree-based algorithms.

Hard Voting Ensemble: As an additional experiment, a Voting Ensemble classifier was made from the combination of the Logistic Regression, Linear SVC, and Random Forest classifiers by using hard voting ('voting'='hard'). In hard voting, the classification decision for each input is made based on the mode of class labels predicted by the classifiers used in the ensemble. The Voting Ensemble classifier was trained to find out if the prediction of the class label by more than one classifier could increase the classification accuracy compared to a single classifier.

Deep Learning Approach: BERT-Based Classification: In order to evaluate the performance of classical machine learning techniques with respect to the current state-of-the-art transformer model, a fine-tuning experiment was carried out with the use of a pre-trained multilingual BERT model (bert-base-multilingual-cased, called as mBERT). In particular, the selection of this model is attributed to its pre-training on 104 languages, which include family languages similar to Urdu.

Tokenization and Dataset Construction: Tokenization was performed on the texts through the BertTokenizer class of the transformers module in Hugging Face's python framework, with the following settings:

- truncation=True: Texts longer than the maximum number of tokens are truncated.
- padding=True: The smaller texts are padded with zeros up to the maximum number of tokens in the batch.
- max_length=128: Token length was kept at 128 tokens.

A special class, RomanUrduDataset, for the dataset was created in PyTorch using the encoding texts and their respective labels. This class facilitated the loading of batches with a batch_size of 16 through the DataLoader class of PyTorch.

Model Architecture and Training: Bert For Sequence Classification architecture provided by Hugging Face was used, initialized with the argument num_labels=2 (as it is a binary classification problem) from the pre-trained bert-base-multilingual-cased checkpoint. The random initialization was used for the classifier head (a linear transformation from the [CLS] token representation into the logits), whereas all other layers of the encoder were loaded from the checkpoint. The fine-tuning was run for three epochs, with the default AdamW optimizer and its learning rate schedule. As a result of limited hardware (CPU was used for training, as it is seen from the output message 'Using device: cpu'), one epoch of the training took 18–23 hours according to the training log messages.

Evaluation: The final tuned model was then tested using the remaining 20% test set (N=4,329 data points). The performance of the model was assessed using Accuracy, Precision, Recall, and F1-Score (Macro Averaging), in the same way as the traditional models were assessed.

Hyperparameter Optimization: The linear support vector machine pipeline was subjected to grid search with 5-fold stratified cross validation (GridSearchCV). Grid search was selected as the search strategy due to the small size of the parameter space associated with the LinearSVM model, which allowed for performing an exhaustive search for optimal parameters. The f1_macro score was used as the optimization metric.

The optimal parameter combination found to be C=0.5 and class_weight="balanced", which yielded the highest CV-F1 macro value of 0.7145. The best model (best_model) obtained through tuning was then tested on the unseen test data and achieved a test accuracy of 0.7421 and an F1 macro of 0.72. This was a significant

increase from the SVM with no parameters tuned.

Table 4. GridSearchCV Configuration and Result.

Hyperparameter	Values Searched	Best Value
classifier C	[0.5, 1, 2, 5]	0.5
classifier class_weight	[None, 'balanced']	'balanced'
Scoring Metric	f1_macro	—
Cross-Validation Folds	5	—
Best CV Score	—	0.7145

Evaluation Metrics: Model performance was assessed using a comprehensive set of classification evaluation metrics, computed via scikit-learn's metrics module

Accuracy: Accuracy for overall classification was calculated by measuring the ratio of correct classifications over the entire number of test samples. Although accuracy is included as a measure for all the models, it is acknowledged to be inadequate on its own when dealing with imbalanced classes.

Precision, Recall, and F1-Score (Macro-Averaged): The values for precision, recall, and F1 were calculated based on each individual class and later averaged through macro averaging, which assigns equal importance to all the classes without considering their support counts. Macro averaging takes precedence as the first metric used for evaluation since it is a stricter measure compared to weighted averaging, especially when dealing with classes that have low support counts such as neutral.

Confusion Matrix: Confusion matrices were created for each model using the confusion_matrix() function from scikit-learn, while heatmap visualization was done using seaborn. This method offered detailed information on individual class errors, allowing the detection of systematic error patterns, specifically the tendency of the classifier to classify neutral and negative classes together in the three-class scenario.

ROC Curve and AUC (Supplementary): For the tuned Linear SVC classifier in the case of a three-class problem, ROC curves and their respective AUC scores were calculated employing the technique of one vs. rest binarization through sklearn. preprocessing. label_binarize. The decision function scores from the decision_function() method served as the probabilities used to represent the model's confidence in its predictions. ROC curves were drawn separately for each class – negative, neutral, and positive

Cross-Validation: Cross validation (cv_score) was done on baseline models along with the train test split evaluation on the full dataset using the TF-IDF pipeline. This method gives a better statistical estimation of model generalization compared to the train test split because it helps reduce variance in estimation due to a train test split. The accuracies of cross-validation for Logistic regression, Naïve Bayes, and Linear SVM were 0.7017, 0.6923, and 0.6994, respectively.

RESULTS

Dataset Overview and Preprocessing: In this research, two datasets of Roman Urdu reviews were used, which were combined and preprocessed into one. One of them was a set of Roman Urdu reviews accompanied by English translations of the latter, and the other one was a dataset containing 11,000 reviews in TSV format. Once the two datasets were combined, they totaled 23,321 records. However, after conducting several preprocessing operations, such as deleting duplicate records and those with less than three words, the final dataset resulted in 22,179 observations, half of which were positive reviews and the rest negative ones.

Table 5. Dataset Statistics after Preprocessing.

Stage	Metric	Value
Raw Combined Data	Total Records	23,321
After Cleaning	Total Records	22,179
Class Distribution	Positive Reviews	11,290 (50.9%)
Class Distribution	Negative Reviews	10,889 (49.1%)
Train/Test Split	Training Samples	17,743 (80%)
Train/Test Split	Test Samples	4,436 (20%)

The process of text preprocessing included the transformation of all texts into lower case, stripping out numbers and punctuation marks, and ensuring that whitespaces were consistent. The labels were encoded using the binary code with the help of LabelEncoder from scikit-learn library (0 for Negative and 1 for Positive).

Feature Engineering: The TF-IDF representation used for the dataset involved a FeatureUnion of two TF-IDF vectorizers. The first one was at the word level and contained unigrams, bigrams, and trigrams with an ngram_range of 1 to 3. It could contain up to 40,000 features. The second vectorizer represented character ngrams from 3 to 6 characters long and included a maximum of 20,000 features. A sublinear term frequency was used in both vectorizers to reduce the significance of common terms

Individual Model Performance: The four classifiers trained on the features set include logistic regression, linear support vector machine (linear SVM), naive bayes classifier (MultinomialNB), and random forest. In all the classifiers using probabilistic margin, balancing of the class weight was employed to take care of the class imbalance that remained.

Logistic Regression provided the best accuracy score of 78.40%, with equally balanced precision and recall scores for both the negative and positive classes (negative F1 = 0.79, positive F1 = 0.78). Linear SVM

provided an accuracy of 77.32% and showed equally balanced performances. Naive Bayes provided an accuracy score of 77.14% and showed more recall for the negative class (negative F1 = 0.83, positive F1 = 0.72). Finally, Random Forest provided the least accuracy among all models, that is, 75.77%. It showed less recall in case of positive reviews (positive F1 = 0.70).

Table 6. Classification Performance of Individual Models and Voting Ensemble.

Model	Accuracy	Prec. (Neg)	Rec. (Neg)	F1 (Neg)	Prec. (Pos)	Rec. (Pos)	F1 (Pos)	Macro F1
Logistic Regression	78.40%	0.77	0.81	0.79	0.80	0.76	0.78	0.78
Linear SVM	77.32%	0.76	0.78	0.77	0.79	0.76	0.77	0.77
Naive Bayes	77.14%	0.74	0.83	0.78	0.81	0.72	0.76	0.77
Random Forest	75.77%	0.73	0.81	0.77	0.80	0.70	0.75	0.76
Voting Ensemble	**79.19%**	—	—	—	—	—	—	—

Voting Ensemble Performance: An ensemble classifier consisting of hard voting based on Logistic Regression, Linear SVM, and Random Forest models has been created using the VotingClassifier class in the Python scikit-learn module. The resulting accuracy of this ensemble classifier was 79.19%, which is greater than those of each of the individual classifiers used in this process, by 0.79% to 3.42%. The performance gain was due to the complementarity of the methods, including the linear boundary estimation of Logistic Regression, maximum margin of SVM, and tree-based decisions of the forest.

DISCUSSION

Effectiveness of Combined TF-IDF Features: Roman Urdu's lack of a uniform orthography made it very important to incorporate the character-based TF-IDF features along with the standard one. Roman Urdu does not have any established writing system in which the same word can have different spellings depending on individual users' preference (e.g., 'acha', 'acha', 'acha'). Character n-grams ranging from three to six letters compensated for this issue by identifying subword similarities in various spellings, whereas word n-grams contained sentiment-bearing phrases such as 'bohat acha' (very good) or 'bilkul bekaar' (completely useless).

Why Logistic Regression Outperformed Other Models: The success of Logistic Regression is due to its natural adaptability to text feature matrices with a large number of dimensions but sparse vectors. Sparse TF-IDF features have mostly zeros, and the decision boundary between two classes is utilized efficiently by linear classifiers such as Logistic Regression and Linear SVM. The probability output by Logistic Regression further ensures better convergence of gradient optimization, which is beneficial when many features do not correlate with class labels, which is frequently observed in under-resourced transliterated languages.

Naive Bayes model, despite being competitive in terms of classification accuracy, exhibited an asymmetric behavior. It performed better in terms of recall on negative reviews but poorer on positive reviews. The reason for this asymmetric behavior might be rooted in the statistical nature of Roman Urdu sentiment lexicons. For instance, negative words such as 'bekar,' 'ganda,' 'kharab' could be more distinctive and occur more frequently in negative reviews than in positive reviews.

The relatively low effectiveness of Random Forest (75.77%) agrees with research findings regarding NLP models: tree algorithms tend to perform poorly on highly dimensional input. In our case, given that we have 60,000 TF-IDF features, each tree uses mostly features carrying no useful information, resulting in increased prediction variance.

Gains from the Voting Ensemble: Considering that the error rate of the voting classifier has improved by about 0.79 percent when compared to the error rate of the best model (logistic regression), it can be stated that diversity helped decrease the error rate. This is because each of the models makes different types of errors. For example, logistic regression will make wrong predictions for some phrases, SVM will be affected by outliers that are close to the margins, and random forest will have difficulties with unknown combinations of words.

Interestingly, although the accuracy rate of random forest was not better than those of the other two algorithms, its inclusion into the voting ensemble did help improve the overall accuracy rate of the classifier. It seems that random forest makes right predictions in certain cases where the two other classifiers cannot.

Challenges of Roman Urdu Sentiment Analysis:

These challenges include, but are not limited to, the following:

1. Phonetic spelling variations make it impossible to ensure that semantically identical tokens are treated as distinct features, thus inflating the number of words used without adding any information.
2. Absence of a standardized Roman Urdu stopwords

list/stemmer makes it challenging to filter out non-content-bearing parts of speech.

3. Finally, sarcasm, code-switching, as well as culturally specific indirect sentiments such as praise/criticism make classification particularly challenging using only shallow features.

Notwithstanding these problems, the models performed quite accurately within the range of 75-79%, which shows that traditional machine learning algorithms with proper feature engineering could act as a baseline method for Roman Urdu sentiment classification. Moreover, the nearly balanced dataset (50.9%/49.1% split between positive and negative classes respectively) prevents accuracy from becoming an issue because of the need for extensive class balancing efforts.

Comparison with Existing Work: Accuracy achieved in this research (75.77%-79.19%) is within the range seen for baseline accuracy for similar tasks involving Roman Urdu and Urdu NLP using TF-IDF coupled with linear classification algorithms, which have been reported to produce accuracies ranging from 74% to 82% on binary sentiment classification. The utilization of features based on both word and character n-grams follows best practices in dealing with low-resource languages where characters add reliability due to inconsistent spelling in Urdu. Future research using transformer-based multilingual models like mBERT or XLM-R would help improve these results.

Conclusion: The paper presents results from an analysis of Roman Urdu reviews with respect to sentiment classification through the use of four different classifiers for machine learning, namely Logistic Regression, Linear SVM, Naïve Bayes, and Random Forest along with the use of a hybrid feature space consisting of TF-IDF features based on both words and characters as n-gram representation. Datasets were prepared by merging two freely available Roman Urdu review datasets, resulting in a corpus comprising 22,179 instances that were evenly divided for training and testing purposes.

In terms of individual models, the Logistic Regression classifier performed the best, showing an accuracy score of 78.40%, followed by Linear SVM with 77.32% accuracy, Naïve Bayes with 77.14% accuracy, and Random Forest with 75.77% accuracy. An ensemble classifier called the Voting Ensemble, consisting of the Logistic Regression, Linear SVM, and Random Forest classifiers, performed best among all others, scoring an accuracy of 79.19%.

These findings have verified that the use of TF-IDF for feature engineering is an effective strategy when dealing with the orthographic variability present in Roman Urdu data. Character n-grams were especially useful in detecting patterns within subwords, which are likely to appear due to phonetic transliterations. Word n-grams were helpful in providing sentiment information at the phrase level.

The findings of this study add value to the existing literature on the NLP techniques used for low-resource and code-switched South Asian languages. They provide a strong baseline in classical machine learning, which can be taken into consideration for future studies using deep learning and transformers. Specifically, multilingual pre-trained models, such as mBERT and XLM-RoBERTa, are expected to be useful in enhancing the performance of Roman Urdu sentiment analysis in the future.

Abbreviations:

AI	Artificial Intelligence
AUC	Area Under Curve
BERT	Bidirectional Encoder Representations from Transformers
CPU	Central Processing Unit
CV	Cross Validation
F1	F1-Score
LR	Logistic Regression
LSTM	Long Short-Term Memory
mBERT	Multilingual BERT
ML	Machine Learning
NB	Naïve Bayes
NLP	Natural Language Processing
POS	Part of Speech
PyTorch	Python Torch (Deep Learning Framework)
RF	Random Forest
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
TF	Term Frequency
TF-IDF	Term Frequency–Inverse Document Frequency
TSV	Tab-Separated Values
CSV	Comma-Separated Values
XLM-R	Cross-lingual Language Model – RoBERTa

REFERENCES

[1] Paul, A., Sharma, R., & Verma, S. (2023). Machine learning approaches for sentiment analysis on textual data. *International Journal of Advanced Computer Science and Applications, 14*(2), 112–120.

[2] Karim, F., Rahman, M., & Islam, S. (2022). Comparative analysis of machine learning algorithms for sentiment classification. *IEEE Access, 10*, 55678–55689. https://doi.org/10.1109/ACCESS.2022.3156789

[3] Said, M., & Ismail, A. (2025). Challenges in sentiment analysis for low-resource and code-mixed languages. *Journal of Natural Language Processing, 32*(1), 45–60.

[4] Analytical Assessment. (2025). Trends in sentiment analysis: From lexicon-based to deep learning approaches. *International Journal of*

Artificial Intelligence Research, 9(1), 1–15.

[5] Khan, S., Ali, Z., & Ahmad, T. (2024). Performance evaluation of support vector machines in sentiment classification. *Expert Systems with Applications, 215*, 119876.

[6] Hamid, R., & Abdulazeez, A. (2024). Efficient text classification using logistic regression and naïve Bayes. *Applied Sciences, 14*(3), 1456. https://doi.org/10.3390/app14031456

[7] Semary, N., Hassan, M., & Elsayed, A. (2023). Hybrid feature engineering for improved sentiment analysis performance. *Knowledge-Based Systems, 260*, 110123.

[8] Sayeed, A., Rahman, M., & Karim, M. (2023). Transformer-based architectures for sentiment analysis: A review. *ACM Computing Surveys, 56*(5), 1–36.

[9] Jim, E., Chen, L., & Wong, K. (2024). Character-level and word-level feature fusion for low- resource language processing. *Neural Computing and Applications, 36*, 5678–5695.

[10] Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2021). Deep learning-based text classification: A comprehensive review. *ACM Computing Surveys, 54*(3), 1–40. https://doi.org/ 10.1145/ 3439726

[11] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. *Information, 10*(4), 150. https://doi.org/ 10.3390/info10040150

[12] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 4171– 4186.

[13] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.