Named Entity Recognition for Hindi-English Code-Mixed Social Media Text

Abhinav Kumar, Anand Keshav, Gholap Sarvesh Sarjerao, Priyanshu Mahawar

I. Problem Statement

Named Entity Recognition (NER) is a core task in Natural Language Processing (NLP) and a subtask of Information Extraction. The main challenge in NER for tweets is that they often lack sufficient information due to the short, informal, and unstructured nature of tweets, especially when code-mixed languages like Hindi-English are involved. Code-mixing complicates NER by combining linguistic units from different languages within a single utterance. This study proposes experiments using machine learning algorithms with word, character, and lexical features to tackle NER in Hindi-English code-mixed tweets.

II. Preprocessing

To handle the noisy, informal nature of Twitter data, extensive preprocessing was carried out to prepare the corpus for NER. Key steps include:

A. Noise Removal

The following steps were used to remove noise from the dataset:

- **Non-Informative Tweets:** Tweets containing only hashtags, URLs, or minimal meaningful content were discarded.
- **Language Filtering:** Tweets containing languages other than Hindi or English were identified as noisy and removed.
- **Script Filtering:** To maintain consistency, tweets entirely in English or using Devanagari script were excluded, keeping only those with Romanized Hindi-English code-mixing.

B. Tokenization and Normalization

To standardize the text:

- **Tokenization:** Each tweet was split into tokens, handling special cases for hashtags, emojis, and URLs.
- **Transliteration:** Hindi tokens in Devanagari script were transliterated to Roman script, ensuring uniform text representation.
- **Removal of Special Characters:** Unnecessary punctuation and symbols were removed to clean up the text.

III. Dataset

A. Data Collection and Filtering

Using the Twitter API, we initially retrieved 110,231 tweets. After a manual filtering process to retain only relevant code-mixed tweets, the final dataset comprised 3,638 tweets. Then we annotated the dataset of 3,638 code-mixed tweets using Named Entity (NE) tags to classify each token. The annotation process resulted in a total of 68,506 labeled tokens. The annotation scheme followed the standard CoNLL format and included three primary NE tags:

- **Person (Per):** Used to tag mentions of individuals.
- **Organization (Org):** Used to tag references to organizations.
- **Location (Loc):** Used to tag geographic locations.

To ensure comprehensive tagging, the dataset also included tags for entities beginning with ”B-” (Beginning) or ”I-” (Inside) based on their position in multi-token entities, and a seventh tag labeled *Other* for non-NE elements. The tag distribution in the dataset is displayed in Table I.

TABLE I: Tag Distribution in the Annotated Dataset

Tag	Count of Tokens
B-Loc	762
B-Org	1,432
B-Per	2,138
I-Loc	31
I-Org	90
I-Per	554
Total NE tokens	**5,007**

B. Inter Annotator Agreement (IAA)

The dataset annotation was conducted by two annotators proficient in both Hindi and English. We calculated the Inter Annotator Agreement (IAA) to ensure high annotation quality. Using Cohen’s Kappa coefficient, we obtained high agreement scores for each tag, as shown in Table II. Notably, the agreement for ”I-Per” and ”I-Org” tags was slightly lower due to ambiguous or infrequent names.

TABLE II: Inter Annotator Agreement for NE Tags

Tag	Cohen’s Kappa
B-Loc	0.98
B-Org	0.96
B-Per	0.94
I-Loc	0.98
I-Org	0.91
I-Per	0.93

The annotated dataset is available online for research purposes at https://github.com/SilentFlame/Named-Entity-Recognition.

IV. Model

In this section, we explain the working of different machine learning algorithms used in experiments on our annotated dataset.

A. Decision Tree

The Decision Tree algorithm belongs to the family of supervised learning algorithms and can be used for both regression and classification tasks. The Decision Tree algorithm solves problems using a tree representation, where each internal node represents an attribute, and each leaf node represents a class label.

To predict a class label for a record, the algorithm starts from the root of the tree. It compares the values of the root attribute with the record's attribute, following the branch corresponding to that value to the next node. The primary challenge in implementing Decision Trees is selecting appropriate attributes for each node, a process known as attribute selection. Popular attribute selection measures include:

- **Information Gain:** Information Gain calculates the expected reduction in entropy due to sorting based on an attribute. The formula for calculating entropy is given by:

$$H(X) = E[I(X)] = -\sum_{x\in X} p(x)\log(p(x))$$

where $p(x)$ is the probability of a class for the feature in question. The attribute with the lowest entropy is chosen as the root, and this process is repeated for selecting features at other levels.

- **Gini Index:** The Gini Index measures how often a randomly chosen element would be incorrectly identified. Lower Gini values indicate better attribute choice. It is calculated as:

$$\text{Gini index} = 1 - \sum_{j} p_j^2$$

where p_j is the probability of a class for a given feature.

B. Conditional Random Field (CRF)

Conditional Random Fields (CRFs) are useful for sequence labeling and structured prediction tasks. In sequence labeling, such as POS tagging or NER, it is advantageous to consider correlations between neighboring labels and to jointly decode the best sequence of labels for an input sentence. For instance, in NER with BIO2 annotation, certain labels are more likely to follow others.

Given a sequence of inputs $X = (x_1, x_2, \ldots, x_m)$ representing the words in a sentence, and the output sequence $S = (s_1, s_2, \ldots, s_m)$ for named entity tags, the conditional probability is modeled as:

$$p(s_1, s_2, \ldots, s_m | x_1, x_2, \ldots, x_m)$$

This is done by defining a feature map $\Phi(x_1, x_2, \ldots, x_m, s_1, s_2, \ldots, s_m) \in R^d$, mapping the input sequence X and state sequence S to a d-dimensional feature vector. The probability is represented by a log-linear model with a parameter vector $w \in R^d$:

$$p(s|x;w) = \frac{\exp(w \cdot \Phi(x,s))}{\sum_{s'} \exp(w \cdot \Phi(x,s'))}$$

where s' ranges over all possible output sequences. The regularized log-likelihood function L is defined as:

$$L(w) = \sum_{i=1}^{n} \log(p(s_i|x_i;w)) - \frac{\lambda_2}{2}\|w\|_2^2 - \lambda_1\|w\|_1$$

The terms $\frac{\lambda_2}{2}\|w\|_2^2$ and $\lambda_1\|w\|_1$ provide regularization, reducing model complexity. The optimal parameter vector w^* is estimated as:

$$w^* = \arg\max_{w\in R^d} L(w)$$

Given w^*, we can find the most likely tag sequence s^* for a sentence x by:

$$s^* = \arg\max_{s} p(s|x;w^*)$$

C. Long Short-Term Memory (LSTM)

Recurrent Neural Networks (RNNs) process sequential data, taking an input sequence $(x_1, x_2, \ldots, x_n)$ and producing an output sequence $(h_1, h_2, \ldots, h_n)$ with information about each input step. However, RNNs struggle to maintain long dependencies, which biases them toward recent inputs. LSTM networks, introduced by Hochreiter and Schmidhuber, address this by learning long-term dependencies through specialized gates controlling memory and forgetting.

In our study, LSTMs are well-suited to handle tweets due to their ability to retain context, even though the tweets are relatively short. We modified the LSTM architecture to better capture context across sequential data, as illustrated in Figure **??**. These improvements help the model maintain important information across the sequence, which is crucial for NER tasks.

D. Gaussian Naive Bayes and Random Forest

Gaussian Naive Bayes (GNB) is a probabilistic classifier based on Bayes' theorem. It is a variant of Naive Bayes, where the likelihood of each feature is assumed to follow a Gaussian (normal) distribution. It's particularly useful for classification tasks where the input features are continuous, and it performs well when the assumptions hold true. However, it doesn't perform well in our case, as the code-mixed texts are highly contextual.

Random Forest (RF) is an ensemble learning model that constructs multiple decision trees during training and merges their outputs for classification. It generally provides robust performance across various tasks by reducing overfitting compared to a single decision tree. In our case, the Random Forest model performed well with the feature vectors and demonstrated good accuracy on both baseline and Word2Vec-embedded data, benefiting from its ability to handle complex interactions between features.

V. Ablation Study

To enhance the performance of our models, we performed hyperparameter tuning on several machine learning algorithms and experimented with various model configurations. This section provides an overview of the models and the tuning techniques we applied.

A. Hyperparameter Tuning and Feature Selection

We conducted hyperparameter tuning and feature selection for the Decision Tree, Naive Bayes, and Random Forest classifiers to improve model performance. Grid search and cross-validation were used to systematically explore combinations of hyperparameters, while feature selection was based on the features identified as optimal by the Conditional Random Fields (CRF) model, which were also effective in these classifiers.

- **Decision Tree:** We tuned parameters including maximum tree depth, minimum samples split, and minimum samples per leaf, and incorporated the most relevant features based on CRF analysis.
- **Naive Bayes:** Hyperparameter tuning included experimenting with different smoothing values to improve classification accuracy on unseen data. Key CRF-identified features were used to boost model performance.
- **Random Forest:** We optimized the number of estimators, maximum depth, and minimum samples split to achieve a balanced model, while also leveraging CRF-selected features for enhanced classification accuracy.

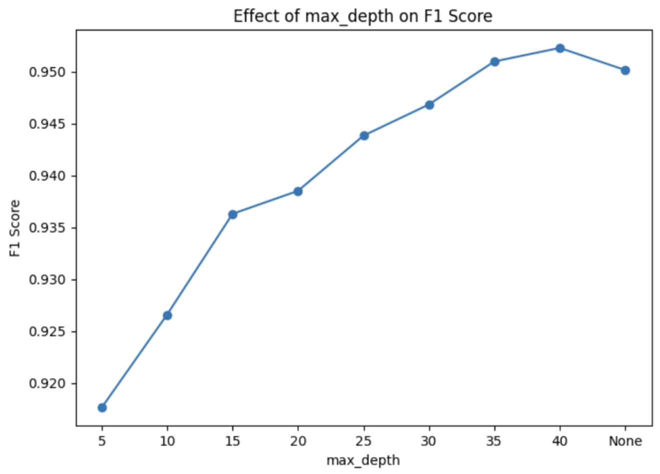

Fig. 1: F1 Score vs Hyperparameter: Max depth

B. BiLSTM with CRF

We employed a Bidirectional Long Short-Term Memory (BiLSTM) network combined with a Conditional Random Field (CRF) layer. The BiLSTM captures sequential dependencies in both directions, while the CRF layer enforces label dependencies in sequence predictions, ensuring more accurate entity boundary detection. This configuration leverages the strengths of both LSTM and CRF for sequence labeling tasks.

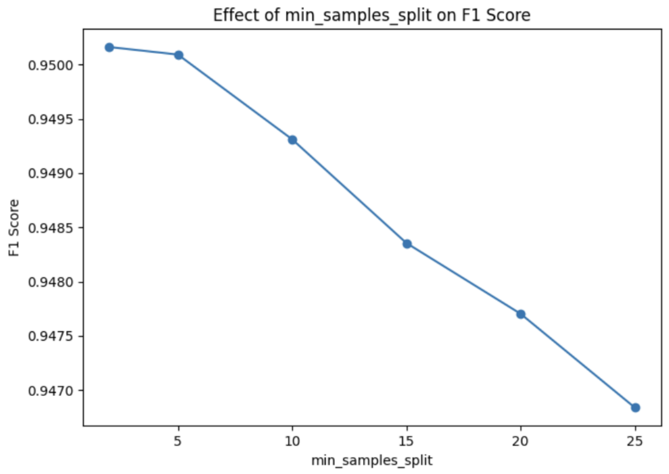

Fig. 2: F1 Score vs Hyperparameter: Min samples split

C. CRF with POS Tags

In this configuration, we used a CRF model with Part-of-Speech (POS) tags as additional features. POS tags help the CRF model identify patterns in named entity structure, enhancing its ability to distinguish entity boundaries, especially in code-mixed text.

D. CRF with Word Embeddings

We also combined CRF with word embeddings to leverage semantic information from words. Word embeddings, trained on a large corpus, provided the CRF model with additional context for each word, helping it identify entity relationships more effectively.

E. Traditional Models with Word2Vec Embeddings

For traditional models such as Decision Tree, Naive Bayes, and Random Forest, we integrated Word2Vec embeddings as input features. Word2Vec embeddings capture semantic similarity between words, which can improve classification accuracy in NER by providing additional context to the models:

- **Decision Tree with Word2Vec:** The embeddings were used as features for each word, allowing the Decision Tree to make classification decisions based on semantic relationships.
- **Gaussian Naive Bayes with Word2Vec:** The underperformance of GNB with Word2Vec embeddings can be attributed to inherent limitations of GNB (independence and distribution).
- **Random Forest with Word2Vec:** Random Forest was trained with Word2Vec features, improving its ability to differentiate between named entities and non-entities in the dataset.

F. Summary

We optimized the Decision Tree, Naive Bayes, and Random Forest models through hyperparameter tuning. We also explored enhancements such as BiLSTM with CRF, CRF with

POS tags, and CRF with word embeddings. Additionally, we integrated Word2Vec embeddings into traditional models, improving performance, especially for Decision Tree and Random Forest, while revealing the limitations of Naive Bayes with embeddings.

VI. Results

A. Performance Metrics

In evaluating our Named Entity Recognition (NER) model, we used the `flat_classification_report` function, which provides several key metrics:

- **Precision**:
$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$
Precision represents the proportion of predicted positive instances that are correctly identified as positive, indicating the classifier's accuracy when it predicts a certain label.
- **Recall**:
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$
Recall, also called sensitivity, shows the proportion of actual positives that are correctly identified by the model.
- **F1-Score**:
$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision + Recall}}$$
The F1-Score is the harmonic mean of precision and recall, combining both to provide a single measure of the model's accuracy, particularly useful when dealing with imbalanced classes.
- **Support**:
$$\text{Support} = \text{Number of true instances for each label}$$
Support shows the count of actual occurrences for each label in the test dataset, which helps interpret other metrics by providing class distribution context.
- **Overall Accuracy**:
$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Predictions}}$$
Accuracy measures the proportion of correct predictions to total predictions, providing an overall performance assessment.

These metrics allow us to evaluate the model's performance across different entity classes and determine areas for potential improvement. To compare between different models we used Overall Accuracy as our only metric.

Model	Accuracy
CRF	0.9660
LSTM	0.9257
CRF with POS	0.9625
CRF with Word Embed	0.9659
BI-LSTM	0.9602
BI-LSTM CRF	0.9457
Targeted Features Classification	0.9681

TABLE III: Accuracies of Advanced Models

Model	Baseline Model	Improved Model	Difference
Decision Tree	0.9364	0.9629	+0.0265
Naive Bayes	0.7414	0.9542	+0.2129
Random Forest	0.9397	0.9648	+0.0251
Logistic Regression	N.A	0.9681	N.A
Support Vector	N.A	0.9638	N.A
K-Neighbours	N.A	0.9563	N.A

TABLE IV: Accuracy Comparison of Baseline and Enhanced Traditional Models

Model	Baseline Accuracy	Word2Vec Accuracy	Difference
Decision Tree	0.9364	0.9578	+0.0214
Random Forest	0.9397	0.9536	+0.0139
Naive Bayes	0.7414	0.3249	-0.4165

TABLE V: Model Accuracy for traditional models with Word2Vec embedding

VII. Observation

A. Impact of Hyperparameter Tuning and Parameter Selection

The performance of Decision Tree, Naive Bayes, and Random Forest models improved significantly after hyperparameter tuning and by selecting specific features for classification. In particular, adjusting the smoothing parameter and focusing on key parameters in Naive Bayes led to a much better classification accuracy.

B. Word2Vec Embeddings

Integrating Word2Vec embeddings into traditional models, resulted in improvements in performance, especially for the Decision Tree and Random Forest model. However, the performance of Naive Bayes with Word2Vec was suboptimal. This can be attributed to the model's assumption of feature independence, which is not well-suited for the high-dimensional nature of Word2Vec embeddings.

VIII. Conclusion

In this work, we addressed the challenge of Named Entity Recognition (NER) in Hindi-English code-mixed tweets. We performed extensive preprocessing to clean and prepare the data, including noise removal, tokenization, and transliteration. The annotated dataset of 3,638 code-mixed tweets was used to train and evaluate machine learning models. Through our experiments, we demonstrated that algorithms such as Decision Trees and Conditional Random Fields (CRFs) were effective in identifying entities in code-mixed text, with high inter-annotator agreement and a robust classification performance.

Future work could explore advanced deep learning models, such as BERT-based architectures, to further improve performance in handling code-mixed text. Additionally, a larger, more diverse dataset could enhance the generalizability of the models to other social media platforms and languages.