TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.04, I.04 (2025)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.44001

Reducing FastText's Limits in Romanized Language Detection

Yashi Bajpai[a], Aditi Joshi [b] and Mr. Amit Srivastava [c]

[a] Student, Computer Science, National P.G. College, Lucknow, India
[b] Student, Computer Science, National P.G. College, Lucknow, India
[c] Assistant Professor, Computer Science, National P.G. College, Lucknow, India
10c.yashibajpai@gmail.com, aditijoshi.141235@gmail.com, amit_sri_in@yahoo.com

KEYWORDS

Romanization; Romanized Text; Multilingual Text Processing; Transliteration; Language Identification; Tokenization in Mixed Scripts; Benchmarking in NLP

ABSTRACT

To identify the language of a given text, language identification models such as FastText are used often. However, these models frequently have trouble accurately categorizing text that is written in the Roman (Latin) nature but have historically used non-Latin scripts like Hindi, Japanese and Chinese. In our research, we analyze FastText's performance on romanized inputs and find a pattern of misinterpretation into unrelated languages and lower confidence scores. We solve this by implementing a score-based thresholding method, which hides the input's anticipated language label and classifies it as romanized if the confidence score that FastText returns is less than the set threshold (0.5).

This threshold-based method increases classification reliability through testing on several languages and romanized inputs. This study identifies a significant weakness in existing language identification systems and suggests a simple, adjustable modification to improve their effectiveness in multilingual, real-world situations.

1. Introduction

An essential component of many multilingual NLP systems is language identification (LID) (Cavnar & Trenkle, 1994; Rehůřek & Kolkus, 2009). But since they are trained on native script data, the majority of popular technologies, like FastText, frequently fail to recognize languages written in Romanized forms. In multicultural nations like India, where many people write their native languages, including Hindi, in Latin script—a technique known as Hinglish—this challenge is even more evident (Jamatia et al., 2015).

1.1 Motivation

Traditional LID systems usually cannot handle the extensive spelling variation and non-standard mappings from native phonemes to Roman letters that come with romanization (e.g., Hindi "ज़" may become "z" or "j") (Sitaram et al., 2019).

The challenges involved in the implementation of Romanized LID are shown by recent datasets like Bhasha-Abhijnaanam and models like IndicLID, which are specifically designed to support native-script and romanized forms of 22 Indic languages using supervised learning techniques (Kakwani et al., 2020; Kakwani et al., 2023).

In addition, more recent studies have shown that engineered training data with natural spelling variations can greatly enhance LID performance on Romanized text, increasing F1 scores on IndicLID benchmarks from approximately 75% to approximately 88% (Bhat et al., 2024).

Corresponding Author: Yashi Bajpai, Department of Computer Science, National P.G. College, Lucknow, India
Email: 10c.yashibajpai@gmail.com

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.04, I.04 (2025)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.44001

1.2 Problem Statement

Whilst these supervised techniques work well, they need a lot of computing power and huge labeled datasets. FastText, on the other hand, is a lightweight, unsupervised model that is often used in practical settings. On Romanized inputs, however, it often generates ambiguous findings or low confidence ratings and incorrect predictions, misclassifying them as unrelated languages. (Grave et al., 2017).

1.3 Proposed Solution

We thoroughly analyze FastText's confidence scores on Romanized versus native-script inputs in Hindi, Japanese, and Chinese in this paper. We found that Romanized inputs regularly result in higher rates of misclassification and lower confidence scores (Fig 1). Taking note of this pattern, we propose a score-based threshold mechanism in which the text is marked as "Romanized" and the model's anticipated language label is removed when FastText returns a confidence below a threshold (for example, 0.5).

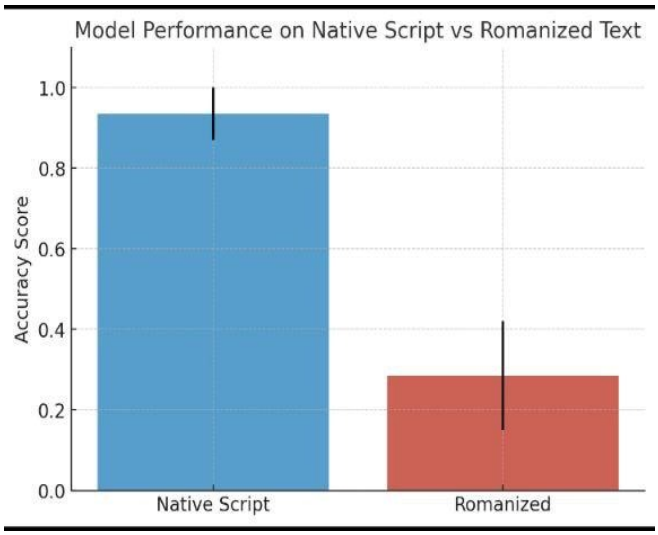

Fig 1: Performance on native vs romanized text

The precise underlying language is not identified by this flagging technique. Instead, it provides a thin detection layer that lets computers know when an input might need special treatment (such as transliteration, manual review, or pipelines for guided processing).

1.4 Contributions

The following contributions are brought forward by this paper:

1) A new analysis that contrasts the confidence behavior of FastText on native and Romanized text.

2) Romanized inputs can be flagged using a threshold-based criterion without the need for external models or retraining.

3) It shows improvement in flag reliability and misclassification handling for test inputs in Chinese, Japanese, and Hindi.

2. Related Work

Most notably in South Asian locations where romanized Hindi (Hinglish) is widely used, research on language detection in code-mixed and romanized text has become more widespread. Previous studies are primarily focused on two areas:

2.1 Romanized Indic Language Identification

Madhani et al. (2023) propose Bhasha-Abhijnaanam, the first broad dataset and model (IndicLID) for understanding texts in 22 Indic languages that are in traditional script and romanized. They show that training on transliterated variations significantly improves the accuracy of detection (Madhani et al., 2023). However, the approach they use is heavy on resources given that it takes training under supervision and organized datasets.

Similarly, utilizing LSTM/CNN models on data with a mix of Hindi and English codes, Joshi & Joshi (2020) evaluated different input formats (character, subword embeddings). While their best models accomplished a

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.04, I.04 (2025)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.44001

prediction accuracy of about 94.5%, they still predominantly utilized word-level annotated samples (Joshi & Joshi, 2020).

2.2 Code-Switching and Hinglish Language Identification

Singh et al. (2018) explored named entity recognition and language identification in tweets with mixed Hinglish symbols. They found that standard tools fail to perform well and that model adjusting must be performed for improved detection (Singh et al., 2018).

Code-mixed preliminary training proved viable, and Ansari et al. (2021) upgraded a code-mixed BERT model (Hindi-English-Urdu) for word-level language recognition, exceeding monolingual norms (Ansari et al., 2021).

2.3 Lightweight & Unsupervised Approaches

Romanized script detectors based on FastText are the focus of some recent endeavors. The langidentification package, for example, creates a refined FastText model that has been trained on South Asian languages that have been romanized, such as Bengali, Tamil, Urdu, and Hindi. It still needs specific data and training, but it attained macro-F1 scores of about 0.66 for romanized detection across multiple languages (absu5530, 202X).

Even though they are still specific to a language and depend on labeled inputs, there are other tools, such as RomanScriptDetect, that make use of supervised machine learning to sort romanized Indian languages across text fields (Abhishek Omray, GitHub, 202X)**.**

2.4 Gap in Literature

At the moment, the majority of LID systems for romanized text count on selected databases or supervised learning (like IndicLID), while some use domain-specific models which need annotating manually (like RomanScriptDetect). A few research studies, however, investigate model-agnostic, zero-shot methods that could make use of already-available resources, such as FastText, without requiring retraining in order to consistently identify a romanized input.

2.5 Positioning of This Work

This paper suggests a simple, threshold-based identifying method that detects romanized inputs without the need to train new models by utilizing FastText's confidence scores. This evaluation layer enhances present LID tools and can be quickly and inexpensively integrated into pipelines, a method not previously explored in the literature.

3. Methodology

This work aims to examine and transform a pretrained FastText language identification model to reliably recognise romanized text inputs (e.g., Roman-script Hindi, Japanese, Chinese) by leveraging its confidence scores.

3.1 Model Foundation

Wikipedia and Tatoeba entries for 176 languages are two of the multilingual datasets that trained the official FastText language identification model (lid.176.bin / lid.176.ftz) (Joulin et al., 2016; Grave et al., 2017). In order to handle rare or misspelt words through partial matching, the model makes use of subword n-gram representations (FastText, 2017).

3.2 Input Testing & Observations

The model was provided with a variety of test sentences written in both native script (such as Devanagari Hindi, Chinese characters, and Japanese kana) and romanized (Latin script) styles. FastText gives the following for every sentence:

A language label that is anticipated, and

A probability score, or confidence score.

Empirically, romanized sentences were often incorrectly categorised into unrelated languages and consistently produced poorer confidence scores (e.g. <0.5). On the other hand, native-script inputs yielded precise detection and scores between 0.8 and 1.0.

These findings support earlier studies that demonstrated how typical LID systems undergo disruption by the ranged character patterns that frequently appear in romanized text (Benton et al., 2025).

3.3 Threshold-Based Flag Mechanism

We present a numeric flag for romanization detection in order to effectively make the most of this constant behaviour:

$$\text{is\_romanized} = (\text{score} < \theta)$$

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.04, I.04 (2025)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.44001

Based on early testing, we empirically established the cutoff at $\theta = 0.5$. The input is identified as "Romanized" and the language label is disabled (set to null) when the score is less than 0.5. If not, the predicted language is maintained.

This method mainly distinguishes between likely native-script and likely romanized text; it does not identify the underlying language.

3.4 Pipeline Overview

The approach suggested commences with a preprocessing-free method of language detection in which a trained FastText model receives input text directly. Though it introduces a modification focused on score interpretation, the detection function, detect (text: str), makes use of the current language identification model.

When the system receives an input text, it uses the FastText prediction algorithm to identify the predicted language label and associated confidence score. To extract the language tag, these labels are coded from their initial format of __label__. For the motives of consistency and outlier inflation mitigation, the relevant score is cast to a float and restricted at 1.0.

The implementation of a confidence metrics is a significant distinctiveness in the revised methodology. For example, the language is marked as Romanized if the score is less than 0.5. Numerous tests using romanized inputs in a variety of non-Latin-script languages, including Hindi, Chinese, and Japanese, were used to experimentally confirm this condition. In such cases, the algorithm clearly indicates low-confidence rating and the missing element of a clear script-based match by setting the "lang" field to "Romanized" and the "score" field to a hyphen ("-").

Alternatively, if the score is greater than or equal to 0.5, the model proceeds according to normal FastText behavior, assigning the identified language and reporting the confidence score. The model can proficiently make distinctions between text written in its original script and displayed in the Latin alphabet courtesy to this decision-making logic.

In multilingual online spaces where users routinely transliterate local languages using Latin characters—a practice that is becoming more and more widespread on informal platforms like social media and messaging apps—this pipeline maintains FastText's efficiency while broadening its scope of use.

3.5 Evaluation Setup

We used a dataset of native-script and romanized texts in different languages—Hindi, Japanese and Chinese—in order to evaluate the usefulness of the improved FastText-based identification system. Both samples written in the original script and their transliterated Latin-alphabet versions were included in each language set.

The accuracy of classification into two categories served as the primary evaluation metric:

Accurately recognising samples of native script

Romanized samples should be correctly marked as "Romanized."

Because of their Latin script, the standard FastText model often incorrectly identified romanized inputs as high-resource languages like English or Spanish, producing inaccurate results. Our adjusted code, on the other hand, used a confidence threshold: inputs that had a prediction score less than 0.5 were regarded to be possibly romanized and were explicitly marked as such.

The model tagging a text as "Romanized," without necessarily specifying the language, was considered a correct detection for romanized texts. The revised logic minimised false positives in Latin-script misclassifications by significantly improving the ability to distinguish between romanized and non-romanized inputs.

4. Experiments and Results

There were two levels to the experiments. Standard text data created by native scripts was used to assess the model in the first phase (Fig. 2). The confidence scores in this case varied from 0.87 to 1.0, implying that the

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.04, I.04 (2025)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.44001

predictions were highly correct. It was confirmed that the model was working successfully in its original version for non-romanized datasets when the predicted languages matched the input text.

```
text = "मन के हारे हार है, मन के जीते जीत|कहे कबीर हरि पाइए,
result = detect(text)
print(result)
```

```
{'lang': 'hi', 'score': 0.9989314675331116}
```

```
text = "冬が来た。白い樹樹の光を体のうちに蓄積しておいて、夜
result = detect(text)
print(result)
```

```
{'lang': 'ja', 'score': 0.9995964765548706}
```

```
text = "これは日本語の文章です。"
result = detect(text)
print(result)
```

```
{'lang': 'ja', 'score': 1.0}
```

```
text = "这是中国人"
result = detect(text)
print(result)
```

```
{'lang': 'zh', 'score': 1.0}
```

Fig. 2: Tests in native scripts

The focus was shifted to romanized text inputs in the second phase. In this specific instance, the original model gave much lower confidence values, frequently between 0.15 and 0.42 . The model often incorrectly identified the language completely, in addition to the decline in confidence. This revealed the model's incompetence to effectively handle romanized data accurately, an observation that was not revealed in the analysis of the normal script.

A thresholding technique has been implemented to narrow this gap. More specifically, the output was placed into a different category called Romanized when the predicted confidence dropped below 0.5. By preventing the wrong native category assignment of low-confidence romanized data, this change has boosted interpretability. Such inputs were identified under a distinctive label rather than being incorrectly classified, strengthening the system's dependability in mixed-script scenarios (Ayush Kashyap et al. (2025)).

The revised approach generated more consistent results. The output for romanized inputs showed Romanized, rather than incorrect language tags. This minimised ambiguities when users used transliterated text while interacting with the system.

Below are visual examples for clarity. These are screen captures of the input text and the matching outputs both before (Fig. 3) and after (Fig. 4) the change. The fundamental structure performs terribly on romanized specimens but well on native scripts in the first set of diagrams. The second set illustrates how low-confidence romanized texts are correctly identified under the Romanized category using the enhanced approach.

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.04, I.04 (2025)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.44001

```
text = "Dàjiā hǎo. Jīntiān tiānqì hěn hǎo."
result = detect(text)
print(result)
```

```
{'lang': 'es', 'score': 0.15342405438423157}
```

```
text="ken san wa omoshiroi desu"
result = detect(text)
print(result)
```

```
{'lang': 'sw', 'score': 0.3367029130458832}
```

```
text="Mera naam Geet hai"
result = detect(text)
print(result)
```

```
{'lang': 'nl', 'score': 0.4298158288002014}
```

Fig. 3: Output prior to the thresholding technique

```
text = "Dàjiā hǎo. Jīntiān tiānqì hěn hǎo."
result = detect(text)
print(result)
```

```
{'lang': 'Romanized', 'score': '-'}
```

```
text="ken san wa omoshiroi desu"
result = detect(text)
print(result)
```

```
{'lang': 'Romanized', 'score': '-'}
```

```
text="Mera naam Geet hai"
result = detect(text)
print(result)
```

```
{'lang': 'Romanized', 'score': '-'}
```

Fig. 4: Output after the implementation of thresholding technique

The results obtained reveal that while the original model performs well on datasets with native script, a simple thresholding technique is capable of managing its limitations with romanized inputs. The claims that this tweak strengthens the reliability of the language detection process is supported by the illustrations shown in the diagrams.

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.04, I.04 (2025)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.44001

5. Conclusion

This study explored language detection with fastText, paying particular attention to the challenges associated with romanized data. The results of the experiments demonstrated that romanized inputs resulted in much lower confidence scores and more regular misclassifications, whereas normal text inputs were identified with excellent accuracy. The implementation of a score-thresholding technique heightened the system's ability to detect romanized text by recognising it instead of making an inaccurate prediction. This highlights the necessity to modify current language detection techniques to keep up with text format changes in the real world, where users commonly adopt mixed scripts for online communication (Gupta and Bali, 2013).

6. Future work

The approach can be refined in the future by incorporating more features like character-level modelling and context-aware embeddings, which might raise reliability across various writing systems (Mikolov et al., 2018). To be able to verify scalability and prediction, further tests might also incorporate larger datasets with different kinds of romanization patterns. In order to offer deeper insights into performance trends across datasets, visualisations and graphical results could be expanded. In regard to the increasing usage of romanized text in online platforms, this work might prove useful as an outline for creating more universal multilingual systems.

References

- Ansari, Z. Z., Beg, M. M. S., Ahmad, T., Khan, M. J., & Wasim, G. (2021). *Language Identification of Hindi-English Tweets Using Code-Mixed BERT*.
- absu5530 (202X). *langidentification: Language identification using fastText for Romanized Scripts* (GitHub).
- Abhishek Omray (202X). *RomanScriptDetect: A tool to identify romanized Indian language text* (GitHub).
- Ayush Kashyap et al. (2025), Design and Implementation of an Intelligent Loan Eligibility System Using Machine Learning Techniques, TEJAS Journal of Technologies and Humanitarian Science, ISSN-2583-5599, Vol.04, I.02 (2025), https://doi.org/10.63920/tjths.42002
- Bhat, R., et al. (2024). *IndicLID: Language Identification for 22 Indic Languages and Romanized Variants*. arXiv preprint arXiv:2504.21540.
- *Bali, K., Choudhury, M., Dasgupta, T., & Basu, A. (2014). Automatic Language Identification in Code-Switched Hindi-English Social Media Text. Journal of Open Humanities Data. — Describes the phenomenon of "Romanagari" and challenges in identifying Hindi written in Latin script on social media.*
- Cavnar, W. B., & Trenkle, J. M. (1994). *N-Gram-Based Text Categorization*. In SDAIR.
- Grave, E., et al. (2017). *Efficient text classification and representation using FastText*. arXiv preprint arXiv:1708.02709.
- *Ghosh, S., Gothe, S. V., Sanchi, C., & Raj Kandur Raja, B. (2021). edATLAS: An Efficient Disambiguation Algorithm for Texting in Languages with Abugida Scripts. arXiv preprint arXiv:2101.03916. — Proposes a method to disambiguate typing variants and romanized word forms for Indic abugida scripts.*
- Gupta, P., & Bali, K. (2013). "Challenges in processing code-mixed and romanized scripts: A case study of Hindi-English." *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.
- Jamatia, A., Das, A., & Gambäck, B. (2015). *Part-of-speech tagging for code-mixed English-Hindi Twitter and Facebook chat messages*. In Proceedings of the International Conference Recent Advances in Natural Language Processing (RANLP).
- Joshi, R., & Joshi, R. (2020). *Evaluating Input Representation for Language Identification in Hindi-English Code Mixed Text*. arXiv.
- Kakwani, D., et al. (2020). *IndicCorp: A multilingual corpus of Indian languages*. arXiv preprint arXiv:2005.08225.
- Kakwani, D., et al. (2023). *Bhasha-Abhijnaanam: A Benchmark for Language Identification of Code-Mixed and Romanized Text in Indian Languages*. ACL Anthology, ACL 2023.

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.04, I.04 (2025)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.44001

- Rehůřek, R., & Kolkus, M. (2009). *Language Identification on the Web: Extending the Dictionary Method*. In CICLing.
- Madhani, Y., Khapra, M. M., & Kunchukuttan, A. (2023). *Bhasha-Abhijnaanam: Native-script and Romanized Language Identification for 22 Indic Languages*. ACL Short Paper.
- Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., & Joulin, A. (2018). "Advances in pre-training distributed word representations." *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*.
- *Nielsen, E., Kirov, C., & Roark, B. (2023). Distinguishing Romanized Hindi from Romanized Urdu. In Proceedings of the Workshop on Computation and Written Language (CAWL 2023). — Explores classification between romanized forms of closely related languages, demonstrating methods to differentiate them.*
- Roark, B., Wolf-Sonkin, L., Kirov, C., Mielke, S. J., Johny, C., Demirsahin, I., & Hall, K. (2020). Processing South Asian Languages Written in the Latin Script: The Dakshina Dataset. arXiv preprint arXiv:2007.01176.
- *Roark, B., Wolf-Sonkin, L., Kirov, C., Mielke, S. J., Johny, C., Demirsahin, I., & Hall, K. (2020). Processing South Asian Languages Written in the Latin Script: The Dakshina Dataset. arXiv preprint arXiv:2007.01176. — Introduces a parallel dataset of native script and romanized South Asian languages, useful for both training and evaluation.*
- Singh, K., Sen, I., & Kumaraguru, P. (2018). *Language Identification and Named Entity Recognition in Hinglish Code-Mixed Tweets*. ACL Student Research Workshop.
- Sitaram, S., et al. (2019). *A Survey of Code-Switched Speech and Language Processing*. In Proceedings of the 27th International Conference on Computational Linguistics (COLING).
- *Sumanathilaka, D., Anuradha, I., Weerasinghe, R., Micallef, N., & Hough, J. (2025). IndoNLP 2025: Shared Task on Real-Time Reverse Transliteration for Romanized Indo-Aryan Languages. arXiv preprint arXiv:2501.05816. — Focuses on converting romanized Indo-Aryan text back into its native script forms in real time.*