

Language recognition using Fasttext

Simon GIRARD and Ewan KNAPIK

May 26, 2025

Abstract

This project explores the effectiveness of language recognition using the FastText model, with a focus on short, informal text data. We evaluated the model’s performance on a curated dataset of tweets, which present unique challenges due to their brevity, slang usage, and multilingual content. The accuracy and speed of FastText’s language identification capabilities were analyzed in this context. To demonstrate a practical application, we developed a prototype chatbot integrated with the Discord platform. The chatbot automatically detects the language of user messages in real time and responds accordingly. Our results highlight the strengths and limitations of FastText in handling noisy, user-generated content and illustrate its potential for integration into conversational AI systems.

1 Introduction

1.1 Aim

The primary aim of this project is to evaluate the capabilities of the FastText language identification model when applied to real-world, user-generated content. Specifically, we focus on testing the model’s effectiveness in identifying the language of tweets—a challenging domain due to the informal, abbreviated, and often multilingual nature of the content.

Beyond static evaluation, we aim to demonstrate the model’s practical application by integrating it into a Discord-based chatbot capable of recognizing and responding to messages in different languages.

1.2 Scope

This project investigates FastText’s performance across various types of input data. The scope includes not only standard, single-language tweets but also edge cases such as tweets containing unsupported languages, code-switched or mixed-language content, and non-linguistic text such as emojis or gibberish. We assess how well FastText handles these variations and examine its limitations.

Furthermore, the project encompasses the development of a simple Discord chatbot to showcase real-time language recognition, thereby extending the evaluation from static analysis to interactive use cases.

For this project, we limit our evaluation to the use of the ‘lid.176.bin’ model, the full 131MB version provided by FastText, which supports 176 languages. We deliberately chose not to use the compressed 1MB version, as our focus was on testing the model’s maximum performance and accuracy in handling challenging, real-world text data. This decision ensures that the results reflect the capabilities of the most comprehensive version of the model. [JGB⁺16]

1.3 Methodology

To evaluate the FastText language identification model, we developed a custom testing tool capable of processing large volumes of short text inputs. This tool was used to run experiments on [the dataset provided](#) in the official repository of *X-Topic (Multilingual Topic Classification in X: Dataset and Analysis, EMNLP 2024)*. This dataset is specifically curated from posts on X (formerly Twitter), making it well-suited for assessing the model’s performance on informal, multilingual, and noisy content. [AUBCC24]

The testing tool processes each text sample through the FastText ‘lid.176.bin’ model and records the predicted language along with its confidence score. We monitored several performance metrics during evaluation, including the number of incorrect predictions, the average confidence across all predictions, the lowest confidence observed, and the specific samples that yielded these lowest-confidence results. These metrics allowed us to better understand the model’s behavior and identify edge cases or failure points.

In addition to offline evaluation, we developed a simple chatbot using the official Discord Python API (‘discord.py’). This bot listens for messages in real time, uses FastText to detect the language of incoming messages, and can respond or tag messages based on their predicted language. This interactive component allowed us to observe the model’s performance in a live, user-facing application and assess its practical applicability.

2 Theoretical Part

2.1 Subword Embeddings and Character n-grams

FastText enhances traditional word embedding techniques by incorporating subword information. Instead of representing each word as a distinct vector, FastText breaks words into a set of character n-grams. For instance, the word “where” with $n = 3$ is represented by the trigrams: `<wh, whe, her, ere, re>`, and the special token `<where>`. Here, angle brackets denote word boundaries, distinguishing prefixes and suffixes. Each of these n-grams has its own vector representation, and the word’s vector is computed as the sum of its subword vectors. This approach allows FastText to generate meaningful embeddings for rare or out-of-vocabulary words by leveraging the shared subword structures among words [JGBM16].

2.2 Model Architecture for Language Identification

In the context of language identification, FastText employs a simple yet efficient architecture. The model processes input text by first tokenizing it into words and then further into character n-grams. Each n-gram is mapped to its corresponding vector, and these vectors are averaged to obtain a representation of the input text. This representation is then passed through a linear classifier to predict the language label. The use of subword information enables the model to capture morphological patterns and handle noisy or misspelled text effectively. FastText’s architecture allows for rapid training and inference, making it suitable for real-time applications [ZLLS24].

2.3 Chatbot Implementation and Workflow

The chatbot was implemented using Python and integrated with the Discord platform via the official `discord.py` library. Its primary function is to identify the language of user messages and provide appropriate responses in the detected language. The bot uses the FastText `lid.176.bin` model to predict the language and evaluate confidence scores in real time.

Once a message is received, the bot passes the text to the FastText model, which returns a language label and an associated confidence score. If the predicted language is supported by the bot (i.e., present in the pre-defined multilingual response set), the chatbot proceeds to match the input message with one of the predefined prompts using semantic similarity.

For this, we pre-process a small set of predefined question-response pairs for each supported language (e.g., English, French, Polish, German, etc.). Each language has its own TF-IDF vectorizer trained on the question prompts. When a user input is received, it is vectorized and compared against the stored questions for the detected language using cosine similarity. The response corresponding to the most similar question is then returned to the user.

This design enables the bot to give language-specific replies with lightweight logic and avoids the need for deep natural language understanding or intent classification. While simple, it demonstrates how language identification can be effectively combined with retrieval-based response generation in a multilingual chatbot context.

3 Practical Part

In this section, we describe the experimental evaluation of the FastText language identification model using real-world social media content. We tested the model on a set of tweets sourced from the X-Topic dataset, containing posts in multiple languages. For this evaluation, we selected four target languages: English (**en**), Spanish (**es**), Greek (**el**), and Japanese (**ja**). For each language, we used a dataset of 1,000 tweets stored in JSON Lines (**.jsonl**) format, where each line corresponds to a JSON object containing a tweet’s text.

3.1 Evaluation Process

We wrote a Python script that loads the FastText **lid.176.bin** model and processes each tweet by extracting its text, predicting the language, and recording the confidence score. For each language, we tracked the number of incorrect predictions (i.e., where the predicted language did not match the expected one), the sum and average of confidence scores for correct predictions, and the lowest confidence observed. Additionally, we saved the tweet associated with the lowest confidence to examine potential causes of uncertainty in the model’s predictions.

3.2 Results and Observations

The model achieved perfect accuracy on all four tested languages, with zero errors across the 1,000 tweets in each case. However, the confidence scores varied, especially in tweets with informal or noisy language.

- **English:** Average confidence was 0.90. The lowest confidence (0.25) occurred on the tweet: *“Mercedes: ”PlEaSe nO sAfeTy CaR” #AbuDhabiGP #F1”*, likely due to the irregular casing and emojis (emojis were excluded from this report).
- **Spanish:** Average confidence was 0.91. The lowest score (0.21) was on: *“Kast: ”Vamos a aplicar la Ley” #CandidatoLlegoTuHora #EsLaHoraDeKast”*, which includes a proper name and political hashtags that may lack strong language signals.
- **Greek:** The model performed exceptionally with an average confidence of 1.00. The lowest confidence was still relatively high at 0.61, for: *“()” Lidl. #cancel_lidl”*, which mixes non-standard punctuation and brand references.
- **Japanese:** The average confidence was 0.99, with the lowest being 0.24 on: *“wwwwwwwwwwwwwwwwwwwww”*, a highly informal tweet with laughter characters and colloquial phrasing.

These results suggest that while FastText is robust even in the presence of informal text and symbols, confidence scores can significantly drop in noisy or stylistically extreme examples. Nevertheless, the overall language identification accuracy remained high across the board.

An interesting observation from our experiments is that languages with distinct alphabets—such as Greek and Japanese—appear to be easier for the FastText model to detect with high confidence. These languages have unique character sets that sharply distinguish them from others, reducing the likelihood of ambiguity during prediction. In contrast, languages that share the Latin script, such as English and Spanish, are more prone to lower confidence scores, especially when the input contains slang, inconsistent capitalization, or limited linguistic context. This suggests that script-based differentiation plays a significant role in FastText’s language identification performance.

3.3 Testing on Unsupported Languages

To further assess the robustness of the FastText language identification model, we conducted tests on text samples written in languages or scripts not included in the model’s supported 176 languages. It is expected that the model will not correctly identify these languages due to lack of training data; however, we anticipated that the model’s confidence scores would generally be lower in these cases, reflecting its uncertainty.

Our experiments involved inputting samples from ancient or less common scripts such as Egyptian hieroglyphs and ancient Semitic alphabets. The model frequently misclassified these texts as languages

it is trained on, for example Japanese, Chinese, Italian, or English. The confidence scores for these predictions ranged from approximately 0.21 to 0.45, significantly lower than typical scores observed for supported languages.

Some specific examples include:

- A string of Egyptian hieroglyphs was detected as Japanese with a confidence of 0.38.
- Another hieroglyphic sequence was predicted as Chinese at 0.21 confidence.
- Text in an ancient alphabet (Ugaritic) was misclassified as Italian with a confidence of 0.33.
- Text in old Persian was also predicted as Chinese (0.27 confidence).
- Phoenician text was classified as English with a confidence as high as 0.45.

While it was to be expected that FastText cannot accurately identify unsupported languages, we were surprised by the high confidence scores. Despite the model’s misclassification of texts in ancient or uncommon scripts, many predictions still had confidence values above 0.2 and in some cases as high as 0.45, even though the alphabets were completely different. This suggests that the model sometimes exhibits unwarranted certainty even when faced with entirely unfamiliar inputs, highlighting a limitation in relying solely on confidence scores to detect unsupported languages. Consequently, additional methods may be needed to more reliably flag or handle such cases in practical applications.





3.4 Testing on Mixed-Language Inputs

We also examined how the model handles inputs containing a mix of languages, often combining supported and unsupported scripts or multiple supported languages. For example, when an unsupported ancient Egyptian script was followed by English text, the model correctly identified the English portion, effectively ignoring the unsupported language segment.

In cases where two supported languages were mixed, the model tended to favor one language based on dominant linguistic cues. For a mix of English and German (*some english text and ein bisschen deutsch dazu*), German was detected as prevailing. Similarly, for English-French mixtures (*some english text and un peu de français*), French prevailed. For English and Polish mixes (*some english troche polskiego*), Polish was dominant in the prediction. This behavior suggests that FastText language identification weighs the relative presence of language-specific features and tends to classify based on the language that contributes stronger or more frequent signals within the text.

3.5 Testing on Symbol-Only Inputs

To further examine the model’s behavior, we tested FastText on inputs composed entirely of non-alphabetic symbols, such as emojis, geometric shapes, diacritical marks, and LaTeX-style commands. These experiments help reveal how the model handles content with no meaningful linguistic structure.

- **Emojis:** Inputting a sequence of emojis yielded a language prediction of Japanese (**ja**) with a confidence of 0.34. This relatively low confidence reflects some uncertainty but still indicates a surprising association with a specific language.
- **Geometric Symbols:** When given input composed entirely of geometric shapes (e.g., , , ), the model also predicted Japanese (**ja**), but with an unusually high confidence of 0.97. This strong confidence is unexpected, as the input lacks any linguistic features and should ideally result in a low-confidence, undefined prediction.
- **Diacritical Marks:** A string of combining diacritical marks (e.g.,  ...) produced a prediction of Russian (**ru**) with a confidence of 0.22. This aligns with expectations, as the input provides little meaningful content, and the low confidence indicates uncertainty.
- **LaTeX-Style Commands:** A sequence of Greek letter LaTeX commands (e.g., `\alpha \beta \gamma ...`) was classified as English (**en**) with a confidence of 0.28, likely due to the recognizable ASCII character pattern resembling English syntax.

These results reveal that while FastText can express some uncertainty when presented with symbol-only or non-linguistic input, it may still return high confidence inappropriately, as seen in the geometric symbols case. This behavior emphasizes the need for cautious interpretation of confidence scores and suggests that additional heuristics may be needed to flag or reject non-linguistic inputs in language identification tasks.

4 Summary

This project explored the effectiveness of FastText’s language identification model, `lid.176.bin`, applied to real-world social media data and a practical chatbot implementation on the Discord platform. Our experiments demonstrated that the model achieves high accuracy and confidence on supported languages, especially those with unique alphabets, while handling noisy and informal text common in tweets.

Testing on unsupported languages revealed that although the model often misclassifies these inputs, confidence scores tend to be lower, signaling uncertainty; however, some unsupported scripts still produced surprisingly high confidence scores, indicating limitations in confidence as a sole reliability metric.

In mixed-language scenarios, FastText generally identified the dominant language, effectively ignoring unsupported segments or favoring the language with stronger linguistic features. Finally, the chatbot implementation illustrated a lightweight yet effective integration of language identification with retrieval-based multilingual responses, showcasing potential for real-time multilingual applications.

Overall, FastText provides a robust and efficient foundation for language recognition tasks, but care should be taken in interpreting confidence scores and handling unsupported or mixed-language inputs in practical deployments.

References

- [AUBCC24] Dimosthenis Antypas, Asahi Ushio, Francesco Barbieri, and Jose Camacho-Collados. Multilingual topic classification in X: Dataset and analysis. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20136–20152, Miami, Florida, USA, November 2024. Association for Computational Linguistics.
- [JGB⁺16] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, H’erve J’egou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.
- [JGBM16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.
- [ZLLS24] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Subword embedding, 2024. Accessed: 2025-05-26.