N-Gram and FastText Based Language Detection System: A Comparative Study

Abhisek Sarkar*

June 12, 2025

Abstract

This work presents a full deployment of two distinct approaches to automatic language identification: a classic N-gram-based statistical model and a cutting-edge FastText-based neural model. Both models were tested and deployed on a multilingual large-scale Wikipedia corpus from 200 languages. The N-gram approach achieved 99.45% accuracy across all 200 languages, while the FastText program achieved 99.83% accuracy on a 30-language subset and 98.45% accuracy on the full 200-language dataset. This comparison reveals the strengths and weaknesses of both older statistical and newer neural approaches in multilingual language identification.

• Introduction

Language identification (LangID) is a basic natural language processing (NLP) task that serves as a prerequisite to most multilingual applications including machine translation, cross-lingual information retrieval, and content filtering systems. The task involves automatically determining the language that a text document has been composed in. With the accelerated growth of multilingual web content and the increased need for automated processing of diverse linguistic data, robust language identification systems are no longer a luxury.

This work presents a comprehensive comparison between two paradigmatically different language identification methods: the conventional N-gram-based statistical method and the state-of-the-art FastText-based neural method. The experimentation covers an enormous gamut of 200 languages, providing insights into the scalability and efficiency of both methods with regard to various linguistic families and writing systems.

The motivation for this comparative study arises from the need to understand the trade-offs between performance and interpretability, efficiency and accuracy, as well as traditional statistical versus modern neural methods in the context of large-scale multilingual language identification.

*Code and data are available at: `https://github.com/abhiseksarkar2001/language_detector`

• Related Work and Theoretical Foundation

• N-Gram Based Approaches

The foundation of this work's N-gram implementation is rooted in the seminal paper by Cavnar and Trenkle (1994) titled *"N-Gram-Based Text Categorisation"* [1]. This pioneering work established the theoretical framework for using character-level N-grams in text classification tasks, including language identification.
Cavnar and Trenkle's approach revolutionized language identification by introducing several key concepts:
Character-level Analysis: Unlike word-based approaches, character-level N-grams capture orthographic patterns that are language-specific, making them robust to vocabulary variations and effective across different writing systems.
Frequency-based Profiling: The method creates language profiles based on the most frequent N-grams, capturing the essential character patterns that distinguish one language from another.
Rank-order Distance Metric: The comparison between language profiles uses rank-order statistics rather than absolute frequencies, making the method robust to text length variations and sampling differences.
The insights from Cavnar and Trenkle's work directly informed several design decisions in our implementation:

- Adoption of character-level trigrams (3-grams) as the optimal balance between specificity and generalization
- Implementation of frequency-based language profile generation
- Utilization of rank-order distance metrics for language comparison
- Focus on the most discriminative N-grams (top 300) for computational efficiency

• FastText and Modern Neural Approaches

The FastText implementation draws inspiration from the work of Joulin et al. (2017) in their paper *"Bag of Tricks for Efficient Text Classification"* [2]. This work demonstrated that simple yet effective neural architectures could achieve state-of-the-art performance on text classification tasks while maintaining computational efficiency.
Key innovations from Joulin et al. that influenced our FastText implementation include:
Subword Information: FastText's ability to handle subword information makes it particularly effective for morphologically rich languages and helps with out-of-vocabulary words.
Hierarchical Softmax: The use of hierarchical softmax enables efficient training on large-scale classification problems with hundreds of classes.
N-gram Features: The incorporation of character and word N-gram features provides a bridge between traditional statistical methods and modern neural approaches.

• Dataset Construction and Preprocessing

• Data Collection

A comprehensive multilingual dataset was constructed using Wikipedia articles across 200 languages. The choice of Wikipedia as the primary data source was motivated by several factors:

- **Coverage:** Wikipedia provides extensive coverage of languages from diverse linguistic families
- **Quality:** The collaborative editing process ensures relatively high-quality, well-formed text
- **Consistency:** Standardized markup and structure across languages
- **Availability:** Freely available and regularly updated content

For each of the 200 languages, the following data collection protocol was implemented:

1. Downloaded 1,000 Wikipedia articles per language using automated web scraping
2. Stored each article as an individual text file with appropriate encoding (UTF-8)
3. Maintained metadata including language code, article title, and collection timestamp

• Text Preprocessing Pipeline

A comprehensive preprocessing pipeline was developed to ensure data quality and consistency:
Markup Removal: All Wikipedia-specific markup, HTML tags, and metadata were stripped from the text while preserving the actual content.
Punctuation and Special Characters: Non-alphabetic characters were handled carefully to preserve language-specific diacritics and writing system characteristics while removing noise.
Normalization: Text was normalized to lowercase for consistency, with special consideration for languages with distinct case systems.
Whitespace Handling: Extra whitespace, line breaks, and formatting artifacts were cleaned while preserving natural text flow.
Encoding Verification: All text files were verified for proper UTF-8 encoding to handle diverse writing systems correctly.

• Data Consolidation

To optimize training efficiency and ensure sufficient data volume per language:

- The 1,000 individual text files per language were merged into 100 consolidated files
- Each consolidated file contained approximately 10 original articles

- This consolidation strategy balanced data volume with computational efficiency
- Final dataset comprised 20,000 text files (100 files × 200 languages)

• N-Gram Based Language Detection

• Theoretical Foundation

N-grams represent contiguous sequences of *n* characters extracted from text. For language identification, character-level N-grams are particularly effective because they capture:

- Orthographic patterns specific to languages
- Phonotactic constraints reflected in writing systems
- Statistical regularities in character combinations
- Morphological patterns in word formation

In this implementation, trigrams (n=3) were selected as the optimal choice based on empirical evidence from Cavnar and Trenkle's work and subsequent research showing that trigrams provide the best balance between specificity and generalization.

• Language Profile Generation

The language profile generation process follows these steps:

1. **Corpus Concatenation:** All 100 text files for each language are concatenated into a single corpus
2. **Trigram Extraction:** All possible trigrams are extracted from the corpus with padding characters for word boundaries
3. **Frequency Counting:** Trigram frequencies are computed and stored in a frequency distribution
4. **Ranking:** Trigrams are ranked by frequency in descending order
5. **Profile Creation:** The top K trigrams (where K=300) form the language profile

For example, from the word "language", the following trigrams are extracted:

```
["_la", "lan", "ang", "ngu", "gua", "uag", "age", "ge_"]
```

where "_" represents word boundary padding.

• Distance Metric and Classification

The classification process employs the out-of-place metric introduced by Cavnar and Trenkle:

Test Profile Generation: Given an input document, trigrams are extracted and ranked by frequency to create a test profile.

Distance Calculation: For each language profile Q and test profile P, the distance is computed as:

$$D(P,Q) = \sum_{t \in P} \begin{cases} |rank_P(t) - rank_Q(t)| & \text{if } t \in Q \\ penalty & \text{if } t \notin Q \end{cases}$$

where *penalty* is typically set to the maximum possible rank difference.

Classification: The language corresponding to the minimum distance is selected as the predicted language.

• Implementation Details

The N-gram system was implemented in Python using standard libraries:

- `collections.Counter` for efficient frequency counting
- `pickle` for serializing language profiles
- `re` for text preprocessing and pattern matching
- `os` for file system operations

Key implementation features include:

- Memory-efficient profile storage using dictionaries
- Optimized distance calculation algorithms
- Robust error handling for encoding issues
- Modular design for easy extension and maintenance

• FastText Based Language Detection

• Methodology and Architecture

FastText, developed by Facebook's AI Research lab, extends the Word2Vec approach by incorporating subword information. The architecture consists of:

Input Layer: Text is represented as a bag of words and character N-grams, allowing the model to handle out-of-vocabulary words effectively.

Embedding Layer: Words and subwords are mapped to dense vector representations learned during training.

Averaging Layer: Word embeddings are averaged to create document-level representations.

Classification Layer: A hierarchical softmax layer efficiently handles large-scale multi-class classification.

• Implementation Configurations

Two distinct FastText models were developed to evaluate scalability:

30-Language Model:

- Selected 30 representative languages from different linguistic families
- 100 training documents per language (3,000 total)
- Designed for rapid prototyping and initial evaluation

200-Language Model:

- Complete dataset with all 200 languages
- 100 training documents per language (20,000 total)
- Demonstrates scalability to large-scale multilingual scenarios

• Training Configuration

We trained two separate FastText models: one for a 30-language classification task and another for a 200-language classification task. The hyperparameters were carefully selected based on the scale and diversity of the dataset in each case.

- **30-Language Model**
  - **Learning Rate:** 1.0 : A higher learning rate allows faster convergence for the relatively simpler 30-language task.
  - **Epochs:** 25 : Sufficient to reach good accuracy without overfitting on a smaller label set.
  - **Word N-grams:** 2 : Incorporates both unigrams and bigrams to capture short-range dependencies and common phrase patterns.
  - **Minimum Word Count:** 1 : Includes all words to ensure even rare tokens are represented, important for under-resourced languages.
  - **Verbose:** 2 : Provides detailed training logs for monitoring progress.

- **200-Language Model**
  - **Learning Rate:** 0.7 : A slightly lower learning rate provides more stable convergence in the more complex 200-language setup.
  - **Epochs:** 50 : More training epochs are necessary to allow the model to generalize well across a much larger and more diverse language set.
  - **Word N-grams:** 3 : Incorporating trigrams captures richer local context, which is essential for distinguishing between a wider variety of languages.
  - **Minimum Word Count:** 1 : As above, ensures inclusion of all available linguistic information, especially critical in low-resource scenarios.
  - **Verbose:** 2 : Maintains detailed logs for monitoring such a large-scale training task.
- **Shared Settings**

In both cases, we used the following additional settings:

  - **Character N-grams:** 3–6 : Enables subword-level understanding, helping to improve performance on morphologically rich and related languages.
  - **Loss Function:** Hierarchical Softmax : Efficient for handling large output spaces like in the 200-language task.
  - **Vector Dimension:** 100 : A balanced embedding size that provides good performance without excessive memory usage.
  - **Context Window Size:** 5 : Captures surrounding context for each word, aiding disambiguation and improving classification.

• Data Preparation

Training data was formatted according to FastText requirements:

```
__label__english This is an English sentence.
__label__french Ceci est une phrase francaise.
__label__german Das ist ein deutscher Satz.
```

Listing 1: FastText Training Data Format

Each line contains a language label followed by the corresponding text, enabling supervised learning for language classification.

• Experimental Results and Analysis

• Performance Metrics

Both systems were evaluated using standard classification metrics:

Table 1: Performance Comparison of Language Detection Systems

System	Languages	Accuracy	Training Time
N-gram Based	200	99.45%	1.5 hours
FastText (30L)	30	99.83%	4 minute
FastText (200L)	200	98.45%	25 minutes

• Detailed Analysis

N-gram System (200 Languages):

- Achieved exceptional accuracy of 99.45% across all 200 languages
- Demonstrated consistent performance across diverse linguistic families
- Showed particular strength in distinguishing languages with different writing systems
- Memory footprint: 50MB for all language profiles
- Inference time: 10ms per document

FastText 30-Language Model:

- Highest accuracy of 99.83% on the reduced language set
- Extremely fast training (under 4 minutes on standard CPU)
- Excellent performance on short texts and noisy inputs
- Model size: 2.1 GB
- Inference time: 1ms per document

FastText 200-Language Model:

- Competitive accuracy of 98.45% on the full dataset
- Scalable architecture handling large-scale classification
- Robust performance across morphologically diverse languages
- Model size: 5.8 GB
- Inference time: 5ms per document

• Error Analysis

Common Error Patterns:

- Confusion between closely related languages (e.g., Norwegian/Danish, Hindi/Urdu)
- Challenges with very short texts ($<$ 20 characters)
- Difficulties with code-switched or multilingual documents
- Performance degradation on highly technical or domain-specific texts

System-Specific Observations:

- N-gram system: More robust to spelling variations and OCR errors
- FastText system: Better performance on informal text and social media content
- Both systems: Excellent performance on clean, well-formed text

• Comparative Analysis

• Advantages and Limitations

N-gram Based Approach:

Advantages:

- High interpretability and explainability
- Robust to domain variations
- Language-agnostic approach
- Minimal computational requirements for training
- Excellent accuracy on clean text

Limitations:

- Sensitive to text length
- Fixed feature representation
- Challenges with highly similar languages
- No learning from prediction errors

FastText Approach:

Advantages:

- Extremely fast training and inference
- Subword information handling
- Robust to vocabulary variations
- Scalable to large numbers of languages
- Good performance on short texts

Limitations:

- Requires larger memory footprint
- Less interpretable than N-gram approach
- Sensitive to hyperparameter tuning
- May overfit on small datasets

• Use Case Recommendations

N-gram System: Recommended for applications requiring high interpretability, limited computational resources, or deployment in resource-constrained environments.
FastText System: Optimal for high-throughput applications, real-time processing, or scenarios with diverse text types and informal language usage.

• Implementation and Code Availability

Both systems were implemented in Python with comprehensive documentation and modular design. The complete implementation is available on GitHub at:
`https://github.com/abhiseksarkar2001/language_detector`

• Future Work and Extensions

Several directions for future research and development have been identified:
Methodological Extensions:

- Hybrid approaches combining N-gram and neural methods
- Ensemble methods leveraging multiple classifiers
- Deep learning approaches using transformer architectures
- Zero-shot learning for new language detection

Application Domains:

- Social media and informal text processing
- Multilingual document processing
- Code-switching detection and handling
- Historical text and manuscript analysis

Technical Improvements:

- Real-time processing optimizations
- Model compression techniques
- Federated learning for privacy-preserving training
- Active learning for continuous model improvement

• Conclusion

This study compares classical N-gram and modern FastText methods to multilingual language identification. Both performed well on a list of 200 languages, with N-gram achieving 99.45% and FastText 99.83% (30 languages) and 98.45% (200 languages). Classical statistical methods are similar to language identification, particularly regarding interpretability and speed. Modern neural approaches such as FastText are superior in terms of training speed, scalability, and stability to text variations. Choose a method based on the requirements of the application, constraints, and performance requirements. N-gram method is suitable in resource-scarce scenarios where interpretability becomes an essentiality. FastText offers better performance in real-time applications with varied text. This work enhances language identification methods and presents real-world applications for multilingual NLP. The evaluation of 200 languages offers scope for further improvements in multilingual detection system research.

References

[1] W. B. Cavnar and J. M. Trenkle, "N-Gram-Based Text Categorisation," in *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, NV, 1994, pp. 161–175. Available: `https://www.let.rug.nl/vannoord/TextCat/textcat.pdf`

[2] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," *arXiv preprint arXiv:1607.01759*, 2017. [Online]. Available: `https://arxiv.org/abs/1607.01759`

[3] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017. Available: `https://arxiv.org/abs/1607.04606`

[4] R. D. Brown, "Non-linear Mapping for Improved Identification of 1300+ Languages," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 627–632. Available: `https://aclanthology.org/D14-1069/`

[5] M. Lui and T. Baldwin, "langid. py: An Off-the-shelf Language Identification Tool," in *Proceedings of the ACL 2012 System Demonstrations*, 2012, pp. 25–30.

Available: `https://aclanthology.org/P12-3005/`