Automated Language Detection System: A Machine Learning Approach to Natural Language Identification

Mohammed Ufraan
Roll No: 160923733152
ufraan1@gmail.com

Faheem Ahmed
Roll No: 160923733148
faheemahmedhyd81@gmail.com

Shaik Mohd Abrar
Roll No: 160923733172
shaikmohdabrar2021@gmail.com

***Abstract*—Language identification is a fundamental task in Natural Language Processing (NLP) with critical applications in content classification and multilingual systems. This paper presents a beginner-friendly implementation of automated language detection using character-level n-gram features and the Naive Bayes classification algorithm. We explore how simple probabilistic models can effectively distinguish between languages without requiring complex deep learning architectures. By extracting character sequences from text, our model learns the unique patterns of each language. Our results demonstrate a high accuracy of 98.07% across 17 diverse languages, with detailed analysis of challenges encountered and their practical solutions during development.**

***Index Terms*—language detection, NLP, character n-grams, Naive Bayes, text classification, machine learning, TF-IDF vectorization**

I. Introduction

Language identification is something you encounter every day without thinking about it. When you use Google Translate, search the web in different languages, or use a content recommendation app, the system must first figure out what language your input is in. This process is called **Language Identification**, and it is the first step in many NLP applications.

From a computer's perspective, text is just a string of characters or symbols. As beginners in Computer Science, we might initially think that identifying a language requires storing giant dictionaries with every word in every language. However, this approach would be extremely slow and use massive amounts of memory. Instead, we can use Machine Learning to analyze the statistical patterns of individual letters and letter combinations.

In this project, we built a simple language detection system using a **Multinomial Naive Bayes** classifier, which is a standard starting point for text classification tasks taught in introductory ML courses. Our approach is practical, efficient, and easy to understand, making it ideal for undergraduate-level implementation.

II. Motivation and Problem Statement

The ability to automatically detect languages has many real-world applications:

- **Content Filtering:** Social media platforms need to identify languages to apply the correct moderation rules and filters.
- **Translation Services:** Before translating text, the system must know which language it is.
- **Information Retrieval:** Search engines use language detection to return results in the correct language.
- **Spam Detection:** Many spam and phishing messages use non-English languages. Identifying the language helps in classification.

Our project focuses on building a simple, effective model that can handle a variety of common languages while remaining interpretable and easy to implement on a beginner's skill level.

III. Technical Background and Concepts

A. What are Character N-Grams?

The core idea behind our approach is that different languages use different combinations of letters. For example, in English, the sequence "the" is very common, while in German, sequences like "der" and "die" are more frequent. Instead of analyzing entire words, we use **character n-grams**.

An n-gram is simply a sequence of n consecutive characters from a text. Let us look at the word "Language" as an example:

- **Unigrams (n=1):** L, a, n, g, u, a, g, e
- **Bigrams (n=2):** La, an, ng, gu, ua, ag, ge
- **Trigrams (n=3):** Lan, ang, ngu, gua, uag, age

By counting how frequently different bigrams and trigrams appear in a text, the model learns the "flavor" or "fingerprint" of a language, even without understanding individual words.

B. Data Preprocessing and Cleaning

Machine Learning models work best when the input data is clean and properly formatted. Our preprocessing pipeline involves three main steps:

1) **Lowercasing:** We convert all text to lowercase so that "Apple" and "apple" are treated identically. This reduces noise and helps the model focus on the actual character patterns rather than capitalization.
2) **Removing Noise:** We strip out special characters, numbers, punctuation marks, and extra whitespace. These

Source Code: https://github.com/botl9/Automated-Language-Detection

elements do not help in identifying the language itself and only add complexity to the model.

3) **Vectorization Using TF-IDF:** Machine Learning models understand numbers, not text. We convert the n-grams into numerical vectors using **TF-IDF** (Term Frequency-Inverse Document Frequency).

C. Multinomial Naive Bayes Classifier

We chose the Naive Bayes algorithm because it is simple to understand, computationally fast, and surprisingly effective for text classification. The algorithm is based on **Bayes' Theorem**.

$$P(\text{Language} \mid \text{Text}) = \frac{P(\text{Text} \mid \text{Language}) \cdot P(\text{Language})}{P(\text{Text})} \tag{1}$$

The "Naive" part refers to the simplifying assumption that every feature (each n-gram) is independent of every other feature. While this is not strictly true in natural language, it works surprisingly well in practice for classification tasks.

IV. Implementation Architecture and Workflow

A. System Pipeline Overview

Our complete system follows these steps:

1) **Data Collection:** We utilized the "Language Detection Dataset" by Basil Saji, sourced from Kaggle. This dataset contains 10,337 text samples distributed across 17 distinct languages, including major European, Asian, and Middle Eastern scripts.
2) **Data Splitting:** We randomly divided the data into 80% for training and 20% for testing.
3) **Model Training:** The classifier learned which n-grams are associated with each language using the training data.
4) **Model Testing:** We evaluated the model on unseen test data to measure its accuracy.
5) **Analysis:** We examined where the model made mistakes to understand its limitations.

B. Code Implementation

1) Step 1: Loading and Preparing Data

```python
import pandas as pd
from sklearn.model_selection import
    ↪ train_test_split

# load dataset
df = pd.read_csv('dataset.csv')

# extract features (text) and labels (language)
X = df['Text']
y = df['Language']

# split 80:20 train:test
X_train, X_test, y_train, y_test =
    ↪ train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"Training samples: {len(X_train)}")
print(f"Testing samples: {len(X_test)}")
```

2) Step 2: Vectorization and Model Training

```python
from sklearn.feature_extraction.text import
    ↪ TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

# pipeline combining vectorization &
    ↪ classification
model = Pipeline([
    ('tfidf', TfidfVectorizer(
        analyzer='char',
        ngram_range=(2, 3),
        max_features=5000
    )),
    ('classifier', MultinomialNB())
])

# train model
model.fit(X_train, y_train)
```

3) Step 3: Evaluation and Prediction

```python
from sklearn.metrics import accuracy_score,
    ↪ classification_report

# make prediction
y_pred = model.predict(X_test)

# calc accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2%}")

print(classification_report(y_test, y_pred))

# function to predict language of new text
def predict_language(text):
    result = model.predict([text])
    confidence = model.predict_proba([text]).
        ↪ max()
    return result[0], confidence
```

V. Results and Performance Analysis

A. Overall Accuracy

After training and testing the model on the Basil Saji dataset, we achieved an overall accuracy of approximately **98.07%**. This performance was consistent across 17 languages, including English, Arabic, Russian, Hindi, and several others. This high accuracy demonstrates the robustness of character-level n-gram analysis even for languages with significantly different scripts.

B. Per-Language Performance

Different languages showed varying levels of accuracy. Languages with unique character patterns (like Arabic or Chinese) were easier to identify. However, languages that share similar alphabets and letter frequencies showed lower accuracy.

C. The Confusion Matrix

A confusion matrix helps us understand not just how accurate the model is, but specifically where it makes mistakes. We discovered two main patterns:

- **Similar Languages:** Portuguese and Spanish share many n-grams, causing occasional misclassification between them.
- **Short Texts:** When the input is only 1–2 words, there are too few n-grams for the model to make a confident prediction.

VI. Challenges Encountered and Solutions

During development, we faced several practical challenges. We documented each challenge and the solution we implemented:

TABLE I
Challenges and Implemented Solutions

Challenge	Solution	Result
Slow training on large data	Limited n-grams to top 5000 features	10× faster
Special characters confusing model	Preprocessed text to remove symbols	Cleaner patterns
Imbalanced language distribution	Ensured equal samples per language	Fairer model
Low confidence on short phrases	Added confidence threshold filter	Fewer mistakes
Similar languages (Spanish/Portuguese)	Increased n-gram range to (2,3)	Better differentiation

A. Why We Chose Bigrams and Trigrams

We initially tried using only bigrams (n=2), but the model struggled with similar languages. By expanding to include trigrams (n=3), the model could see longer patterns and distinguish between related languages more effectively. We did not go beyond trigrams because it would increase computation time significantly without proportional accuracy gains.

VII. Limitations and Future Improvements

Our current model has several limitations that are natural for a beginner-level system:

- **Language Coverage:** While we expanded our scope to 17 languages, increasing coverage to hundreds of languages would require more diverse data and higher computational resources for feature engineering.
- **Sensitivity to Mixed Languages:** Our model assumes the input is in a single language. Code-switching or multilingual text confuses the classifier.
- **Script Dependency:** Languages using the same script (like English and German) are harder to distinguish than languages using different scripts (like English and Arabic).

For future work, we could:

- Train on more languages and larger datasets to improve generalization.
- Use more advanced algorithms like Support Vector Machines (SVM) or neural networks.
- Handle multilingual detection by identifying dominant languages in mixed-language text.
- Deploy the model as a REST API for real-world applications.

VIII. Conclusion

This project provided valuable hands-on experience with the complete machine learning workflow: from data collection and preprocessing to model training, evaluation, and analysis. By using character n-grams and Naive Bayes, we built a system that is both computationally efficient and easy to understand.

A key takeaway for us as CS undergraduates is that good data preprocessing and thoughtful feature engineering often matter more than using complex algorithms. In many cases, a simple, well-tuned model beats a complicated one that is poorly understood.

The simplicity of our approach does not diminish its effectiveness—it demonstrates that fundamental ML concepts, when applied correctly, can solve real-world problems. We hope this project serves as a useful reference for other students beginning their journey into machine learning.

Acknowledgments

We are grateful to our Machine Learning instructor for providing the theoretical foundation required to complete this project. Special thanks to the scikit-learn documentation and open-source community for the excellent libraries that made this implementation straightforward.

References

[1] T. Dunning, "Statistical identification of language," CRL Technical Report, 1994.
[2] Scikit-learn developers, "Working with Text Data," scikit-learn documentation, [Online]. Available: https://scikit-learn.org/.
[3] P. Norvig, "Natural Language Corpus Data," 2009. [Online]. Available: https://norvig.com/ngrams/.
[4] A. Ng and K. Katanforoosh, "Machine Learning Specialization," Coursera, 2022.
[5] M. A. Ahsan, K. Ahmad, J. Ahamed, M. Omar, and K. A. B. Ahmad, "PAPQ: Predictive analytics of product quality in industry 4.0," Sustainable Operations and Computers, vol. 4, pp. 53–61, 2023. doi: 10.1016/j.susoc.2023.02.001.

Scan to view Live Project & Source Code
https://lang.ufraan.dev