Kaggle Competition Report: Language Identification via Text Classification

Abdellah Oumida, Elisa Faure, Idriss Mortadi, Marianne Brugidou
CentraleSupélec, France
{abdellah.oumida, elisa.faure, idriss.mortadi, marianne.brugidou}@student-cs.fr

Abstract

This report outlines our approach for the Advanced NLP Class Kaggle Competition, which involves predicting the language of a given text using the ISO-639-3 standard. Our solution employs the `xlm-roberta-base` model with parameter-efficient fine-tuning (LoRA), complemented by data augmentation and a cosine annealing scheduler. This method achieved a leaderboard score of **0.87996**.

1 Introduction

Language identification is a crucial preprocessing step in many NLP applications. The Advanced NLP Class Kaggle Competition requires classifying a given text into one of 389 languages or dialects, which form a curated subset of the ISO-639-3 standard (SIL International, 2025). This high-granularity classification presents challenges due to the large label space and diverse linguistic features. Our approach leverages a pre-trained multilingual transformer model, adapted through Low-Rank Adaptation (LoRA) (Hu et al., 2022), to address these challenges efficiently.

2 Solution

Our solution consists of several key components:

2.1 Preprocessing

Our text preprocessing pipeline is designed to clean and normalize text data efficiently. We developed a custom dataset **class** to read CSV files, tokenize texts with the HuggingFace tokenizer, and apply data augmentation. Specifically, any text exceeding 128 tokens is truncated or randomly sliced (if `AUGMENT` is enabled) to improve generalization, especially for low-resource languages. Then we proceed to clean the dataset.

The first step involves script detection and filtering, where we use Unicode codepoint ranges to identify the script of each character in the text. By leveraging a lookup table from "unicode_ranges.csv", we map Unicode ranges to specific language groups. To optimize performance, we employ a binary search ("bisect") to determine the script of each character and maintain a frequency count of detected scripts. Once the majority script is identified, the function "filter_majority_script" ensures text uniformity by retaining only characters that belong to the most frequently occurring script in the input text.

In addition to script filtering, we remove unnecessary elements such as URLs, tags, and hashtags. The function ‘remove_links_and_tags‘ eliminates links that start with ‘http://‘, ‘https://‘, or ‘www.‘, as well as tags that begin with ‘@‘ and hashtags that start with ‘#‘. To enhance readability, multiple consecutive spaces are reduced to a single space.

By implementing these preprocessing steps, our approach ensures that the text is clean, consistent, and free from unnecessary, making it more suitable for training our models.

2.2 Training

In Kaggle's cloud environment, we have access to two NVIDIA Tesla T4 GPUs (NVIDIA Corporation, 2018). Each T4 GPU features 2,560 CUDA cores and 16 GB of GDDR6 memory. By leveraging Hugging Face's Accelerate library (Gugger et al., 2022), we can efficiently train large-batch models using these GPUs.

- **Model Architecture:** We build upon a fine tuned version of **xlm-roberta-base** we found on HuggingFace: **papluca/xlm-roberta-base-language-detection**[1], an XLM-RoBERTa variant fine-tuned for language identification. It relies on the XLM-RoBERTa architecture (Conneau et al., 2019) with a classification head on top (i.e., a linear layer over the pooled output). This model can be used directly as a language

[1] papluca/xlm-roberta-base-language-detection

detector for 20 languages, including Arabic, English, French, Hindi, and Chinese, among others. We integrate LoRA (Hu et al., 2022) via the PEFT framework (Mangrulkar et al., 2022), updating only the query and value modules in the transformer blocks. This parameter-efficient approach substantially reduces the number of trainable parameters, which is particularly beneficial given the large number of language classes.

- **Data Processing:** We set the maximum sequence length to 128 tokens. Random slicing is applied for texts exceeding 128 tokens to diversify the training examples.
- **Training:** We train for 40 epochs with a learning rate of 4e-5 and weight decay of 0.01. Large batch sizes (320 samples per device) ensure stable gradient estimates. We use the AdamW optimizer with a learning rate scheduler that employs cosine annealing with warm restarts, where $T_0 = 5$, $T_{\text{mult}} = 2$, and $\eta_{\text{min}} = $ 1e-6. We log training metrics every 100 steps.
- **LoRA parameters:** We use LoRA (Low-Rank Adaptation) for efficient fine-tuning. Key parameters include:
  - **Rank (`r`)**: 8, rank of LoRA update matrices.
  - **Scaling Factor (`lora_alpha`)**: 32, scales LoRA updates.
  - **Dropout (`lora_dropout`)**: 0.1, prevents overfitting.
  - **Target Modules**: `"query"` and `"value"`.

3 Results and Analysis

Metric	Precision	Recall	F1-score
Macro Average	0.88	0.88	0.88
Weighted Average	0.88	0.87	0.87

Table 1: Overall evaluation metrics on the validation set.

We evaluated our model on a held-out validation set, achieving an overall accuracy of 0.88. Table 1 shows the macro and weighted average metrics, both indicating strong performance across classes. However, a detailed classification report revealed that while certain languages (e.g., `abk`, `ace`, `ach`) reached near-perfect scores, others (e.g., `acm`, `afb`) performed less consistently, suggesting a need for further optimization for low-resource or highly similar dialects.

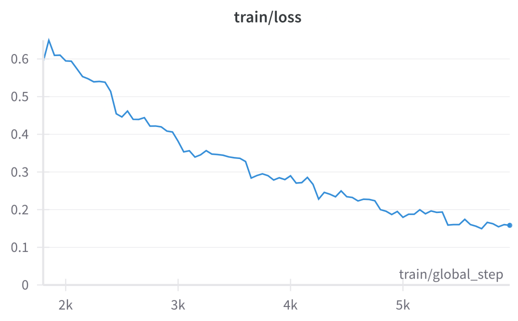

Figure 1: Validation loss over training steps.

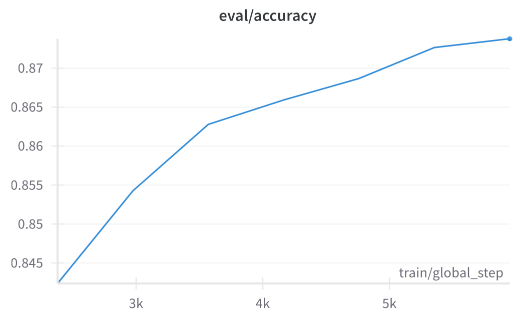

Figure 2: Validation accuracy over training steps.

Figures 1 and 2 illustrate the validation loss and accuracy trends across epochs. The validation loss decreases steadily down to 0.15, while the accuracy improves up to 0.87, indicating consistent learning progress. These curves suggest that the chosen hyperparameters and learning rate schedule allowed for stable training without noticeable overfitting.

Limitations

We initially attempted full fine-tuning but were limited by time and computational resources. Instead, we used LoRA for efficiency, though it has trade-offs. Our model's fixed maximum sequence length may affect performance on longer inputs. Evaluation was done on a single pre-trained model and data split. Ensemble methods and further hyperparameter tuning could improve performance, especially for underrepresented languages.

References

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

NVIDIA Corporation. 2018. Nvidia tesla t4 tensor core datasheet. Accessed: 2025-03-07.

SIL International. 2025. ISO 639-3 Macrolanguage Mappings. Accessed: 2025-02-25.