# NLI Project Report

- Mohamad Hijazi
- Mias ghantous

## Native Language Identification with User Generated Content- paper:

### Introduction:

Native Language Identification (NLI) is a computational linguistics task aimed at determining an author's native language (L1) based on their writing in a second language. It's important for language learning, sociolinguistics, and authorship profiling. This paper addresses the complex scenario of identifying the native language of advanced non-native English speakers based on user-generated content on social media platforms, specifically Reddit.

### Dataset:

The study employs a vast corpus of English Reddit posts, accurately annotated in 23 native languages across 29 countries, with over 230 million sentences and 3.5 billion tokens. The dataset, which includes texts from highly proficient non-native speakers, presents a more challenging task than traditional NLI on learner corpora.

the data is combined of 2 sets, the first is Europe data which consist of focused subreddits (Europe, AskEurope, Euro peanCulture) whose content is generated by users specifying their country as a flair, and the second is out-of-domain which consist of other subreddits that were written by the same authors as the first set.

### Methodology:

- Binary Classification: Distinguishing between native and non-native authors
- Language Family Classification: Determining the language family of the user (Germanic, Balto-Slavic, Romance, or native English)
- Native Language Identification: Identifying the specific native language of the user (23-way classification).

**<u>Features and Results:</u>**

The authors employ a wide range of features, which can be categorized as follows:

| Feature | Binary | Families | NLI |
|---|---|---|---|
| Character trigrams | 89.59 | 82.34 | 72.07 |
| Token unigrams | 90.12 | 83.72 | 73.50 |
| Spelling errors | 73.97 | 55.66 | 27.00 |
| Grammer errors | 77.23 | 49.65 | 14.44 |
| Function words | 81.09 | 61.25 | 28.21 |
| POS trigrams | 75.50 | 55.83 | 26.31 |
| Sentence length | 52.85 | 28.82 | 6.07 |

In addition to the individual features above, we train different features combinations as described on the paper.

| features combinations | Binary | Families | NLI |
|---|---|---|---|
| character trigrams, tokens, and spelling errors(edit distance) | 91.39 | 83.27 | 75.57 |
| Function words, POS trigram and sentence length | 81.09 | 53.19 | 28.32 |
| Spelling errors and grammar errors | 81.90 | 56.06 | 23.77 |

Features extraction process:

1) <u>Character Trigrams</u> - are extracted by removing spaces from text and creating sequences of three consecutive characters, capturing unique sub word patterns in specific languages. By counting their frequency, the code identifies common sequences within a dataset, aiding in language distinction. We tokenize text into trigrams using "nltk" library, calculates trigram frequency and use the top 1000 most frequent character trigrams. the feature is a normalization over the number of trigrams in the dataset.

2) <u>Token unigram</u> - are key to understanding text vocabulary distribution. The model uses tokenization, frequency counting, top unigrams, and feature vector creation to identify essential aspects of a language's lexicon and distinguish between native languages, we extracted the tokens using "nltk" library and we excluded the stop words, and we just used the top 1000 most frequent unigram, also the feature is a normalization over the number of tokens in the text.

3) Edit distance - The average edit distance between misspelled words and their corrections measures the severity of spelling errors. It is determined by detecting spelling errors using "language_tool_python", computing the "Levenshtein" distance for each misspelled word and the corrected word that was provided as the first replacement of the language tool, averaging the edit distances by dividing the total number of tokens (this time we count stop words), and creating a feature vector to represent the general tendency of spelling errors.

4) Spelling errors - The analysis of spelling errors helps identify the influence of a writer's native language on their second language usage. The process involves detecting errors using the "language_tool_python" library, analyzing the corrections, and calculating the "Levenshtein" distance between the misspelled word and the correction. The frequency of each edit operation is counted, providing a profile of common spelling error patterns (insert of a letter, deletion of a latter, and substituting 2 letters). A feature vector consists of 400 features (most common patters) representing the frequency of each edit operation is created for each document.

5) Grammar errors – we used a language tool that detects and counts grammatical errors in text, categorizing them by type and text length. The frequency of these errors can reveal the writer's native language, as non-native speakers often make specific mistakes influenced by their native language, providing insight into grammatical challenges in second language writing. The "language_tool_python" library is used to identify and count grammatical errors in text, we normalizing these counts based on text length to create a feature vector. Also we used "collections.defaultdict" to store the error counts in a way that allows easy accumulation and later normalization by text length.

6) Function words – these words are used differently across languages and are less likely to be consciously altered by writers. They provide a stable basis for linguistic analysis and are effective in Natural Language Processing (NLI) tasks. The extraction process involves compiling a list of function words, tokenizing each document, counting their frequency, and creating a feature vector for each document.

7) POS trigram - are language-specific sentence structures extracted by tagging words with their grammatical roles and generating sequences of three consecutive POS tags. The text is tokenized into words and tagged with part-of-speech using the "nltk.pos_tag" function. Trigrams representing common syntactic patterns are generated, and the frequency of these trigrams is calculated across the dataset. Analyzing their frequency helps identify patterns in sentence construction.

8) Sentence Length - The text is tokenized into sentences and words to calculate the average word count per sentence, a linguistic feature that can vary across languages. We used "nltk" library for tokenizing text into sentences (nltk.sent_tokenize) and words (nltk.word_tokenize). For each document, we calculate the average number of words per sentence, which serves as a measure of sentence length. This average is stored as a feature vector that can be helps identify the native language by capturing typical sentence structure and length.

## **Out-of-domain-test:**

We tested the SVMs that we got on the out of domain data to check the robustness of the combined features, for each country we picked random 20 users to make the test faster.

| features combinations | Binary | Families | NLI |
|---|---|---|---|
| character trigrams, tokens, and spelling errors(edit distance) | 81.9 | 56.06 | 23.77 |
| Function words, POS trigram and sentence length | 75.73 | 42 | 12.4 |
| Spelling errors(edit distance and the patterns) and grammar errors | 68.4 | 34.02 | 15.06 |

## **A Deep Generative Approach to Native Language Identification – paper:**

We fine-tune a GPT-2 language model using text data from different countries to adapt its performance to specific regional languages and styles. This process involves merging text files from different individuals within each country into a consolidated dataset, tokenizing this text, and managing it with a TextDataset object. The text data is split into training (80%), testing (10%), and validation (10%) sets for each country. The GPT-2 model is fine-tuned using the Trainer class from the Hugging Face transformers library, with specified training arguments and parameters. The model is saved for evaluation, ensuring it is well-adapted to the linguistic characteristics of each target region.

After that, we evaluate the performance of fine-tuned GPT-2 models on unseen texts from different languages. It loads validation texts from files corresponding to each language, prepares a dataset for each language, and evaluates it against fine-tuned models using the

GPT-2 model and tokenizer. The model with the lowest loss on unseen texts is identified as the best-performing model. The results, including the model with the lowest loss and its match to the expected language, are stored in a dictionary and saved to a JSON file for further analysis. In this approach we get an accuracy of 91%.

On the other hand, the text is processed into chunks, we dividing it into smaller segments, evaluated against all models individually, and calculated the loss for each model on these segments. This method offers a more granular evaluation, assessing model performance on smaller text samples rather than the entire dataset. The code aggregates results, computes accuracy, and saves performance metrics to a JSON file. In this approach we get an accuracy of 72.59%.

We also did an evaluation over the out-of-domain data, we choose random 100 chunks for each language, and we got 25.13 accuracy.