Language proximity analysis

Adrian Bagiński, Mateusz Cakała, Krzysztof Olczyk

October 2025

1 Introduction

Program's goal is to compare words from different languages and find out how similar they are to each other using special algorithms. Then it creates a graph picturing how similar are they to each other. Levenshtein's distance method is going to be used in process. Code is written in Python programming language.

2 Prototype version - October 15th 2025

First and very basic version of the program that will be developed later. It has few files with various function realizing functionalities as taking or saving data and creating a comparison of languages.

2.1 file_utils.py

This part of the program defines a set of utility functions responsible for handling input and output operations related to word lists and similarity matrices. They serve as supporting tools for the main part of the program, which compares words between two languages and calculates their similarity.

2.1.1 `get_words_from_file(file_path)`

Purpose: Reads a list of words from a text file and returns them as a Python list.

Description:

- Opens the file specified by `file_path` in UTF-8 encoding.
- Reads all lines, strips unnecessary whitespace using `.strip()`.
- Filters out empty lines so that only valid words remain.

Relevance: This function is likely used to load vocabulary lists for each language being compared. The resulting lists are then used in similarity calculations between languages.

2.1.2 `save_words_to_file(words, file_path)`

Purpose: Saves a list of words to a text file, ensuring that the output directory exists.

Description:

- Creates the output directory using `os.makedirs()` if it does not already exist.
- Writes each word from the `words` list to a new line in the file.

Relevance: This function allows saving processed or filtered word lists that can later be used for further comparison or analysis.

2.1.3 `save_similarity_matrix(words1, words2, matrix, file_path)`

Purpose: Saves a complete similarity matrix to a CSV file, where each cell represents the similarity between a word from the first language (`words1`) and a word from the second language (`words2`).

Description:

- Ensures the output directory exists before saving.
- Opens a CSV file in UTF-8 encoding and writes:
  - A header row containing all words from `words2` (the column labels).
  - For each word in `words1`, a new row containing the similarity scores from `matrix`.
- Each similarity value is formatted to two decimal places using `f"{v:.2f}"`.

Comment translation (from Polish):

> "Saves the full similarity matrix to a CSV file. Rows: `words1`, Columns: `words2`."

Relevance: This function provides a structured and human-readable representation of the computed word similarities, facilitating analysis of how closely related the two compared languages are.

2.2 graph_utils.py

2.2.1 `save_similarity_matrix_csv(matrix, languages, topic, folder="results/graphs")`

Purpose: Saves a language similarity matrix to a CSV file. Each cell of the matrix represents the similarity value between two languages. The file is named according to a given topic, and stored in a specified folder.

Description:

- Ensures that the output directory exists using `os.makedirs()` with `exist_ok=True`.

- Constructs the output file path dynamically, combining the given `folder` and the file name based on the provided `topic` (e.g., `topic_matrix.csv`).
- Opens a new CSV file for writing in UTF-8 encoding.
- Writes the first row as a header, where columns correspond to the list of `languages`.
- Iterates over the `languages` and their corresponding rows in the `matrix`, writing each language name followed by its similarity values.

Relevance: This function is designed to export a matrix that shows the degree of similarity between multiple languages, possibly computed for a particular `topic` or lexical domain. It helps visualize or analyze cross-linguistic relationships by providing a structured and easily readable CSV representation.

2.3 similarity.py

2.3.1 `compute_similarity(word1, word2)`

Purpose: Computes the similarity between two words and returns a numerical value in the range from 0 to 1, where 1 indicates identical words and 0 represents complete dissimilarity.

Description:

- Utilizes the `textdistance` library, specifically the `Levenshtein` distance algorithm.
- The function calls `textdistance.levenshtein.normalized_similarity(word1, word2)`, which measures the similarity by comparing the minimum number of character edits (insertions, deletions, substitutions) required to transform one word into the other.
- The resulting similarity score is normalized to a range between 0 and 1, providing a standardized metric of word resemblance.

Relevance: This function represents the core computational step in the program's language comparison process. By quantifying how similar individual words are, it enables the construction of broader similarity matrices and contributes to the analysis of linguistic relationships between languages.

2.4 translate.py

2.4.1 `translate_word(word, lang)` and `translate_words(words, lang)`

Purpose: Provides automatic translation of words into a specified target language using the Google Translate service. The functions support both single-word and batch translation.

Description:

- The implementation utilizes the `GoogleTranslator` class from the `deep_translator` library.
- `translate_word(word, lang)`:
  - Automatically detects the source language (`source='auto'`) and translates the input `word` into the target language specified by `lang`.
  - Converts the translation result to lowercase to ensure consistency.
  - Handles exceptions gracefully — if an error occurs during translation, an informative error message is printed, and the original word is returned unchanged.
- `translate_words(words, lang)`:
  - Performs batch translation by applying the `translate_word()` function to each word in the provided `words` list.
  - Returns a list of translated words corresponding to the input list.

Relevance: These functions enable multilingual comparison by ensuring that words from different languages are translated into a common reference language before similarity computation. This step is essential for fair and consistent cross-linguistic analysis, especially when measuring lexical similarity between language pairs.

2.5 `main.py` — Program Orchestration and Workflow

Purpose: This script serves as the main entry point of the entire project. It coordinates the process of loading word lists, translating them into multiple languages, computing pairwise word similarities, and saving the resulting data for each topic.

Description:

- **Imports and Configuration:**
  - Imports utility functions from separate modules:
    - `get_words_from_file()`, `save_words_to_file()`, `save_similarity_matrix()` from `utils.file_utils`.
    - `translate_words()` from `utils.translate`.
    - `compute_similarity()` from `utils.similarity`.
  - Defines key configuration variables:
    - `languages = [‘‘en’’, ‘‘pl’’, ‘‘es’’]` — specifies which languages will be compared.
    - `data_dir = "data"` — input folder containing source word lists.
    - `results_dir = "results"` — base folder for all output files.
  - Ensures that required subdirectories exist:

        * `results/translations` — stores translated word lists.
        * `results/similarities` — stores CSV files with similarity matrices.

- **Main Processing Loop:**
    - Iterates through all `.txt` files in the `data` directory. Each file represents a separate *topic* or semantic field.
    - For each file:
        1. Extracts the topic name by removing the `.txt` extension.
        2. Loads the list of words using `get_words_from_file()`.
        3. Translates the list of words into each target language using `translate_words()`.
        4. Saves each set of translated words into separate files within the `results/translations` folder.

- **Similarity Computation:**
    - For every unique pair of languages (`lang1`, `lang2`), computes a word-by-word similarity matrix using `compute_similarity()`.
    - Each element of the matrix represents the normalized similarity score between a word from `lang1` and a word from `lang2`.
    - The resulting matrix is saved as a CSV file in `results/similarities`, named according to the topic and the two compared languages (e.g., `animals_en_pl.csv`).

- **Program Output:**
    - After all topics are processed, a confirmation message is displayed: `‘‘All topics processed successfully!  Check results folder.’’`

Relevance: The `main.py` script integrates all project components into a unified workflow. It automates the full pipeline — from data loading and translation to similarity computation and result storage. This modular structure ensures scalability, allowing new languages or topics to be easily added without modifying the underlying logic of translation or similarity calculation.