Authorship Identification System

Tharake Wijenayake and Gade Aski Reddy

Harrisburg University of Science and Technology

CISC 691-50-B

Prof. Donald O'Hara & Prof. Majid Shaalan

May 28, 2025

Summary

We implemented the authorship identification system described in chapter 11 of Learn AI-Assisted Python by Leo Porter using GitHub Copilot to assist in writing functions. We then improved the program by adding new signature metrics, optimizing the code to improve performance, and structuring the code so that a batch of mystery texts can be run conveniently. The new metrics added were the ratio of contractions to number of words, density of punctuation, and ratio of stop words. The main addition to improving performance was to save the known signature data in a database on the first run and rerunning the function get_all_signatures only if the user indicates it's a new training dataset by setting save_signatures=True in the make_guess function. Copilot is clearly a very useful and powerful tool that can be used to accelerate the development process. However, it sometime produces code that seems to be doing what you requested and even passes a few test cases but there are subtle differences and would fail some edge cases. We found that adding carefully thought out test cases, going over the logic of the AI generated code, and refining the Copilot prompt iteratively are all extremely important.

Base Authorship identifier program

Introduction

We developed the authorship Identification system described in chapter 11 of "Learn AI-Assisted Python” by Leo Porter using the top down problem decomposition principle described in chapter 7. It uses the following five signatures to identify a likely author:

- Average word length
- Different words divided by total words
- Words used exactly once divided by total words
- Average number of words per sentence
- Average sentence complexity

Code documentation

The code consists of the following main functions:

- make_guess: a wrapper function for process_data
- process_data: gets the signatures from get_all_signatures and calculates lowest score using lowest_score
- lowest_score: calculates the score given two signatures and a list of weights
- get_all_signatures: calculates the signature for all the files in a specified directory
- make_signature: calculates the five signature metrics for a given text
- average_sentence_complexity: Calculate average sentence complexity
- average_sentence_length: Average number of words per sentence
- exactly_once_to_total: calculate words used exactly once divided by total words
- different_to_total: Calculate different words divided by total word

- average_word_length: Calculate average word length

We used GitHub Copilot to assist writing the functions using carefully structured prompts. We tested each function using a number of test cases to confirm it performed as required.

Improved authorship Identification program

Assigned Duties

Group Member	Task
Gade	Add signature metric punctuation density
Gade	Add signature metric stopword ratio
Gade	Reviewed performance bottlenecks and restructured the logic to streamline execution
Tharake	Add signature metric contraction density
Tharake	Structure code to save and use the known author signatures in database
Tharake	Modify the code to run a batch of mystery text

Design Process & Improvements

As the authors of "Learn AI-Assisted Python" comment the most computationally heavy task is constructing the dictionary of known signatures. There is no reason to recalculate it unless we are changing the training dataset. Saving the data in a database and giving the user the option to rerun it will improve the efficiency of the code. In fact, for any training dataset of significant size (not just four samples as in our case) any thing else would be completely unviable. We modified the process_data function to save the signature unless the save_signatures is set to False. We also modified it to accept a list of mystery filenames. Copilot was used to help update the function.

In order to come up with some new signature metrics we though of possible simple measures that could capture the idiosyncrasies and style associated with each author and ran some ad hoc tests to determine if their addition influenced the prediction accuracy. Copilot was used to write functions used to calculate the new metrics by setting prompts for the required logic.

We also reviewed performance bottlenecks and restructured the logic to streamline execution.

Finaly we set up the make_guess function to accept any number of mystery text file inputs from the command line using sys.argv so that a batch of mystery files can be conveniently run directly from the command line.

Code Documentation

The code consists of the same main functions as the base authorship identification program, other than the functions used to calculate the new signature metrics.

- make_guess: Can now be run from the command line with any number of mystery text files. It will save the known signature database if save_signatures=True
- process_data: modified to accept a list of mystery text files and will save and use the known signature database if save_signatures=True
- lowest_score: calculates the score given two signatures and a list of weights. The list of weights has to have the length of the new number of signatures.
- get_all_signatures: uncahnged
- make_signature: modified to include the new signatures
- contraction_density: calculates the ratio of contractions to total words in the text

Each function in the code has a clear docstring outlining its inputs, return values, purpose, and a doctest.

The code can be run from the command line using:

Python improved_authorship_identification.py known_author_data_path' 'boolean_save_signatures_flag' 'mystery_file1' 'mystery_file2' …

Challenges & Future Work

Challenges

Copilot sometimes generates code that marginally deviates from the desired logic. They can sometimes pass a few test cases but will fail on edge cases. Copilot is clearly a very useful tool for accelerating code development but care needs to be taken to make sure the logic is doing what is required and test for edge cases and.

The selection of new signature metrics was done in an ad hoc manner. We simply thought of a few signals that will reasonably distinguish between various authors and ran a few tests to see how they affected accuracy. Then we set a random weight to it. However, to properly develop signal metrics we would need a much larger dataset, and we would need to properly analyze each of the proposed signals to isolate the ones with predictive power and systematically come up with an optimal weight for the signals.

Future Work

Some potential future work is to use machine learning algorithms to construct a classification model for the authors using all the signals we have. We can also try to develop some new signals by systematically analyzing a large data set such as the corpus of text from Gutenberg. We can also try to extract contextual information from the text using natural language processing in order to use as a new signal. This will involve extracting features from the text using either a simple bag-of-words with N-grams or a pretrained model like sentence BURT and then training a classification model to identify the author from the text features.