Bohan Yang
9/28/2025
CISC 691
A01 - A Simple Authorship Identification System

For this assignment, I was using Python as the programming language, the code is running in Visual Studio Code (VS Code) on a Mac. For AI assistance, I was interacting with GitHub Copilot, which uses the GPT-4.1 model from OpenAI (model vendor). The way I used it included coding, prompting debugging and reflection feedback for testing. Besides that, I also have used it for research on specific techniques/terminalogies in NLP including what has been used for cleaning and summarizing statistics for the text. Details are as follows:

Function	Purpose	Why / Returns
clean_word	Cleans up a word by converting it to lowercase, removing punctuation and non-alphabetic characters, and stripping whitespace.	Ensures consistency in word analysis by standardizing the format.
average_word_length(words)	Calculates the average length of words in a list.	Authors may have characteristic word lengths, which helps in stylometric analysis.
different_to_total(words)	Computes the ratio of unique words to total words.	Measures vocabulary richness, which can differ by author.
exactly_once_to_total(words)	Finds the ratio of words that appear exactly once to the total number of words.	Captures the use of rare words, another stylometric feature.

split_string(text)	Splits a text into a list of words using whitespace.	Prepares the text for word-level analysis.
make_signature(words)	Generates a tuple of stylometric features from a list of words, such as average word length, vocabulary richness, and more.	Creates a numerical "signature" for each text to compare writing styles.
get_all_signatures(texts)	For each known author, reads their text, cleans and splits it into words, and computes a stylometric signature (tuple of features).	Returns a dictionary mapping each author's name to their signature.
make_guess	For an unknown text, computes its signature, normalizes it, and compares it to all known authors' normalized signatures using weighted Euclidean distance.	Returns the author whose signature is closest (smallest distance) to the unknown text.

Version I gives guess that is has lower accuracy to the actual results (accuracy 25%):

	Version I	Actual Results
Unknown Text 1	Charles Dickens	Arthur Conan Doyle
Unknown Text 2	Arthur Conan Doyle	Charles Dickens
Unknown Text 3	Jane Austen	Jane Austen
Unknown Text 4	Charles Dickens	Mark Twain

Based on this, I implemented the following improval through experiments and investigation -

1. I enhance the function of make signature, to basically have it capture features of certain authors with more aspects using more metrics such as unique word %, long word % (since some authors favor long words and I would like the algorithm to capture this feature) and etc. Details are as follows:

Metric	**Purpose**	**Why**
Proportion of words longer than 7 letters	Measures the fraction of words in the text that are longer than 7 letters	Some authors favor longer, more complex words, which helps distinguish their style
Proportion of short words (≤3 letters)	Calculates the fraction of words that are very short (three letters or fewer)	Frequent use of short words may indicate a simpler or more conversational style
Ratio of unique long words to unique words	Finds the proportion of unique words that are long (over 7 letters)	Shows whether an author's vocabulary includes many distinct long words
Hapax Legomena Ratio (once / unique words)	Measures the fraction of unique words that appear only once in the text	A high ratio suggests diverse vocabulary and less repetition, which can be author-specific
Type–Token Ratio (unique / total words)	Calculates the ratio of unique words to the total number of words	Indicates vocabulary richness; higher values mean more varied word use

Results given by these added five features are (accuracy 25%):

	Version II - 1	Actual Results
Unknown Text 1	Mark Twain	Arthur Conan Doyle
Unknown Text 2	Mark Twain	Charles Dickens
Unknown Text 3	Jane Austen	Jane Austen

Unknown Text 4	Charles Dickens	Mark Twain

I noticed unknown txt 1 and 2 are both classified as Mark Twain, so I went back and looked into the algorithm, noticed that potentially some added features are assigned equal weights in the make_guess function in calculation of Euclidean distance. This makes some features that naturally have larger numbers within their range have dominant weights than other features. For example, within the all 8 features (3 from version I and 5 added feature), Feature 1 might have a range of [1,100] while feature 6 might have a range [0.01,0.2] which give them equal weights to compare is unfair and won't provide information for the algorithm to make decisions. This gives my second enhancement -

2. I normalize each feature to [0,1] make sure they are comparable. Updated results are (accuracy 50%):

	Version II -2	Actual Results
Unknown Text 1	Mark Twain	Arthur Conan Doyle
Unknown Text 2	Charles Dickens	Charles Dickens
Unknown Text 3	Jane Austen	Jane Austen
Unknown Text 4	Charles Dickens	Mark Twain

3. To further enhance the accuracy, I noticed that
    a. All normalized features have equal weights in calculating Euclidean distance in make_guess function - some of them should have different weights to emphasize its importance
    b. Some features are duplicated that might cause co-effect to the result - e.g, Type-Token Ratio (unique words / total words) and Different to total; Hapax Legomena Ratio (once / unique words) and exactly_once_to_total. So I deleted two of the added features to avoid noises and the final kept feature

becomes [avg_word_len, diff_to_total, once_to_total, prop_long_words, prop_short_words, unique_long_ratio].

Since Feature 2,6 has the largest differences between authors, especially for Mark Twain and Arthur Conan Doyle given their style, I give them higher weights. Feature 1 and 3 is not so distinguishable from others but still important to tell Jane Austen and Charles Dickens from others given their text length and vocabulary richness, so I give them moderately high weights. With weight = [8,18,8,1,1,15], I get the final results (accuracy - 100%):

	Version II -3	Actual Results
Unknown Text 1	Arthur Conan Doyle	Arthur Conan Doyle
Unknown Text 2	Charles Dickens	Charles Dickens
Unknown Text 3	Jane Austen	Jane Austen
Unknown Text 4	Mark Twain	Mark Twain

Couple reflections:

1. I like this assignment because it gets me familiar with how AI can best help in coding, adding scope (like the metrics) to existing frameworks and debugging. I didn't see any new challenges since I used to use AI tools a lot in learning new knowledge but this is my first time incorporating it into coding.
2. The most boring part I guess would be tuning the parameter to get 100% accuracy which I tried both myself and leveraging AI to understand the fact and see which approach of tuning is more efficient.

Future work and development:

How to improve accuracy

1. Adding more / more dedicated metrics (like considering in unit of sentence) into make signatures
2. Change calculation method in make_guess from Euclidean distance to other norms
3. Better Normalization (like Z Normalization)
4. Systematically tune weights (upgrade from current approach of based on observations)

5. Adding more text data for training
6. Ensemble methods - use the results of several feature sets or algorithms and vote for the best guess.

More comprehensive testings:

1. Adding external / longer text into for testing
2. Adding blinding testing samples, e.g give text from five different authors but the algorithm makes a guess on which four.