A Simple Authorship Identification System

By: Divyansh Aggarwal, Sangram Gaikwad, Vishal Varadharajan (Group –4)

These are the videos in order:

1. Divyansh video link - Meeting with Divyansh Aggarwal-20250217_001722-Meeting Recording.mp4
2. Sangram video link -
   https://myharrisburgu-my.sharepoint.com/:v:/g/personal/sgaikwad1_my_harrisburgu_edu/EUxdDB1KSr5OIOoFLFhlHUQByCKfvoFdg2Bov8nBWMJAnQ?e=zwQTav&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifSwicGxheWJhY2tPcHRpb25zIjp7fX0%3D
3. Vishal video link, GitHub Link (Individual Branch) -
   https://myharrisburgu-my.sharepoint.com/personal/vvaradharajan_my_harrisburgu_edu/_layouts/15/stream.aspx?id=%2Fpersonal%2Fvvaradharajan%5Fmy%5Fharrisburgu%5Fedu%2FDocuments%2FRecordings%2FMeeting%20with%20Vishal%20Varadharajan%2D20250216%5F172751%2DMeeting%20Recording%2Emp4&ga=1&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2Ec6404fe2%2D8f1d%2D4605%2D837a%2Dd1cb3bae4a5d

We have submitted the combined code file through Divyansh's submission. We just feel we need to structure our report in terms of three parts explaining contributions from three of us individually which also involves our individual comments on challenges we faced while implementing our own parts. That being said, we want to emphasize that we all three have been working in sync with each other and have collaborated with each other through multiple team meetings while working on the project.

GitHub repo link - https://github.com/daggarwal2806/AuthorshipIdentificationSystem

More information is given in Divyansh's contribution.

Divyansh’s contribution

My submission contains the group’s improved_authorship_identification.py file. I am also going to submit the gutenberg_metadata.csv file which will be used by the program to train on the dataset. I will also try to add a trained BERT-based model (only on 5 authors with multiple books) so that it can be used for the program directly but it’s a large model so there might be space constraints.

I did all the functional implementation and testing on Google Colab using a Jupyter notebook and used Gemini AI and at times co-pilot to get code using prompts.

Design Process of Improvements to the Initial Code

The initial implementation aimed to build an authorship identification system using a machine learning model. The initial models tested were traditional machine learning models such as Random Forest, SVM, and Logistic Regression. However, these models performed poorly, with accuracies in the range of 0.001 to 0.009 when trained on a dataset containing all authors with text from just one book each. Given the nature of the problem—where sequence and contextual information play a crucial role—I decided to explore deep learning models.

The first approach used a pre-tokenized dataset where tokens were pre-calculated before being passed into the model. However, upon further examination, it became evident that this method had several drawbacks, including inefficiencies in handling text sequences dynamically and potential loss of flexibility in tokenization.

To address this, I iteratively refined the approach by dynamically tokenizing input text within the data processing pipeline. This modification ensured that the model could process previously unseen text while maintaining consistency in tokenization. I explored various embedding techniques, including pretrained models like BERT and GloVe, to improve feature representation.

I first experimented with an LSTM-based model, which yielded an accuracy of **76.71%** on the same dataset. Seeing a significant improvement, I realized that sequence-based models were more suited for this task.

To speed up training and evaluation, I created a subset of the dataset by selecting only five authors and using text from just one book per author. This allowed for faster iteration and experimentation, especially considering resource constraints on Google Colab, where I used a T4

GPU. On this reduced dataset, I trained a BERT-based model, which achieved an impressive accuracy of **91.82%**.

Encouraged by these results, I expanded the dataset by including all available books from the selected five authors. This improved the model's ability to generalize across different books by the same author. The BERT model on this dataset reached an accuracy of **96.47%**, while the LSTM model achieved **87.56%**.

Despite BERT's strong performance, I faced challenges related to computational efficiency and inference speed. Given these constraints, I continued refining the LSTM model to make it a viable alternative. This involved fine-tuning hyperparameters, optimizing embeddings, and structuring the input sequences more effectively.

Through multiple iterations and extensive testing, I systematically moved from traditional machine learning models to deep learning architectures, ultimately balancing performance and efficiency by choosing LSTM as a practical alternative to BERT.

Although LSTM required a lot of hyper-parameters tuning and further exploration, I implemented the BERT model for my assignment considering the higher accuracies.

Overview of Code Functionality

The final version of the code implements a BERT-based authorship identification model. The process follows several key steps:

1. **Data Preprocessing & Tokenization**: Initially, raw text data was loaded and tokenized dynamically rather than using a pre-tokenized dataset. We utilized transformers from Hugging Face to tokenize text using the BERT tokenizer, ensuring consistency and compatibility with the BERT model.
2. **Word Embeddings**: The BERT model inherently generates contextualized embeddings, capturing word meanings based on surrounding context. This significantly improved model performance compared to static embeddings like GloVe.
3. **Data Transformation**: The dataset was structured into tokenized sequences of fixed length, with padding applied where necessary. This ensured uniformity in input dimensions for training.
4. **Model Architecture**: The final model was based on the pretrained BERT transformer with a classification head added for authorship prediction. Fine-tuning was performed on the authorship dataset to optimize for accuracy.

5. **Training & Evaluation**: The model was trained using cross-entropy loss and optimized with AdamW. Leveraging Google Colab's T4 GPU, we fine-tuned BERT for multiple epochs, achieving an accuracy of **96.47% on five authors (multiple full book text dataset)**.
6. **Comparison with Traditional Models and LSTM**:
    a. On a dataset containing all authors with text from just one book each:
        i. Random Forest Accuracy: **0.0062**
        ii. SVM Accuracy: **0.0093**
        iii. Logistic Regression Accuracy: **0.0010**
        iv. LSTM Accuracy: **76.71%**
    b. On a dataset containing only five authors and text from just one book per author:
        i. BERT Model Accuracy: **91.82%**
    c. On a dataset containing five authors with all their books combined:
        i. BERT Model Accuracy: **96.47%**
        ii. LSTM Accuracy: **87.56%**
7. **Inference on Real Data**: To assess real-world applicability, actual book passages from different authors were used for testing.

Challenges & Future Work

Throughout this implementation, several challenges were encountered and addressed:

1. **Pre-tokenization Issues**: The initial approach of pre-tokenized text led to inconsistencies. Dynamically tokenizing the text using BERT's tokenizer solved this problem, improving the model's flexibility.
2. **Computational Constraints**: The BERT-based model, while powerful, required significant computational resources. Optimization strategies such as mixed-precision training and model distillation could help improve inference speed.
3. **Embedding Integration**: Removing torchtext and switching to gensim for loading GloVe embeddings improved compatibility but introduced complexity in handling unknown words and vocabulary size management.
4. **Handling Small Data Per Author**: Since some authors had very few books available, splitting book texts into smaller chunks helped increase the dataset size and improved training performance.
5. **Model Performance on Real Text**: While the model performed well on the test dataset, its accuracy declined when tested on longer, more complex passages. This suggests the need for further improvements in handling diverse linguistic styles.

6. **Hyperparameter Optimization**: The choice of hyperparameters, such as sequence length, batch size, and learning rate, had a significant impact on model performance. Further tuning could enhance results.

For future work, we propose:

- Exploring transformer-based models with optimized architectures to balance accuracy and efficiency like DistilBERT.
- Enhancing dataset diversity to improve generalization across different writing styles.
- Experimenting with attention mechanisms in LSTM to capture long-term dependencies better.
- Evaluating the model on multilingual text data to test its robustness.
- Implementing additional preprocessing steps, such as stopword removal and stemming, to refine input text representation.
- Investigating alternative embedding techniques, such as contextualized word embeddings from models like FastText, to improve performance.

References -

https://www.kaggle.com/datasets/mateibejan/15000-gutenberg-books?resource=download&select=guteberg_download.py

Sangram’s contribution

Overview

As we know and as discussed earlier in this report, this project implements an authorship identification program that predicts the author of an unknown text by comparing its writing style characteristics with those of known authors. The program utilizes a signature-based approach, where a signature is a set of features extracted from a text that represents the author's writing style. I have added a functionality where the program also incorporates natural language processing (NLP) techniques and interacts with a language model (Gemini) for additional analysis and information retrieval.

Added Features

Named Entity Recognition (NER) -

Named Entity Recognition (NER) is a Natural Language Processing (NLP) technique that automatically identifies and classifies named entities in text into predefined categories such as people, organizations, locations, dates, etc. In this program, NER is implemented using the spaCy library, which provides pre-trained language models and tools for various NLP tasks. OK, let's dive deeper into the Named Entity Extraction (NER) component of the project.

The NER Process -

1. Text Loading and Preparation: The extract_named_entities function begins by loading the content of the specified text file. The text might undergo some preprocessing steps (like cleaning and normalization) depending on the spaCy model used.
2. spaCy's NER Model: The core of the NER process lies in spaCy's pre-trained NER model (en_core_web_sm in this case). This model has been trained on a massive dataset to identify and categorize named entities based on their context within the text.
3. Entity Recognition: The loaded text is processed by the spaCy pipeline, which includes the NER component. The NER model analyzes the text, recognizing and labeling various named entities such as:
   - PERSON: Names of people (e.g., "Sherlock Holmes", "John Watson")
   - GPE: Geopolitical entities (countries, cities, states)
   - ORG: Organizations (companies, institutions, agencies)
   - LOC: Locations (mountains, bodies of water, landmarks)
   - DATE: Dates and times

- MONEY: Monetary values

and several other entity types.

4. Entity Categorization: Each recognized entity is assigned a label indicating its type (e.g., "PERSON", "ORG", "LOC"). This allows for structured organization and analysis of the extracted entities.
5. Filtering for Unique Persons: The code specifically focuses on extracting unique PERSON entities. This is because the primary goal is to identify characters in the text to aid in guessing the author and book.

Why NER is Important Here -

1. Character Identification: Identifying the key characters in a book is crucial for understanding its plot, themes, and authorship. NER helps automate this process, extracting the names of people who are likely to be significant characters.
2. AI Interaction: The extracted character names are used as input to the Gemini AI model. This enables the AI to leverage its knowledge base and make informed guesses about the author and book based on the presence of specific characters. This also provides a smaller data set to AI model for predictions. The NER reduces the whole text of the book to a list of characters which can be fed to AI model for prediction, resulting in significant reduction in processing times.

AI Interaction -

This project leverages the power of the Gemini AI model to enhance the basic signature analysis with external knowledge and reasoning capabilities.

The AI Interaction Process -

1. Named Entity Extraction: The extract_named_entities function uses spaCy's NER capabilities to identify and extract named entities from the input text. The focus is on extracting unique PERSON entities, which are assumed to represent characters in the text.
2. Prompt Engineering for Gemini: The guess_author_and_book function takes the set of unique character names and constructs a prompt for the Gemini AI model. The prompt explicitly asks Gemini to "guess" the author and book based on the provided character information. This prompt engineering step is crucial for guiding Gemini's response and ensuring it aligns with the project's goal.
3. Gemini's Response: The constructed prompt is sent to Gemini using model.generate_content(). Gemini processes the prompt, leveraging its knowledge base and language processing capabilities to analyze the character names and generate a

response. The response typically includes a guessed author and book title based on the provided information.

4. Further Information Retrieval: The get_author_description function takes Gemini's guessed response (author and book) and formulates another prompt. This prompt asks Gemini to provide more information about the guessed author and book, enriching the analysis with additional context.
5. Output and Presentation: The responses from Gemini are printed to the console, providing the user with insights into the AI's predictions and the reasoning behind them.

Why AI Integration is important here -

1. Limited ML Expertise - Not having any experience with the ML technologies, I was unable to build a complete text recognition model on my own within the timelines of this project.
2. Leveraging Readily Available AI: To overcome this limitation, the project strategically utilizes a readily available AI model like Gemini. This approach allows the project to benefit from advanced AI capabilities without the need to build and train a model from scratch.
3. Gemini as a Prediction Engine: Essentially, Gemini acts as a powerful prediction engine, leveraging its vast knowledge base and language processing abilities to make informed guesses about the author and book based on the provided character information.

Challenges and Learnings

1. Python Programming and GitHub Copilot: While I had some initial challenges with Python syntax and best practices, utilizing GitHub Copilot significantly aided my development process. Copilot helped identify and correct syntax errors, suggested code improvements, and even assisted in writing clear and concise comments.
2. Generative AI Integration: Integrating generative AI, specifically Gemini, with Python was a new endeavor. Researching third-party resources, such as Stack Overflow articles and documentation, proved invaluable in understanding the integration process and successfully connecting Gemini to the code.
3. Natural Language Processing with spaCy: This project provided my first hands-on experience with Natural Language Processing (NLP) using spaCy. I was particularly interested in learning how spaCy's NER capabilities could be applied to extract meaningful information from text, which broadened my understanding of NLP applications.

4. Exploring Parallel Processing: Although I attempted to implement parallel processing for NLP tasks using nlp.pipe, I was unable to fully integrate it before the project deadline.

Future Scope

1. Optimizing NLP with Parallel Processing: A key area for future improvement is implementing parallel processing for NLP tasks. This could significantly enhance the efficiency and speed of the analysis, especially for larger texts.
2. Expanding AI Model Integration: To further improve the accuracy and robustness of the system, incorporating additional AI models could be explored. This could involve experimenting with different language models or ensemble methods to combine the strengths of multiple AI approaches.
3. Expanding this program with UI: Adding an attractive User Interface with the help of some web-development can be another improvement one can explore for this project. A UI will complete this program in a true sense where a user can directly upload the unknown file on website and get the results presented in more readable format.

Vishal's contribution

Overview of the Problem

Authorship identification is a challenging task in computational linguistics, where the objective is to determine the author of a given text. The problem arises from the need to analyze and compare linguistic patterns across different texts. Authorship attribution has applications in areas like literary analysis, plagiarism detection, and forensic investigations. The primary challenge lies in accurately capturing and comparing stylistic features that distinguish one author's writing from another.

Main Solution

The provided Python code addresses this problem by implementing a system that analyzes text signatures from known authors and compares them against an unknown text. The core idea is to use linguistic features to identify the most likely author of a mystery text. The program calculates five distinct linguistic features for each text: average word length, ratio of different words to total words, ratio of words used exactly once to total words, average sentence length, and average sentence complexity. These features are then compared using a weighted scoring mechanism to determine the closest match. Principal Component Analysis (PCA) is also integrated to reduce the dimensionality of signatures, enhancing computational efficiency.

Future Takeaways and Recommendations

The authorship identification code demonstrates a solid implementation of text signature analysis. However, there are areas for improvement. Performance could be enhanced by introducing multiprocessing to handle larger datasets efficiently. The weighting factors, currently hardcoded, should be made adjustable to improve adaptability across different datasets. Additionally, better error handling during file I/O operations can increase the robustness of the system. By addressing these aspects, the tool can become more reliable and versatile for various authorship attribution tasks.