So I Write Like Shakespeare? Explainable Writing Style Detection with User Interface

Andrew Hoang `ahhoang@ucdavis.edu`
Qiwen Xiao `qwxiao@ucdavis.edu`
Ching Hao Chang `cccchang@ucdavis.edu`
Weifeng Liu `wfliu@ucdavis.edu`
Abdullah Al Rawi `aialrawi@ucdavis.edu`
Department of Computer Science
University of California, Davis
Davis, USA

Abstract

In this project, we explore writing style detection and authorship attribution using machine learning techniques. Leveraging `fastText`, we developed a model capable of classifying text given by users based on the authors whose previous works share the most writing style with it. To enhance user comprehension, the model also conducts analysis using SHAP and LIME algorithms, which aids in understanding how a given text aligns with the styles of other writers. In addition, to facilitate user interaction and visualization, we implemented a `Tornado` server, enabling user interactions and real-time analysis. This project is not only a useful tool to writers, but also contributes to the growing field of computational linguistics and authorship analysis, providing a basic framework for future research, development, and enhancement.

1. Introduction

The rise of digital communication and the explosion of online content have led to an increased interest in authorship attribution—determining the author of a given text by analyzing its linguistic and stylistic features. Despite the accuracy of modern machine learning models, a significant challenge remains: these models often function as "black boxes," providing little insight into why certain predictions are made. This lack of transparency becomes problematic, especially in applications where understanding the rationale behind a prediction is crucial, such as in forensic linguistics, academic plagiarism detection, or even creative writing feedback.

In our project, we aim to address this gap by developing a machine learning-based approach that not only classifies text according to its author but also provides a layer of explainability through SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations). By separating the given text into sentences and using `fastText` [2] to train a model that inherently captures and leverages the contextual relationships within the text, we can classify sentences with a high degree of accuracy. Additionally, the similarity scoring mechanism along with the analysis from SHAP and LIME allows users to see how their text compares to the writing styles of other authors, offering a more transparent and understandable classification process.

To further enhance user interaction, we implemented a `Tornado` [12] server that visualizes these results in real time. This interactive component ensures that users can actively engage with the analysis, exploring how specific features of their text contribute to the overall classification and similarity scores.

In summary, this project makes the following contributions:

- **An explainable writing style detection system** based on SHAP and LIME that goes beyond mere accuracy in authorship attribution by incorporating explainability. Unlike many traditional models, our approach provides users with insights into how their text is classified and what stylistic features contribute to these classifications.

- **A similarity scoring mechanism** that allows users to see how their text compares to the writing styles of other authors. This feature enhances interpretability, making it easier for users to understand the stylistic similarities and differences between their work and that of other writers.

- **An efficient sentence classification approach with `fastText`** that handles sentence-level classification, capturing the nuanced stylistic differences between authors. This contributes to the growing body of work in computational linguistics by demonstrating the effectiveness of fastText in the context of authorship attribution.
- **A user interface for interactive visualization and user engagement** that provides real-time visualization and interaction. This user-friendly interface allows users to input their text, view the classification results, and explore similarity scores in an intuitive manner, thereby enhancing the usability and accessibility of our system.

Our replication package is available at [3].

2. Background and Related Work

In this section, we review the relevant literature and technologies that form the foundation of our project. We focus on three key areas: Authorship Attribution, `fastText`, and `Tornado`. Each of these areas contributes to the overall framework we developed for writing style detection and authorship attribution.

2.1. Authorship Attribution

Authorship attribution (AA) is a field of study concerned with determining the authorship of a given text based on its linguistic and stylistic features [13]. This area of research has gained prominence with the rise of digital texts and the need for verifying authorship in various domains such as literature, historical documents, legal disputes and social media [7, 11]. With the advent of machine learning, modern approaches leverage computational models to analyze textual features more efficiently and accurately [7, 9, 11].

Recent approaches involve models that focus on two different methods. Embedding based models analyze word embeddings to assign contextual meaning to a text. These models typically take form as deep neural networks or transformers that, based on training, correlates relationship between words. In recent literature, the PART [9] model aims to address text length bottlenecks by using a multi-stage encoder to extract all contextual and semantic embedding from a text. Another embedding based model, DeepStyle [7] uses deep learning to better classify smaller texts. They do this by extracting different author embeddings and fusing them together.

Feature based models focus on lexical and syntatic behaviours such as punctuation, sentence length, and word frequencies. These models show up in the form of SVM's and random forrest models which extract the aforementioned features and use statistical classifiers to predict authorship. STEL and LUAR [11] are feature based methodologies that feature models use to capture variations of punctuation, casing, contraction spelling, as well as, stylistic and content information to distinguish between authors.

Most of the current literature revolving around author attribution involves proposing new models to overcome bottlenecks or using AA to confront problems such as determining if a text has been written by a large language model [8]. While these approaches may achieve high accuracies classifying texts within their own niche, there is seldom effort into incorporating user interpretability. Many of these models act as black boxes in which the user inputs a text and receives a predicted value with no explanation as to why a text is attributed to a certain author. In our project, we aim to address this gap by exploring and comparing two different AI explainability frameworks **SHAP** and **LIME**.

2.2. fastText

`fastText` is an open-source, library developed by Facebook AI Research (FAIR) that is designed for efficient text classification and representation learning. Unlike traditional word embedding models, `fastText` represents each word as a bag of character n-grams, enabling it to capture subword information and handle out-of-vocabulary words more effectively. This feature is particularly advantageous in authorship attribution, where subtle stylistic differences can play a significant role in distinguishing between authors.

`fastText` has been widely adopted in various natural language processing (NLP) tasks due to its speed and simplicity. It has been used for tasks ranging from sentiment analysis to language identification, demonstrating its versatility. In our project, we leveraged `fastText`'s capabilities to perform sentence-level classification, allowing us to capture the nuanced stylistic differences between authors. The efficiency of `fastText` also made it possible to handle large datasets and perform real-time analysis, which is crucial for our interactive application.

2.3. Tornado

`Tornado` is a scalable, non-blocking web server and web application framework written in Python. It is well-suited for building real-time web applications that require long-lived network connections, such as WebSockets. `Tornado`'s asynchronous networking capabilities make it ideal for handling multiple connections simultaneously, which is essential for applications that involve real-time data processing and visualization.

In the context of our project, `Tornado` was used to create an interactive web-based interface that allows users to input their text and receive real-time feedback on authorship attribution and similarity scoring. The `Tornado` server handles user requests, processes the input text using our `fastText`-based model, and displays the results through a user-friendly interface. The choice of `Tornado` was driven by its ability to provide low-latency responses, ensuring that users can interact with the system smoothly and efficiently.

3. Data Analysis

The dataset comprises 3,036 English-language books authored by 142 different individuals. This collection, sourced from [5], has undergone meticulous cleaning to eliminate extraneous metadata, licensing information, and annotative notes. It serves as a foundation for developing a model capable of identifying and elucidating writing styles. Due to the dataset's extensive size, we selected works from ten authors for model training. The chosen authors—Charles Dickens, Agatha Christie, Jane Austen, Mark Twain, O Henry, Oscar Wilde, P. G. Wodehouse, Walt Whitman, Winston Churchill, and Zane Grey—were selected to represent a diverse range of literary styles, thereby enabling the model to discern varied writing styles.

3.1. Length and Complexity

Fig. 1 displays the average text length for each author within the dataset. Notably, while the average text length is relatively consistent across most authors, Walt Whitman's text length is significantly different. This discrepancy arises because Whitman is the sole poet among the authors, necessitating a greater volume of text to capture his distinctive style. Consequently, including Whitman aids in achieving a balanced representation of diverse writing styles.

Author	Average Sentence Length
Agatha Christie	9.78
Charles Dickens	19.49
Jane Austen	20.41
Mark Twain	20.22
O Henry	15.20
Oscar Wilde	16.36
P G Wodehouse	11.74
Walt Whitman	**28.48**
Winston Churchill	14.75
Zane Grey	12.44

Table 1. The average sentence length of each author.

Tab. 1 presents the average sentence length, which varies among authors and reflects the idiosyncrasies of their writing styles.

Sentence complexity is quantified by counting the number of commas, conjunctions, and semicolons within each sentence, as shown in Tab. 2. This metric highlights variations in writing complexity among authors. For instance, Walt Whitman's poetry, characterized by extensive use of sentences and semicolons, results in a higher complexity value compared to other authors. Conversely, Agatha Christie's preference for clear and concise prose results in fewer complex sentence structures and consequently lower complexity values. These insights into sentence complexity provide valuable indicators for distinguishing between authors' writing styles, thereby informing the model's development.

Author	Average Sentence Complexity
Agatha Christie	1.32
Charles Dickens	3.55
Jane Austen	3.46
Mark Twain	3.50
O Henry	2.36
Oscar Wilde	2.73
P G Wodehouse	1.60
Walt Whitman	**6.39**
Winston Churchill	2.34
Zane Grey	1.86

Table 2. The average sentence complexity of each author.

3.2. Sentiment and Diversity

Tab. 3 summarizes various features that differentiate the writing styles of the selected authors. These attributes—sentence structure, lexical diversity, and sentiment polarity—offer a holistic view of how each author's work aligns with their genre, tone, and narrative voice.

Fig. 2 illustrates the average sentiment polarity for the ten selected authors, providing a clear comparison of their emotional expressiveness. Walt Whitman and Jane Austen exhibit the highest average sentiment polarity, indicating the optimistic and positive undertones prevalent in their works. Whitman's free verse poetry often conveys transcendence, unity, and democracy, while Austen's novels typically revolve around romantic resolutions, hence the higher positive sentiment.

In contrast, Zane Grey and Winston Churchill present the lowest sentiment polarities, reflecting the often serious, rugged, or politically complex nature of their writing. Zane Grey's focus on conflict and survival within the Western genre lends itself to more neutral or negative sentiment, while Churchill's speeches and historical writings tend to evoke serious, sober reflections on war and governance.

Moreover, authors such as Mark Twain and O. Henry, who are known for their wit and satire, maintain relatively positive sentiment values. Their humorous and ironic styles contribute to an engaging narrative tone, often resulting in lighter, more entertaining reads despite the social critiques embedded in their work.

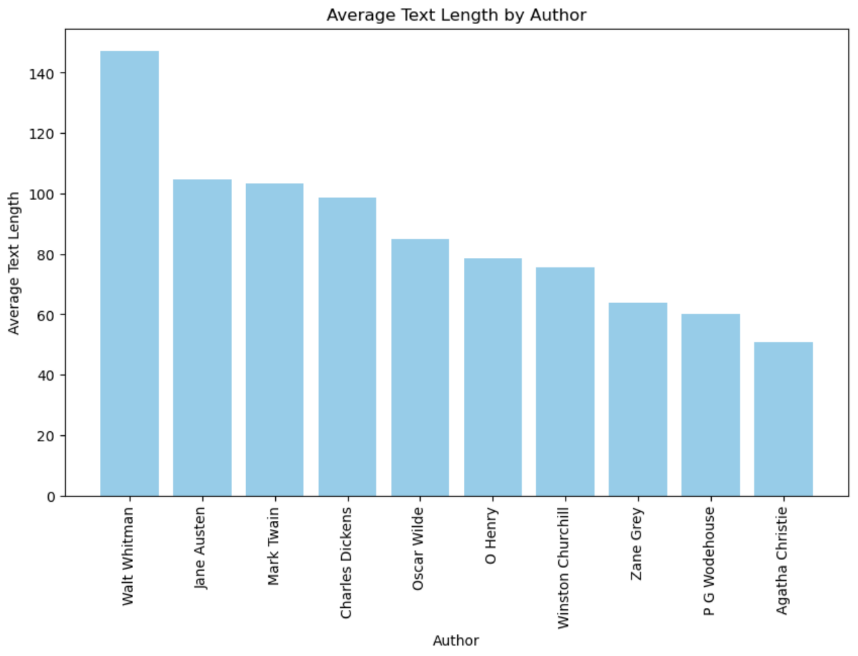


Figure 1. The average text length by author.

Author	Sentence Length and Structure	Lexical Diversity	Sentiment Polarity
Agatha Christie	Compact, suitable for clarity and conciseness in mystery novels.	High	Positive, engaging and suspenseful narratives with positive resolution.
Charles Dickens	Varied, reflecting detailed descriptive style and complex social settings.	Moderate	Moderate positive, critical yet hopeful depiction of Victorian England.
Jane Austen	Shorter, direct and conversational style.	High	Highest, romantic and often satirical with positive endings.
Mark Twain	Diverse, aligns with versatile, humorous, and critical narratives.	Lower	Positive, humorous and satirical style.
O Henry	Compact, typical for wit and wordplay in short stories.	Very high	Moderately positive, ironic and humorous nature.
Oscar Wilde	Varies, suitable for plays and philosophical essays with a range of expressions.	High	Moderately positive, ironic and humorous nature.
P G Wodehouse	Moderate, balanced for comedic and light-hearted narratives.	High	Moderate positive, upbeat and whimsical style.
Walt Whitman	Extended, reflecting broad themes in free verse poetry.	Lower	Highest, themes of transcendence, democracy, and love.
Winston Churchill	Diverse, from impassioned speeches to historical narratives.	High	Moderate, serious tone on complex political and historical topics.
Zane Grey	Shorter and uniform, fitting for action-driven Western adventure genres.	High	Lowest, reflects conflict and rugged landscapes typical of Western genre.

Table 3. Analysis by author.

4. Approach

In this section, we detail the methodology employed in our project for writing style detection. Our approach is structured into four main components: the use of `fastText` for sentence classification, the implementation of a similarity scoring mechanism, the algorithms to achieve explainability, and the development of an interactive `Tornado` server for real-time analysis and visualization.

4.1. Sentence Classification with fastText

To address the task of authorship attribution, we utilize `fastText`, which operates by representing words as bags of character n-grams, allowing it to capture subword information and handle out-of-vocabulary terms effectively. This capability is particularly useful for distinguishing between the subtle stylistic differences that characterize individual authors.

4.1.1 Text Tokenization

Considering that classifying the text as a whole may compromise explainability, we first divide the entire text into separate sentences. For each sentence, a tokenizer is deployed to transform it into a sequence of tokens, $\mathbf{t} = \{t_1, t_2, \ldots, t_n\}$, where t_i represents an individual token. This sequence is then converted into a bag of character n-grams. For instance, a 3-gram representation of the token “authorship” would include “aut”, “uth”, “tho”, “hor”, “ors”, “rsh”, “shi”, “hip”.

4.1.2 Vector Representation

`fastText` generates vector representations for each token and n-gram, $\mathbf{v}_{t_i}$, which are aggregated to form a sentence vector $\mathbf{v}_s$. The sentence vector $\mathbf{v}_s$ is computed as the average of the vectors of its constituent tokens:

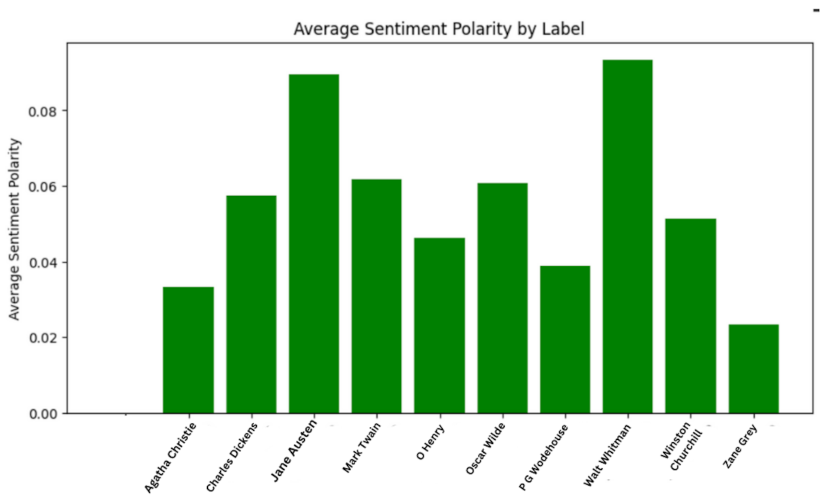


Figure 2. The average sentiment polarity by author.

$$\mathbf{v}_s = \frac{1}{n}\sum_{i=1}^{n}\mathbf{v}_{t_i} \tag{1}$$

where n is the number of tokens in the sentence. This vector is then used as input to the classification model.

4.1.3 Classification Model

The sentence vector $\mathbf{v}_s$ is fed into a softmax classifier to predict the probability distribution over possible authors. If there are k authors, the classifier outputs a vector $\mathbf{p} = \{p_1, p_2, \ldots, p_k\}$, where p_j is the probability of the sentence belonging to author j. The classification is determined by:

$$\hat{y} = \arg\max_j p_j \tag{2}$$

where $\hat{y}$ denotes the predicted author for the sentence.

4.2. Similarity Scoring

To provide deeper insights into the stylistic similarities between texts, we compute similarity scores between the vector representations of the input text and reference texts.

The similarity score between the vector representation of the input sentence $\mathbf{v}_s$ and a reference sentence vector $\mathbf{v}_r$ is calculated using cosine similarity:

$$\text{Cosine Similarity}(\mathbf{v}_s, \mathbf{v}_r) = \frac{\mathbf{v}_s \cdot \mathbf{v}_r}{\|\mathbf{v}_s\|\|\mathbf{v}_r\|} \tag{3}$$

where $\cdot$ denotes the dot product, and $\|\cdot\|$ represents the Euclidean norm of the vectors. The resulting score ranges from -1 (completely dissimilar) to 1 (completely similar), providing a quantitative measure of how closely the input sentence matches the stylistic features of the reference texts.

4.3. Explainability with SHAP and LIME

To further enhance the interpretability of our authorship attribution model, we employed SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations), both of which are powerful tools for explaining individual predictions in machine learning models.

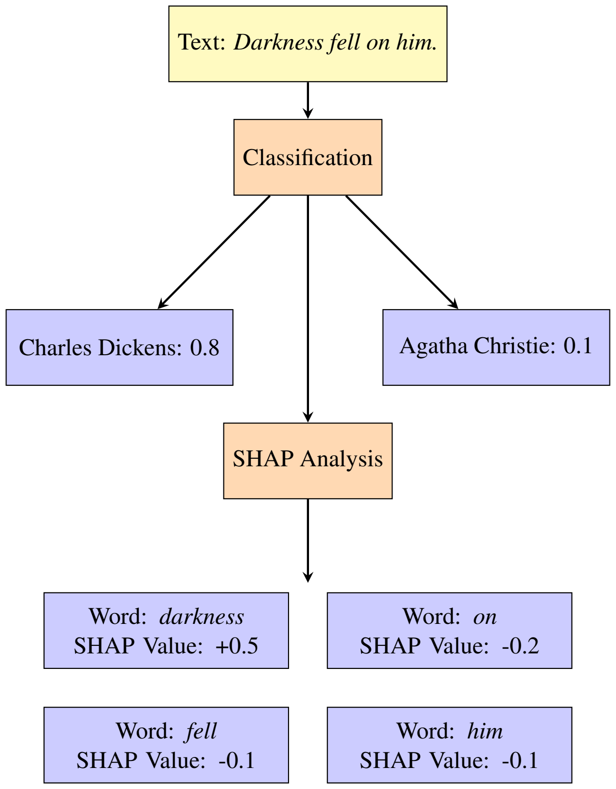
4.3.1 SHAP Explanation

SHAP values are based on cooperative game theory and provide a way to explain the contribution of each feature to a particular prediction. For our text classification task, SHAP values help us understand how different tokens and n-grams contribute to the classification of a sentence.

$$\phi_i = \frac{1}{|S|}\sum_{S\subseteq N\setminus\{i\}}[f(S\cup\{i\}) - f(S)] \tag{4}$$

where ϕ_i is the SHAP value for feature i, $f(S)$ is the model output with feature set S, and N is the set of all features. This equation calculates the average contribution of feature i across all possible subsets S of the remaining features.

Application to Text: For each sentence, SHAP values are computed for the tokens and n-grams, showing their impact on the classification outcome. Higher SHAP values indicate a greater contribution to the predicted author, allowing us to identify which parts of the text most influence the model's decision.



Text: *Darkness fell on him.*
Classification
Charles Dickens: 0.8
Agatha Christie: 0.1
SHAP Analysis
Word: *darkness* SHAP Value: +0.5
Word: *on* SHAP Value: -0.2
Word: *fell* SHAP Value: -0.1
Word: *him* SHAP Value: -0.1

Figure 3. This diagram shows the SHAP analysis for the text "*Darkness fell on him*". Each word's SHAP value indicates its impact on the classification result. Positive SHAP values increase the likelihood of the text being attributed to Charles Dickens, while negative values reduce it.

4.3.2 LIME Explanation

LIME provides a local, interpretable explanation by approximating the decision boundary of the model with a simpler, interpretable model in the vicinity of the instance being explained. For our task, LIME generates explanations for individual sentences by approximating the model's behavior using a linear model on perturbed data.

LIME Algorithm: Given an instance x to be explained, LIME perturbs x to create a dataset of $(x', \tilde{y})$ pairs where x' are perturbed instances and $\tilde{y}$ are the corresponding predictions. A linear model is then trained on this dataset to approximate the model's decision boundary locally.

$$\hat{f}(x) = \arg\min_{g} \left[\text{Loss}(g(x), f(x)) + \text{Penalty}(g)\right] \tag{5}$$

where $\hat{f}(x)$ is the local approximation, $f(x)$ is the model's prediction, g is the interpretable model, and the Loss term measures how well g approximates f while the Penalty term controls the complexity of g.

Application to Text: LIME produces a linear approximation of the classification model around each sentence, allowing us to interpret the influence of specific tokens and n-grams on the classification outcome. This provides insights into which features are most relevant for the prediction in the local context of each sentence.

4.4. Real-Time Interaction with Tornado

The `Tornado` server facilitates real-time processing and visualization of classification results, similarity scores, SHAP values, and LIME explanations.

4.4.1 Server Workflow

Upon receiving a user's text input, the Tornado server performs the following steps.

Tokenization: Breaks the text into sentences and tokens.

Vector Computation: Converts each sentence into its vector representation using `fastText`.

Classification: Applies the trained `fastText` model to classify each sentence and aggregates the results.

Similarity Scoring: Computes similarity scores between the input text and reference texts.

Explanation: Computes and visualizes SHAP or LIME values to explain the classification.

Response Generation: Formats and sends the results back to the user interface for display.

4.4.2 Real-Time Visualization

Brian Goetz [6] recommends using the following formula to determine the optimal number of threads:

$$\text{\#Threads} = \text{\#Cores} \times \left(1 + \frac{\text{Wait time}}{\text{Service time}}\right) \tag{6}$$

In line with this, the server dynamically visualizes classification results, similarity scores, SHAP values, and LIME explanations by offloading CPU-intensive tasks, such as predicting and explaining, to a thread pool. This allows users to interact with the system, exploring how their text aligns with various authors' styles and understanding the impact of individual features on the classification outcome efficiently.

5. Experiment Design

5.1. Research Question

- **RQ1: Effectiveness of our models.** How do our models perform upon evaluation?
- **RQ2: Comparison between SHAP and LIME.** In production, should we use SHAP or LIME for this purpose?

- **RQ3: Effectiveness of our server.** How does our server perform under varying conditions?

5.2. Measurement

P@1 (Precision at 1) measures the fraction of samples for which the top-1 predicted class is the correct class. If the model correctly predicts the true class as the top-1 prediction, it's considered a success.

R@1 (Recall at 1) measures how often the true class appears as the top-1 prediction across all samples. Since this is a multi-class task, R@1 typically equals P@1 because for each sample, there is only one relevant class in this case. In the rest of this work, we will use the metric P@1.

5.3. Implementation

Data collection. As specified in Sec. 3.

Hyperparameters. For our basic model, we use a learning rate of 0.5 and an embedding size of 300 dimensions. We set the number of word n-grams to 2 to capture bi-gram relationships within the text. The model is trained for 20 epochs to ensure sufficient learning without overfitting. Additionally, we employ `fastText`'s autotuning feature to further optimize the model's performance. We use a validation set with the same origin as the training set's to guide the autotuning process, with a target model size of 12M, an autotuning duration of 600 seconds, and F1 score as the optimization metric. Default values are used for other parameters such as minimum word count and loss function.

5.4. Threats to Validity

Threats to internal validity lie in technique implementations and experimental scripts. To mitigate the threat, we manually check our code and build them on state-of-the-art frameworks, *e.g.*, PyTorch [4]. We also directly use the original implementations from prior work [1]. **Threats to external validity** lie in datasets used in our study. To reduce this threat, we perform our experiments on the widely-used datasets such as [5] and [10].

6. Result Analysis

6.1. RQ1: Effectiveness of different fastText models for authorship classification

The performance of two models is evaluated based on varying hyper-parameters. One model is trained using standard hyper-parameters described in Sec. 5.3, while the second is optimized through the `fastText` auto-tune feature. Confusion matrices for both models (Fig. 4 5) are generated and analyzed to assess the models' classification accuracy. In addition, detailed classification reports and `fastText` precision and recall metrics are produced and are presented in Tab. 4.

The auto-tuned model consistently outperforms the standard model in terms of the `fastText` precision and recall report, achieving a P@1 score of 94.7% compared to the standard model's score of 75.4%. This indicates that the auto-tuned model has a 94.7% probability of correctly predicting the top label.

The performance difference between the models can also be attributed to the large number of classes in the dataset. In multi-class classification problems, the likelihood of misclassifying instances increases as the number of classes grows, particularly in terms of precision and recall. This is because the model must discriminate between a larger number of potential labels, making it more challenging to assign the correct one with high confidence.

While the standard model demonstrates higher precision in the classification report, this does not necessarily indicate superior performance. In the context of a large class set, the model may struggle with recall, as it is more prone to falsely predicting a negative class. This tendency to misclassify increases the difficulty of achieving balanced performance across both precision and recall metrics.

For this reason, the F1 score becomes particularly important in evaluating model performance. It offers a balanced view by considering both precision and recall, which is crucial when dealing with many classes. The auto-tuned model, with its higher F1 score, demonstrates better adaptability to the complexities posed by the large number of classes, outperforming the standard model in this regard.

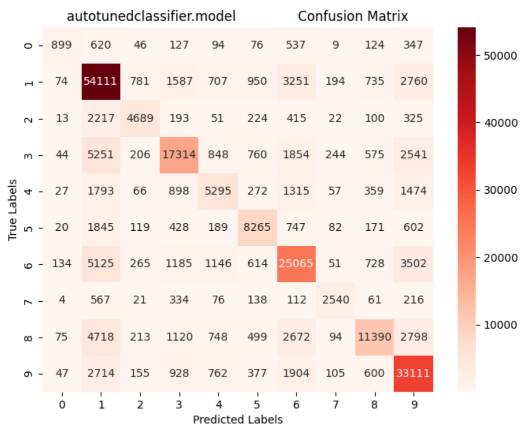


Figure 4. Auto-tuned model confusion matrix.

6.2. RQ2: Comparison between SHAP and LIME

We evaluate the computational efficiency of SHAP and LIME by measuring the time taken to generate explanations across various text instances. The experiment involves using a auto-tuned `fastText`-based text classifier, which achieves a P@1 score of 94.7% on the test set, to predict the

	Auto-tuned			Basic		
Label	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Agatha Christie	0.67	0.31	**0.43**	0.98	0.20	0.34
Charles Dickens	0.69	0.83	0.75	0.80	0.86	**0.83**
Jane Austen	0.71	0.57	**0.63**	0.98	0.32	0.48
Mark Twain	0.72	0.58	0.64	0.76	0.80	**0.78**
O Henry	0.53	0.46	**0.49**	0.95	0.25	0.40
Oscar Wilde	0.68	0.66	0.67	0.67	0.84	**0.74**
P G Wodehouse	0.66	0.66	0.66	0.71	0.83	**0.76**
Walt Whitman	0.75	0.62	**0.68**	1.00	0.26	0.41
Winston Churchill	0.77	0.47	0.58	0.90	0.51	**0.65**
Zane Grey	0.69	0.81	0.75	0.69	0.92	**0.79**
Mean	-	-	**0.628**	-	-	**0.618**

Table 4. Comparison of classification performance between two models.

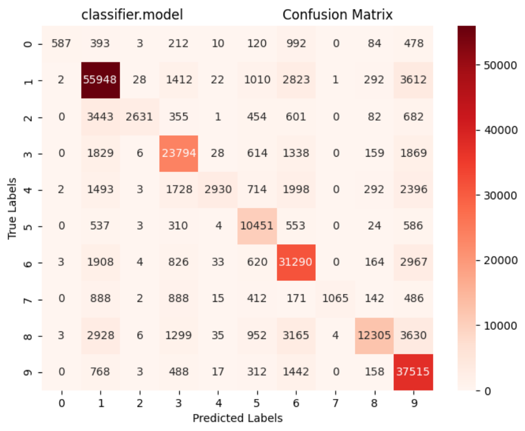


Figure 5. Basic model confusion matrix.

top three authors for each instance, followed by generating explanations using both SHAP and LIME. We record the time taken by each method to produce these explanations, considering different text lengths (measured by the number of tokens). The experiment is conducted over 1000 instances on a 12th Gen Intel(R) Core(TM) i7-12650H CPU, with the time taken for each instance recorded and plotted against the number of tokens in the text.

The results, shown in Fig. 6, indicate a clear difference in the computational efficiency between SHAP and LIME.

SHAP: The time taken by SHAP to generate explanations remains relatively stable across different text lengths, with only a modest increase as the number of tokens increases. This stability is evident from the lower and more consistent scatter of data points along the time axis.

LIME: In contrast, LIME exhibits a more significant increase in time with the growing number of tokens. The scatter plot shows a steeper slope for LIME, particularly for longer texts, indicating that LIME is less efficient as the text length increases.

Based on the result, we speculate that LIME's reliance on generating numerous perturbed samples and fitting a local surrogate model makes it more time-consuming, particularly for longer texts. On the other hand, SHAP's more direct approach to calculating feature attributions, coupled with optimization techniques, results in a faster and more consistent performance.

Overall, SHAP demonstrates superior computational efficiency compared to LIME, particularly for longer text instances. This finding suggests that **SHAP may be more suitable for applications where processing time is a critical factor, especially when dealing with lengthy textual data.**

6.3. RQ3: Effectiveness of our server

Fig. 7 illustrates the performance of our `Tornado` server with the explainer implemented in SHAP based on the result from Sec. 6.2 under varying conditions. The 3D surface plot shows the average time per request as the number of concurrent requests and the length of the text being processed increase.

The results demonstrate that as the number of requests increases, the average time per request also increases, but at a relatively modest rate. This indicates that our server can handle a substantial number of simultaneous requests with minimal performance degradation. Similarly, increasing the length of the input text causes only a slight increase in the average time per request, which suggests that our server is well-optimized for processing larger inputs.

Overall, **the server exhibits robust performance characteristics, effectively managing concurrent workloads while maintaining low latency, even as input size and request frequency increase.** This robustness makes our

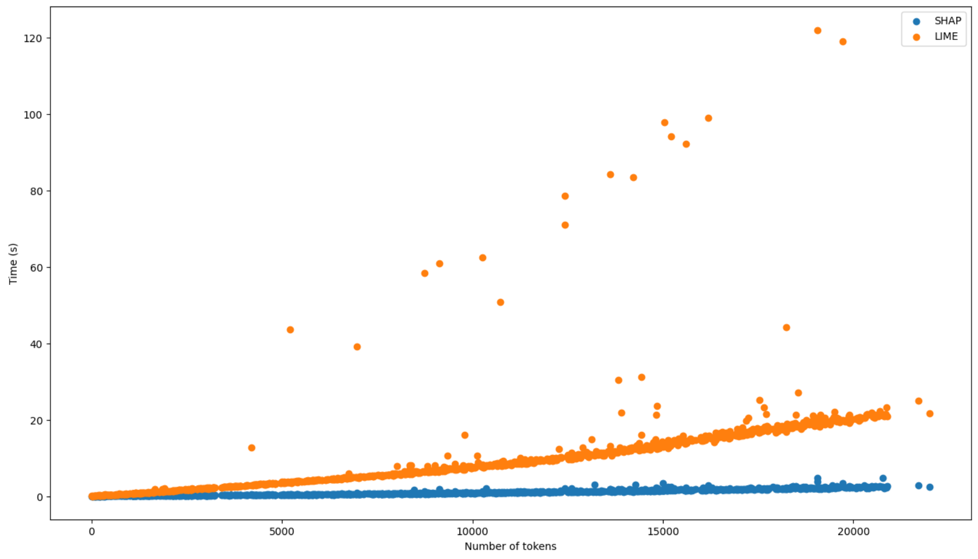


Figure 6. Scatter plot comparing the computational efficiency of SHAP and LIME.

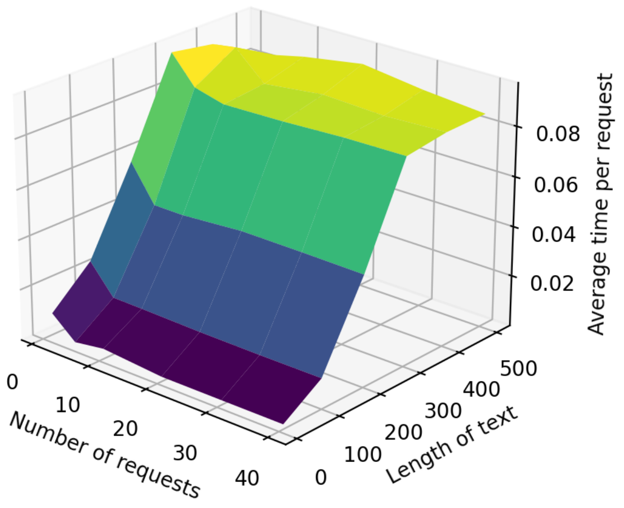


Figure 7. Average time per request as a function of the number of requests and the length of the text. (Vertical axis unit in seconds)

server well-suited for applications requiring real-time text processing and analysis.

7. Conclusion and Discussion

In this work, we present a novel pipeline for writing style detection and authorship attribution, which combines a `fastText`-based text classifier with SHAP and LIME for model explainability and features real-time visualization.

Our evaluation shows that the auto-tuned `fastText` model outperforms the basic model in terms of classification accuracy. The auto-tuned model's adaptability to the multi-class nature of authorship attribution makes it a strong candidate for future applications in this domain.

In our comparison of SHAP and LIME for explaining model predictions, SHAP exhibited better computational efficiency, particularly for longer text instances. SHAP's ability to provide faster explanations with minimal increase in processing time as text length grows suggests that it is better suited for production environments where performance and scalability are critical. On the other hand, LIME may still be useful for smaller datasets or instances where interpretability and local explanations are prioritized over speed. Further work could explore hybrid approaches that leverage both methods based on the context and requirements of the task.

Our server, built on the `Tornado` framework, demonstrated robust performance under increasing workloads. Even with a growing number of concurrent requests and larger input sizes, the server maintained low latency, which is crucial for real-time text analysis and authorship attribution. This scalability positions the system as a viable solution for interactive applications, such as online writing analysis platforms, where fast response times are essential. Future improvements could focus on optimizing memory usage and further reducing latency, particularly as the dataset size and number of concurrent users increase.

In conclusion, this work contributes to the field of computational linguistics and authorship attribution by presenting an efficient, explainable, and scalable solution that can be applied to real-time writing analysis. Further research could explore expanding the model to support multilingual datasets and integrating additional explainability techniques to enhance user interaction.

Acknowledgements

This work is the final project of the course ECS 171 SS2 2024 delivered by instructor Xiwei Xuan at University of California, Davis.

Note: This work was not funded by any external sources.

References

[1] Ankie Fan. Test your text. https://github.com/AnkieFan/testurtext-algo, 2022. 7

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. In *Transactions of the Association for Computational Linguistics*, volume 5, pages 135–146. MIT Press, 2017. 1

[3] Charley-xiao. Ecs 171 project. https://github.com/Charley-xiao/ecs171-project, 2024. 2

[4] Facebook, Inc. PyTorch. https://pytorch.org/, 2016. 7

[5] Martin Gerlach and Francesc Font-Clos. A standardized project gutenberg corpus for statistical analysis of natural language and quantitative linguistics. *CoRR*, abs/1812.08092, 2018. 3, 7

[6] Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea. *Java Concurrency in Practice*. Addison-Wesley, 2006. 6

[7] Zhiqiang Hu, Roy Ka-Wei Lee, Lei Wang, Ee-peng Lim, and Bo Dai. Deepstyle: User style embedding for authorship attribution of short texts. *Arxiv*, Mar 2021. 2

[8] Baixiang Huang, Canyu Chen, and Kai Shu. Authorship attribution in the era of llms: Problems, methodologies, and challenges, 2024. 2

[9] Javier Huertas-Tato, David Camacho, Alejandro Martin, and Álvaro Huertas-García. Part: Pre-trained authorship representation transformer. *Arxiv*, Sep 2022. 2

[10] Shibamouli Lahiri. Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 96–105, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. 7

[11] Gaspard Michel, Elena V. Epure, Romain Hennequin, and Christophe Cerisara. Distinguishing fictional voices: a study of authorship verification models for quotation attribution, 2024. 2

[12] Tornado Project. Tornado web server. https://www.tornadoweb.org/en/stable/, 2024. 1

[13] Jacob Tyo, Bhuwan Dhingra, and Zachary C. Lipton. On the state of the art in authorship attribution and authorship verification, 2022. 2

Appendix: Task Distribution

Task	**Responsible Person**
Data Collection and Preprocessing	Abdullah Al Rawi Weifeng Liu
Model Building and Training	Qiwen Xiao Andrew Hoang
Interface Development and Pipelining	Ching Hao Chang

Table 5. Task Distribution Table