



PDAN 2

Assignment 1

Mukeba Mbunda Eliezer

ST10486340

Varsity College Cape Town Newlands

LECTURE NAME:

Dr Rudolf

Due:

October 03, 2025

Table of content

| | |
|--|----|
| 1. Introduction | 3 |
| 2. Justification of the dataset..... | 4 |
| 3. Exploratory Data Analysis (EDA) | 5 |
| 4. Feature engineering | 11 |
| 5. Train the model..... | 12 |
| 6. Evaluate the model | 14 |
| 7. Retrained the model | 16 |
| 8. Evaluate the retrained model..... | 18 |
| 9. Conclusion | 20 |
| References | 21 |

1. Introduction

This report provides a walkthrough and presents findings from an author identification analysis. The primary aim of this study is to develop a model capable of distinguishing or predicting an author based on their unique writing style. The goal is to create a system that can analyze the way an individual writes and infer the likely author. Such a system has significant practical applications: the ability to accurately identify an author based on stylistic markers is crucial in fields such as plagiarism detection, forensic linguistics, and the analysis of historical manuscripts (Abbasi & Chen, 2009).

Identifying an author is a challenging task. Different authors exhibit variations in vocabulary, sentence structure, and stylistic choices, and even a single author may demonstrate stylistic changes across different works. Consequently, a robust model is required, one that can capture subtle textual patterns, learn the flow of sentences, recognize preferred phrases, and understand overall writing style to accurately differentiate authors (Savoy, 2020).

To address this complexity, we employ Long Short-Term Memory (LSTM) networks. LSTMs are a type of deep learning architecture that excels at capturing long-term dependencies in sequential data, making them particularly suitable for text-based sequence prediction tasks (Geeksforgeeks, 2025). In this context, LSTMs can effectively “remember” word sequences, understand long-term relationships between sentences, and capture the structure of ideas, which is essential for modeling writing style.

The analysis begins with loading the dataset and performing an initial exploration to understand its characteristics. This is followed by a data cleaning process, which prepares the text for analysis. Subsequently, we visualize the data using bar charts, pie charts, histograms, box plots, and heatmaps to uncover patterns and distributions. After understanding the dataset, we proceed to feature preparation, which includes text preprocessing, tokenization, padding, and label encoding. These steps are essential for transforming the raw text into a format suitable for model training.

The first LSTM model is trained using basic hyperparameters to assess initial performance and identify areas for improvement. Based on this evaluation, the model is retrained with tuned hyperparameters to enhance accuracy and overall performance. Finally, the report concludes with a discussion of findings and insights derived from the author identification analysis.

2. Justification of the dataset

The dataset used for this analysis was obtained from Kaggle, a platform that provides secondary datasets for academic purposes. Originally, Kaggle provided two versions of the dataset: a labeled corpus and an unlabeled corpus. For this analysis, we downloaded the labeled dataset, which was used in the Spooky Author Identification competition.

Original dataset link: [Kaggle - Identify the Author](#)

Competition dataset link: [Spooky Author Identification](#)

The dataset contains texts from works of fiction by Edgar Allan Poe (EAP), H.P. Lovecraft (HPL), and Mary Shelley (MWS). The texts were collected from open-source repositories such as Project Gutenberg and the Internet Archive, and then segmented into sentences using the NLTK sentence tokenizer.

While the original dataset included over 50,000 individual sentences, the version used in this analysis (and in the competition) contains 19,579 records. This dataset provides a suitable foundation for building models to identify authors based on their unique writing styles.

This dataset was particularly suitable for the author identification task with LSTM models for several reasons:

1. Clear data structure

The dataset contains two main columns:

- id: Represents the unique identifier for each record
- Text: representing the author's writing,
- Author: identifying the corresponding author.

This structure ensures that the dataset can be directly used for classification tasks without additional restructuring. The dataset provided labeled sentences paired with their corresponding authors. This structure allowed the model to learn stylistic patterns by mapping text to its author (Savoy, 2020).

2. High volume of data

With 19,579 sentences, the dataset offers sufficient examples for an LSTM to capture long-term dependencies in writing style. Large datasets are critical for deep learning models, as they provide the diversity and volume needed to avoid overfitting and improve generalization (Google Cloud, [s.a].).

3. Sequential nature of the data

LSTMs are designed to process sequential data such as text, where words follow an order and dependencies span across different time steps. Since the dataset consists of

continuous sentences, it matches the requirements of sequence models. As AWS (s.a.) notes, recurrent neural networks like LSTM are most effective when trained on sequential data with complex semantic and syntactic structures.

4. Cleaned and structured text

The data has been segmented into individual sentences using the NLTK sentence tokenizer. This preprocessing step improves model performance by providing clean, structured input sequences that LSTMs can effectively process, reducing noise and inconsistencies in the training phase (Lehau 2025) .

5. Relevance for stylistic learning

The dataset is drawn from authors with distinctive writing styles. This diversity enhances the ability of the model to learn stylistic differences, which is crucial for the authorship attribution task (Savoy, 2020).

3. Exploratory Data Analysis (EDA)

In this section, we describe the key steps undertaken to prepare the data for analysis. First, we imported the necessary libraries, created a Spark session, and loaded the dataset. To gain an initial understanding, we displayed the first five rows of the dataset. Next, we proceeded with data cleaning by removing duplicate rows, handling null values, and eliminating extra whitespace and non-printable characters.

After cleaning, we filtered the dataset to retain only the three main authors, ensuring the data is relevant for the analysis. We then checked the filtered dataset for any remaining missing values or duplicates to ensure data quality. Finally, we explored the distribution of classes using various visualization techniques, including bar plots, pie charts, histograms, box plots, and heatmaps, to better understand the dataset and its characteristics.

Step 1: Import Libraries, Create Spark Session, and Load Data

```

# Importing libraries for data processing, visualization, and modeling

# Spark for distributed data processing
from pyspark.sql import SparkSession # Create Spark session
from pyspark.sql.functions import col, udf, from_json, length, lower, when, count, regexp_replace, explode, split, avg # Useful Spark SQL functions
from collections import Counter # Counter for counting elements
# Pandas and Numpy for data manipulation
import pandas as pd # DataFrames and data analysis
import numpy as np # Numerical operations
# Matplotlib and Seaborn for visualization
import matplotlib.pyplot as plt # Plotting graphs
import seaborn as sns # Statistical data visualization
# Scikit-learn for preprocessing and metrics
from sklearn.preprocessing import LabelEncoder # Encode categorical labels as integers
from sklearn.model_selection import train_test_split # Split data into train and test sets
from sklearn.utils.class_weight import compute_class_weight # Compute class weights for imbalanced data
from sklearn.metrics import confusion_matrix, classification_report # Model evaluation metrics
# NLTK for text preprocessing
import nltk # Natural Language Toolkit
from nltk.corpus import stopwords # List of stopwords
from nltk.stem import WordNetLemmatizer # Lemmatization of words
# TensorFlow Keras for deep learning
from tensorflow.keras.preprocessing.text import Tokenizer # type: ignore # Convert text to sequences
from tensorflow.keras.preprocessing.sequence import pad_sequences # type: ignore # Pad sequences to same length
from tensorflow.keras.models import Sequential # type: ignore # Sequential model architecture
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional # type: ignore # Neural network layers
from tensorflow.keras.optimizers import Adam # type: ignore # Optimizer for training
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau # type: ignore # Training callbacks
# Regular expressions for text cleaning
import re # Regular expressions
# OS for environment configuration
import os # Operating system functions
os.environ["PYSPARK_PYTHON"] = "python" # Set PySpark Python interpreter
os.environ["PYSPARK_DRIVER_PYTHON"] = "python" # Set PySpark driver Python interpreter
✓ 422s

```

Figure 1: Import libraries

```

# Creating a spark session and loading the dataset

# Create Spark session
spark = SparkSession.builder.appName("AuthorIdentification").getOrCreate()

# Load dataset
df = spark.read.csv("train.csv", header=True, inferSchema=True)

# Print dimensions
print((df.count(), len(df.columns)))

# Show first 5 rows
df.show(5, truncate=900)
✓ 0.6s

(19579, 3)
+-----+
|   id|
+-----+
|id26305|This process, however, afforded me no means of ascertaining the dimensions of my dungeon; as I
|id17569|
|id11008|
|id27763|
|id12958|
+-----+
only showing top 5 rows

```

Figure 2: Creating Spark session, loading the data and display the first 5 rows

Step 2: Data cleaning: Removing duplicate rows, handling null values, and eliminating extra whitespace and non-printable characters.

```

# Function to clean the data
def deepclean(df):
    """
    Function to clean the dataframe by:
    - Dropping duplicates
    - Dropping rows with null values
    - Removing extra whitespaces
    - Removing non-printable characters
    """
    # Initial number of rows
    initial_count = df.count()
    # Drop duplicates
    df_clean = df.dropDuplicates()
    # Drop rows with null values
    df_clean = df_clean.dropna(subset=["text", "author"])
    # Remove extra whitespaces
    df_clean = df_clean.withColumn("text", regexp_replace(col("text"), "\s+", " "))
    # Remove non-printable characters
    df_clean = df_clean.withColumn("text", regexp_replace(col("text"), "[^\x20-\x7E]", ""))
    # Final number of rows
    final_count = df_clean.count()

    return df_clean, initial_count, final_count

```

✓ 0.0s

Figure 3: Function to clean the data

```

# Clean the dataframe

df_clean, initial_count, final_count = deepclean(df)
print(f"Initial number of rows: {initial_count}")
print(f"Final number of rows: {final_count}")

# Show first 5 rows of the cleaned dataframe
df_clean.show(5, truncate=900)

```

✓ 54s

```

Initial number of rows: 19579
Final number of rows: 19579
+-----+-----+-----+-----+
|      id|                                     text|          author|
+-----+-----+-----+-----+
[id19806|                                     ""But that Kidd's accumulations were immense| is well known."|
[id21409|    The bandage lay heavily about the mouth but then might it not be the mouth of the breathing Lady of Tremaine?|          EAP|
[id22107|                                     We expelled the bodies through the double hatches and were alone in the U .|          HPL|
[id23537|    Of the various tales that of aged Soames, the family butler, is most ample and coherent.|          HPL|
[id20251| Besides, the estates, which were contiguous, had long exercised a rival influence in the affairs of a busy government.|          EAP|
+-----+-----+-----+-----+
only showing top 5 rows

```

Figure 4: Apply the function and display the first 5 rows of the cleaned data

Step 3: Filter the data for valid authors and check the data

```

# Filter for valid authors
authors = ["HPL", "EAP", "MWS"]
df_filter = df_clean.filter(col("author").isin(authors))

print(f"Final number of rows after cleanning : {final_count}")
print("Final number of rows after flitering : ", df_filter.count())

# group by author and count number of sentences per author
author_counts = df_filter.groupBy("author").count().orderBy(col("count").desc())
author_counts.show()

```

✓ 5.6s

Final number of rows after cleanning : 19579
Final number of rows after flitering : 18047

| author | count |
|--------|-------|
| EAP | 7044 |
| MWS | 5552 |
| HPL | 5451 |

Figure 5: Filter the data for 3 valid authors

```

# Check for missing values

print("Number of missing values per column:")
df_filter.select([count(when(col(c).isNull(), c)).alias(c) for c in df_filter.columns]).show()

# check for duplicates
print("Number of duplicate rows: ", df_filter.count() - df_filter.dropDuplicates().count())

# Print schema
df_filter.printSchema()

```

✓ 7.4s

Number of missing values per column:

| id | text | author |
|----|------|--------|
| 0 | 0 | 0 |

Number of duplicate rows: 0

```

root
 |-- id: string (nullable = true)
 |-- text: string (nullable = true)
 |-- author: string (nullable = true)

```

Figure 6: Checking missing and duplicated values on filtered data

Step 4: Visualization techniques, including bar plots, pie charts, histograms, box plots, and heatmaps

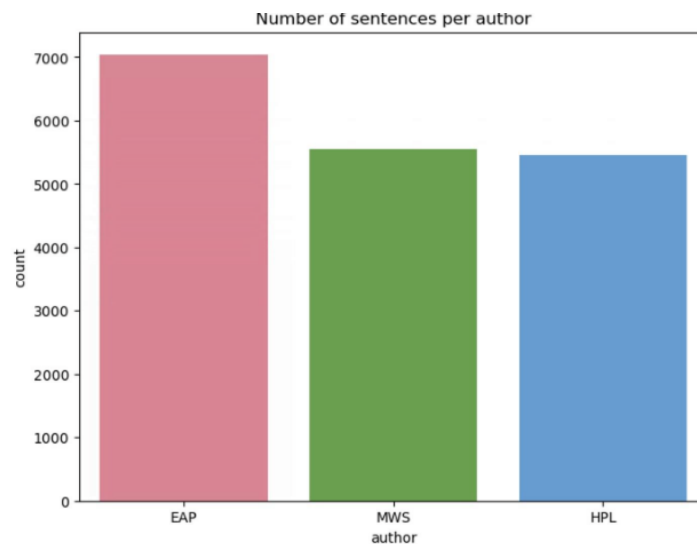


Figure 7: Number of sentences per author

Figures 7 and 8 illustrate the number of sentences in the dataset for each author. It is evident that H.P. Lovecraft (HPL) has the highest number of sentences, followed by Edgar Allan Poe (EAP), and then Mary Shelley (MWS). This suggests that HPL's works are more extensively represented in the dataset compared to the other authors.

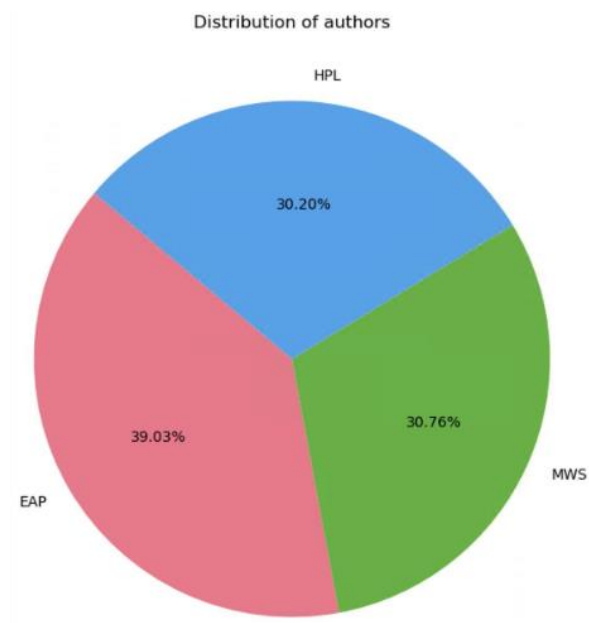


Figure 8: Distribution of authors

The figure above shows the distribution of text lengths in the dataset. It can be observed that texts with an average length of 314 words are the most frequent. This value was determined through calculation (please refer to the accompanying code file for details).

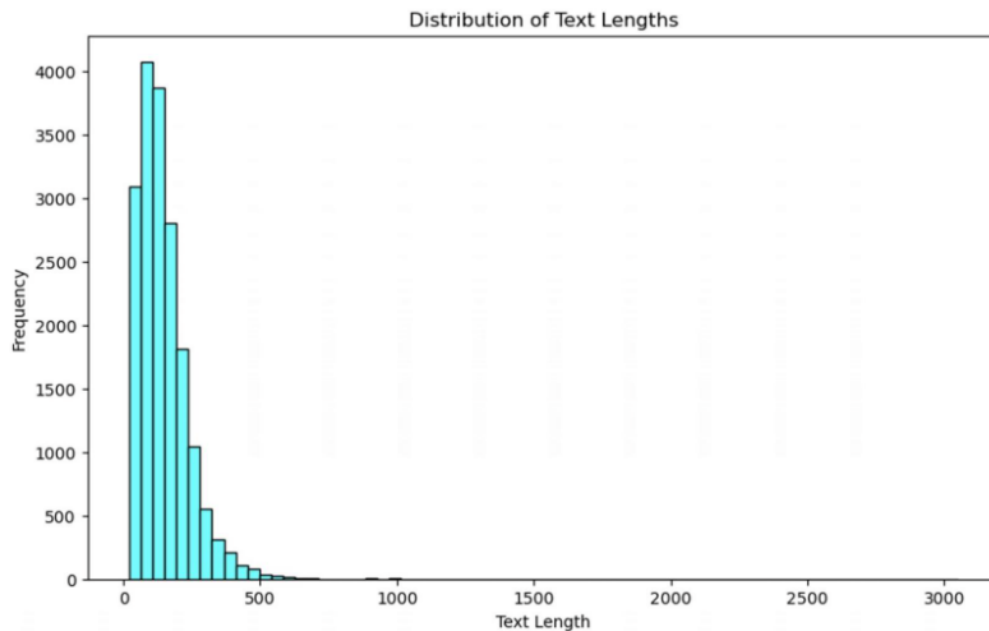


Figure 9: Distribution of text lengths

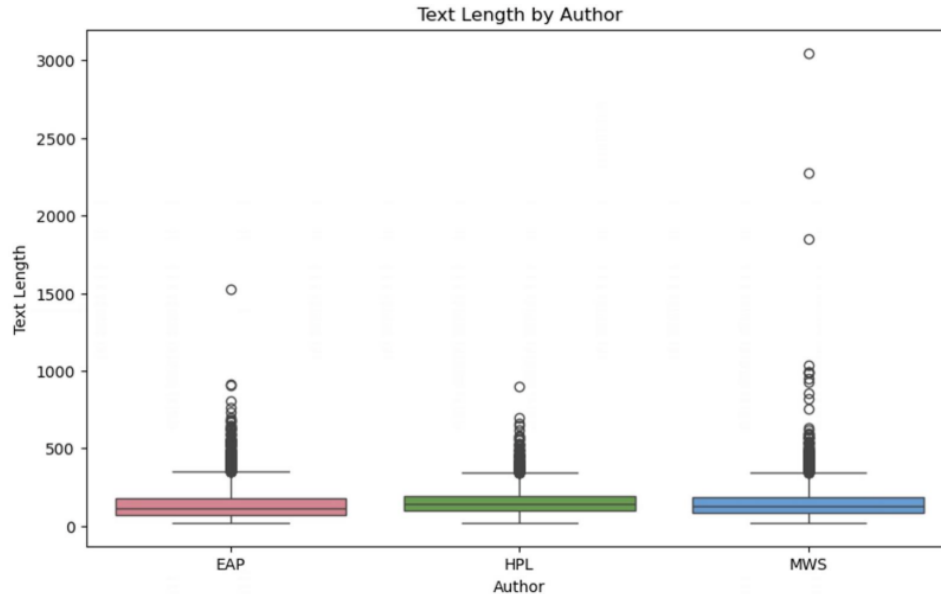


Figure 10: Text length by author

The figure above illustrates the presence of outliers for each author. Texts with lengths close to or exceeding 500 words are considered outliers, which may introduce noise into the dataset.

4. Feature engineering

After visualizing the dataset and gaining initial insights, the next step was text preprocessing to prepare the data for model training. We began by calculating the maximum sequence length (maxlen), which provided the average text length in the dataset.

This step was important for determining the sequence length to be used in the LSTM model. We then implemented a preprocessing function to clean the text. The function converted all text to lowercase, removed special characters, eliminated stopwords, and applied lemmatization to reduce words to their root form. Following this, we tokenized the data to transform text into sequences of integers representing words.

To ensure consistent input length for the model, we applied padding to the sequences, allowing shorter texts to match the maximum sequence length. We also created a function to encode labels, mapping each author to a numerical value suitable for training the model. Finally, the dataset was split into training and testing subsets, providing the foundation for building, training, and evaluating the LSTM model.

```
# Find maxlen that covers 95% of the texts
maxlen_ = int(np.percentile(df_length_pandas, 95))
print("Recommended maxlen:", maxlen_)

✓ 0.0s

Recommended maxlen: 314
```

Figure 11: Function to find the average text length

```
# NLTK downloads
nltk.download('stopwords')
nltk.download('wordnet')

# Initialize stopwords and lemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Functions for text preprocessing
> def preprocess_text(text): ...

# Function to tokenize texts
> def tokenize_texts(texts, num_words=None): ...

# Function to pad sequences
> def pad_text_sequences(sequences, maxlen=310): ...

# Function to encode labels
> def encode_labels(labels): ...

# Function to prepare data for modeling
> def prepare_data(df, text_col="text", label_col="author", maxlen=314, num_words=None, test_size=0.2, random_state=42): ...

✓ 0.0s
```

Figure 12: Functions for preparing the data

```
# Use of prepare_data function

# convert Spark DataFrame to Pandas DataFrame
df_pandas = df_filter.select("text", "author").toPandas()

# Split data into training and testing sets
X_train, X_test, y_train, y_test, tokenizer, vocab_size, label_encoder = prepare_data(
    df_pandas, text_col="text", label_col="author", maxlen=314, num_words=10000
)

# Print shapes of the datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
print("Vocabulary size:", vocab_size)
print("Classes:", label_encoder.classes_)

✓ 4.7s

X_train shape: (14437, 314)
X_test shape: (3610, 314)
y_train shape: (14437,)
y_test shape: (3610,)
Vocabulary size: 21127
Classes: ['EAP' 'HPL' 'MWS']
```

Figure 13: Split data into training and testing sets

The output of the data split shows that we obtained 14,437 training samples and 3,610 test samples. Each sample is represented as a sequence of 314 tokens, which corresponds to the maximum sequence length defined during preprocessing. The vocabulary size is 21,127 unique tokens, and the task involves classifying texts into three author classes: Edgar Allan Poe (EAP), H.P. Lovecraft (HPL), and Mary Shelley (MWS). This representation ensures that the model receives inputs of consistent length, making it suitable for training an LSTM-based architecture.

5. Train the model

In this section, we began by selecting several hyperparameters to build, compile, and train the model. The table below shows the chosen hyperparameters.

| Hyperparameter | Value | Description |
|-------------------|-------|--|
| embedding_dim | 18 | Embedding vector dimension |
| vocab_size | 21127 | Vocabulary size from tokenizer |
| input_length | 100 | Max sequence length |
| lstm_units | 128 | Number of LSTM units |
| dense_units | 64 | Dense hidden layer size |
| dropout_rate | 0.3 | Dropout rate for regularization |
| recurrent_dropout | 0.2 | Dropout for LSTM recurrent connections |
| learning_rate | 0.001 | Optimizer learning rate |
| epochs | 5 | Training epochs |

| | | |
|------------|----|------------|
| batch_size | 64 | Batch size |
|------------|----|------------|

The selected hyperparameters guided both the training process and the performance of the LSTM model. The embedding dimension determined the size of the vector representations for each word, allowing the model to capture semantic relationships within the text (Manik, 2020). The input sequence length standardized all samples, ensuring consistent training across the dataset. The number of LSTM units controlled the model's capacity to learn long-term dependencies in the sequences, while the dense layer facilitated the extraction of higher-level features. Dropout and recurrent dropout were applied to reduce overfitting and improve generalization (Adrian, 2018). Additionally, the learning rate regulated the pace of weight updates during training, and the chosen number of epochs and batch size balanced computational efficiency with effective model convergence (Kuldeep, 2021).

Following this, we built the models using `Sequential()`, which is the simplest type of model in Keras. It consists of a linear stack of layers, allowing us to configure the network layer by layer (Databricks, [s.a.]).

First, the Embedding layer transformed words into dense vectors to capture semantic relationships. Next, the LSTM layer processed the sequences, learning long-term dependencies in the text. A Dense layer followed to extract higher-level features, and a Dropout layer was added to prevent overfitting. Finally, the last Dense layer produced class probabilities for author prediction. The overall structure and flow of these layers, after compiling the model using the Adam optimizer and sparse categorical cross-entropy as the loss function, are illustrated in the figure below.

| Layer (type) | Output Shape | Param # |
|-------------------------|--------------|-------------|
| embedding_1 (Embedding) | ? | 0 (unbuilt) |
| lstm_1 (LSTM) | ? | 0 (unbuilt) |
| dense_2 (Dense) | ? | 0 (unbuilt) |
| dropout_1 (Dropout) | ? | 0 |
| dense_3 (Dense) | ? | 0 (unbuilt) |

Figure 14: Architecture of the LSTM model

6. Evaluate the model

To evaluate the model, we employed several metrics. Loss was used to measure the difference between the predicted values and the actual target values (BTB, 2024), while accuracy assessed the overall performance. The classification report provided detailed insights into precision, recall, and F1-score. The confusion matrix visualized both correct and incorrect predictions, helping to understand how well the model distinguished between different authors (Aniruddha, 2025). Additionally, we visualized the loss and accuracy during training and validation to monitor the model's learning process.

First, we printed the test loss and accuracy to get an initial understanding of the model's performance.

| Test Loss | Test Accuracy |
|-----------|---------------|
| 1.0915 | 0.39 (39%) |

The high test loss suggests that the model struggled to correctly classify many of the test samples, indicating underfitting or insufficient learning. On the other hand, the accuracy for this task is relatively low, considering that there are only three main authors, even a random classifier could achieve comparable or slightly better results. These findings indicate that the configuration used to build, compile, and train the model was not sufficient for effectively identifying authors based on their writing style.

We then used the classification report to further evaluate the model (see figure below).

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| EAP | 0.39 | 1.00 | 0.56 | 1409 |
| HPL | 0.00 | 0.00 | 0.00 | 1090 |
| MWS | 0.00 | 0.00 | 0.00 | 1111 |
| accuracy | | | 0.39 | 3610 |
| macro avg | 0.13 | 0.33 | 0.19 | 3610 |
| weighted avg | 0.15 | 0.39 | 0.22 | 3610 |

Figure 16: Classification report

We analyzed precision, which measures the proportion of correctly predicted instances for each author among all instances predicted as that author, and recall, which indicates how effectively the model identified all actual instances of each author. The F1 score

combines both precision and recall to provide a more balanced assessment of the model’s performance in distinguishing between authors.

| | |
|-----------|--------------|
| Precision | 0.1523 (15%) |
| Recall | 0.3903 (39%) |
| F1-score | 0.2191 (21%) |

The confusion matrix below shows that the model predicted only a single class (EAP) for all samples. This indicates that the model is underfitting and failing to distinguish between the different authors. As a result, there are no correct classifications for H.P. Lovecraft (HPL) and Mary Shelley (MWS), highlighting the model’s inability to learn the distinctive writing styles of each author.

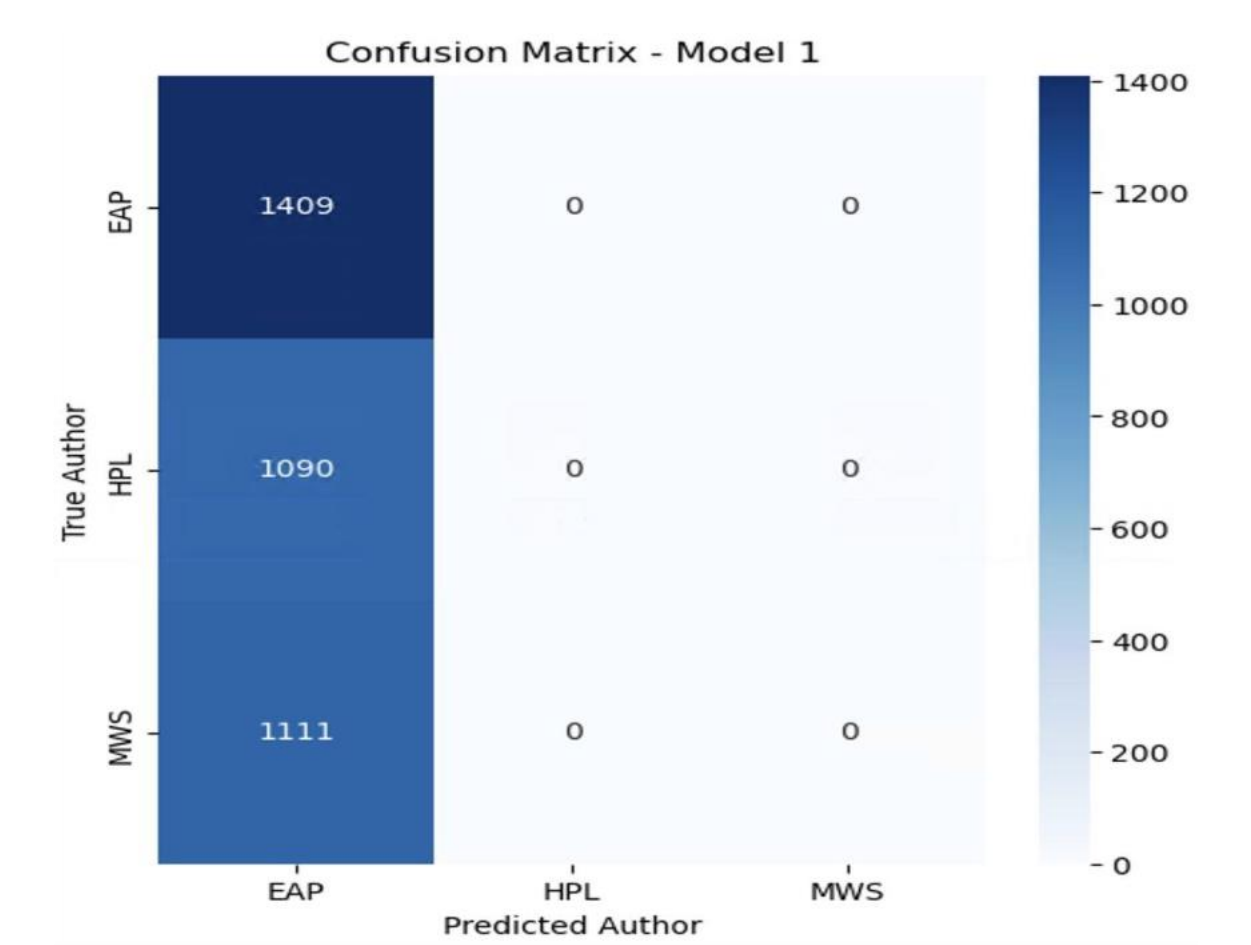


Figure 17: Confusion matrix

The figure below shows a comparative graph of training and validation accuracy and loss. The training accuracy gradually increases but remains relatively low, while the validation

accuracy stays higher throughout. The training loss initially decreases but then fluctuates, and the validation loss shows a similar pattern. In general, the training loss indicates how well the model is learning, while the validation loss shows its ability to generalize to new data (Frederik, 2023). Ideally, loss should decrease steadily and accuracy should increase, suggesting effective learning without overfitting (Frederik, 2023). The observed fluctuations and relatively low training accuracy indicate that the model is still struggling to learn the patterns in the data and its performance needs improvement.

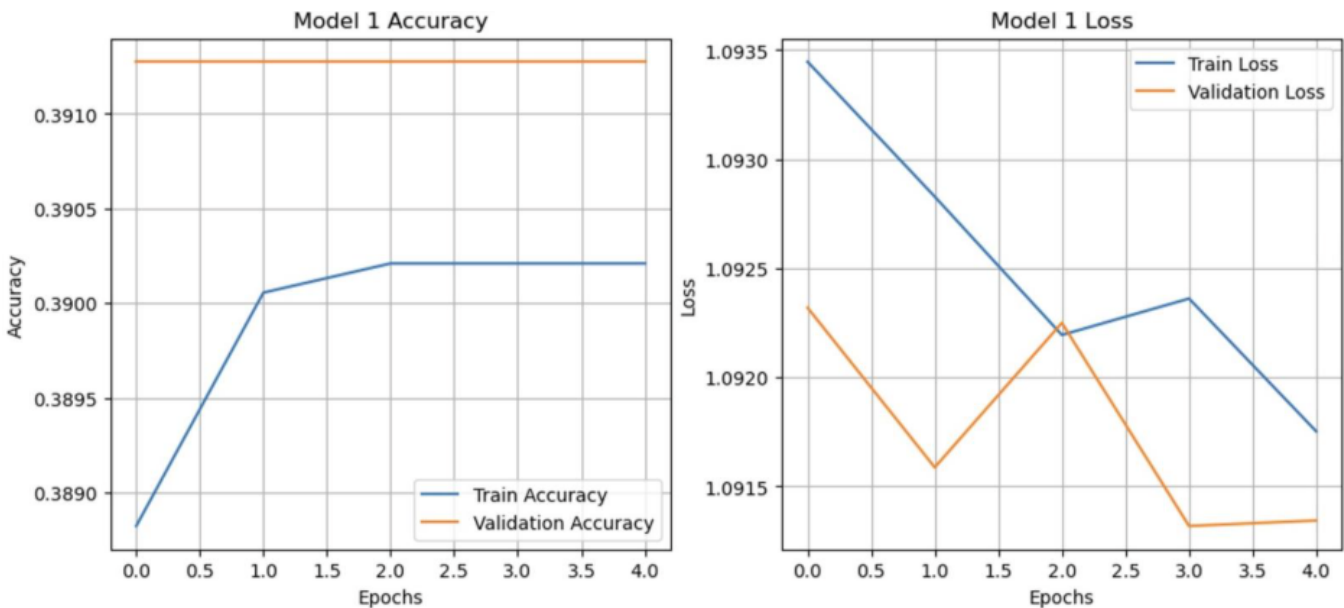


Figure 19: Model 1 Accuracy Vs Loss

Based on the evaluation, the first model clearly needed improvement for several reasons. The test loss was relatively high at 1.0915, while the test accuracy was low at 39%, indicating that the model struggled to make correct predictions on unseen data. The precision was only 15%, showing that many of the model's positive predictions were incorrect. The recall was 39%, suggesting that the model failed to identify a large portion of the true positive cases. The F1-score of 21% confirms that the model's overall predictive performance is weak. These metrics demonstrate that the model was limited in its ability to learn and generalize, justifying the need for improvements in architecture, embeddings, or training strategy.

7. Retrained the model

We retrained the model and tuned the hyperparameters to improve performance.

| Hyperparameter | Value | Description |
|----------------|-------|----------------------------|
| embedding_dim | 200 | Embedding vector dimension |

| | | |
|-------------------|-------|--|
| vocab_size | 21127 | Vocabulary size from tokenizer |
| input_length | 314 | Max sequence lengt |
| lstm_units | 256 | Number of LSTM units |
| dense_units | 128 | Dense hidden layer size |
| dropout_rate | 0.4 | Dropout rate for regularization |
| recurrent_dropout | 0.3 | Dropout for LSTM recurrent connections |
| learning_rate | 0.001 | Optimizer learning rate |
| epochs | 20 | Training epochs |
| batch_size | 64 | Batch size |

By tuning the hyperparameters, we optimized the model’s ability to learn meaningful patterns from the data while reducing overfitting and improving generalization.

We built the model again using a Sequential architecture. Unlike the first model, this version includes a Bidirectional LSTM. First, the Embedding layer transforms words into dense vectors to capture semantic relationships. Next, the Bidirectional LSTM layer allows information to flow both forward and backward, enabling the model to capture richer contextual information while learning long-term dependencies in the text (Geeksforgeeks, 2025). A Dense layer follows to extract higher-level features, and a Dropout layer is added to prevent overfitting. Finally, the last Dense layer produces class probabilities for author prediction. The overall structure and flow of these layers are illustrated in the figure below.

| Layer (type) | Output Shape | Param # |
|---------------------------------|--------------|-------------|
| embedding_3 (Embedding) | ? | 0 (unbuilt) |
| bidirectional_2 (Bidirectional) | ? | 0 (unbuilt) |
| dense_6 (Dense) | ? | 0 (unbuilt) |
| dropout_3 (Dropout) | ? | 0 |
| dense_7 (Dense) | ? | 0 (unbuilt) |

Figure 20: Retrained model architecture

On top of that, we added EarlyStopping and ReduceLROnPlateau callbacks to prevent overfitting and adjust the learning rate dynamically during training, further improving the model’s ability to generalize (Himanshu, 2020).

8. Evaluate the retrained model

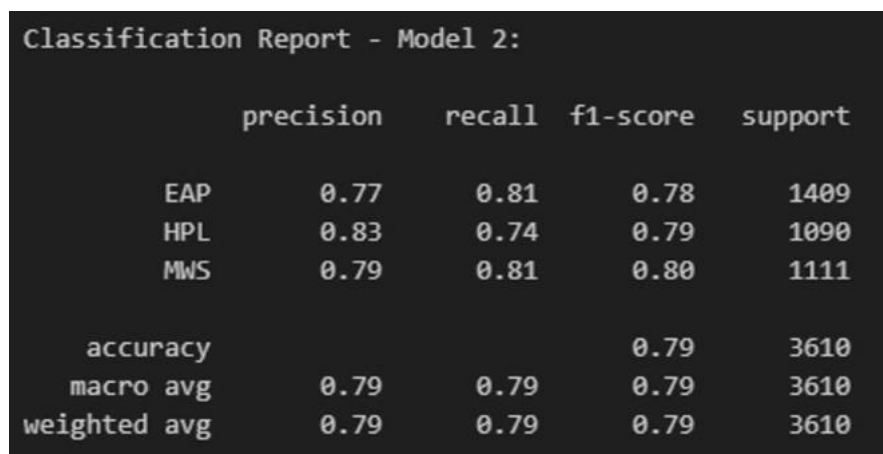
As for the evaluation of the first model, we used the same metrics such as loss, accuracy, precision, recall, F1-score, and the confusion matrix to evaluate the trained model.

First, we printed the test loss and accuracy to get an initial understanding of the model's performance.

| Test Loss | Test Accuracy |
|-----------|---------------|
| 0.5172 | 0.79 (78%) |

The lower test loss indicates that the error between predicted and actual values has decreased, which is better for a model because it reflects improved learning and generalization. At the same time, the higher test accuracy demonstrates that the model is making more correct predictions, showing that it performs better than the previous version.

We then used the classification report to further evaluate the model (see figure below).



```
Classification Report - Model 2:

              precision    recall  f1-score   support

   EAP         0.77        0.81        0.78        1409
   HPL         0.83        0.74        0.79        1090
   MWS         0.79        0.81        0.80        1111

 accuracy              0.79              3610
 macro avg           0.79        0.79        0.79        3610
 weighted avg       0.79        0.79        0.79        3610
```

Figure 21: Classification report model 2

Based on this report, we calculated the average precision, recall, and F1-score to better evaluate the model's overall performance, as shown in the table below.

| | |
|-----------|--------------|
| Precision | 0.7913 (79%) |
| Recall | 0.7884 (78%) |
| F1-score | 0.7894 (78%) |

We observed an improvement in precision, which measures the proportion of correctly predicted instances for each author among all instances predicted as that author, and recall, which indicates how effectively the model identified all actual instances of each author. The F1-score combines both precision and recall to provide a balanced assessment of the model's performance in distinguishing between authors.

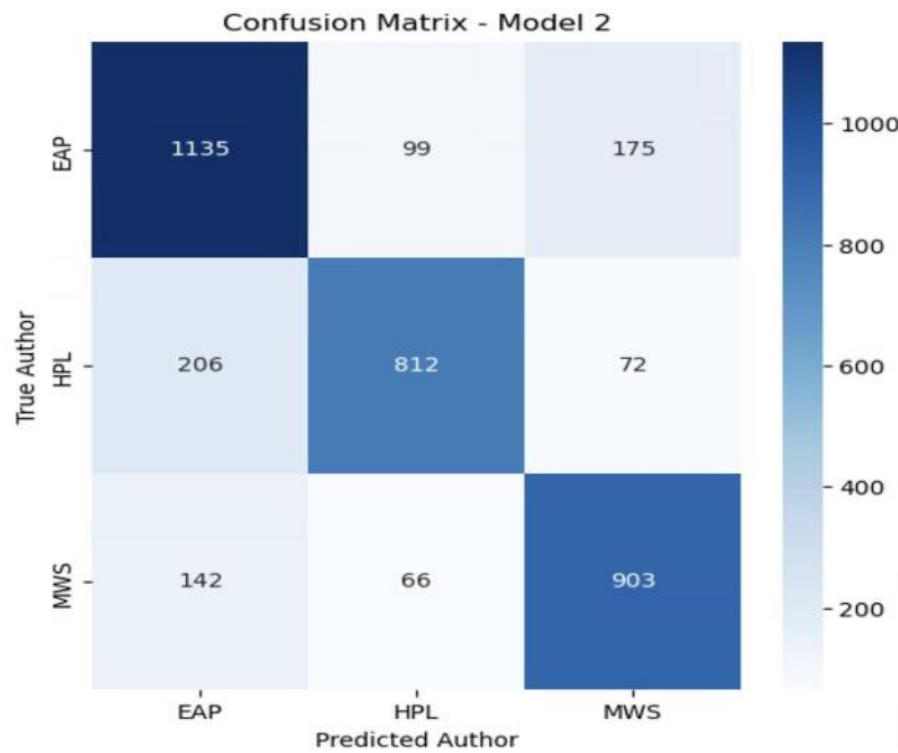


Figure 22: confusion matrix Model 2

The confusion matrix below shows that the model correctly predicted many instances for each author but also made several misclassifications. For instance, the model coorectly predicted 1135 sentences for EAP, while 89 and 175 were misclassified as HPL and MWS, respectively. Similarly, for HPL, 812 instances were correctly predicted, with 206 misclassified as EAP and 72 as MWS. For MWS, 903 were correctly classified, while 142 and 66 were misclassified as EAP and HPL.

This indicates that, although the model can distinguish between authors to some extent, there is still overlap in predictions, suggesting that further improvements may be necessary to enhance its ability to accurately capture each author's distinctive writing style

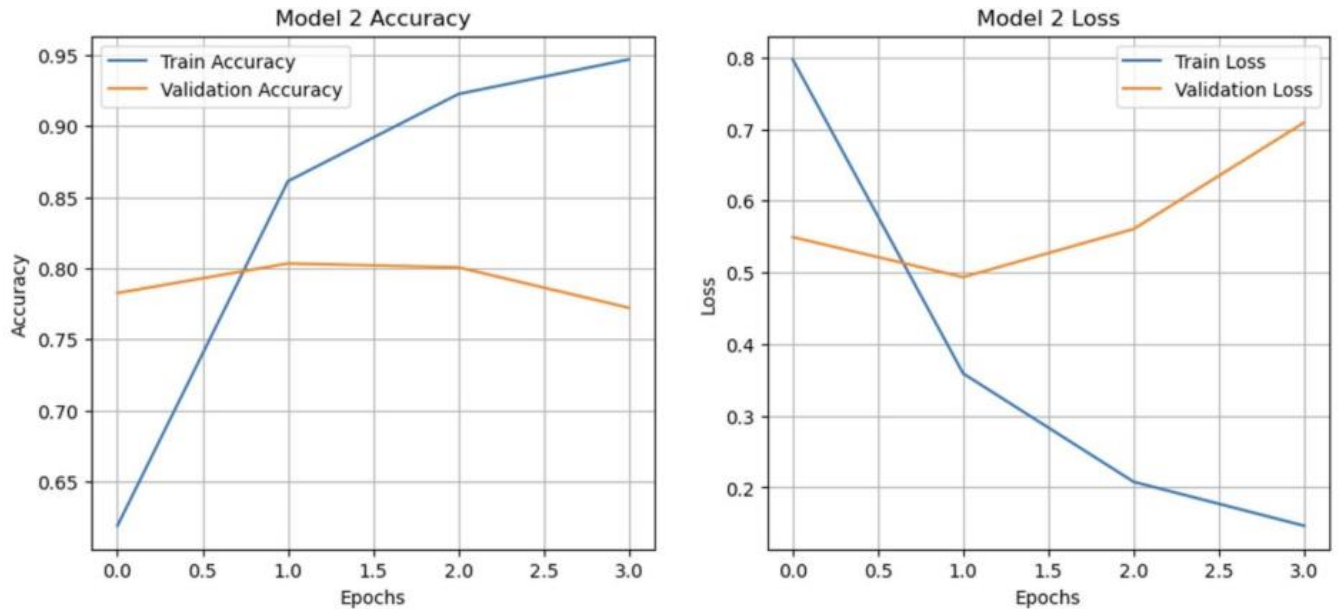


Figure 23: Model 2 Accuracy Vs Loss

Compared to the previous version, this model shows an improvement in the graph. The training loss indicates how well the model is learning, while the validation loss reflects its ability to generalize to new data (Frederik, 2023). We observed that the loss decreased steadily and accuracy increased, suggesting effective learning with minimal overfitting.

9. Conclusion

We have reached the end of this analysis, where our goal was to build a model capable of distinguishing or predicting an author based on their unique writing style. We used a dataset for author identification from Kaggle, which allowed us to conduct the analysis. After cleaning and exploring the data, we gained a clear understanding that enabled us to build an LSTM model from scratch. The first model achieved a test accuracy of 39% with a loss of 1.0915, indicating that it was not able to correctly predict the authors. To address these limitations, we tuned the hyperparameters and added a bidirectional layer to allow information to flow both forward and backward in the sequences. After retraining, the model achieved a significantly improved accuracy of 79% with a loss of 0.5127, demonstrating a clear enhancement in performance. Despite this improvement, there is still room for further refinement, as the model occasionally misclassified sentences for other authors, as shown in the confusion matrix (Figure 22). Implementing techniques such as GloVe embeddings could further increase the model's accuracy and overall performance.

References

Abbasi, A. and Chen, H., 2005. Applying authorship analysis to extremist-group web forum messages. IEEE Intelligent Systems, 20(5), pp.67-75.

Adrian, G. 2018. A review of Dropout as applied to RNNs. [Online]. Available at: <https://adriangcoder.medium.com/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b> (Accessed: 01 October 2025).

Aniruddha, B. 2025. Confusion Matrix in Machine Learning. [Online]. Available at: <https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/> (Accessed: 01 October 2025).

AWS. [s.a]. What is RNN (Recurrent Neural Network)? [Online]. Available at: <https://aws.amazon.com/what-is/recurrent-neural-network/> (Accessed: 01 October 2025).

BTB. [s.a]. Performance Insights: Train Loss vs. Test Loss in Machine Learning Models. Available at: <https://baotramduong.medium.com/machine-learning-train-loss-vs-test-loss-735ccd713291> (Accessed: 01 October 2025).

Databricks. [s.a]. Keras Model [Online]. Available at: <https://www.databricks.com/glossary/keras-model> (Accessed: 01 October 2025).

Geeksforgeeks, 2025. Bidirectional LSTM in NLP. [Online]. Available at: <https://www.geeksforgeeks.org/nlp/bidirectional-lstm-in-nlp/> (Accessed: 01 October 2025).

GoogleCloud. [s.a]. What is deep learning? [Online]. Available at: <https://cloud.google.com/discover/what-is-deep-learning/> (Accessed: 01 October 2025).

Himanshu, S. 2018. Is your epochs not performing? Try callbacks. [Online]. Available at: <https://medium.com/@himanshuit3036/is-your-epochs-not-performing-try-callbacks-76519f0368a9> (Accessed: 01 October 2025).

Manik, S. 2020. Understanding Word Embeddings from scratch | LSTM model. [Online]. Available at: <https://medium.com/data-science/word-embeddings-and-the-chamber-of-secrets-lstm-gru-tf-keras-de3f5c21bf16/> (Accessed: 01 October 2025)

Savoy, J., 2020. Machine learning methods for stylometry: Authorship attribution and author profiling. Cham: Springer International Publishing [online]