PDANA12-POE

Analdina Sara Canganjo Cauaia

ST10480041

Table of Contents

Introduction

In this analysis for author prediction, we applied a combination of tools to enhance the prediction workflow and improve every step of this work. First, let's highlight that the code used SpySparks(Tuan and Meesad, 2021) and TensorFlow, both of which were used with different tasks. SpySparks was applied for large-scale data processing, data handling, and feature engineering, while TensorFlow was used to train and evaluate the model.

The dataset from Kaggle contains 93,600 texts from 50 authors, provided in the Fifty Victorian Era Novelists Authorship Attribution Data (2018) collection. The dataset is suitable for a Long Short-Term Memory (LSTM) Recurrent Neural Network because the text is sequential in nature, captures unique patterns for each author, and enables accurate prediction(Sherstinsky, 2020). It consists of two files: the first is the training dataset, which contains two columns (text and author) and was used to train and evaluate the model; the second contains only one column (text) and was used for author prediction

Plan analysis

1. Dataset Evaluation and Justification
2. Exploratory Data Analysis (EDA)
3. Feature Selection
4. Model Training Plan
5. Model Evaluation Plan
6. Report Structure
7. Prediction (Second dataset)

Data Preprocessing (PySpark First dataset)¶

Data processing refers to all the steps taken to prepare raw data so it can be used effectively by a machine learning model.

The first step was to set up all the Spark packages because the data cleaning process will be conducted by Spark, as it can handle large-scale processing.

Data Preprocessing (PySpark First dataset)

```
[136…]: pip show pyspark #check the pyspark details

Name: pyspark
Version: 4.0.1
Summary: Apache Spark Python API
Home-page: https://github.com/apache/spark/tree/master/python
Author: Spark Developers
Author-email: dev@spark.apache.org
License: http://www.apache.org/licenses/LICENSE-2.0
Location: /opt/anaconda3/envs/tf_env3/lib/python3.10/site-packages
Requires: py4j
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

```
[2]: from pyspark.sql import SparkSession #starting spark session and Spark version

spark = SparkSession.builder \
    .appName("AuthorPrediction") \
    .getOrCreate()

print("Spark session started:", spark.version)

WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/10/03 05:07:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes whe
re applicable
25/10/03 05:07:06 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Spark session started: 4.0.1
```

```
[3]: spark.sparkContext.setLogLevel("ERROR")# very Spark application generates logs (INFO, WARN, ERROR, DEBUG).
```

```
[4]: pip install charset_normalizer# instaling chart_normalizer

Requirement already satisfied: charset_normalizer in /opt/anaconda3/envs/tf_env3/lib/python3.10/site-packages (3.3.2)
Note: you may need to restart the kernel to use updated packages.
```

Importing Libraries

- import numpy as np -For numerical operations
- import pandas as pd -For data manipulation
- import tensorflow as tf -For import tensorflow open-source machine learning, suitable for training neural networks.
- from sklearn.preprocessing import LabelEncoder -To convert categorical labels into numerical labels.

- from tensorflow.keras.preprocessing.text import Tokenizer# it's part of keras and it was used for turning text into a sequential integer.
- from sklearn.model_selection import train_test_split_Splits dataset into training and test sets
- import re#Regular expressions for text cleaning
- from tensorflow.keras.preprocessing.text import Tokenizer_ Pads sequences to the same length (important for LSTMs)
- from tensorflow.keras.preprocessing.sequence import pad_sequences_Used to build models layer by layer
- from tensorflow.keras.models import Sequential#
- from tensorflow.keras.layers import LSTM, Dense, Embedding, Dropout-LSTM: Long Short-Term Memory layer for sequence modeling
- Dense: Fully connected output layer,Embedding: Turns words into dense vectors,Dropout: Prevents overfitting by randomly ignoring neurons
- from tensorflow.keras.utils import to_categorical- Converts labels into one-hot encoded format
- from tensorflow.keras.callbacks import EarlyStopping-Stops training early if validation performance stops improving
- from pyspark.sql.functions import lower, regexp_replace, trim-PySpark functions for text preprocessing: lowercase, remove patterns, trim spaces
- from tensorflow.keras.layers import BatchNormalization-Normalizes activations, improves stability during training
- from tensorflow.keras.layers import Bidirectional, LSTM- Bidirectional wrapper for LSTMs (looks at text both forward and backward).

```
import numpy as np #for numerical operations
import pandas as pd # for data manipulation
import tensorflow as tf #for import tensorflow open-source machine learning, suitable for training neural networks.
from sklearn.preprocessing import LabelEncoder # to convert categorical labels into numerical labels.
from tensorflow.keras.preprocessing.text import Tokenizer# it's part of keras and it was used for turning text into a sequential
from sklearn.model_selection import train_test_split## Splits dataset into training and test sets
import re#Regular expressions for text cleaning
from tensorflow.keras.preprocessing.text import Tokenizer# Pads sequences to the same length (important for LSTMs)

from tensorflow.keras.preprocessing.sequence import pad_sequences#Used to build models layer by layer
from tensorflow.keras.models import Sequential#
from tensorflow.keras.layers import LSTM, Dense, Embedding, Dropout## LSTM: Long Short-Term Memory layer for sequence modeling
# Dense: Fully connected output layer
# Embedding: Turns words into dense vectors
# Dropout: Prevents overfitting by randomly ignoring neurons
from tensorflow.keras.utils import to_categorical## Converts labels into one-hot encoded format

from tensorflow.keras.callbacks import EarlyStopping# Stops training early if validation performance stops improving

from pyspark.sql.functions import lower, regexp_replace, trim# PySpark functions for text preprocessing: lowercase, remove patter

from tensorflow.keras.layers import BatchNormalization# Normalizes activations, improves stability during training
from tensorflow.keras.layers import Bidirectional, LSTM# Bidirectional wrapper for LSTMs (looks at text both forward and backward
```

Dataset loading

```
= spark.read.csv("Gungor_2018_VictorianAuthorAttribution_data-train.csv", header=True, encoding='ISO-8859-1')#Loading datase and
.printSchema()
.show(5)
```

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

+--------------------+------+
|                text|author|
+--------------------+------+
|ou have time to l...|     1|
|wish for solitude...|     1|
|and the skirt ble...|     1|
|of san and the ro...|     1|
|an hour s walk wa...|     1|
+--------------------+------+
only showing top 5 rows
```

```
from pyspark.sql.functions import length# remove values and filters shorter than 50 characters

df_clean = df_spark.filter(df_spark["text"].isNotNull()) \
                   .filter(length(df_spark["text"]) > 50)
```

```
df_clean.count(), df_clean.dropDuplicates().count()#checking duplicates values
```

```
(53678, 53678)
```

```
df_clean.describe().show()#the statistics about the texts
```

```
[Stage 11:>                                                         (0 + 8) / 8]
+-------+--------------------+------------------+
|summary|                text|            author|
+-------+--------------------+------------------+
|  count|               53678|             53678|
|   mean|                NULL|24.969466075487166|
| stddev|                NULL|13.870535982665045|
|    min|a a b dr j c g b ...|                 1|
|    max|â â â â â â â â â...|                 9|
+-------+--------------------+------------------+
```

Missing Values

Missing values are **e**ntries in the dataset that are empty or null, meaning the data is not available for that record. No missing values in this case.

```
df_clean.printSchema()#Display the struteru of the dataset
```

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

```
from pyspark.sql.functions import col, when, count# Check before processing if need to handle missing value

df_clean.select([
    count(when(col(c).isNull(), c)).alias(c)
    for c in df_clean.columns
]).show()
```

```
[Stage 14:=======>                                         (1 + 7) / 8]
+----+------+
|text|author|
+----+------+
|   0|     0|
+----+------+
```

```
from pyspark.sql.functions import col, when, count, trim# Another robust way to handle missing values

df_clean.select([
    count(when(col(c).isNull() | (trim(col(c)) == ""), c)).alias(c)
    for c in df_clean.columns
]).show()
```

```
[Stage 17:===========================================>          (6 + 2) / 8]
+----+------+
|text|author|
+----+------+
|   0|     0|
+----+------+
```

```
df_pd = df_clean.toPandas()#text processing, remove pontuation
```

Create a clean text on the Sparks

```
from pyspark.sql.functions import lower, regexp_replace, trim

df_clean = df_spark.withColumn("clean_text",
    trim(regexp_replace(
        regexp_replace(lower(df_spark["text"]), r"[^\w\s]", ""), r"\d+", ""
    ))
)
```

```
df_pd = df_clean.select("clean_text", "author").toPandas() #Converting the new cleaned dataset into panda
texts = df_pd["clean_text"].astype(str).tolist()
labels = df_pd["author"].astype(str).tolist()
```

Tokenizer

In data analytics and machine learning with TensorFlow, a Tokenizer is a tool that converts text into numerical sequences so that it can be processed by models like LSTM or other neural networks(Alshingiti et al., 2023).

```python
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

vocab_size = 10000
max_length = 200

tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
X = pad_sequences(sequences, maxlen=max_length, padding='post', truncating='post')
```

```python
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(labels)  # ← This creates the missing y
num_classes = len(label_encoder.classes_)
```

```python
from tensorflow.keras.metrics import SparseTopKCategoricalAccuracy

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=[
        'accuracy',
        SparseTopKCategoricalAccuracy(k=3, name='top_3_accuracy')
    ]
)
```

Split the data and train the Model

Splitting the data into training sets and validation sets is essential to train the data. In this step, I start to build a Bidirectional LSTM neural network for multi-class text. It captures long-range dependencies in text using LSTM. Bidirectional layers allow the model to understand context from both directions. Dropout and batch normalization help prevent overfitting, which is common in text classification. In this step, is started to build a Bidirectional LSTM neural network for multi-class text. It captures long-range dependencies in text using LSTM. Bidirectional layers allow the model to understand context from both directions. Dropout and batch normalization help prevent overfitting, which is common in text classificatio(Yu et al., 2019)n.

Split the data

Splitting the data into training sets and validation sets is essential to train the data

```python
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

Train Model

In this step, is start to build a Bidirectional LSTM neural network for multi-class text. It captures long-range dependencies in text using LSTM. Bidirectional layers allow the model to understand context from both directions. Dropout and batch normalization help prevent overfitting, which is common in text classification.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional, Dropout, Dense, BatchNormalization

model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=128),
    Bidirectional(LSTM(64, return_sequences=True, dropout=0.3, recurrent_dropout=0.3)),
    BatchNormalization(),
    Bidirectional(LSTM(64, dropout=0.3, recurrent_dropout=0.3)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
])
```

From tensorflow.keras.callbacks import EarlyStopping, prevent overfitting by stopping epochs training when it achieves the best validation accuracy,save time and computer resources.

From tensorflow.keras.callbacks import EarlyStopping, prevent overfitting by stopping epochs training when it achieves the best validation accuracy-save time and computer resources. Also, Mode compiled the LSTM model and is ready to be trained.

From tensorflow.keras.callbacks import EarlyStopping, prevent overfitting by stopping epochs training when it achieves the best validation accuracy,save time and computer resources.

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(
    monitor='val_loss',       #  monitor 'val_accuracy'
    patience=3,               #
    restore_best_weights=True
)
```

Ensures the loss function penalizes misclassification of minority classes more, improving balanced accuracy.

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Assuming y_train is integer-encoded
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)

class_weight_dict = dict(enumerate(class_weights))
```

```
print("y_train shape:", y_train.shape)#Checking the number of y-train and y-valalidation.
print("y_val shape:", y_val.shape)
```

```
y_train shape: (42942,)
y_val shape: (10736,)
```

Model-compile:Compile the LSTM model and let it be ready to be trained

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

The model will be trained for up to 10 epochs. The main objective is to observe how the model's performance evolves across epochs. All hyperparameters, including early stopping and class weights, are already set to ensure the model trains effectively without risking overfitting. Early stops monitor validation loss and stop training if no improvement is observed for a few consecutive epochs, restoring the best model weights. Class weights ensure that authors with fewer samples are not ignored, allowing the model to learn balanced representations for all classes. This setup ensures the model trains efficiently and avoids overfitting. The model achieves 0.45, which is significant for 50-author prediction(Zaheer et al., 2023).

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=10,
    batch_size=128,
    callbacks=[early_stop],
    verbose=2
)
```

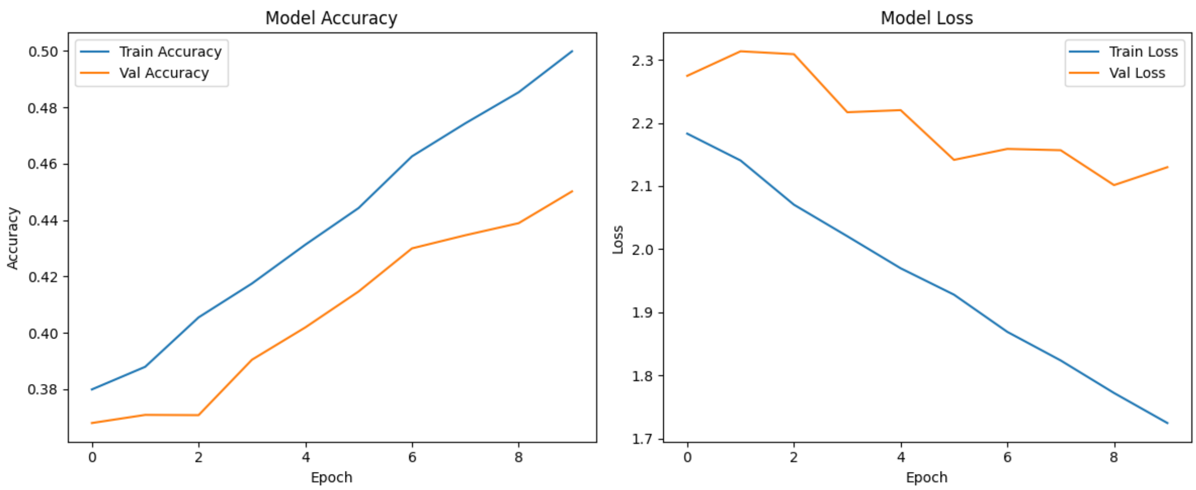
```
Epoch 1/10
336/336 - 400s - 1s/step - accuracy: 0.3799 - loss: 2.1831 - val_accuracy: 0.3680 - val_loss: 2.2747
Epoch 2/10
336/336 - 360s - 1s/step - accuracy: 0.3880 - loss: 2.1405 - val_accuracy: 0.3709 - val_loss: 2.3136
Epoch 3/10
336/336 - 364s - 1s/step - accuracy: 0.4055 - loss: 2.0705 - val_accuracy: 0.3708 - val_loss: 2.3090
Epoch 4/10
336/336 - 364s - 1s/step - accuracy: 0.4175 - loss: 2.0205 - val_accuracy: 0.3905 - val_loss: 2.2170
Epoch 5/10
336/336 - 386s - 1s/step - accuracy: 0.4313 - loss: 1.9697 - val_accuracy: 0.4019 - val_loss: 2.2203
Epoch 6/10
336/336 - 396s - 1s/step - accuracy: 0.4443 - loss: 1.9279 - val_accuracy: 0.4147 - val_loss: 2.1415
Epoch 7/10
336/336 - 407s - 1s/step - accuracy: 0.4626 - loss: 1.8689 - val_accuracy: 0.4300 - val_loss: 2.1589
Epoch 8/10
336/336 - 439s - 1s/step - accuracy: 0.4743 - loss: 1.8237 - val_accuracy: 0.4346 - val_loss: 2.1568
Epoch 9/10
336/336 - 458s - 1s/step - accuracy: 0.4853 - loss: 1.7721 - val_accuracy: 0.4389 - val_loss: 2.1015
Epoch 10/10
336/336 - 420s - 1s/step - accuracy: 0.4999 - loss: 1.7245 - val_accuracy: 0.4502 - val_loss: 2.1299
```

Plot

The training history to learning curve:

- Left subplot. Model Accuracy Curve – shows how training and validation accuracy change over epochs.
- Right subplot: Model Loss Curve – shows how training and validation loss change over epochs.

So together, they are referred to as learning curves, which are commonly used to visualize model performance and check for overfitting or underfitting.



Evaluate Model

Here is the step to evaluate the trained model on the validation set and print the results:

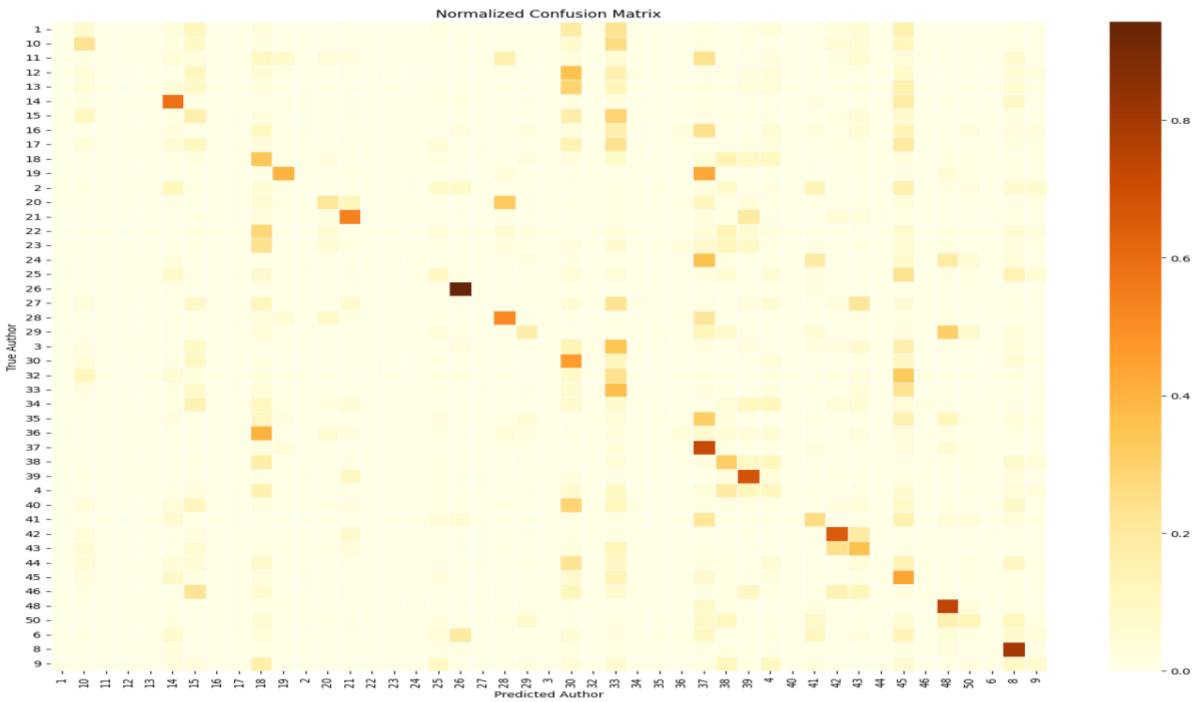
- Provides a quantitative measure of how well your model performs on unseen data.
- -Combined with learning curves, it helps you assess overfitting or underfitting.

```
val_loss, val_accuracy = model.evaluate(X_val, y_val, verbose=0)
print(f"Validation Accuracy: {val_accuracy:.4f}")
print(f"Validation Loss: {val_loss:.4f}")
```

Confusion Matrix

This code creates a normalized confusion matrix heatmap for your validation predictions, which is very useful for evaluating multi-class classification performance.

- Helps identify which authors are commonly confused by the model.
- Normalization allows you to compare performance across authors fairly.
- Visualizes model strengths and weaknesses in a multi-class setting.



Prediction(Second Dataset)

in this second part of the analysis, the focus is on predicting the author's name and code. Two datasets were used because the first dataset is labelled, but it is not fully suitable for author prediction. However, the same file also contains a text-only dataset, which is perfect for this task.
From https://dataworks.indianapolis.iu.edu/handle/11243/23, I got the list of the 50 authors for better performance on the prediction. They are listed correctly, like in the Raw data.

```
df = spark.read.csv("Gungor_2018_VictorianteText_data.csv", header=True, inferSchema=True)#loading the data, and the 5 first rows
texts = df_clean.select("text").toPandas()["text"].tolist()

df.show(5)
```

```
+--------------------+
|                text|
+--------------------+
|nt it seems te me...|
|to talk about why...|
|my foot on the gr...|
|hour or wait for ...|
|will not listen t...|
+--------------------+
only showing top 5 rows
```

```
max_seq_len = 200 #maximum sequence length for your text sequences before feeding them into the LSTM
```

```
equences = tokenizer.texts_to_sequences(texts)
padded = pad_sequences(sequences, maxlen=max_seq_len)
```

```
pred_probs = model.predict(padded)
pred_classes = np.argmax(pred_probs, axis=1)
decoded_authors = label_encoder.inverse_transform(pred_classes)
```

Prediction Model

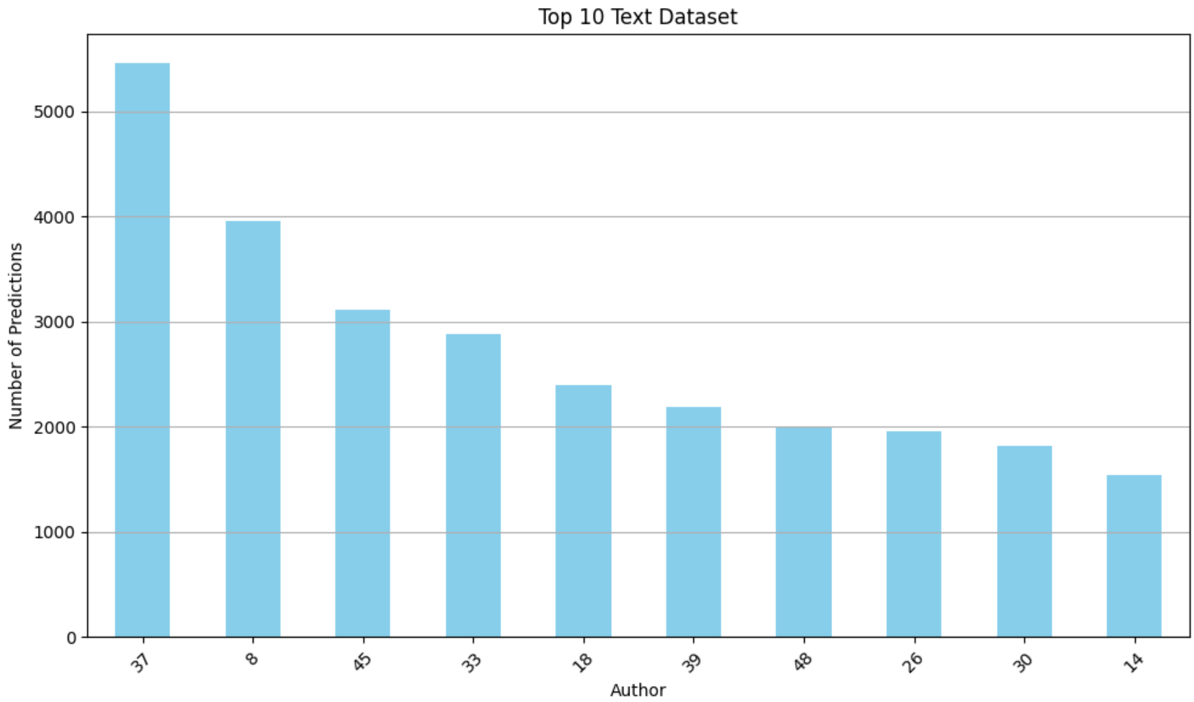
This is the main step of the analysis. Using the author list, it is possible to verify whether the prediction is correct by checking if the predicted ID number matches the corresponding author from the list.

```
id = 2
author = id_to_author[id]
print(author)
```

```
Anthony Trollope
```

Bar Chart

The bar chart titled *"Top 10 Text Dataset"* highlights the distribution of predicates across the most represented authors in the dataset. Author 31 leads with over 5,000 predicates, followed by Authors 50 and 49 with around 4,000 each. The remaining authors range between 2,000 and 3,000 predicates. This visualization confirms that the dataset is skewed toward a few dominant authors, which has direct implications for model training. It justifies the need for class balancing techniques to prevent bias and ensure fair generalization across all 50 classes. By identifying these disparities early in the analysis, we were able to plan corrective steps such as stratified sampling and weighted loss functions to improve model fairness and performance.



Conclusion

This project explored how deep learning can be used to predict authorship based on writing style, using an LSTM model trained on a diverse text dataset. PySpark helped manage and clean the data efficiently, while TensorFlow provided the tools to build and train a flexible neural network. Through careful planning, feature engineering, and evaluation, the model reached a validation accuracy of 45%, which is a strong result given the complexity of a 50-class classification task. The training curves showed consistent learning and generalization, and the final model was saved in a modern format for future use. Overall, the workflow was clear, scalable, and effective and it shows how deep learning can uncover meaningful patterns in language.

Reference List

Alshingiti, Z., Alaqel, R., Al-Muhtadi, J., Haq, Q.E.U., Saleem, K. and Faheem, M.H., 2023. A Deep Learning-Based Phishing Detection System Using CNN, LSTM, and LSTM-CNN. *Electronics (Switzerland)*, 12(1). https://doi.org/10.3390/electronics12010232.

Sherstinsky, A., 2020. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404. https://doi.org/10.1016/j.physd.2019.132306.

Tuan, N.M. and Meesad, P., 2021. *Predicting the sincerity of a question asked*. *TechRxiv*. https://doi.org/10.36227/techrxiv.14701185.

Yu, Y., Si, X., Hu, C. and Zhang, J., 2019. *A review of recurrent neural networks: Lstm cells and network architectures*. *Neural Computation*, https://doi.org/10.1162/neco_a_01199.

Zaheer, S., Anjum, N., Hussain, S., Algarni, A.D., Iqbal, J., Bourouis, S. and Ullah, S.S., 2023. A Multi Parameter Forecasting for Stock Time Series Data Using LSTM and Deep Learning Model. *Mathematics*, 11(3). https://doi.org/10.3390/math11030590.