COIMBRA BUSINESS SCHOOL

iscac Politécnico de Coimbra

100 ANOS

since 1921

Authorship identification based on Natural Language Processing and Machine Learning

Luis Coimbra

ISCAC | 2025

COIMBRA BUSINESS SCHOOL

iscac Politécnico de Coimbra

100 ANOS

Luis Miguel Chaves Coimbra

Authorship Identification of Texts based on Natural Language Processing and Machine Learning

Coimbra, abril de 2025

Luis Miguel Chaves Coimbra

Authorship Identification of Texts based on Natural Language Processing and Machine Learning

Thesis submitted to the Higher Institute of Accounting and Administration of Coimbra to fulfil the requirements for obtaining a master's degree in Data Analysis and Decision Support Systems, conducted under the guidance of Professor Dora Regina Oliveira Melo.

Coimbra, abril de 2025

STATEMENT OF RESPONSABILITY

I declare that I am the author of this Dissertation, which is an original and unpublished work, which has never been submitted to another Higher Education Institution to obtain an academic degree or other qualification. I further certify that all citations are properly identified and that I am aware that plagiarism is a serious lack of ethics, which could result in the annulment of this dissertation.

"Leave me where the moonbeams

Carve through the leaves like blades

Lay me in the tall grown grass in a shallow grave

Let it have me

I've got a place in the world

And I found my way

Out in the night all alone

In the way out there

I ain't lonely

I'm long lost

Say goodbye (long lost)

Stay with my mind (long lost)

Send me to the mountains

Let me go free forever (ever)

I'll be running through the forest

Dancing in the fields like this forever (ever)"

- Ben Schneider, Long Lost

DEDICATORY

I dedicate this dissertation to my family, whose continuous support and encouragement were essential throughout this journey. Their patience in listening to my reflections and helping me organize my thoughts played a crucial role in helping me stay focused and motivated.

I also wish to extend this dedication to my friends and colleagues from the master's program, Mário Simão and Ricardo Neto. Their companionship transformed long hours of study into moments of collaboration and lightness, making this academic journey all the more meaningful and rewarding.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to Professor Dora Regina Oliveira Melo, my supervisor, for her unwavering support throughout this challenging journey. Her belief in my abilities and her invaluable guidance were instrumental in helping me overcome the many obstacles I faced along the way.

I would also like to extend my sincere thanks to Professor Joana Leite for her assistance with various tasks related to the master's program. Her willingness to help, even on short notice, made a significant difference and was greatly appreciated.

RESUMO

O aumento contínuo na produção e disponibilização de conteúdos textuais apresenta tanto desafios quanto oportunidades para o campo de identificação de autores. Esta dissertação tem como principal objetivo desenvolver uma estratégia que utilize metodologias de Processamento de Língua Natural (PLN) e algoritmos de Aprendizagem Automática para identificar a autoria de textos.

A identificação de autoria consiste em determinar a identidade de um autor a partir de um determinado texto, com base nos seus padrões linguísticos e estilísticos exclusivos, assumindo um papel crucial, não só para compreender estilos de escrita, mas também para combater práticas como o plágio. O estudo, enquadrado como um problema de classificação supervisionada, utilizando dados provenientes da plataforma Twitter (atualmente conhecida como X) para explorar estratégias de identificação de autoria e construir modelos capazes de prever a autoria de textos com base em dados rotulados.

O estudo realizado propõe uma abordagem que utiliza metodologias de PLN para transformar textos em dados estruturados, permitindo a aplicação de técnicas de Aprendizagem Automática para criar modelos capazes de identificar padrões de escrita e associá-los a autores específicos. Para avaliar a eficácia dos modelos, serão explorados algoritmos tradicionais de Aprendizagem Automática, como *Naive Bayes* (NB) e *Random Forest* (RF), bem como modelos avançados de *Deep Learning* (DL), incluindo Redes Neuronais Convolucionais (CNN) e Redes Neuronais Recorrentes (RNN).

Esta dissertação adota a metodologia KDD (*Knowledge Discovery* in *Databases*), iniciando-se com a definição do problema e dos objetivos relacionados com a identificação de autoria de textos. Durante as etapas de preparação e exploração de dados, são aplicadas técnicas de PLN, como a tokenização e remoção de *stopwords*, de modo a estruturar os dados para os modelos de Aprendizagem Automática. A fase de modelação consiste na implementação e otimização de algoritmos de Aprendizagem Automática e DL com o desenvolvimento de um caso prático de identificação de autoria na plataforma Twitter. Por fim, a fase de avaliação dos modelos será realizada com base em métricas de desempenho, onde o modelo Classificador Passivo Agressivo (PAC) foi identificado

como o mais eficaz, destacando-se pela sua precisão na distinção de estilos de escrita contribuindo de forma relevante para a melhoria do processo de identificação de autoria.

Palavras-chave: Identificação de Autoria, Processamento de Língua Natural (PLN), Aprendizagem Automática.

ABSTRACT

The continuous increase in the production and availability of textual content presents both challenges and opportunities for the field of Authorship Identification. The main objective of this dissertation is to develop a strategy that utilizes Natural Language Processing (NLP) methodologies and Machine Learning algorithms to identify the authorship of texts.

Authorship Identification consists of determining the identity of an author based on a specific text, relying on their unique linguistic and stylistic patterns. This plays a crucial role, not only in understanding writing styles but also in combating practices such as plagiarism. The study, framed as a supervised classification problem, uses data from the Twitter platform (currently known as X) to explore Authorship Identification strategies and build models capable of predicting the authorship of texts based on labeled data.

The study proposes an approach that utilizes NLP methodologies to transform texts into structured data, enabling the application of Machine Learning techniques to create models capable of identifying writing patterns and associating them with specific authors. To evaluate the models' effectiveness, traditional Machine Learning algorithms, such as Naive Bayes (NB) and Random Forest (RF), as well as advanced Deep Learning (DL) models, including Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), will be explored.

This dissertation adopts the KDD (Knowledge Discovery in Databases) methodology, starting with the definition of the problem and objectives related to Authorship Identification. During the data preparation and exploration stages, NLP techniques, such as tokenization and stopwords removal, are applied to structure the data for the Machine Learning models. The modeling phase consists of the implementation and optimization of Machine Learning and DL algorithms, with the development of a practical case of Authorship Identification on the Twitter platform. Finally, the model evaluation phase will be carried out based on performance metrics, where the Passive-Aggressive Classifier (PAC) model was identified as the most effective, standing out for its precision

in distinguishing writing styles and contributing significantly to the improvement of the Authorship Identification process.

Keywords: Authorship identification, Natural Language Processing (NLP), Machine Learning.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Abbreviations, Acronyms, and Initials

PLN	Processamento de Língua Natural
NLP	Natural Language Processing
ML	Machine Learning
DL	Deep Learning
NB	Naive Bayes
RF	Random Forest
SVM	Support Vector Machine
K-NN	K-Nearest Neighbors
LR	Logistic Regression
PAC	Passive-Aggressive Classifier
XGB	Extreme Gradient Boosting
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
CV	Count Vectorizer
TF-IDF	Term Frequency-Inverse Document Frequency
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer
BPE	Byte Pair Encoding

LSTM	Long Short -Term Memory Networks
POS	Part-of-speech
BOW	Bag-of-words
CV	Cross-validation
CA	Classification Accuracy
Avg CV	Average cross-validation
OvR	One-vs-Rest
KDD	Knowledge Discovery in Databases
NLTK	Natural Language Toolkit
ReLU	Rectified Linear Unit

INTRODUCTION

The continuous increase of digital content production over the years has presented new challenges and opportunities in the field of Authorship Identification. The vast amount of textual data shared online, whether in articles, emails, news or social media, raises concerns about the authenticity of the information.

Identifying the author of a given text is a well-researched problem in the field of Natural Language Processing (NLP), traditionally applied to literary works and academic papers. However, in the digital age, this challenge has evolved due to the rise of short and informal texts, particularly those found on social media platforms. Twitter (currently known as "X") serves as a prime example of such case, where its users usually communicate in brief, often fragmented messages.

The language used in social media is highly informal, frequently incorporating slang, abbreviations, hashtags, and irregular grammar. These characteristics make Authorship Identification in short-form texts significantly more complex compared to traditional written content(Suryaningrum, 2023).

This dissertation presents a supervised Machine Learning (ML) approach to Authorship Identification, determining the task as a supervised classification problem focused on predicting the author of a given text. The main objective is to develop a methodology that combines Machine Learning and Natural Language Processing (NLP) techniques to effectively identify the author of a text. Specifically, the objectives include analyzing the linguistic and stylistic challenges of Authorship Identification in short texts, evaluating the effectiveness of traditional ML algorithms such as Naïve Bayes (NB), Random Forest (RF), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Logistic Regression (LR), Passive-Aggressive Classifier (PAC), and Extreme Gradient Boosting (XGB) in authorship classification, exploring deep learning techniques, particularly Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to enhance the identification process, implementing NLP preprocessing techniques such as tokenization and stopword removal to improve model robustness, and finally comparing

the performance of ML and DL models based on their strengths and limitations(Singh et al., 2021; Khan et al., 2023).

The work developed in this dissertation follows the Knowledge Discovery in Databases (KDD) methodology, a widely adopted framework in data analysis (Fayyad et al., 1996). The study is structured into several phases, beginning with problem definition and data selection, where key challenges in Authorship Identification are determined, and pre-existing Twitter dataset, available on GitHub, is used. This is followed by data preprocessing and feature engineering, where NLP techniques such as tokenization, stopword removal and vectorization are applied.

The next stage involves model development, in which various Machine Learning and Deep Learning algorithms are tested using the Python programming language. Pre-implemented models from libraries such as scikit-learn and TensorFlow are employed, and hyperparameter tuning is carried in order to enhance their predictive performance.

Finally, evaluation is conducted, assessing model performance using performance metrics such as accuracy, precision, recall and F1-score in order to compare traditional and deep learning approaches.

This dissertation is structured into several chapters. The first chapter, "Authorship Identification" introduces the research problem and its significance. It provides an overview of modern applications, research developments, and challenges in authorship identification. As Stamatatos (2009) suggests, the application of automated techniques can enhance the efficiency of Authorship Identification. This chapter also explores the potential of enhancing authorship identification through machine learning automation and presents the research objectives of the study.

The second chapter, "Natural Language Processing for Authorship Identification" focuses on the core NLP techniques used in Authorship Identification. It covers essential preprocessing methods such as tokenization, stopword removal, noise reduction, stemming, lemmatization, and padding, emphasizing their role in preparing the data for the classification’s models at hand. Additionally, this chapter will also address the issue

of imbalanced data, discusses feature extraction methods, and delves into vectorization techniques, including word embeddings.

The third chapter, "Machine Learning for Authorship Identification" examines the various modeling approaches used in this study. It distinguishes between traditional machine learning algorithms, such as Naïve Bayes, Random Forest, Support Vector Machines, and Logistic Regression, and advanced deep learning models like Convolutional Neural Networks and Recurrent Neural Networks. This chapter also explores the integration of model evaluation and validation within the modeling process, focusing on the use of cross-validation and performance metrics to ensure robust model development.

The fourth chapter, "Case Study: Authorship Identification in Tweets," presents the application of the discussed techniques to a case study on Authorship Identification in short, informal text like tweets. It includes an in-depth exploratory data analysis, dataset preprocessing, and model preparation. The chapter details the modeling, creation, and evaluation of classification models, with a focus on model selection, training, and performance improvement. It also discusses model validation and its limitations, highlighting the most successful models in each of the experiments that will be conducted.

Finally, the dissertation concludes with a summary of the research findings, an assessment of the limitations of the study, and recommendations for future work in the field of authorship identification using Natural Language Processing and Machine Learning techniques. The final section, Conclusion and Future Work, integrates these aspects to provide a comprehensive closure to the research.

1 Authorship Identification

Authorship identification is the task of determining the author of a given text based on their unique writing style (Neal et al., 2018). This problem falls within the broader category of text classification and is typically approached as a supervised learning task (Talukdar et al., 2024). By leveraging Natural Language Processing (NLP) techniques and Machine Learning (ML) models, it is possible to analyze linguistic patterns and stylistic markers to attribute authorship with high accuracy (Stamatatos, 2009).

The origins of Authorship Identification can be traced back to when scholars used stylistic analysis to authenticate the authorship of classical works. This stylistic analysis involves the extraction of stylometric features (Ramnial et al., 2016), which are distinctive writing style markers that can help differentiate documents authored by different individuals.

Over the centuries, these efforts evolved into a sophisticated field of study, now significantly increased by computational advancements. The importance of authorship identification spans various domains, from literary scholarship and historical analysis to security, forensic linguistics, and even social media monitoring (Koppel et al., 2009).

Authorship Identification is commonly framed as a text classification problem, where the task is to classify a given text into one of several potential authors based on linguistic patterns, writing style, and other relevant features. In cases where the number of authors is limited to two, this becomes a binary classification problem (Iyer et al., 2019).

Modern approaches to the field have generated the incorporation of features such as the the words chosen and the stylistic way of writing, which make the task more accurate and robust. (Stamatatos, 2013).

Furthermore, while direct Authorship Identification remains the main goal, referring to the process of assigning a specific text directly to its author based on linguistic features, researchers may, in cases on uncertainty, turn to author profiling. This alternative approach focuses on categorizing an author based on their attributes such as the language chosen. (Villa-Cueva et al., 2022).

1.1 Modern Applications

Today, the field of Authorship Identification has become increasingly more significant in applications of both academic and practical contexts (Crockett, 2024).

Authorship Identification has expanded with the development of digital media to various fields (Koppel et al., 2009). Modern techniques now extend beyond traditional domains and are now being applied to areas such as music, for example, in recognizing musical styles and resolving disputes over authorship of a song, art and painting, plagiarism (Foltýnek et al., 2019), spam filtering (e.g., identifying unsolicited and virus-containing emails), and forensic investigations (e.g., identifying authors of anonymous or phishing emails) (Rashid et al., 2020).

There has also been a growing focus on analyzing software code to determine authorship by examining programming way of writing. This approach helps address issues like computer viruses, cyberattacks, unauthorized copying of code and plagiarism of software (Malik et al., 2021).

1.2 Research Developments

In recent years, Authorship Identification has witnessed significant advancements in its field, driven by the adoption of Deep Learning techniques. Traditional approaches relied on manual feature engineering combined with Machine Learning Algorithms, such as Random Forest (RF) or Support Vector Machine (SVM), focusing most of the time on lexical and character-level features to capture aspects of an author´s writing style (Stamatatos, 2009).

Subsequent research has explored the application of deep neural networks, mainly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNS) to this task of Authorship Identification. CNN´s have demonstrated the ability to effectively capture patterns both at the world and character level, while RNNs, particularly Long Short-Term Memory (LSTM) architectures, have shown strength in modeling sequential dependencies, that is, effectively remembering the order of previous words in a text to capture its contextual meaning, presents in text (Shrestha et al., 2017). These models also

have achieved notable results across various NLP tasks, including the most recent applications to authorship attribution and profiling (Villa-Cueva et al., 2022).

1.3 Challenges and limitations in Authorship identification

Authorship Identification is a critical task within the broader field of text mining, which involves the extraction of useful information and various patterns from large volumes of textual data. Text mining uses NLP to analyze and interpret textual content in order to identify the author of a given text. This process is increasingly important in today's digital age, where electronic communication has become a constant, and questions of authorship frequently arise in literary contexts.

One of the main challenges in Authorship Identification comes from issues related to ghostwriting and the authenticity of copyrighted literary works, where a text is authored by someone other than the current attributed author. A growing trend in recent years comes from the increase activity of student ghostwriters. As highlighted by Crockett (2024), ghostwriters often take deliberate steps to make their work appear student-written, with the aim of minimizing the changes of detection during the grade of the paper. This increasing professionalism in ghostwriting has introduced additional challenges in the tasks of Authorship Identification. Moreover, Crockett´s investigations suggest that is not uncommon for ghostwritten assignments to receive poor grades, suggesting that the quality of writing alone may not be a reliable factor in detecting ghostwriting.

Authorship Identification tasks are generally categorized into three types: author verification, closed-set attribution, and open-set attribution (Barlas & Stamatatos, 2020). Among these, open-set attribution, where the true author may not be included in the list of possible choices, presents a considerable challenge. This form of attribution is inherently more difficult than closed-set attribution or author verification, where the true author is either known to be within a limited set of candidates or is the only candidate being considered. While author verification and closed-set attribution have shown encouraging results, the same cannot be said for open-set attribution, which remains a challenging and less explored area.

Another significant challenge is the cross-domain attribution, where the training data and test data differ in various topics, leading to the issue cross-topic or cross-genre authorship attribution. This is due to the fact that stylistic features can differ dramatically between genres, as stated by Kestemont et al. (2018). In this context, a "genre" refers to different types or categories of written texts, such as fiction, non-fiction or poetry, or technical writing. These stylistic variations can complicate the task of Authorship Identification, making it difficult to accurately capture an author´s unique writing style.

Furthermore, another complex task comes from defining appropriate stylometric that can effectively capture the individual style of an author. This challenge comes from the necessity of selecting the right classification method which often needs to be customized for the case at hand. Barlas & Stamatatos (2020) emphasize that different texts may require different approaches, making the identification of the correct stylometric features and classification techniques a difficult task.

In summary, while significant progress has been made in Authorship Identification, this field still faces several persistent challenges. Issues such as ghostwriting make it more difficult to identify the true author by disguising their writing style. Additionally, cross-domain attribution introduces difficulties when the text may be different in genre or topic, affecting the consistency of the identification. The task is then further complicated by the need to define and select appropriate classification methods for each specific case. Furthermore, open-set attribution remains particularly challenging, as it involves situations where the true author may not be among the known candidates.

Addressing these challenges is a crucial step for improving the accuracy and reliability of the field of Authorship Identification.

1.4 Enhancing Authorship Identification through Machine Learning Automation

From the perspective Machine Learning field, Authorship Identification is approached as a text classification problem. This means that the main objective of the task is to assign a given document to predefined set of possible authors, relying on stylistic features such as the author´s way of writing (Ikonomakis et al., 2005). This involves not only recognizing

patterns in vocabulary but also identifying subtler stylistic nuances such as syntax, punctuation, and word usage.

The roots of authorship identification date back to the late 19th century, with pioneering work by Mendenhall (1887), who developed a stylometric method for determining authorship by analyzing basic textual features. Mendenhall proposed creating a curve that relates world length of a text to its frequency, which ultimately could be used to distinguish between authors.

In the 20th century, the field took a significant leap with the introduction of multivariate analysis methods. Mosteller and Wallace (1964) advanced this approach in their landmark study of the Federalist Papers.

Currently, the most widely adopted methods of Authorship Identification go to supervised Machine Learning Techniques, which seek to identify the distinctions between different classes in order to minimize classification errors. In this context, Machine Learning algorithms are trained under labeled datasets, which are then converted into numerical vectors that represent the stylist features of the text.

In recent years, deep learning methods, particularly neural networks, have gained significant attention in the field of Authorship Identification. These techniques, such as Convolutional Neural Networks (CNNs), have shown promising results in capturing more complex patterns in text. For example, Shrestha et al. (2017) applied a character-level CNN to short text Authorship Identification, achieving an accuracy of 76.1% when tested on a dataset of 1,000 tweets per author. This CNN model, with its convolutional and pooling layers, is able to capture local interactions between characters and learn patterns reflective of an author's writing style. However, like many deep learning models, the performance of CNNs tends to degrade when working with smaller datasets, as highlighted by Barz & Denzler (2019).

In addition to the evolution of techniques, the field has seen significant advancements in evaluation measures. Traditional metrics such as accuracy and precision are commonly used to assess model performance. Accuracy measures the proportion of correct predictions made by the model, while precision evaluates how many of the predicted

positive instances are actually correct. However, as datasets often present imbalances between classes, where some classes are more represented than others, alternative evaluation metrics have emerged to provide a more comprehensive view of the model effectiveness. These include, for example, the F1-score, which balances precision and recall and recall itself, which measures how many positive instances the model successfully predicted.

Overall, the progression of Authorship Identification methods demonstrates a continuous quality increase in the field´s ability to correctly identify the author of a text.

2 Natural Language Processing for Authorship identification

The identification and analysis of information in large collection of unlabeled and unstructured text written in natural language is a crucial task for information systems of any complexity and scale. In the context of Authorship Identification, Natural Language Processing (NLP) plays a fundamental role by transforming raw data into structured representations, allowing for the discovery of linguistic patterns specific to individual authors (Stamatatos, 2009). Furthermore, the use of stylometric features, which expose systematic patterns within an author´s writing style, is essential for this process. Various features have been proposed in the literature, including vocabulary richness measures, syntactic structures, function word frequencies, and character n-gram distributions (Yülüce & Dalkılıç, 2022).

This chapter introduces key NLP techniques utilized for text preprocessing and feature extraction. It begins with the discussion of tokenization, which is the process of segmenting text into meaningful units, or tokens. As highlighted by Mullen et al. (2018), tokenization is language-dependent and computationally demanding, yet fundamental for subsequent analysis.

Following this, the chapter addresses stopwords, with the objective of filtering out common words that carry irrelevant information for authorship tasks (Khalid, 2021). Next, the removal of noise including irrelevant characters and formatting inconsistencies, is also discussed. According to Al Sharou et al. (2021), noise significantly impacts model performance and must be carefully managed during preprocessing. The concepts of stemming and lemmatization are then explored. These techniques, essential for many text analysis applications, reduce words to their root or canonical forms, thus supporting vocabulary normalization (Abidin et al., 2024). The importance of padding techniques is also highlighted, particularly for machine learning models that require inputs of fixed size. Padding ensures that variable-length sequences are standardized, typically by appending zeros.

The chapter further examines the issue of data imbalance, which can introduce training-test discrepancies and reduce classification performance. As noted by Li et al. (2019),

imbalanced datasets may cause models to overly favor majority classes, necessitating specialized handling strategies.

Feature extraction is another critical component of the pipeline, with particular emphasis on character-level and word-level features. These features capture stylistic and structural nuances in the texts that are key to distinguishing authors (Stamatatos, 2009). Finally, the chapter discusses vectorization techniques, which are essential for converting text into numerical forms that Machine Learning algorithms can process.

By structuring the chapter this way, the main goal is to provide a comprehensive view of the preprocessing and feature extraction techniques that support the whole process of Authorship Identification.

2.1 Tokenization: Understanding tokenization in NLP

Tokenization is a fundamental step in NLP, that involves breaking down data into smaller units, such as words, subwords or characters. This will enable the machines to better interpret and process the textual data and is essential for transforming unstructured data into a format that can be used by computational models. Tokenization is a foundational step in natural language processing (NLP) tasks, bridging raw text and language models (Mullen et al., 2018).

Figure 1 shows an example of how tokenization transforms raw text into a series of tokens. The image serves to illustrate how the process works by breaking down a piece of text into smaller, manageable components. This tranformation allows for the free-form text to be turned into units that can be processed by the automated systems.

By converting text into tokens, it becomes more structured and ready for tasks such as stopword removal or feature extraction, which are necessary steps for preparing data.

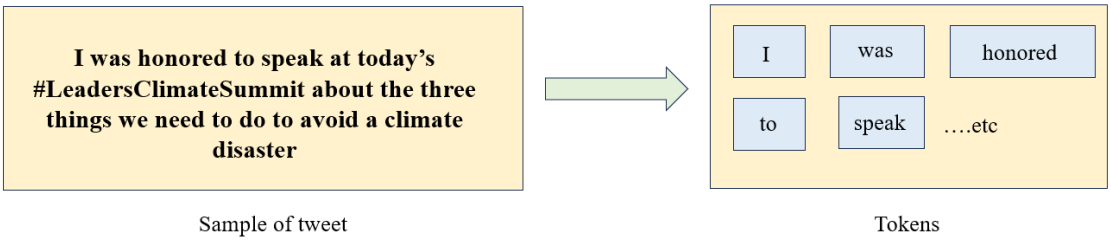

Figure 1 - Example of Sentence Tokenization

2.1.1 Types of Tokenization: Word, Subword and Character-Level

This subsection of the study presents an overview of the main tokenization techniques that are used in NLP, focusing on their specific characteristics for different types of textual data. By examining the various strategies, such as words, subword, and character-level tokenization, it becomes possible to understand how these choices can influence the overall structure of the input data, and ultimately the performance of the classification models. With this, the analysis conducted supports the interpretation of differences in model accuracy, highlighting the importance of selecting the appropriate tokenization method for a given context.

2.1.1.1 Word Tokenization

Work tokenization corresponds to the simplest form of tokenization where the data is split based on spaces.

For example, the sentence "Communities of color have been hit hard by COVID-19…" would be split into individual words like ["Communities", "of"," color"," have"," been"," hit"," hard"," by"," COVID-19"].

This type of method doesn't require any vocabulary training, as it simply separates words using whitespace characters.

2.1.1.2 Subword Tokenization

Subword tokenization, unlike traditional word-based tokenization, breaks the text into smaller units. This method is usefull for complex word structures and for handling out-of-vocabulary terms. Subword tokenization methods like BPE (Byte Pair Encoding) are

widely used in modern models, making it a suitable word segmentation strategy for neural network models (Sennrich et al., 2015).

For example, the sentence “Communities of color have been hit hard by COVID-19…” would be split into ["com", "mun", "ities", "of", "col", "or", "have", "been", "hit", "hard", "by", "COVID", "-", "19"].

2.1.1.3 Character Tokenization

Character tokenization is particularly effective for data where individual characters can convey significant meaning. By breaking down text into its constituent characters, models can better capture the nuances of language, especially in scenarios where word boundaries are not clearly defined. Ribeiro et al., (2018) demonstrated that character-level tokenization can outperform traditional word-level approaches in specific NLP tasks, such as dialog act recognition, by capturing morphological and orthographic patterns.

For example, the sentence “Communities of color have been hit hard by COVID-19…” would be split into ["C", "o", "m", "m", "u", "n", "i", "t", "i", "e", "s", " ", "o", "f", " ", "c", "o", "l", "o", "r", " ", "h", "a", "v", "e", " ", "b", "e", "e", "n", " ", "h", "i", "t", " ", "h", "a", "r", "d", " ", "b", "y", " ", "C", "O", "V", "I", "D", "-", "1", "9"].

To illustrate how different tokenization techniques segment text, table 1 presents the output of the three approaches previously mentioned, word, subword, and character-level tokenization, applied to the same sentence. Each method breaks the text into smaller units, affecting how the input is represented for machine learning models

Table 1 - The outputs of different tokenization methods

Method	Tokenized sentence
Word Tokenization	[“Communities”, “of”,” color”,” have”,” been”,” hit”,” hard”,” by”,” COVID-19”].

Subword Tokenization	["com", "mun", "ities", "of", "col", "or", "have", "been", "hit", "hard", "by", "COVID", "-", "19"]
Character Tokenization	["C", "o", "m", "m", "u", "n", "i", "t", "i", "e", "s", " ", "o", "f", " ", "c", "o", "l", "o", "r", " ", "h", "a", "v", "e", " ", "b", "e", "e", "n", " ", "h", "i", "t", " ", "h", "a", "r", "d", " ", "b", "y", " ", "C", "O", "V", "I", "D", "-", "1", "9"]

2.2 Stopwords: Filterning Non-Informative Words

This section of the work will explore the concept of stopwords in Natural Language Processing (NLP).

In Natural Language Processing, optimizing text data is a crucial step, with stopword removal being particularly significant. As Khalid. (2021) noted, the stopwords removal process involves checking the existence of predefined (and not allowed) words in the text. Stopwords are common words in language, such as “the,” “a,” “an,” and “in,” that need to be eliminated. This is because they occupy space in the database without adding meaningful value. Typically, stopwords carry less meaningful content compared to more substantive words, making their removal a crucial step in enhancing the overall performance of the models.

In Table 2, it is possible to see an example of stopword removal on a sentence. As shown, common function words such as pronouns, prepositions, and auxiliary verbs (e.g., “I”, “to”, “at”) are eliminated, leaving only the most important terms.

Table 2 - Example of Stopword removal

Sentence (With stopwords)	Sentence (Without stopwords)
Communities of color have been hit hard by COVID-19	Communities color hit hard COVID-19

I was honored to speak at today's #LeadersClimateSummit about the three things we need to do to avoid a climate disaster.	Honored speak today #LeadersClimateSummit three things need avoid climate disaster.

2.3 Noise removal

The term "noise" encompasses both harmful and meaningful types of nonstandard data. However, it is essential to differentiate between content that should be eliminated (harmful noise) and content that should be retained to enhance the system's performance (useful noise) (Sharou et al., 2021). Furthermore, there are specific situations where certain types of noise should be preserved, such as punctuation patters in sentiment analysis or emojis, as they can help maintain different sentiments.

Textual data from platfroms like social media often contains content that is considered non-standard, which poses challenges for NLP systems that are typically trained on cleaner data. As a result of this, these systems often struggle to process this content accurately. Studies like Hataya and Nakayama (2018) have investigated the impact of noise in machine learning. Rolnick et al. (2017) introduced a model designed to adapt to various patterns of artificial noise.

A key goal is going to be vocabulary standardization, which involves normalizing inconsistent casing by converting all text into lowercase. This is particularly important in informal writing, where informal writing often leads to irregular capitalization, and lowercasing the text is going to help the models to recognize and process words more effectively, improving overall task performance.

In addition to converting text to lowercase, various other techniques for noise removal are available. These include:

- Punctuation removal: Eliminating unnecessary characters that typically do not contribute meaning.

- Link removal: URLs generally don't provide meaningful information for Authorship Identification, so they should be treated as noise.
- Whitespace removal: Excessive spaces can cause formatting issues, and their removal ensures cleaner input for models.
- Hashtag removal: Hashtags are often used to categorize content rather than reflect an individual's unique writing style, so they can obscure the linguistic patterns necessary for identifying the author.
- Non-alphanumeric and special character removal: Special characters like @, $, or & are often used in social media but do not carry any valuable information for the task.

All these preprocessing steps are important to allow models to focus on the relevant content and linguistic patterns, ultimately enhancing their overall accuracy and performance in achieving the main goal of this work: correctly identifying the author based on textual data.

2.4 Stemming and Lemmatization

Stemming and lemmatization are preprocessing techniques that allow for the information retrieval system to have enhanced recalling capabilities. Their main purpose is to improve overall consistency on textual data by grouping together similar terms, which can result in a better model generalization and simplify analysis. Singh and Gupta (2017) discuss how stemming techniques are effective in handling sparse data.

Stemming algorithms work by removing suffixes from words allowing them to be reduced to their common “stem”. In contrast to this, the process of lemmatization employs resources in order to convert a word into its dictionary form. Unline stemming, lemmatization takes into accounts the context of the text and meaning of the words, ensure that different forms of the same word are groupe together. For example, stemming transforms the word “better” into “bett”, while lemmatizations identifies “better” as a comparative term of “good”, returning “good” as the lemma.

Table 3 demonstrates how different original words are converted into their basic forms through lemmatization.

Table 3 - Example of Lemmetization

Original Word	Lemmetized Word
Rocks	Rock
Corpora	Corpus
Better	Good

Table 4 demonstrates how stemming reduces words to their stem by removing common affixes. For instance, the words “Liked”, “Likes” or “Likely” are reduced to the root “Like”.

Table 4 - Example of Stemming

Original Word	Stemmed Word
Like	Likes / Liked / Likely
Chocolate	Chocolates / Chocolatey / Choco

Despite their purpose, the application of these techniques to languages like Portuguese introduced additional complexity. Unlike English, Portuguese consists of high gender variation and verb conjugation, which makes stemming and lemmatization more challenging and less accurate. For instance, the words *“bonito”* (masculine form of “beautiful”) and *“bonita”* (feminine form) share the same root but refer to different genders. Many normalization algorithms have limited effectiveness when applied to Portuguese texts, often failing to capture the linguistic subtleties of the language (Silva et al., 2023).

Furthermore, it is also crucial to differentiate between basic linguistic normalization and the deeper stylistic patterns that machine learning algorithms can identify. While these techniques help standardize text to further analysis, they can also remove important stylist features that are helpful in the task of Authorship Identification.

2.5 Padding: Handling variable-length input in NLP models

This section will explore the concept and importance of padding in NLP models, addressing the challenge of handling variable-length input sequences and ensuring consistent input dimensions for model processing.

Padding is a technique commonly used in Natural Language Processing (NLP) to standardize the length of text sequences, ensuring uniformity when working with models that require fixed-size inputs (Xu et al., 2023).

It involves adding extra elements, like zeros or special tokens, at the beginning or end of a sequence to make it match the required length. This allows for more efficient batch processing.

Figure 2 presents an example of this preprocessing technique, illustrating how padding is used to standardize the length of input sequences.

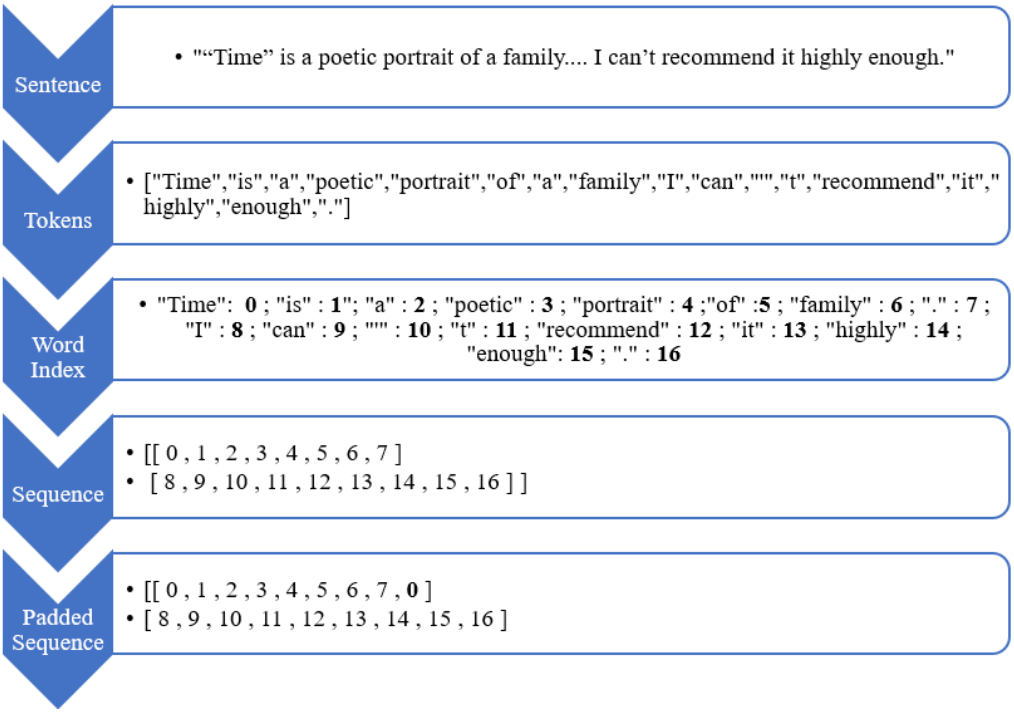

Figure 2 - Example of Padding

2.6 Handling Imbalanced Data

Imbalanced data is a common challenge in machine learning, where some classes of data are significantly underrepresented compared to others. This imbalanced data can lead to biased models that tend to favor majority classes, resulting in poor performance when predicting minority classes. In natural language processing (NLP), this problem is often

intensified in tasks like authorship identification, where certain authors may contribute far more text data than others (Li et al., 2019).

Section 2.6 focuses on the regularization techniques for handling imbalanced data in machine learning and NLP tasks. Feature and data regularization techniques can be used to improve model performance in imbalanced contexts. For instance, feature regularization methods like Term Frequency-Inverse Document Frequency (TF-IDF) help reduce the impact of overly frequent words that may be less informative, by assigning higher weights to terms are more distinctive across different classes. This enhances the model's ability to focus on meaningful linguistic differences rather than on repetitive or generic terms. On the other hand, data regularizations involves removing elements such as links, hashtags, special characters, and emojis, which may not meaningfully contribute to core linguistic patterns. By focusing on cleaner and more informative features, regularization helps models learn more representative patterns, improving their ability to generalize across different classes and reducing the impact of imbalanced data.

2.7 Feature extraction: Capturing author patterns and writing style

In Natural Language Processing (NLP), lexical feature extraction is a crucial step in converting unstructured textual data into structured representations that can be effectively processed by machine learning models. Lexical features focus on the words and characters used in the data, making them particularly relevant for authorship identification, as they capture key elements on an individual´s writing style (Stamatatos, 2009).

Lexical features can be divided into character-level and word-level features. Character-level features include patterns like punctuation usage and character sequence which are key features for analyzing the unique writing style of different authors. On the other hand, word-level features capture the vocabulary used by the author, reflecting their choice of words and topics they talk about. These word-level features can also reveal stylist choices, such as the use of frequent word combinations and play a crucial role in capturing

important stylistic or linguistic patterns associated with each word, providing valuable information for algorithms (Stamatatos, 2009; Koppel et al., 2009).

The choice between character-level and word-level features depends on the nature of the text and the problem at hand. For instance, word-level features are more suitable for tasks that involve longer, more formal text. On the other hand, as noted by Stamatatos (2009) character-level features are particularly effective for short and informal texts, as they better capture individual writing patterns in such contexts.

2.8 Vectorization: Converting text into numerical representations

Vectorization is a fundamental step in NLP, as it enables the transformation of textual data into numerical representations that can then be processed by the machine learning algorithms. This process involves converting tokens, such as words or characters, into vectors, allowing computation models to interpret and analyze language-based information. Among the most widely used vectorization techniques are CountVectorizer and Term Frequency-Inverse Document Frequency (TF-IDF). CountVectorizer creates feature vectors by counting the number of occurrences of each word in a document, while TF-IDF assigns weights based on the frequency of words relative to their importance

The primary objective of using these techniques is to capture both the meaning and structure of the data while preserving the contextual relationships between words. The vectorization process facilitates efficient analysis of large volumes of data, allowing for improved performance in the models used for authorship identification. As highlighted by Suryaningrum (2023), the choice of vectorization method significantly influences the performance of classification models, with approaches like TF-IDF and CountVectorizer offering different advantages in capturing term importance and frequency.

2.8.1 The importance of text vectorization in NLP

Text vectorization is essential in NLP as it transforms unstructured textual data into structured numerical representations, making it interpretable for machine learning models. By converting text into vectors, this process enables models to recognize patterns and meanings within the data.

One traditional technique for vectorization is the Bag-of-Words (BoW) model, which represents textual data as a collection of individual words. This model, implemented through CountVectorizer, treats each word independently, resulting in a sparse vector where the value for each word indicates its frequency within a document. While this approach is straightforward and effective for various tasks, it has significant limitations: it ignores the context in which words appear. For example, it treats phrases like "good" and "not good" as similar due to shared words, despite their differing meanings.

To enhance this limitation, the Term Frequency-Inverse Document Frequency (TF-IDF) method is utilized. This technique assigns different weights to words based on their frequency within the dataset, diminishing the influence of common, less significant words like "the" or "and" while emphasizing more important but infrequent terms.

However, despite its advantages, TF-IDF also has limitations. It fails to capture the contextual meaning of words and does not account for word semantics or the order in which words appear (Liu et al., 2018).

2.8.2 Word embeddings

Word embeddings are a technique in NLP that represent words as dense, continuous vectors in a high-dimensional space. Unlike traditional vectorization techniques such as Bag-of-Words or TF-IDF, which treat words as independent units and ignore semantic relationships, word embeddings capture relationships between words by grouping words that share similar meanings closer together in the vector space. This allows models to better interpret the meaning and usage of words based on their context.

Several methods have been developed to generate word embeddings. Early approaches include Word2Vec, introduced by Mikolov et al. (2013), which learns word representations by predicting a word based on its surrounding words, or vice versa. Another important method is GloVe (Global Vectors for Word Representation), developed by Pennington et al. (2014), which generates word embeddings by analyzing how often words appear together across large textual data. This enables the model to learn both the meaning of individual words and the overall patterns in the language.

3 Machine Learning for Authorship Identification

In the context of the field of Authorship Identification, Machine Learning algorithms play an important role by analyzing textual features within data and distinguishing between author´s writing styles. According to Yülüce and Dalkılıç (2022), with the development and widespread use of machine learning models over the last decade, machine learning-based approaches have become a promising solution for Authorship Identification.

Traditional algorithms, such as Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forests (RF), use statistical properties and features extraction techniques in order to classify authorship. On the other hand, deep learning techniques, such as neural networks, are capable of capturing complex linguistic patterns through automated representation learning, offering automatic feature selection, as stated by Hossain et al. (2021).

In order to assess the performance of these algorithms, it is important to evaluate them using cross-validation. This method splits data into several subsets, that helps the model to generalize new data and reduce the risk of overfitting, as highlighted by Moss, Leslie and Rayson (2018). Furthermore, the effectiveness of classification is quantified using evaluation metrics, including accuracy, precision, recall, and F1-score. These metrics allow for a comparison of the performance of different algorithms based on their predictive performance.

This section of the study explores the theoretical foundations of these algorithms. By understanding their mechanism, it is possible to better evaluate their effectiveness in tasks like Authorship Identification.

3.1 Machine Learning Modeling Techniques

This section provides a theoretical overview of the machine learning algorithms used in this study. Each algorithm is discussed in terms of its mechanisms and general application in Authorship Identification.

3.1.1 Naïve Bayes (NB)

Naïve Bayes is a simple, yet efficient probabilistic classification algorithm based on Bayes' theorem and is widely used for tasks like Authorship Identification. It works by calculating the probability of each class (in this case, authors) based on the features that were observed within the text, such as the frequency of words, character patterns and other linguistic characteristics. The algorithm then estimates the likelihood of each feature appearing within a specific class and then combines these probabilities to predict the most likely author. Despite its simplicity, Naïve Bayes can be highly effective and as stated by Kowsari et al. (2019), it can be computationally effective, requiring minimal memory for its task, making it a practical choice for many classification tasks.

3.1.2 Random Forest (RF)

Random Forest is an ensemble learning algorithm that boosts classification accuracy by building various decision trees instead of relying on just one. With this, each tree is then trained on a random subset of the data and a random subset of features, which helps reduce overfitting and improves the overall model's ability to generalize to new, unseen data. This makes Random Forest particularly useful for tasks such as Authorship Identification were picking up different in an author's writing is crucial. According to Alzamzami (2020), Random Forest have shown strong resistance to problems like noise and overfitting, as it is also capable of handling large datasets efficiently. Abbasi et al. (2022) emphasizes that ensemble learning techniques like Random Forest benefit from aggregating multiple decision trees to make more stable and accurate predictions.

Additionally, as noted by Kowsari (2019), Random Forest models are relatively quick to train on text datasets compared to deep learning models. However, making predictions with Random Forests can be slower because every tree needs to contribute its vote during inference.

In terms of Authorship Identification, the input data is typically text (e.g., tweets, articles, etc.). This text is first transformed into numerical representations, such as word vectors, which can then be fed into the Random Forest model. The decision trees in the forest

make predictions based on various linguistic features (e.g., word frequencies, character patterns, sentence structure).

Once all the trees have made their predictions, the final classification is determined by majority voting: the author predicted by the most trees becomes the final prediction (Figure 3).

By leveraging multiple decision trees and their collective decision-making process, Random Forest ensures higher accuracy and generalization when predicting the authorship of a given text.

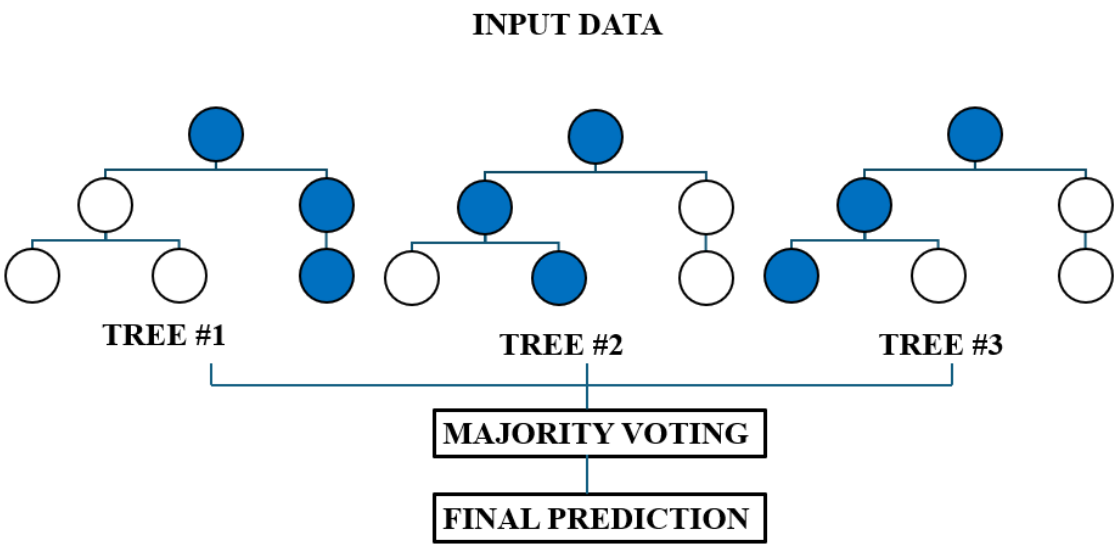

Figure 3 - Random Forest

3.1.3 Support vector machine (SVM)

Support Vector Machines (SVMs) are supervised learning models that analyze data for classification and regression tasks. The main goal is to create a line that can divide data into different groups. The goal is to position this line to be as far away as possible from the other data points of each group, which makes the models make more accurate prediction on new data (Yülüce & Dalkılıç, 2022).

SVM can use different techniques (kernels) to create complex decision boundaries when the data isn't easily separable in a straight line. (figure 4)

As highlighted by Alzamzami (2020), SVMs have consistently demonstrated robust performance in text classification tasks, making them a reliable choice for problems like Authorship Identification.

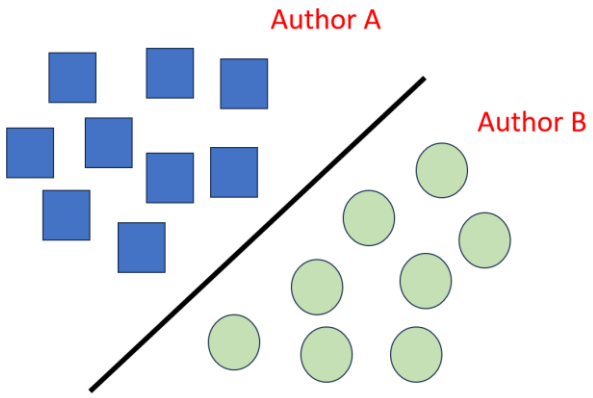

Figure 4 - Support vector machine

3.1.4 K-Nearest neighbors

K-Nearest Neighbors (k-NN) is widely applied within Natural Language Processing (NLP) for tasks such as text classification (Kowsari, 2019). It works by finding the "k" closest data points to a new data point and classifying the new data based on the majority of classes that are nearby. This method presents several advantages, including fast training and simplicity. It also requires little training time and is less sensitive to irrelevant and noisy features (Munazhif et al., 2023). Its simplicity and adaptability make it a good choice for working on smaller datasets. (Ghaeini, 2013).

3.1.5 Passive-Aggressive classifier (PAC)

The Passive-Aggressive Classifier is an online learning algorithm that remains "passively" when it makes correct predictions and "aggressive" when it wrongly predicts an outcome (Varada et al., 2022). After this, the model adapts by updating its initial parameters, creating a feedback loop that improves accuracy performance over time (figure 5).

While it might not be a go-to choice when it comes to working with smaller datasets, as evidenced by Varada (2022), the PAC works well on environments where new data is constantly being created.

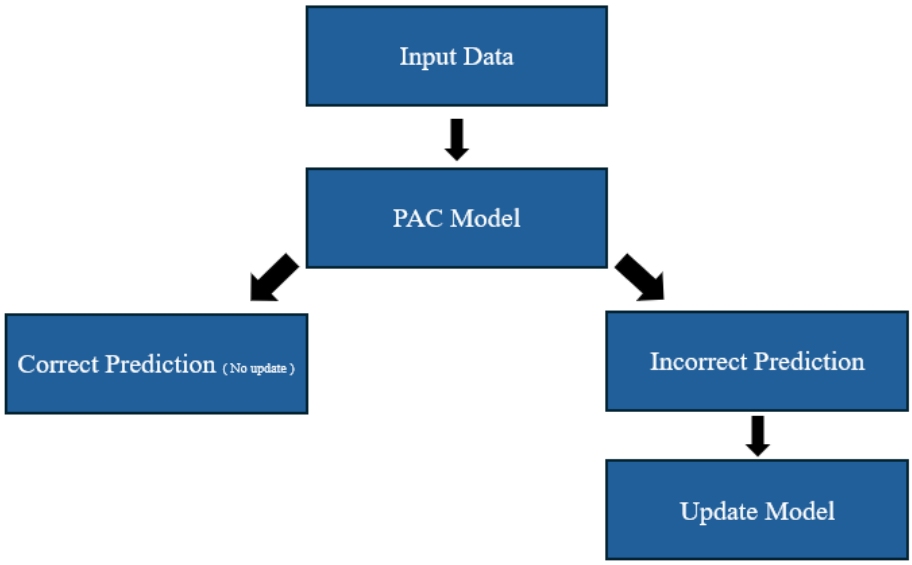

Figure 5 - Passive-Aggressive Classifier

3.1.6 Logistic Regression (LR)

Logistic Regression is a widely used supervised learning algorithm that analyzes the relationships between variables (Khalid ,2021). Although it was originally created with the intent of handling binary classification tasks, it can easily be adapted to handle multi-class problem through the One-vs-Rest (OvR) strategy, where a separate binary classifier is trained for each of the classes (Alhessi & Wicentowski, 2015). In this setup, the model selects the class (in this case, the author) with the highest predicted probability.

Due to its simplicity and effectiveness, Logistic Regression is determined as a good option for handling noisy and high-dimensional data. Its robustness, scalability to large datasets, and interpretability also make it a reliable choice for many NLP tasks (Shaik , 2023).

3.1.7 Extreme Gradient Boosting

Extreme Gradient Boosting is a powerful algorithm proposed by Chen and Guestrin and is designed for both classification and regression problems. As show in figure 6, XGBoost works by building an ensemble of decision trees, and each new tree aims to correct the mistakes made by the previous one (Fattahi, Mejri, & Ziadia, 2021). This process helps improve the model's accuracy and robustness making it a reliable choice for Authorship Identification.

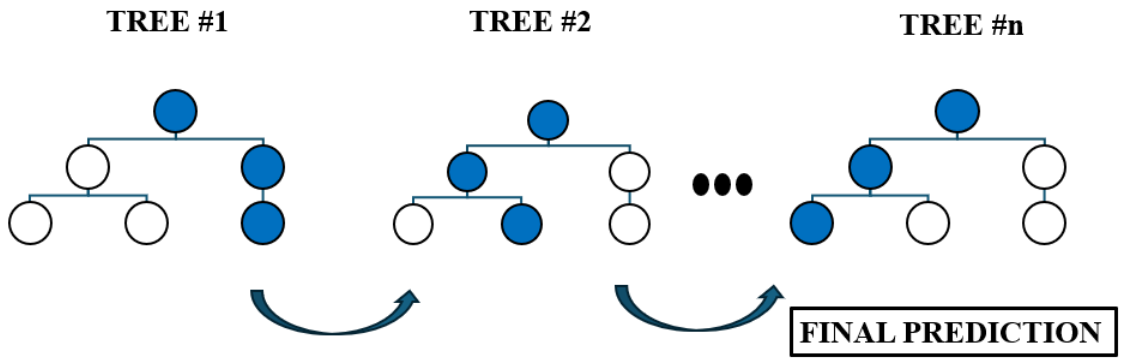

Figure 6 - Extreme Gradient Boosting

3.2 Deep Learning Modeling Techniques

This section will provide a brief overview of the deep learning models used in this study.

Deep learning models, as mentioned by Hossain (2021), are particularly effective for handling tasks like Authorship Identification due to their ability to automatically learn complex patterns from raw data. By leveraging multiple layers of neural networks, deep learning models can capture subtle relationships in language, which is beneficial for distinguishing between authors based on their writing style which is beneficial for distinguishing between authors based on their writing style, even in noisy and high-dimensional data.

3.2.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a type of artificial neural network that is specifically designed to identify patterns in sequential data, making them effective in the task of Authorship Identification. RNNS process input sequences by maintaining a memory of previous inputs that it received thought hidden states, allowing the use of past prediction to influence newer ones. As new data is processed, the hidden state is continuously updated, enabling the model to recognize dependencies and stylistic patterns across a text (Figure 7).

These characteristics make RNN a good option for handling tasks like language modeling and pattern recognition, however, the internal dynamic of RNNS, particularly how information is stored within the hidden states are complex and considered a “black box” (Ming et al., 2017). Word embeddings are also employed during training where words are connected into dense, continuous vector spaces, capturing similarities and relationships between words resulting in an improvement in the model´s ability to generalize across different contexts.

Among RNN variants, Long Short-Term Memory networks (LSTMs) are particularly effective where a structure is introduced compromised of input, output and forget gates. These mechanisms allow for the network to selectively retain or discard information, enabling for the capture of long-term dependencies that are critical for processing sequential data (Ghosh, 2016).

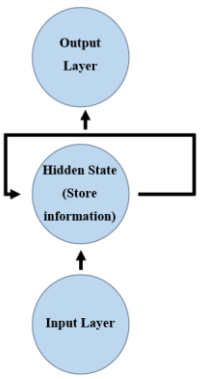

Figure 7 - Recurrent Neural Networks

3.2.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a type of neural network that consist of convolutional layers followed by pooling and fully connected layers, which enable the network to extract and refine relevant features from textual data (Tang, X., 2024). These convolutional layers apply filters within the network in order to capture author features, such as way of writing, while the pooling layers reduce dimensionality, allowing for the model to better focus on the most important and relevant patterns. These networks are then able to identify key patterns within word sequences, such as recurring phrases within text or used word combinations, which are important to distinguish between authors.

In classification tasks, such as multi-class Authorship Identification, CNNs often use loss functions like sparse categorical cross-entropy, which helps optimize the model to quantify the difference between the predicted and actual class distributions. This optimization allows for the models to reliably distinguish between authors based on the textual features it has learned (Palaniappan, 2023).

3.3 Model evaluation metrics

The performance of the selected algorithms can be evaluated using an assessment framework that incorporates multiple key metrics. These metrics are accuracy, recall, precision, F1-score and average cross-validation score and they serve to provide a comprehensive understanding of the model effectiveness in identifying the author of a text based on their writing style and patterns. Given the importance of each metric for this work, they will be explained in detail.

- Accuracy – Measures how often the model makes correct predictions out of all predictions made. It indicates whether the model is being trained effectively to distinguish the patters between authors.

$$Accuracy\ (\ CA\ ) = \frac{Correct\ predictions}{Total\ number\ of\ predictions}$$

For instance, if the model correctly identifies the author in 85 out of 100 instances, the accuracy would be 85%. However, while this provides a general overview of

performance, accuracy alone can be misleading, especially in cases of imbalanced datasets (Fernández , 2018). In such datasets, certain authors may have significantly more data, giving the model more information about those authors and potentially leading to an inflated accuracy score.

- Recall - Represents the percentage of data correctly attributed to a specific author out of all inputs that belong to that author.

$$Recall = \frac{True\ positive\ predictions}{Total\ Positive + False\ Negative\ (\ All\ of\ predicted\ Results\ )}$$

For example, when predicting data written by Author X, the model correctly identifies 80 tweets but misses 20. This indicates that the model successfully captured 80% of Author X's texts while failing to identify the remaining 20%.

- Precision – Evaluates how many of the identified textual data instances truly belong to the assigned author.

$$Precision = \frac{True\ positive\ predictions}{Total\ Positive + False\ Positive\ (\ Actual\ Results)}$$

For example, the model identifies 40 texts belonging to that author, but only 30 of them are actually correct, the precision would be 30/40 (75%). A high precision ensures that the model avoids falsely attributing data to the wrong author.

- The F1-score - It calculates the balance between precision and recall, providing a comprehensive measure of the model's performance.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

For example, if the precision is 90% and recall is 80%, the F1-score would be approximately 84.2%. This metric ensures that neither precision nor recall dominates the evaluation.

- The Average Cross-validation Score - Calculates the mean performance of the model across multiple data splits, providing a comprehensive measure of the model's generalization ability.

$$Avg\ CV = \frac{Sum\ of\ Cross\ Validation\ Scores\ for\ Each\ Fold}{Number\ of\ Folds}$$

3.4 Cross-validation: Ensuring robust model evaluation

As stated by Moss, Leslie & Rayson (2018) training an algorithm and evaluating its statistical performance on the same dataset can lead to overly optimistic results. Cross-validation (CV) was created in order to address this issue by testing the model´s performance on new, unseen data, offering a more accurate estimate of its performance. There are many scenarios where the available data is limited, prompting the need to split the dataset into training samples while the remaining portion is used to evaluate the algorithm performance.

One popular cross-validation technique, particularly useful in Natural Language Processing is K-Fold Cross-Validation. As described by Moss (2018) and Chungku et al., (2010), in this method, the dataset is divided into K equal parts. The model is then trained on K-1 parts, and the part that remains is used to test the model. This process is repeated K times, with each part being used for testing once. The average result of these testes will give a better idea of how the model will perform on new unseen data, helping to avoid problem such as overfitting.

4 Case Study: Authorship Identification in Tweets

After discussing the theoretical foundations of Natural Language Processing (NLP) and Machine Learning, this section shifts to the practical aspects of the study. It provides an overview of the dataset collected, compromising of tweets, which are usually short and informal texts rich in unique linguistic features such as slang and abbreviations. The main goal of this case study consists in developing and fine-tuning the most robust models capable of correctly identifying authorship.

Additionally, this section will discuss the preprocessing steps applied in the dataset, which are essential to ensure that the selected models are applied to clean data, maximizing their effectiveness in the task at hand.

4.1 Exploratory data analysis

The dataset used in this study was sourced from GitHub (https://github.com/vegarrsm/twitterAuthorIdentification/blob/master/authors.tsv) , consisting of a collection of tweets along with their corresponding authors. The dataset includes a diverse group of individuals, such as celebrities, politicians, and renowned soccer players, with data collected over the course of their Twitter activity and despite the lack of information regarding the time period of the data collected, the dataset deemed suitable for this study due to its diversity of authors and textual content. The content varies across different topics, ranging from politics and trending events to religion, providing a broad spectrum of linguistic styles. The practical work in this dissertation aims to analyze distinctive writing patters and linguistic features present in these tweets.

4.1.1 Dataset

The dataset comprises 13,515 entries attributed to 15 distinct authors. Particularly, there are no rows with null values, indicating that each tweet is fully written. Table 5 presents a detailed summary for each author, including the number of samples, average word count per tweet, maximum word count per tweet, and maximum character count per tweet. These metrics provide insight into the length and structure of the tweets across different authors.

Table 5 - Author dataset summary

Author Name	**Number of Samples**	**Average Word Count**	**Max Word Count**	**Max Character Count**
Ariana Grande	901	7	56	281
Barack Obama	901	26	57	304
Bill Gates	901	27	49	283
CNN	901	29	50	280
Cristiano Ronaldo	901	9	55	281
Kim Kardashian	901	14	59	291
Youtube	901	11	47	281
Britney Spears	901	12	62	282
Demi Lovato	901	13	57	286
Jimmy Fallon	901	12	55	283
Justin Bieber	901	7	61	281
Katy Perry	901	16	56	289
Neymar	901	7	49	288
Rhianna	901	12	55	292
Shakira	901	16	53	298

It is possible to observe that each author contributes an equal number of samples, ensuring a balanced dataset for Authorship Identification. Additionally, the analysis reveals that the CNN author has the highest average word count per tweet, while Britney Spears has the highest maximum word count, and Barack Obama holds the record for the highest maximum character count per tweet.

To further visualize the unique writing style that each author possesses in the data collected, word clouds were generated for each one in figure 8 and 9. These highlight the most frequently used words of each author, offering a view of the distinct linguistic patters

and vocabulary preferences used in their tweets. The code used for generating these visualizations is provided in Appendix 12.

Figure 8 - Wordcloud Combination for Different Authors (part 1)

Figure 9 - Wordcloud Combination for Different Authors (part 2)

4.1.2 Dataset preprocessing and model preparation

Once the dataset was explored and its structure understood, the next crucial step involved preparing the data for model training. To ensure that the models could operate effectively with clean data, a series of preprocessing techniques, previously described in Sections 2.1, 2.2, 2.3, and 2.6, were applied. The following steps will help reduce noise and standardize the text, particularly the removal of elements that, while not traditionally

classified as stopwords, allow for the models to work more effectively with meaningful linguistic features:

1. Removed unnecessary elements such as links.
2. Remove extra whitespaces.
3. Remove #.
4. Remove punctuation.
5. Remove special characters.
6. Remove numeric values.
7. The text is all converted to lowercase.
8. Remove non-alphanumeric characters.
9. Remove @.

These data transformations are crucial for ensuring consistency and eliminating irrelevant information, which significantly enhances the performance of the models. (Sami et al., 2021)

4.2 Modeling of Classification Models for Authorship Identification

This section of the study presents the modeling process, which consists of algorithm selection, model creation, model evaluation, and model selection. It begins with the choice of suitable algorithms for the task of identifying the author of a written tweet. Subsequently, the construction and training of the models are described, covering data processing techniques, feature extraction, and model tuning. Following this, the models are evaluated using relevant metrics to assess their performance. Finally, based on the evaluation results, the best-performing model is chosen.

4.2.1 Algorithm Selection

The selection of the classification algorithms is a crucial step in the modeling process for Authorship Identification. Various Machine Learning and Deep Learning models were considered, with attention given to their parameters in order to generate the best results.

Several of the traditional Machine Learning algorithms were implemented using the Scikit-learn library. Naïve Bayes was constructed utilizing the MultinomialNB() function. Logistic Regression utilized the LogisticRegression() function where parameters such as max_iter, which specifies the maximum number of iterations for the solver to converge, were considered. For Support Vector Machine (SVM), the SVC() function was used, considering important parameters such as kernel (which determines the type of kernel used for transformation, e.g. “linear”, ”rbf”)

Random Forest was built using the function RandomForestClassifier() , where the number of trees created is controlled by the parameter n_estimators. In order to ensure consistency in the results across different runs, the random_state parameter was set. This means that the selection of features and data points for each tree follows the same sequence, producing identical results. The max_depth parameter is also considered to control the depth of the trees.

For the K-Nearest Neighbors (KNN), the KNeighborsClassifier() function was employed, with the n_neighbors parameter specifying the number of neighbors used in the classification and weights parameter determining the weight function used in prediction. Additionally, XGBoost (Extreme Gradient Boosting) was incorporated using the XGBClassifier() function from the XGBoost library, with key parameters such as learning_rate (contribution of each individual tree to the final prediction), n_estimators (number of trees created) and max_depth (number of layers for each individual tree) being considered.

Regarding the Deep Learning models, Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) units was developed using the LSTM() and Dense() layers from the TensorFlow/Keras framework. The most important parameters considered

included units (which control the number of LSTM units) and dropout (which helps cases of overfitting by randomly setting input units to zero during training).

Similarly, a Convolutional Neural Network (CNN) was constructed utilizing the Conv1D() function, where key parameters including filters (the number of output filters in the convolution), kernel_size (the length of the 1D convolution window), and activation (the activation function applied) were considered.

Each algorithm´s parameters were chosen in order to tune for optimized performance. With this, experiments were conducted in order to compare the effectiveness of these algorithms before moving to the evaluation phase, where their performance is being analyzed using performance metrics.

4.2.2 Model Creation

The model creation phase involved applying the selected algorithms to the preprocessed twitter dataset and training them accordingly. In this stage, various preprocessing techniques are applied in order to ensure consistency in the textual data. These techniques were applied using TensorFlow/Keras for tokenization and sequence padding, and NLTK for stopword removal.

The models were implemented in python (version 3.11.11) using Google Colaboratory as the environment. Traditional Machine Learning models were built using the Scikit-learn library, while deep learning models utilized the TensorFlow/Keras framework.

An 80/20 train-test split was used for all algorithms, with 10-fold cross-validation employed in order to evaluate the model stability and generalization. Naïve Bayes was applied with default parameters and a random_state of 42 to ensure reproducibility. Random Forest was evaluated using various parameters configurations in order to assess its performance on different text representation techniques, such as TF-IDF and CountVectorizer. The Support Vector Machine, implemented with the SVC function, was used for its ability to work well in high-dimensional spaces. Key parameters such as the kernel (set to RBF) and the regularization parameter were adjusted during model tuning. K-Nearest Neighbour was considered using 10 neighbors and the weight parameter

selected as “distance” resulting in giving more influence to neighbors that are closer. Additionally, during tokenization for KNN, a maximum of 10,000 words was allowed to limit vocabulary size.

The Passive-Aggressive Classifier was applied using defaults parameters and a random_state of 42. Logistic Regression was utilized with an increased iteration limit (max_iter = 1000) to ensure the model had enough time to converge. The XGBoost model was applied with the max_features function to restrict the number of features to 10,000, using default settings and a random_state of 42 to ensure generalization across the model creation process.

Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) units was built using the LSTM and Dense layers from the TensorFlow/Keras framework. The model was trained using the Sparse Categorical Cross Entropy loss function with the Adam optimizer and dropout regularization was applied at the rate of 0.2 in order to mitigate overfitting. Tokenization for both the RNN and CNN model allowed a maximum of 10,000 words with a maximum sequence length of 100 tokens in order to ensure optimal computational efficiency. When it comes to Convolutional Neural Network (CNN), it was applied using Conv1D layers in order to detect local patterns within the data. Its architectures consisted of convolutional layers with 128 filters of size 5 with the activation “relu”. A max pooling layer was also used in order to reduce the dimensionality of data, followed by a Global Max Pooling layer to retain the most significant features. A final dense layer used softmax activation to produce probability distribution over all the potential authors. Like the RNN, the CNN was trained with the Sparse Categorical Cross Entropy and the Adam optimizer. Both Deep Learning models were evaluated using a 10-fold cross-validation and an 80-20 split. The classification performance of these Machine Learning and Deep Learning models were evaluated using performance metrics under the effects of different text representations, such as TF-IDF and CountVectorizer. Character-level and word-level feature extraction methods were also explored to assess their impact on classification accuracy.

By applying these algorithms and tuning their respective parameters, the models were trained and evaluated to identify the most effective approach for Authorship Identification from Twitter data.

4.3 Experiments and Model Improvement

This section of the study focuses on presenting and analyzing the results that were generated from the application of both Machine learning (ML) and Deep Learning (DL) algorithms. The main goal was to evaluate the performance of TF-IDF and Count Vectorizer feature extraction techniques when applied to the selected ML models, as well as to assess the performance of the chosen DL models in the context of Authorship Identification.

The decision to use these approaches originates from their proven effectiveness in Natural Language Processing (NLP) tasks, particularly in dealing with short and informal text, such as tweets (Feroze, Feroze, & Abbasi, 2024)

All models were applied to preprocessed data, which involved cleaning the dataset by removing links, hashtags, and other irrelevant elements. The baseline results, summarizing the initial model performances, are presented in Table 6 and 7, with further detailed results for each experiment outlined in subsequent sections.

The programming code used to generate the baseline results for ML and DL models can be found in Appendix 1 and Appendix 2, respectively. The code corresponding to the additional experimental scenarios is provided in Appendices 3 to 11, with each appendix aligned to a specific experiment.

A total of five additional experiments were conducted following the baseline analysis. These experiments were designed to assess the overall performance of the models under various scenarios, providing deeper insights into the dataset and the challenges of Authorship Identification in NLP. Each experiment is justified in its respective section. The experiments included:

1. Word-Level vs. Character-Level Classification: Comparing model performance when features are extracted at the word level versus the character level.

2. 10-Fold Cross-Validation vs. 15-Fold Cross-Validation: Evaluating whether increasing the number of folds impacts the robustness of the model evaluation.

3. Impact of Stopwords: Assessing model performance with and without stopword removal to understand the influence of non-informative words.

4. Excluding a Major Author: Removing tweets from the Portuguese author Cristiano Ronaldo to analyze the effect of class imbalance on classification.

5. Training-Test Split Variations: Testing a 70/30 train-test split as an alternative to the baseline 80/20 split to observe changes in model performance.

Each of these experiments provided insights into different aspects of the dataset at hand and the behavior it holds under varying conditions. Ultimately, the model with the overall performance will be selected as the most suitable for this task.

4.3.1 Evaluation of the models Created

The following tables summarize the results obtained using the baseline configuration: 10-fold cross-validation, all authors included, stopwords removed, and an 80/20 train-test split. These results serve as a foundation for comparison with the subsequent experimental outcomes.

4.3.1.1 Machine learning models

Table 6 - ML results - Baseline

Model		**CA**	**F1**	**Precision**	**Recall**	**Avg Cv**
Naive Bayes	TF-IDF	0.66	0.67	0.71	0.67	0.66
	Count Vec	0.65	0.65	0.69	0.65	0.64
Random Forest	TF-IDF	0.60	0.61	0.66	0.59	0.60
	Count Vec	0.58	0.60	0.67	0.58	0.58

Support Vector Machine	TF-IDF	0.66	0.67	0.71	0.66	0.65
	Count Vec	0.50	0.54	0.69	0.50	0.50
K-Nearest Neighbours	TF-IDF	0.49	0.50	0.54	0.49	0.49
	Count Vec	0.46	0.47	0.51	0.46	0.46
Passive-Aggressive Classifier	TF-IDF	0.67	0.67	0.70	0.67	0.67
	Count Vec	0.61	0.62	0.65	0.61	0.59
Logistic Regression	TF-IDF	0.65	0.66	0.70	0.65	0.65
	Count Vec	0.64	0.66	0.70	0.64	0.64
Extreme Gradient Boosting	TF-IDF	0.56	0.58	0.64	0.56	0.56
	Count Vec	0.59	0.61	0.67	0.59	0.59

As it is possible to observe, the Passive-Aggressive Classifier (PAC) paired with TF-IDF vectorization was able to generate the best performance amongst all the models tested, achieving a classification accuracy of 67%, and F1 score of 0.67, a precision of 0.70 and recall value of 0.67.

While the Naïve Bayes and Support Vector Machine (SVM) models, were able to performance well with TF-IDF, both achieving scores of 66% accuracy and F1 scores of 0.67, PAC slightly outperformed these models in terms of recall and average cross-validation values. On the other hand, models that used CountVectorizer were generally underperforming. This suggests that TF-IDF, with the purpose of giving more weight to terms that matter most within text, is better at capturing more relevant features within data that can help the task of Authorship Identification.

4.3.1.2 Deep learning models

Table 7 - DL results - Baseline

Model	CA	F1	Precision	Recall	Avg CV
RNN	0.66	0.67	0.69	0.66	0.66
CNN	0.54	0.55	0.61	0.54	0.51

In the case of the Deep Learning models, the Recurrent Neural Network (RNN) was able to outperform the Convolution Neural Network (CNN), with a classification accuracy of 66%, an F1 score of 0.67 and a recall of 0.66. These results showcase that RNNs are better at being able to handle sequential data, making them valuable in tasks where the order of words is important.

4.3.1.3 Model selection

After analyzing the results obtained, it is possible to assert that the PAC model, combined with TF-IDF vectorization, was able to generate the best performance. Its competitive accuracy and precision highlight its ability to correctly identify the author of the tweets.

While the RNN model also demonstrated strong metrics, PAC was ultimately chosen due to its faster training time. This supports the claim that PAC is a practical and efficient choice for tasks with limited computational resources.

4.3.2 Model Performance Improvement

4.3.2.1 Character Level vs Word Level

The results showcased in these subsections focus on character-level representation of the textual data. With this, the experiment was conducted with the same setup as the baseline, including a 10-fold cross-validation, all authors included and an 80-20 train-test split. Since the analysis is performed at the character level, stopwords were retained during preprocessing.

It is important to mention that deep learning models were not applied in this case, as character-level features require significant input sizes. This means that, since this experiment had limited resources, using deep learning would not have been practical or efficient. Additionally, Extreme Gradient Boosting (XGBoost) posed considerable computational challenges when applied. So, to mitigate this, the max_features parameter was changed to 10,000, ensuring that the experiment could be completed within a reasonable timeframe.

Table 8 - ML results - Character level vs Word Level

Model		**CA**	**F1**	**Precision**	**Recall**	**Avg Cv**
Naive Bayes	TF-IDF	0.55	0.56	0.69	0.55	0.54
	Count Vec	0.62	0.62	0.69	0.62	0.60
Random Forest	TF-IDF	0.62	0.62	0.65	0.62	0.62
	Count Vec	0.61	0.62	0.65	0.61	0.62
Support Vector Machine	TF-IDF	0.68	0.69	0.74	0.68	0.70
	Count Vec	0.58	0.60	0.69	0.58	0.57
K-Nearest Neighbours	TF-IDF	0.58	0.58	0.60	0.58	0.58
	Count Vec	0.53	0.53	0.55	0.53	0.53
Passive-Aggressive Classifier	TF-IDF	0.72	0.72	0.73	0.72	0.71
	Count Vec	0.67	0.67	0.68	0.67	0.66
Logistic Regression	TF-IDF	0.68	0.68	0.70	0.68	0.69
	Count Vec	0.67	0.67	0.69	0.67	0.66

Extreme Gradient Boosting	TF-IDF	0.62	0.63	0.64	0.62	0.62
	Count Vec	0.63	0.64	0.67	0.63	0.62

The results obtained from this character-level representation highlight notable differences when compared to the word-level baseline. The PAC with TF-IDF stood out as the best-performing model, achieving a classification accuracy of 72%, an F1 score of 0.72, and an average cross-validation score of 0.71. This marks a significant improvement over the values obtained in the word-level configuration, which resulted in 67% accuracy and a cross-validation score of 0.67.

Similarly, the SVM with TF-IDF showed a slight increase in accuracy and F1 score, going from 66% to 68% and from 0.67 to 0.69, respectively. The Naïve Bayes model, on the other hand, displayed a decrease in accuracy, dropping from 66% to 55%.

For other models, performance remained relatively consistent across both character-level and word-level representations, with only minimal differences in accuracy and other metrics.

Overall, these findings reinforce the strengths of the PAC with TF-IDF, which outperformed other models across both feature representations. Its adaptability and results make it the most reliable choice for this task.

4.3.2.2 15-fold cross validation

The next evaluation conducted assesses model performance using a 15-fold cross-validation approach instead of the 10-fold configuration used in the baseline. This adjustment was made to determine if increasing the number of folds would generate better and more consistent performance metrics. While a 5-fold cross-validation was considered, it was ultimately avoided as it would reduce the amount of training data used in each fold, leading to less reliable results.

The original data preprocessing setup was maintained throughout this experiment, including stopword removal, the inclusion of all authors, and the 80/20 train-test split.

Table 9 - ML results - Experiment 2 - 15-fold cross validation

Model		**CA**	**F1**	**Precision**	**Recall**	**Avg Cv**
Naive Bayes	TF-IDF	0.67	0.67	0.71	0.67	0.66
	Count Vec	0.65	0.65	0.69	0.65	0.65
Random Forest	TF-IDF	0.60	0.61	0.66	0.59	0.60
	Count Vec	0.58	0.60	0.67	0.58	0.58
Support Vector Machine	TF-IDF	0.66	0.67	0.71	0.66	0.66
	Count Vec	0.50	0.54	0.69	0.50	0.51
K-Nearest Neighbours	TF-IDF	0.49	0.50	0.54	0.49	0.49
	Count Vec	0.46	0.47	0.51	0.46	0.46
Passive-Aggressive Classifier	TF-IDF	0.67	0.67	0.70	0.67	0.67
	Count Vec	0.61	0.62	0.65	0.61	0.60
Logistic Regression	TF-IDF	0.65	0.66	0.70	0.65	0.66
	Count Vec	0.64	0.66	0.70	0.64	0.64
Extreme Gradient Boosting	TF-IDF	0.58	0.62	0.70	0.58	0.59
	Count Vec	0.58	0.63	0.74	0.58	0.59

Table 10 - DL results - 15-fold cross validation

Model	CA	F1	Precision	Recall	Avg CV
RNN	0.66	0.67	0.68	0.66	0.66
CNN	0.52	0.53	0.59	0.52	0.52

The results obtained from this experiment provide some insights into how increasing the number of folds impacts the model's performance. Overall, the results showcase consistency with those from the 10-fold cross-validation, indicating that the addition of folds does not impact model performance but provides more reliable and stable metrics. This is particularly evident in models such as Naïve Bayes (NB), PAC, and SVM, which showed minimal variations in the performance metrics. For instance, Naïve Bayes with TF-IDF achieved an F1 score of 67% in both 10-fold and 15-fold cross-validation. Similarly, PAC with TF-IDF maintained a consistent 67% accuracy and F1 score of 0.67, reinforcing its robustness across different cross-validation configurations. The SVM with TF-IDF also demonstrated stable performance, maintaining an accuracy of 66%, suggesting that increasing the number of folds had little impact on its effectiveness.

In the case of Deep Learning models, the increase in folds did not lead to any significant performance improvements. The RNN maintained its 66% accuracy and an F1 score of 0.67, while the CNN continued to underperform with an accuracy of 52%, indicating that the number of folds had little effect on these models' overall outcomes.

While the additional folds were able to generate more stable metric performance, the results indicate that the changes were not transformative for most models. Therefore, the computational cost of increasing cross-validation folds by 5 suggests that maintaining the default 10-fold configuration remains a more efficient choice for evaluating these models.

Ultimately, the results analyzed showcase that PAC remains the leading model, demonstrating consistent results across the different cross-validation setups, maintaining an accuracy score of 67% with TF-IDF and an F1 score of 0.67.

4.3.2.3 No stopword removal

The next assessment evaluates the impact of retaining stopwords during the preprocessing of data. This analysis investigates whether removing commonly used and non-informative words affects the overall performance of the chosen models. In the context of the Twitter dataset, stopwords may carry stylistic information on the way authors write, such as their tendency to use specific filler words.

This experiment is especially relevant for the TF-IDF and CountVectorizer methods, as both approaches are sensitive to word frequency. The other preprocessing setups remain consistent with the baseline experiment, which includes including all authors, employing a 10-fold cross-validation, and using an 80/20 train-test split.

Table 11 - ML results - Experiment 3 - No stopword removal

Model		**CA**	**F1**	**Precision**	**Recall**	**Avg Cv**
Naive Bayes	TF-IDF	0.63	0.63	0.69	0.63	0.62
	Count Vec	0.60	0.60	0.68	0.60	0.59
Random Forest	TF-IDF	0.59	0.60	0.64	0.59	0.58
	Count Vec	0.59	0.60	0.64	0.59	0.59
Support Vector Machine	TF-IDF	0.64	0.65	0.69	0.64	0.64
	Count Vec	0.50	0.52	0.64	0.50	0.50
K-Nearest Neighbours	TF-IDF	0.50	0.51	0.54	0.50	0.50
	Count Vec	0.44	0.44	0.47	0.44	0.43
Passive-Aggressive Classifier	TF-IDF	0.68	0.69	0.71	0.68	0.69
	Count Vec	0.63	0.64	0.66	0.63	0.63

Logistic Regression	TF-IDF	0.65	0.66	0.70	0.65	0.64
	Count Vec	0.65	0.67	0.70	0.65	0.65
Extreme Gradient Boosting	TF-IDF	0.60	0.63	0.69	0.61	0.61
	Count Vec	0.60	0.62	0.70	0.60	0.60

Table 12 - DL results - No stopword removal

Model	**CA**	**F1**	**Precision**	**Recall**	**Avg CV**
RNN	0.67	0.67	0.69	0.67	0.65
CNN	0.54	0.55	0.59	0.54	0.52

Upon analyzing the results, it is possible to observe slight changes and consistent patterns compared to the baseline experiment, where stopwords were removed.

The Naïve Bayes (NB) model with TF-IDF generated an accuracy of 63%, similar to the baseline, while CountVectorizer achieved 60%, showcasing minimal change. PAC also achieved similar results to its baseline performance, and SVM with TF-IDF showed a slight decrease in accuracy, going from 66% to 64%.

Logistic Regression performed similarly with both TF-IDF and CountVectorizer, with a slight improvement in the CountVectorizer case, reaching 65% accuracy, compared to the baseline's 64%. Deep Learning models like RNN and CNN performed as expected, with the RNN achieving 67% accuracy and 0.67 F1 score, similar to the baseline. CNN continued to underperform, with an accuracy of 54%.

Ultimately, the inclusion of stopwords does not drastically alter the performance of most models, even offering slight improvements in some cases. The PAC model with TF-IDF once again remained the most effective, regardless of the preprocessing choice.

4.3.2.4 Without the author "Cristiano Ronaldo"

The next experiment conducted evaluates the performance of the models after excluding the Portuguese author Cristiano Ronaldo from the dataset.

Ronaldo's writing style, consisting of tweets written in the Portuguese language, may have unique characteristics that contribute to the overall classification task. By removing this author, it is possible to analyze whether his inclusion had a significant impact on the model's ability to identify the authors.

This experiment will be conducted using the same preprocessing steps as the baseline, with all features retained, a 10-fold cross-validation, and an 80-20 train-test split.

Table 13 - ML results - Without the author "Cristiano Ronaldo"

Model		**CA**	**F1**	**Precision**	**Recall**	**Avg Cv**
Naive Bayes	TF-IDF	0.67	0.68	0.72	0.67	0.67
	Count Vec	0.65	0.66	0.71	0.65	0.65
Random Forest	TF-IDF	0.62	0.64	0.70	0.62	0.61
	Count Vec	0.60	0.62	0.70	0.59	0.59
Support Vector Machine	TF-IDF	0.67	0.68	0.73	0.67	0.67
	Count Vec	0.51	0.56	0.73	0.52	0.51
K-Nearest Neighbours	TF-IDF	0.51	0.53	0.56	0.51	0.51
	Count Vec	0.49	0.50	0.54	0.49	0.47
Passive-Aggressive Classifier	TF-IDF	0.70	0.70	0.72	0.70	0.68
	Count Vec	0.62	0.64	0.69	0.62	0.61

Logistic Regression	TF-IDF	0.67	0.68	0.71	0.67	0.67
	Count Vec	0.66	0.68	0.74	0.65	0.65
Extreme Gradient Boosting	TF-IDF	0.61	0.64	0.74	0.61	0.61
	Count Vec	0.60	0.64	0.77	0.60	0.61

Table 14 - DL results - Without the author "Cristiano Ronaldo"

Model	**CA**	**F1**	**Precision**	**Recall**	**Avg CV**
RNN	0.68	0.69	0.72	0.68	0.68
CNN	0.53	0.54	0.59	0.52	0.53

After analyzing the results, it is possible to conclude that the performance of the models demonstrated slight changes when compared to their baseline setup.

The NB model with TF-IDF achieved a 67% accuracy, an F1 score of 0.68, and a recall of 0.67. These results showed similar performance to the baseline, indicating that Ronaldo's stylistic features did not alter the model's ability to identify other authors.

SVM with TF-IDF showed a slight improvement in all metrics, with a 67% classification accuracy and a 0.73 precision score. This increase implies that the model was able to focus more on the remaining authors with fewer noisy features from Ronaldo's writing style.

The PAC model showed the most significant improvement, with an accuracy, an F1 score, and recall of 70%, outperforming other models in this experiment. The other models, such as Logistic Regression and K-Nearest Neighbors, also demonstrated slight improvements to their baseline metrics.

In the case of the Deep Learning models, the absence of the author Ronaldo had little impact. The RNN continued to perform strongly with an accuracy of 68% and an F1 score of 0.69, while the CNN model underperformed with an accuracy of 53%, showing no significant change from the baseline.

Based on the results, the Passive-Aggressive Classifier (PAC) with TF-IDF emerged as the leading model in this experiment. With an accuracy of 70%, an F1 score of 0.70, a recall of 0.70, and an average cross-validation score of 0.68, PAC displayed the best performance metrics after removing Ronaldo.

In conclusion, excluding the author Ronaldo from the Twitter dataset resulted in slight improvements for some models, particularly PAC. The overall effect of his exclusion was not dramatic but provided some insights into the importance of specific authors in the dataset. Ultimately, PAC emerged as the leading model, showcasing its robustness and effectiveness in authorship identification tasks, even when certain authors are excluded.

4.3.2.5 70/30 train-test split ratio

The final experiment conducted modifies the train-test split ratio, shifting from the previous 80/20 split to a 70/30 split. By allocating 70% of the data for training and 30% for testing, it is possible to assess how a larger testing set will affect the model's performance.

A larger test set can provide more robust evaluation metrics and potentially reveal insights into how well the model generalizes to unseen data.

The preprocessing remains consistent with prior experiments, including the use of TF-IDF and Count Vectorizer as feature extraction methods. Additionally, 10-fold cross-validation continues to be used to ensure stability in performance evaluation, and all authors are included.

Table 15 - ML results - 70/30 train-test split ratio

Model		**CA**	**F1**	**Precision**	**Recall**	**Avg Cv**
Naive Bayes	TF-IDF	0.65	0.66	0.70	0.65	0.65
	Count Vec	0.64	0.64	0.68	0.63	0.63
Random Forest	TF-IDF	0.59	0.61	0.66	0.59	0.58
	Count Vec	0.58	0.60	0.66	0.58	0.56
Support Vector Machine	TF-IDF	0.65	0.66	0.71	0.65	0.64
	Count Vec	0.49	0.53	0.69	0.49	0.48
K-Nearest Neighbours	TF-IDF	0.50	0.52	0.56	0.50	0.49
	Count Vec	0.48	0.49	0.53	0.47	0.47
Passive-Aggressive Classifier	TF-IDF	0.67	0.67	0.70	0.67	0.66
	Count Vec	0.58	0.60	0.65	0.58	0.58
Logistic Regression	TF-IDF	0.65	0.66	0.70	0.65	0.65
	Count Vec	0.64	0.65	0.71	0.64	0.63
	TF-IDF	0.58	0.61	0.69	0.58	0.58

Extreme Gradient Boosting	Count Vec	0.59	0.62	0.73	0.58	0.59

Table 16 - DL results - Experiment 5 - 70/30 train-test split ratio

Model	**CA**	**F1**	**Precision**	**Recall**	**Avg CV**
RNN	0.66	0.67	0.69	0.66	0.66
CNN	0.54	0.55	0.61	0.54	0.51

With the analysis of the results, it is possible to determine that the models showed varied performance compared to the baseline, with most models generating a slight decrease in performance.

The Naïve Bayes (NB) with TF-IDF showed a slight decrease in accuracy, going from 66% to 65%, and a decrease in F1 score, going from 0.67 to 0.66. The Random Forest (RF) model maintained consistent performance metrics, showcasing little to no decline. The Support Vector Machine (SVM) exhibited a decline in most metrics, with precision being the only metric that remained unchanged.

The Passive-Aggressive Classifier (PAC), once again, remained as the top performer with an accuracy of 67%, F1 score of 0.67, and recall of 0.67. The larger test set did not heavily affect this model, suggesting its robustness across different data splits.

Overall, the remaining models showed minimal improvement in their metrics, with only slight increases in some cases, such as the F1 score of Extreme Gradient Boosting, which rose from 0.58 to 0.61.

In the case of the Deep Learning models, these maintained the same values, indicating that their performance was unaffected by the larger test set.

In conclusion, the 70/30 train-test split experiment demonstrated that while most models showed slight decreases in performance compared to the baseline, PAC maintained its strong position as the leading model.

4.3.3 Leading model for each experiment

This section provides a detailed comparison between the baseline and the various modifications (experiments) tested. The results are summarized in Table 17, highlighting the leading model selected for each experiment.

Table 17 - Summary of the experiment results

Experiment	**Model Selected**	**CA**	**F1**	**Precision**	**Recall**	**Avg Cv**
Baseline	PAC (TF-IDF)	0.67	0.67	0.70	0.67	0.67
Character Level vs Word Level	PAC (TF-IDF)	0.72	0.72	0.73	0.72	0.71
15-fold cross validation	PAC (TF-IDF)	0.67	0.67	0.70	0.67	0.67
No stopword removal	PAC (TF-IDF)	0.68	0.69	0.71	0.68	0.69
Without the author "Cristiano Ronaldo"	PAC (TF-IDF)	0.70	0.70	0.72	0.70	0.68
70/30 train-test split ratio	PAC (TF-IDF)	0.67	0.67	0.70	0.67	0.66

The results analyzed highlight that the PAC model with TF-IDF vectorization outperformed other approaches across all experiments, reaffirming its suitability for authorship identification tasks on the Twitter dataset. The variations in preprocessing and dataset composition, such as retaining stopwords, led to slight improvements in all the respective metrics. For instance, retaining stopwords increased the F1 score from 0.67 to 0.69, indicating that stylistic cues in common words can enhance the model's ability to distinguish authors. Other modifications, such as changing the train-test split ratio, showed negligible impact, demonstrating the robustness of the baseline setup.

4.3.4 Model Selection

For the task at hand of correctly identifying the author of a tweet, the process of model selection was heavily influenced by the nature of the dataset. Since the data consists of short and informal texts, a high level of unpredictability arises, requiring a model capable of handling such variability.

After evaluating the different models, the Passive-Aggressive Classifier (PAC) emerged as the most suitable choice, delivering strong results in the baseline setup and the experiments conducted.

The PAC model, being specifically designed for online learning, means that it can update its parameters continuously as new data arrives. While it might not be the first choice when working with smaller-sized datasets, as evidenced by Varada (2022), PAC works well in environments where new data is constantly being created. This makes it an ideal option for processing real-time streams of tweets or for handling dynamic datasets that are frequently updated.

In this study, combining PAC with TF-IDF feature representation showed an improvement in accuracy, a critical factor for the task of Authorship Identification. Ultimately, the decision to position PAC as the leading model was validated through model evaluation and the experiments conducted. Given its efficiency and adaptability in handling dynamic datasets, meaning that it can support real-time text classification systems, PAC proves to be a good choice for real-world applications.

4.3.5 Model Interpretation and Knowledge Representation

The Passive-Aggressive Classifier (PAC) demonstrated a strong ability to detect stylistic patterns across different authors when combined with TF-IDF representations. These patterns, embedded in the model's decision-making process, go beyond simple keyword memorization. Instead, the model is capable of generalizing writing behaviors, such as the use of different punctuation styles, length patterns of phrases, and particular lexical choices.

Ultimately, the best performance was achieved using a character-level TF-IDF representation, where each text was transformed into vectors based on overlapping sequences of characters. The vectorizer was configured with the parameters analyzer='char' and ngram_range= (2, 5), meaning that the model considered all character sequences from bigrams (2 characters) to 5-grams (5 characters).

The insights extracted from the models were effectively visualized through performance metrics (such as accuracy and F1-score) and word representations, such as word clouds. These tools played a crucial role in transforming data into interpretable and actionable knowledge. They provided an understanding of which words and linguistic traits most influenced the model's decisions, which is essential for explaining the model's outputs.

Through the model's evaluation, it became evident that the choice of TF-IDF representation significantly enhanced the PAC model's performance. The weighted scheme of TF-IDF emphasizes the most discriminative terms an author uses, helping to draw decision boundaries between authors. In this context, where the data is often noisy, giving importance to unique and unusual terms proved to be relevant in helping the model to learn more effectively within the data, generating a better predictive performance.

Moreover, the practical implications of this model extend beyond academic experimentation. Given its efficiency and ability to adapt to incoming data, the Passive-Aggressive Classifier holds potential for real-world applications in domains such as forensic linguistics, plagiarism detection, and content moderation. For instance, it can assist in detecting harmful or impersonated content on social media platforms by continuously learning and adapting to new writing patterns. With this, the PAC model

represents a practical and scalable solution for real-time environments. Its robustness in handling evolving textual inputs makes it a valuable asset in fields requiring rapid adaptation and precise textual attribution.

CONCLUSION AND FUTURE WORK

This dissertation provides an in-depth study of the field of Authorship Identification using Machine Learning (ML) and Deep Learning (DL) techniques applied to Twitter data. The research conducted showcases the challenges of handling noisy and informal data when it comes to identifying the author of a given text. Among the experiments conducted, the Passive-Aggressive Classifier (PAC) consistently outperformed others, especially when changes like maintaining stopwords or excluding certain authors were applied. This highlights the model's ability to effectively handle Twitter data structures.

Ultimately, these findings suggest that PAC, particularly when paired with appropriate feature representations, holds great promise for identifying authorship in noisy and informal text. However, the study also identified key areas for future exploration. One promising direction is the development of real-time Authorship Identification systems, such as early detection of harmful behavior, where PAC models show great promise. Achieving this would require balancing computational efficiency with accuracy, and addressing the constantly dynamic nature of authors' writing styles, influenced by current trends and personal changes, would be an important factor to consider.

Furthermore, expanding this work to other social media platforms like Reddit or Instagram also presents valuable opportunities for future research. These platforms offer distinct linguistic features and user behaviors that could enhance the robustness of the techniques applied in Authorship Identification. The constantly updated "slang" used on these platforms is an important factor that needs to be considered when applying these techniques. Incorporating these distinct features across multiple platforms, while also maintaining a real-time identification capabilitie, would allow for more effective content moderation, preventing harmful content as it is being created, tackling issues like cyberbullying or hate speech.

As social media platforms are of global scale, future work should also focus on extending Authorship Identification models to handle multilingual data. This would involve developing algorithms capable of accounting for language-specific features, such as

grammatical structures and specific idiom expressions, which when analyzed correctly can portray meaning that differ from the literal interpretation of the phrases.

Ultimately, future work should aim to improve the scalability of these systems, ensuring their applicability to a wider range of scenarios while maintaining strong performance. This challenge will ensure that the systems are capable of handling large volumes of textual data while maintaining good precision and speed in classification. This direction is particularly relevant today, as the continuous growth of information online demands solutions that are capable of adapting to an exponential increase of data.

REFERENCES

Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text Classification Algorithms: A Survey. *Information 2019, Vol. 10, Page 150*, *10*(4), 150. https://doi.org/10.3390/INFO10040150

Feroze, S. A., Feroze, S. M. B., & Abbasi, U. (2024). The Unveiling Distress: Harnessing NLP and Deep Learning to Identify Suicidal Signals in Tweets. *International Journal of Technology Innovation and Management (IJTIM)*, *4*(1), 20–31. https://doi.org/10.54489/YMY5BP91

Chungku, C., Rabgay, J., & Faaß, G. (2010). *Building NLP resources for Dzongkha: A Tagset and A Tagged Corpus* (pp. 103–110). https://aclanthology.org/W10-3214/

Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., & Herrera, F. (2018). Learning from Imbalanced Data Sets. *Learning from Imbalanced Data Sets*. https://doi.org/10.1007/978-3-319-98074-4

Palaniappan, J. R. (2023). A Wide Analysis of Loss Functions for Image Classification Using Convolution Neural Network. *Proceedings - 2023 3rd International Conference on Innovative Sustainable Computational Technologies, CISCT 2023*. https://doi.org/10.1109/CISCT57197.2023.10351209

Ghosh, S., Vinyals, O., Strope, B., Roy, S., Dean, T., & Heck, L. (2016). *Contextual LSTM (CLSTM) models for Large scale NLP tasks*. https://doi.org/10.1145/1235

Ming, Y., Cao, S., Zhang, R., Li, Z., Chen, Y., Song, Y., & Qu, H. (2017). Understanding Hidden Memories of Recurrent Neural Networks. *2017 IEEE Conference on Visual Analytics Science and Technology, VAST 2017 - Proceedings*, 13–24. https://doi.org/10.1109/VAST.2017.8585721

Fattahi, J., Mejri, M., & Marwa Ziadia. (2021). Extreme Gradient Boosting for Cyberpropaganda Detection (pp 99 – 112). *Frontiers in Artificial Intelligence and Applications*. https://doi.org/10.3233/faia210012

Shaik, M. A., Sree, M. Y., Vyshnavi, S. S., Ganesh, T., Sushmitha, D., & Shreya, N. (2023). Fake News Detection using NLP. *International Conference on Innovative*

Data Communication Technologies and Application, ICIDCA 2023 - Proceedings, 399–405. https://doi.org/10.1109/ICIDCA56705.2023.10100305

Varada Rajkumar, K., Vallabhaneni, P., Marlapalli, K., Koti Mani Kumar, T. N. S., & Revathi, S. (2022). Detection of Fake News Using Natural Language Processing Techniques and Passive Aggressive Classifier. *Smart Innovation, Systems and Technologies*, *289*, 593–601. https://doi.org/10.1007/978-981-19-0011-2_53

Ghaeini, M. (2013). Intrinsic Author Identification Using Modified Weighted KNN Notebook for PAN at CLEF 2013 https://ceur-ws.org/Vol-1179/CLEF2013wn-PAN-Ghaeini2013.pdf

Moss, H. B., Leslie, D. S., & Rayson, P. (2018). Using J-K fold Cross Validation to Reduce Variance When Tuning NLP Models. *COLING 2018 - 27th International Conference on Computational Linguistics, Proceedings*, 2978–2989. https://arxiv.org/pdf/1806.07139

Hossain, M. R., Hoque, M. M., Dewan, M. A. A., Siddique, N., Islam, M. N., & Sarker, I. H. (2021). Authorship classification in a resource constraint language using convolutional neural networks. *IEEE Access*, *9*, 100319–100338. https://doi.org/10.1109/ACCESS.2021.3095967

Liu, C. Z., Sheng, Y. X., Wei, Z. Q., & Yang, Y. Q. (2018). Research of Text Classification Based on Improved TF-IDF Algorithm. *2018 IEEE International Conference of Intelligent Robotic and Control Engineering, IRCE 2018*, 69–73. https://doi.org/10.1109/IRCE.2018.8492945

Xu, J., Jiang, Y., Yuan, B., Li, S., & Song, T. (2023). Automated Scoring of Clinical Patient Notes Using Advanced NLP and Pseudo Labeling. *2023 5th International Conference on Artificial Intelligence and Computer Applications, ICAICA 2023*, 384–388. https://doi.org/10.1109/ICAICA58456.2023.10405427

Li, X., Sun, X., Meng, Y., Liang, J., Wu, F., & Li, J. (2019). Dice Loss for Data-imbalanced NLP Tasks. *Proceedings of the Annual Meeting of the Association for*

Computational Linguistics, 465–476. https://doi.org/10.18653/v1/2020.acl-main.45

Singh, J., & Gupta, V. (2017). A systematic review of text stemming techniques. *Artificial Intelligence Review*, *48*(2), 157–217. https://doi.org/10.1007/S10462-016-9498-2/METRICS

Sennrich, R., Haddow, B., & Birch, A. (2015). Neural Machine Translation of Rare Words with Subword Units. *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Long Papers*, *3*, 1715–1725. https://doi.org/10.18653/v1/p16-1162

Abidin,Z., Junaidi, A., & Wamiliana. (2024). Text Stemming and Lemmatization of Regional Languages in Indonesia: A Systematic Literature Review. *Journal of Information Systems Engineering and Business Intelligence*, *10*(2), 217–231. https://doi.org/10.20473/JISEBI.10.2.217-231

Sharou, K. Al, Li, Z., & Specia, L. (2021). *Towards a Better Understanding of Noise in Natural Language Processing* (pp. 53–62). https://doi.org/10.26615/978-954-452-072-4_007

Mullen, L. A., Benoit, K., Keyes, O., Selivanov, D., & Arnold, J. (2018). Fast, Consistent Tokenization of Natural Language Text. *Journal of Open Source Software*, *3*(23), 655. https://doi.org/10.21105/JOSS.00655

Barz, B., & Denzler, J. (2019). Deep Learning on Small Datasets without Pre-Training using Cosine Loss. *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020*, 1360–1369. https://doi.org/10.1109/WACV45572.2020.9093286

Shrestha, P., Sierra, S., González, F. A., Rosso, P., Montes-Y-Gómez, M., & Solorio, T. (2017). Convolutional Neural Networks for Authorship Attribution of Short Texts. In *the Association for Computational Linguistics* (Vol. 2, pp. 669–674). https://aclanthology.org/E17-2106/

Khalid, N. (2021). Authorship identification based on nlp. *European Journal of Computer Science and Information Technology*, *9*(1), 1–26. https://www.eajournals.org/wp-content/uploads/Author-Identification-Based-on-NLP.pdf

Yülüce, İ., & Dalkılıç, F. (2022). Authorship identification with machine learning algorithms. *International Journal of Multidisciplinary Studies and Innovative Technologies*, *6*(1), 45. https://doi.org/10.36287/ijmsit.6.1.45

Abbasi, A., Javed, A. R., Iqbal, F., Jalil, Z., Gadekallu, T. R., & Kryvinska, N. (2022). Authorship identification using ensemble learning. *Scientific Reports*, *12*(1), 1–16. https://doi.org/10.1038/S41598-022-13690-4

Tang, X. (2024). Author identification of literary works based on text analysis and deep learning. *Heliyon*, *10*(3), e25464. https://doi.org/10.1016/j.heliyon.2024.e25464

Nanda Fahrezi Munazhif, Gomal Juni Yanris, & Nirmala, M. (2023) Implementation of the k-nearest neighbor (kNN) method to determine outstanding student classes. *Sinkron: Jurnal Dan Penelitian Teknik Informatika*, *8*(2), 719–732. https://doi.org/10.33395/sinkron.v8i2.12227

Neal, T., Sundararajan, K., Fatima, A., Yan, Y., Xiang, Y., & Woodard, D. (2018). Surveying Stylometry Techniques and Applications. *ACM Computing Surveys*, *50*(6), 1–36. https://doi.org/10.1145/3132039

Barlas, G., & Efstathios Stamatatos. (2020). Cross-domain authorship attribution using pre-trained language models. *IFIP Advances in Information and Communication Technology*, 255–266. https://doi.org/10.1007/978-3-030-49161-1_22

Mendenhall, T. (1887). The characteristic curves of composition. *Science*, *9*(37), 37–49.

Mosteller, F., & Wallace, D. L. (1964). *Inference and disputed authorship: The Federalist*. Addison-Wesley.

De Freitas, L., Calaigian, S., & Varges Da Silva, M. (2023). *The effect of stemming and lemmatization on Portuguese fake news text classification*. https://arxiv.org/pdf/2310.11344

Ribeiro, E., Ribeiro, R., & de Matos, D. M. (2018). A Study on Dialog Act Recognition using Character-Level Tokenization. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11089 LNAI*, 93–103. https://doi.org/10.1007/978-3-319-99344-7_9

Koppel, M., Schlier, J., & Argamon, S. (2009). Computational methods in authorship attribution. *Journal of the American Society for Information Science and Technology*, *60*(1), 9–26. https://doi.org/10.1002/ASI.20961

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*. https://arxiv.org/pdf/1301.3781

Rolnick, D., Veit, A., Belongie, S., & Shavit, N. (2017). *Deep Learning is Robust to Massive Label Noise*. https://arxiv.org/pdf/1705.10694

Kristien Margi Suryaningrum. (2023). Comparison of the TF-IDF Method with the Count Vectorizer to Classify Hate Speech. *Engineering, Mathematics and Computer Science Journal (EMACS)*, *5*(2), 79–83. https://doi.org/10.21512/emacsjournal.v5i2.9978

Singh, B., Desai, R., Ashar, H., Tank, P., & Neha Katre. (2021). A Trade-off between ML and DL Techniques in Natural Language Processing. *Journal of Physics*, *1831*(1), 012025–012025. https://doi.org/10.1088/1742-6596/1831/1/012025

Khan, W., Daud, A., Khan, K., Muhammad, S., & Haq, R. (2023). Exploring the frontiers of deep learning and natural language processing: A comprehensive

overview of key challenges and emerging trends. *Natural Language Processing Journal*, *4*(100026), 100026. https://doi.org/10.1016/j.nlp.2023.100026

Villa-Cueva, E., González-Franco, I., Sanchez-Vega, F., & Pastor López-Monroy, A. (2022). *NLP-CIMAT at PoliticEs 2022: PolitiBETO, a Domain-Adapted Transformer for Multi-class Political Author Profiling*. https://ceur-ws.org/Vol-3202/politices-paper2.pdf

Efstathios Stamatatos, Ph. D. (2013). On the Robustness of Authorship Attribution Based on Character *N*-gram Features. *Journal of Law and Policy*, *21*(2). https://brooklynworks.brooklaw.edu/jlp/vol21/iss2/7

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, *17*(3), 37–37. https://doi.org/10.1609/AIMAG.V17I3.1230

Foltýnek, T., Meuschke, N., & Gipp, B. (2019). Academic plagiarism detection: A systematic literature review. *ACM Computing Surveys*, *52*(6). https://doi.org/10.1145/3345317

Iyer, R. R., & Rose, C. P. (2019). *A Machine Learning Framework for Authorship Identification From Texts*. https://arxiv.org/pdf/1912.10204

Alhessi, Y., & Wicentowski, R. (2015). SWATAC: A Sentiment Analyzer using One-Vs-Rest Logistic Regression (pp. 636–639). *Association for Computational Linguistics.* https://aclanthology.org/S15-2106.pdf

Ilyas, M., Malik, N., Bilal, A., Razzaq, S., Maqbool, F., & Abbas, Q. (2021). Plagiarism Detection Using Natural Language Processing Techniques. *Technical Journal*, *26*(01), 90–101. https://tj.uettaxila.edu.pk/index.php/technical-journal/article/view/1484

Alzamzami, F., Hoda, M., & Saddik, A. E. (2020). Light Gradient Boosting Machine for General Sentiment Classification on Short Texts: A Comparative Evaluation. *IEEE Access*, *8*, 101840–101858. https://doi.org/10.1109/access.2020.2997330

Talukdar, W., & Biswas, A. (2024). Synergizing Unsupervised and Supervised Learning: A Hybrid Approach for Accurate Natural Language Task Modeling. *International Journal of Innovative Science and Research Technology (IJISRT)*, 1499–1508. https://doi.org/10.38124/IJISRT/IJISRT24MAY2087

Kestemont, M., Tschuggnall, M., Stamatatos, E., Daelemans, W., Specht, G., Stein, B., & Potthast, M. (2018). Overview of the Author Identification Task at PAN-2018 Cross-domain Authorship Attribution and Style Change Detection. https://repository.uantwerpen.be/docman/irua/9fa518/153293.pdf

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. https://aclanthology.org/D14-1162/

Ikonomakis, M., Kotsiantis, S., & Tampakas, V. (2005). Text classification using machine learning techniques. *WSEAS Transactions on Computers*, *4(8)*, *966–974* https://www.researchgate.net/publication/228084521_Text_Classification_Using_Machine_Learning_Techniques

Ryuichiro Hataya, & Nakayama, H. (2018). Investigating CNNs' Learning Representation under label noise. *OpenReview*. https://openreview.net/forum?id=H1xmqiAqFm

Crockett, R. (2024). Forensic Assignment Stylometry. *Springer International Handbooks of Education*, *Part F2304*, 1447–1465. https://doi.org/10.1007/978-3-031-54144-5_154

Stamatatos, E. (2009). A survey of modern authorship attribution methods. *Journal of the American Society for Information Science and Technology*, *60*(3), 538–556. https://doi.org/10.1002/ASI.21001

Ramnial, H., Panchoo, S., & Pudaruth, S. (2016). Authorship Attribution Using Stylometry and Machine Learning Techniques. *Advances in Intelligent Systems and Computing*, *384*, 113–125. https://doi.org/10.1007/978-3-319-23036-8_10

Rashid, J., Mahmood, T., Nisar, M. W., & Nazir, T. (2020). Phishing Detection Using Machine Learning Techniques. *Proceedings - 2020 1st International Conference of Smart Systems and Emerging Technologies, SMART-TECH 2020*, 43–46. https://doi.org/10.1109/SMART-TECH49988.2020.00026

APPENDIX

APPENDIX 1 – Baseline Machine Learning Implementation

This appendix contains the code used to generate the values used in the using the baseline configuration: 10-fold cross-validation, all authors included, stopwords removed, and an 80/20 train-test split for the machine learning models. The code was developed in Python (version 3.11.11), using the Scikit-learn library, and the environment Google Colaboratory.

```
#### Imports

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import PassiveAggressiveClassifier, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import TruncatedSVD
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn import metrics


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored
to speak at today's #LeadersClimateSummit about the three things we need to do to
avoid a climate disaster"]})
```

```
# Add the first line for Bill Gates
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today's #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())

#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)



#### Naive Bayes - TF-IDF
```

```
# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', MultinomialNB())
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)




#### Naive Bayes - Count Vectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2),stop_words='english')),
    ('classifier', MultinomialNB())
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())
```

```
pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)



#### Random Forest - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

 and Random Forest classifier
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', RandomForestClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - Count Vectorizer

# 80/20 split
```

```
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', RandomForestClassifier(random_state=42))
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Support Vector Machine - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# SVM
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', SVC(random_state=42))
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)
```

```
# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Support Vector Machine - Count Vectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# SVM
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', SVC(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Passive-Aggressive Classifier - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)
```

```
#PAC
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
#### Passive-Aggressive Classifier - Count Vectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# PAC
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)
```

```
# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Logistic Regression - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

#LR
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Logistic Regression - Count Vectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)
```

```
# LR
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# XGBOOST
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())
```

```
pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - Count Vectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# XGBOOST
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
```

```
#### K-nearest Neighbours - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)

# KNN
pipeline_knn = Pipeline([
    ('vectorizer', TfidfVectorizer(
        ngram_range=(1, 2),
        stop_words='english',
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())

pipeline_knn.fit(x_train, y_train)

# KNN
y_pred_knn = pipeline_knn.predict(x_test)

# Results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)
#### K-nearest Neighbours - Count Vectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)
```

```
# KNN
pipeline_knn = Pipeline([
    ('vectorizer', CountVectorizer(
        ngram_range=(1, 2),
        stop_words='english',
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())


pipeline_knn.fit(x_train, y_train)

# KNN
y_pred_knn = pipeline_knn.predict(x_test)

# results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")

print("Classification Report (KNN):\n", classification_report_knn)
```

APPENDIX 2 – Baseline Deep Learning Implementation

This appendix contains the code used to generate the results with the baseline configuration: 10-fold cross-validation, all authors included, stopwords removed, and an 80/20 train-test split for the Deep Learning models. The code was implemented in Python (version 3.11.11) within the Google Colaboratory environment, using the TensorFlow/Keras library.

```
#### Imports

import pandas as pd
import numpy as np
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Conv1D,
MaxPooling1D, GlobalMaxPooling1D


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
```

```
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today's #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)



### RNN - Baseline
```

```
# Function to remove stopwords
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Remove stopwrods
data['Tweet'] = data['Tweet'].apply(remove_stopwords)

X = data["Tweet"]
y = data["Author_Name"]

# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)

# Label Encoding
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 80 / 20 split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.20,
random_state=42)

# Function to created the model
def create_rnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        LSTM(units=50, dropout=0.2, recurrent_dropout=0.2),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 10 K-FOLD cross validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

cv_scores = []
```

```
for train_index, val_index in kf.split(X_train, y_train):
    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]
    model = create_rnn_model()

    # Model fit
    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    # Evaluate model
    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)

# Results
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_rnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

# Predict on the test data
y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


### CNN - Baseline

def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)


data['Tweet'] = data['Tweet'].apply(remove_stopwords)

X = data["Tweet"]
y = data["Author_Name"]
```

```
# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)


label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 80/20 Split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.20,
random_state=42)

# Model Creation
def create_cnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        Conv1D(128, 5, activation='relu'),
        MaxPooling1D(5),
        Conv1D(128, 5, activation='relu'),
        GlobalMaxPooling1D(),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 10 k-fold CV
kf = KFold(n_splits=10, shuffle=True, random_state=42)

cv_scores = []

for train_index, val_index in kf.split(X_train, y_train):

    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]

    model = create_cnn_model()

    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)
```

```
    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)

# CV results
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_cnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
```

APPENDIX 3 – Comparison of Character-Level and Word-Level Representations in Machine Learning

This appendix showcases the code used to obtain a character-level representation of the textual data. Since the analysis is performed at the character level, stopwords were retained during preprocessing. The evaluation includes all authors, an 80/20 train-test split and a 10-fold cross-validation. The code was implemented with the same setup as the baseline presented in appendix 1, which includes Python (version 3.11.11), the Scikit-learn library, and Google Colaboratory as the environment.

```
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import PassiveAggressiveClassifier



#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
```

```
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today's #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)



#### Naive Bayes - Character-level  TD-IDF
```

```
# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', MultinomialNB())
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Naive Bayes - Character-level  - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', MultinomialNB())
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())
```

```
pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - Character-level  - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', RandomForestClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - Character-level  - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)
```

```
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', RandomForestClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


####  Support Vector Machine - Character-level  - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', SVC(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)

# Results
```

```
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


####  Support Vector Machine - Character-level  - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', SVC(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


####  Passive Aggressive Classifier - Character-level  - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
```

```
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


####  Passive Aggressive Classifier - Character-level  - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


####  Logistic Regression - Character-level  - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


####  Logistic Regression - Character-level  - Count Vectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(analyzer='char', ngram_range=(2, 5))),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])
```

```
cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


####  Extreme Gradient Boosting - Character-level  - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(analyzer='char', ngram_range=(2, 4),
max_features=10000)),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
```

```
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - Character-level - Count Vectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(analyzer='char', ngram_range=(2, 4),
max_features=10000)),  # Character-level CountVectorizer
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### K-Nearest Neighbours - Character-level - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)


pipeline_knn = Pipeline([
    ('vectorizer', TfidfVectorizer(
        analyzer='char',
```

```
        ngram_range=(2, 5),
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(
        metric='cosine',
        weights='distance',          # Weight by distance
        n_neighbors=10               # Number of neighbors
    ))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())


pipeline_knn.fit(x_train, y_train)


y_pred_knn = pipeline_knn.predict(x_test)

# Results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)


#### K-Nearest Neighbours - Character-level  - Count Vectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)

pipeline_knn = Pipeline([
    ('vectorizer', CountVectorizer(
        analyzer='char',
        ngram_range=(3, 5),
        max_df=0.9,
```

```
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(
        metric='cosine',
        weights='distance',          # Weight by distance
        n_neighbors=10               # Number of neighbors
    ))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())


pipeline_knn.fit(x_train, y_train)


y_pred_knn = pipeline_knn.predict(x_test)

# Results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)
```

APPENDIX 4 – Performance Evaluation with 15-Fold Cross-Validation in Machine Learning Models

This appendix presents the code used to generate the Machine Learning model results, with a switch from 10-fold to 15-fold cross-validation to assess whether it produces better or more stable outcomes. Stopwords were removed during preprocessing, an 80/20 train-test split was applied, and all authors were included in the evaluation. The code was developed using Python (version 3.11.11), the Scikit-learn library and Google Colaboratory as the environment.

```
import pandas as pd
import numpy as np
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import PassiveAggressiveClassifier


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
```

```
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today’s #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)

#### Naive Bayes - 15 CV - TF-IDF

# 80/20 Split
```

```
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', MultinomialNB())
])



cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Naive Bayes - 15 CV - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2),stop_words='english')),
    ('classifier', MultinomialNB())
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())
```

```
pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - 15 CV - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', RandomForestClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - 15 CV - Count Vectorizer

# 80/20 Split
```

```
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# Create a pipeline for CountVectorizer with stopword removal and Random Forest
classifier
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', RandomForestClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Support Vector Machine - 15 CV - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', SVC(random_state=42))
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())
```

```
pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Support Vector Machine - 15 CV - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', SVC(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Passive-Aggressive Classifier - 15 CV - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)
```

```
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Passive-Aggressive Classifier - 15 CV - CountVectorizer


# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)
```

```
# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Logistic Regression - 15 CV - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Logistic Regression - 15 CV - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
```

```
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - 15 CV - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
```

```
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
#### Extreme Gradient Boosting - 15 CV - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### K-Nearest Neighbours - 15 CV – TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)

pipeline_knn = Pipeline([
    ('vectorizer', TfidfVectorizer(
        ngram_range=(1, 2),
        stop_words='english',
```

```
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])

cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=15,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())

pipeline_knn.fit(x_train, y_train)

y_pred_knn = pipeline_knn.predict(x_test)


classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)


#### K-Nearest Neighbours - 15 CV - CountVectorizer

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)


pipeline_knn = Pipeline([
    ('vectorizer', CountVectorizer(
        ngram_range=(1, 2),
        stop_words='english',
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
```

```
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=15,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())


pipeline_knn.fit(x_train, y_train)


y_pred_knn = pipeline_knn.predict(x_test)

# Results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)
```

APPENDIX 5 – Performance Evaluation with 15-Fold Cross-Validation in Deep Learning Models

This appendix presents the code used to generate the Deep Learning model results, with a switch from 10-fold to 15-fold cross-validation to assess whether it produces better or more stable outcomes. Stopwords were removed during preprocessing, an 80/20 train-test split was applied, and all authors were included in the evaluation. The code was developed using Python (version 3.11.11), the TensorFlow/Keras library and Google Colaboratory as the environment.

```
import nltk
import pandas as pd
import re
import numpy as np
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Embedding, LSTM, Conv1D, MaxPooling1D,
GlobalMaxPooling1D
from keras.preprocessing.text import Tokenizer
from nltk.corpus import stopwords
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.utils import to_categorical


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})
```

```
# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today's #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
```

```
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)

#### Naive Bayes - 15 CV - TF-IDF

# 80/20 Split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', MultinomialNB())
])



cv_scores = cross_val_score(pipeline, x_train, y_train, cv=15)  # 15 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### RNN - 15 CV

# Stopword Removal
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

data['Tweet'] = data['Tweet'].apply(remove_stopwords)

X = data["Tweet"]
y = data["Author_Name"]
```

```
# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 80/20 Split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.20,
random_state=42)

# Model creation
def create_rnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        LSTM(units=50, dropout=0.2, recurrent_dropout=0.2),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 15 k-fold cross validation
kf = KFold(n_splits=15, shuffle=True, random_state=42)

cv_scores = []

for train_index, val_index in kf.split(X_train, y_train):
    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]

    model = create_rnn_model()

    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)

# CV Results
print("Cross-validation scores:", cv_scores)
```

```
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_rnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### CNN- 15 CV

#Stopword function
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

data['Tweet'] = data['Tweet'].apply(remove_stopwords)

X = data["Tweet"]
y = data["Author_Name"]

# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 80/20 Split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.20,
random_state=42)
```

```
# Model creation
def create_cnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        Conv1D(128, 5, activation='relu'),
        MaxPooling1D(5),
        Conv1D(128, 5, activation='relu'),
        GlobalMaxPooling1D(),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 15 K fold CV
kf = KFold(n_splits=15, shuffle=True, random_state=42)

cv_scores = []

for train_index, val_index in kf.split(X_train, y_train):

    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]
    model = create_cnn_model()

    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_cnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
```

APPENDIX 6 – Impact of No Stopword Removal on Machine Learning Models

This appendix presents the code used to generate the results of the ML models without the removal of stopwords, in order to evaluate their impact on model performance. A 10-fold cross-validation was employed, alongside an 80/20 train-test split, considering all authors. The code was developed using Python (version 3.11.11), the Scikit-learn library and Google Colaboratory as the environment.

```
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import PassiveAggressiveClassifier


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored
to speak at today's #LeadersClimateSummit about the three things we need to do to
avoid a climate disaster"]})
```

```
# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)


#### Naive Bayes - No stopwords removal - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42)
```

```
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2))),
    ('classifier', MultinomialNB())
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Naive Bayes - No stopwords removal - Count Vectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2))),
    ('classifier', MultinomialNB())
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
```

```
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - No stopwords removal - TF-IDF
# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2))),
    ('classifier', RandomForestClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - No stopwords removal - Count Vectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2))),
    ('classifier', RandomForestClassifier(random_state=42))
```

```
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)



#### Support Vector Machine - No stopwords removal - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2))),
    ('classifier', SVC(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Support Vector Machine - No stopwords removal - CountVectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2))),
    ('classifier', SVC(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
#### Passive Aggressive Classifier - No stopwords removal - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2))),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])
```

```
cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)



#### Passive Aggressive Classifier - No stopwords removal - CountVectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2))),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
```

```
#### Logistic Regression - No stopwords removal - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2))),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Logistic Regression - No stopwords removal - CountVectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2))),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10) # 10 CV
```

```
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - No stopwords removal - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words=None)),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
```

```
#### Extreme Gradient Boosting - No stopwords removal - CountVectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words=None)),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
#### K-Nearest Neighbours - No stopwords removal - TF-IDF

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)


pipeline_knn = Pipeline([
    ('vectorizer', TfidfVectorizer(
        ngram_range=(1, 2),
        stop_words=None,
        max_df=0.9,
        min_df=3,
        max_features=10000
```

```
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())

pipeline_knn.fit(x_train, y_train)
y_pred_knn = pipeline_knn.predict(x_test)

# Results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)


#### K-Nearest Neighbours - No stopwords removal - CountVectorizer

# 80/20 split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)


pipeline_knn = Pipeline([
    ('vectorizer', CountVectorizer(
        ngram_range=(1, 2),
        stop_words=None,
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])
```

```
cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())


pipeline_knn.fit(x_train, y_train)
y_pred_knn = pipeline_knn.predict(x_test)

# Results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)
```

APPENDIX 7 – Impact of No Stopword Removal on Deep Learning Models

This appendix presents the code used to generate the results of the Deep Learning models without the removal of stopwords. A 10-fold cross-validation was employed, alongside an 80/20 train-test split and considering all authors in the dataset. The code was implemented in Python (version 3.11.11) within the Google Colaboratory environment, using the TensorFlow/Keras library.

```
import nltk
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Embedding, LSTM, Conv1D, MaxPooling1D,
GlobalMaxPooling1D
from keras.preprocessing.text import Tokenizer
from nltk.corpus import stopwords
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.utils import to_categorical


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)
```

```
# Add the first line for Bill Gates
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today's #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)


#### RNN - No stopword Removal
```

```
X = data["Tweet"]
y = data["Author_Name"]

# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 80/20 Split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.20,
random_state=42)

#Model Creation
def create_rnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        LSTM(units=50, dropout=0.2, recurrent_dropout=0.2),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

#10 k - crossvalidation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

cv_scores = []

for train_index, val_index in kf.split(X_train, y_train):
    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]

    model = create_rnn_model()

    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)
```

```
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_rnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### CNN - No stopword Removal

X = data["Tweet"]
y = data["Author_Name"]

# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 80/20 split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.20,
random_state=42)

# Model Creation
def create_cnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        Conv1D(128, 5, activation='relu'),
        MaxPooling1D(5),
        Conv1D(128, 5, activation='relu'),
```

```
        GlobalMaxPooling1D(),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 10 k-fold CV
kf = KFold(n_splits=10, shuffle=True, random_state=42)

cv_scores = []

for train_index, val_index in kf.split(X_train, y_train):

    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]

    model = create_cnn_model()

    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)

# CV results
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_cnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
```

APPENDIX 8 – Excluding the Author 'Cristiano Ronaldo' in Machine Learning Models

This appendix presents the code used to generate the results of the Machine Learning models after removing the author 'Cristiano Ronaldo'. A 10-fold cross-validation was performed, using an 80/20 train-test split and applying stopword removal during preprocessing. The code was developed using Python (version 3.11.11), the Scikit-learn library and Google Colaboratory as the environment.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import PassiveAggressiveClassifier, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import TruncatedSVD
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn import metrics


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored
to speak at today's #LeadersClimateSummit about the three things we need to do to
avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)
```

```
# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)
# Remove author Ronaldo
data = data[data['Author_Name'] != 'Cristiano']



#### Naive Bayes - TF-IDF

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42)
```

```
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', MultinomialNB())
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)



#### Naive Bayes - Count Vectorizer

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2),stop_words='english')),
    ('classifier', MultinomialNB())
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)
```

```
# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)



#### Random Forest - TF-IDF

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

 and Random Forest classifier
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', RandomForestClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - Count Vectorizer

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)
```

```
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', RandomForestClassifier(random_state=42))
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Support Vector Machine - TF-IDF

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# SVM
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', SVC(random_state=42))
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Support Vector Machine - Count Vectorizer

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# SVM
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', SVC(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Passive-Aggressive Classifier - TF-IDF

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

#PAC
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
```

```
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Passive-Aggressive Classifier - Count Vectorizer

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# PAC
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
```

```
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Logistic Regression - TF-IDF

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

#LR
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Logistic Regression - Count Vectorizer

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# LR
```

```
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - TF-IDF

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# XGBOOST
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
```

```
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - Count Vectorizer

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.20, random_state=42)

# XGBOOST
pipeline = Pipeline([
   ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
   ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### K-nearest Neighbours - TF-IDF
```

```
# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)

# KNN
pipeline_knn = Pipeline([
    ('vectorizer', TfidfVectorizer(
        ngram_range=(1, 2),
        stop_words='english',
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())

pipeline_knn.fit(x_train, y_train)

# KNN
y_pred_knn = pipeline_knn.predict(x_test)

# Results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)



#### K-nearest Neighbours - Count Vectorizer

# 80/20 split – No author “Cristiano Ronaldo”
x_train, x_test, y_train, y_test = train_test_split(
```

```
    data["Tweet"], data["Author_Name"], test_size=0.20, random_state=42
)
# KNN
pipeline_knn = Pipeline([
    ('vectorizer', CountVectorizer(
        ngram_range=(1, 2),
        stop_words='english',
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())


pipeline_knn.fit(x_train, y_train)

# KNN
y_pred_knn = pipeline_knn.predict(x_test)

# results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)
```

APPENDIX 9 – Excluding the Author 'Cristiano Ronaldo' in Deep Learning Models

This appendix presents the code used to generate the results of the Deep Learning models after removing the author 'Cristiano Ronaldo'. A 10-fold cross-validation was performed, using an 80/20 train-test split and applying stopword removal during preprocessing. The code was implemented in Python (version 3.11.11) with the Google Colaboratory environment, using TensorFlow/Keras as the library.

```
#### Imports

import pandas as pd
import numpy as np
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Conv1D,
MaxPooling1D, GlobalMaxPooling1D


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
```

```
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today’s #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)
# Remove author Ronaldo
data = data[data['Author_Name'] != 'Cristiano']
```

```
### RNN – No author Cristiano Ronaldo

# Function to remove stopwords
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Remove stopwrods
data['Tweet'] = data['Tweet'].apply(remove_stopwords)

X = data["Tweet"]
y = data["Author_Name"]

# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)

# Label Encoding
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 80 / 20 split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.20,
random_state=42)

# Function to created the model
def create_rnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        LSTM(units=50, dropout=0.2, recurrent_dropout=0.2),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 10 K-FOLD cross validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)
```

```
cv_scores = []

for train_index, val_index in kf.split(X_train, y_train):
    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]
    model = create_rnn_model()

    # Model fit
    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    # Evaluate model
    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)

# Results
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_rnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

# Predict on the test data
y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


### RNN – No author Cristiano Ronaldo

def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)


data['Tweet'] = data['Tweet'].apply(remove_stopwords)
```

```
X = data["Tweet"]
y = data["Author_Name"]

# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)


label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 80/20 Split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.20,
random_state=42)

# Model Creation
def create_cnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        Conv1D(128, 5, activation='relu'),
        MaxPooling1D(5),
        Conv1D(128, 5, activation='relu'),
        GlobalMaxPooling1D(),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 10 k-fold CV
kf = KFold(n_splits=10, shuffle=True, random_state=42)

cv_scores = []

for train_index, val_index in kf.split(X_train, y_train):

    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]

    model = create_cnn_model()
```

```
    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)

# CV results
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_cnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
```

APPENDIX 10 –Evaluation with 70/30 Train-Test Split in Machine Learning Models

This appendix presents the code used to generate the results of the Machine Learning models. A 10-fold cross-validation was performed, using a 70/30 train-test split and applying stopword removal during preprocessing. The code was developed using Python (version 3.11.11), the Scikit-learn library and Google Colaboratory as the environment.

```
#### Imports

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import PassiveAggressiveClassifier, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import TruncatedSVD
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn import metrics


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored
to speak at today's #LeadersClimateSummit about the three things we need to do to
avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)
```

```
# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)



#### Naive Bayes - TF-IDF

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"], test_size=0.30, random_state=42)


pipeline = Pipeline([
```

```
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', MultinomialNB())
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)




#### Naive Bayes - Count Vectorizer

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2),stop_words='english')),
    ('classifier', MultinomialNB())
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
```

```
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)



#### Random Forest - TF-IDF

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

 and Random Forest classifier
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', RandomForestClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Random Forest - Count Vectorizer

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)


pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', RandomForestClassifier(random_state=42))
```

```
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Support Vector Machine - TF-IDF

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

# SVM
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', SVC(random_state=42))
])

cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())

pipeline.fit(x_train, y_train)

y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)
```

```
#### Support Vector Machine - Count Vectorizer

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

# SVM
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', SVC(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Passive-Aggressive Classifier - TF-IDF

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

#PAC
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])
```

```
cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Passive-Aggressive Classifier - Count Vectorizer

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

# PAC
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', PassiveAggressiveClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
```

```
print("Classification Report:\n", classification_report)


#### Logistic Regression - TF-IDF

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

#LR
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Logistic Regression - Count Vectorizer

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

# LR
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])
```

```
cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10 cv

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - TF-IDF

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

# XGBOOST
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)
```

```
# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### Extreme Gradient Boosting - Count Vectorizer

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(data["Tweet"], data["Author_Name"],
test_size=0.30, random_state=42)

# XGBOOST
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2), stop_words='english')),
    ('classifier', GradientBoostingClassifier(random_state=42))
])


cv_scores = cross_val_score(pipeline, x_train, y_train, cv=10)  # 10-fold cross-
validation

print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", cv_scores.mean())


pipeline.fit(x_train, y_train)


y_pred = pipeline.predict(x_test)

# Results
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


#### K-nearest Neighbours - TF-IDF

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.30, random_state=42
```

```
)

# KNN
pipeline_knn = Pipeline([
    ('vectorizer', TfidfVectorizer(
        ngram_range=(1, 2),
        stop_words='english',
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())

pipeline_knn.fit(x_train, y_train)

# KNN
y_pred_knn = pipeline_knn.predict(x_test)

# Results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)


#### K-nearest Neighbours - Count Vectorizer

# 70/30 split
x_train, x_test, y_train, y_test = train_test_split(
    data["Tweet"], data["Author_Name"], test_size=0.30, random_state=42
)
# KNN
pipeline_knn = Pipeline([
    ('vectorizer', CountVectorizer(
```

```
        ngram_range=(1, 2),
        stop_words='english',
        max_df=0.9,
        min_df=3,
        max_features=10000
    )),
    ('svd', TruncatedSVD(n_components=300, random_state=42)),
    ('classifier', KNeighborsClassifier(metric='cosine', weights='distance',
n_neighbors=10))
])


cv_scores_knn = cross_val_score(pipeline_knn, x_train, y_train, cv=10,
scoring='accuracy', n_jobs=-1)

print("Cross-validation scores (KNN):", cv_scores_knn)
print("Average cross-validation score (KNN):", cv_scores_knn.mean())


pipeline_knn.fit(x_train, y_train)

# KNN
y_pred_knn = pipeline_knn.predict(x_test)

# results
classification_report_knn = metrics.classification_report(y_test, y_pred_knn)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (KNN): {accuracy_knn:.3f}")
print("Classification Report (KNN):\n", classification_report_knn)
```

APPENDIX 11 – Evaluation with 70/30 Train-Test Split in Deep Learning Models

This appendix presents the code used to generate the results of the Deep Learning models. A 10-fold cross-validation was performed, using a 70/30 train-test split and applying stopword removal during preprocessing. The code was implemented in Python (version 3.11.11) within the Google Colaboratory environment, using the TensorFlow/Keras library.

```
#### Imports

import pandas as pd
import numpy as np
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Conv1D,
MaxPooling1D, GlobalMaxPooling1D


#### Import Dataset

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv',
sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I
was honored to speak at today's #LeadersClimateSummit about the three things we
need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
```

```
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today's #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())
# Data cleaning
print(data.isnull().sum())



#### Data Preparation


# Remove Links
data['Tweet'] = data['Tweet'].str.replace(r'http\S+', '', regex=True)

# Remove #
data['Tweet'] = data['Tweet'].str.replace(r'#', '', regex=True)

# Remove Ponctuation or special characters
data['Tweet'] = data['Tweet'].str.replace(r'[^\w\s]', '', regex=True)

# Remove extra whitespace .
data['Tweet'] = data['Tweet'].str.replace(r'\s+', ' ', regex=True)

# Remove numeric values
data['Tweet'] = data['Tweet'].str.replace(r'\d+', '', regex=True)    # \d means digital which is 0-9

# Put text into lower casing
data['Tweet'] = data['Tweet'].str.lower()

# Remove non-alphanumeric characters from the 'Text' column
data['Tweet'] = data['Tweet'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))

# Remove @ from author name
data["Author_Name"] = data["Author_Name"].str.replace("@", "", regex=False)



### RNN - Baseline
```

```
# Function to remove stopwords
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Remove stopwrods
data['Tweet'] = data['Tweet'].apply(remove_stopwords)

X = data["Tweet"]
y = data["Author_Name"]

# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)

# Label Encoding
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 70/30 split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.30,
random_state=42)

# Function to created the model
def create_rnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        LSTM(units=50, dropout=0.2, recurrent_dropout=0.2),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 10 K-FOLD cross validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

cv_scores = []
```

```
for train_index, val_index in kf.split(X_train, y_train):
    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]
    model = create_rnn_model()

    # Model fit
    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    # Evaluate model
    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
    cv_scores.append(val_acc)

# Results
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_rnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)

# Predict on the test data
y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")
print("Classification Report:\n", classification_report)


### CNN - Baseline

def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)


data['Tweet'] = data['Tweet'].apply(remove_stopwords)

X = data["Tweet"]
y = data["Author_Name"]
```

```
# Tokenize and pad sequences
max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)


label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 70/30 Split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_encoded, test_size=0.30,
random_state=42)

# Model Creation
def create_cnn_model():
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=50, input_length=max_len),
        Conv1D(128, 5, activation='relu'),
        MaxPooling1D(5),
        Conv1D(128, 5, activation='relu'),
        GlobalMaxPooling1D(),
        Dense(len(label_encoder.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# 10 k-fold CV
kf = KFold(n_splits=10, shuffle=True, random_state=42)

cv_scores = []
for train_index, val_index in kf.split(X_train, y_train):

    X_val_train, X_val_test = X_train[train_index], X_train[val_index]
    y_val_train, y_val_test = y_train[train_index], y_train[val_index]

    model = create_cnn_model()

    model.fit(X_val_train, y_val_train, epochs=5, batch_size=32, verbose=0)

    val_loss, val_acc = model.evaluate(X_val_test, y_val_test, verbose=0)
```

```
    cv_scores.append(val_acc)

# CV results
print("Cross-validation scores:", cv_scores)
print("Average cross-validation score:", np.mean(cv_scores))

final_model = create_cnn_model()
final_model.fit(X_train, y_train, epochs=5, batch_size=32)
y_pred_prob = final_model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)
# Classification Report
classification_report = metrics.classification_report(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.3f}")
print("Clasification Report:\n", classification_report)
```

APPENDIX 12 – Data Analysis for Authorship Patterns

Appendix 12 presents the code utilized for generating word clouds and conducting data analysis, which played a crucial role in exploring and interpreting the author´s patterns.

```
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import nltk
from nltk.corpus import stopwords

# Read TSV
data = pd.read_csv('/content/drive/MyDrive/TESE/Author_Identification/authors.tsv', sep='\t')

# Rename the columns
data = data.rename(columns={'0': 'Drop1', '@BillGates': 'Author_Name', 'a': 'Drop2', 'I was honored to speak at today’s #LeadersClimateSummit about the three things we need to do to avoid a climate disaster. ': 'Tweet'})

# Drop the unnecessary columns
data.drop(columns=['Drop1', 'Drop2'], inplace=True)

# Add the first line for Bill Gates
new_row_df = pd.DataFrame({'Author_Name': '@BillGates', 'Tweet': ["I was honored to speak at today’s #LeadersClimateSummit about the three things we need to do to avoid a climate disaster"]})

# Append the new row to the DataFrame
data = pd.concat([data, new_row_df], ignore_index=True)

# Display DF
display(data.head())

# Shape
data.shape

# Details
data.groupby('Author_Name').size()

data['Word_Count'] = data['Tweet'].apply(lambda x: len(str(x).split()))

# Average Word Count
```

```
avg_word_count_per_author =
data.groupby('Author_Name')['Word_Count'].mean().reset_index(name='Average_Wor
d_Count')

# Result
print(avg_word_count_per_author)

data['Char_Count'] = data['Tweet'].apply(lambda x: len(str(x)))

# Max and Min character Count
char_count_stats_per_author = data.groupby('Author_Name')['Char_Count'].agg(['max',
'min']).reset_index()

char_count_stats_per_author.columns = ['Author_Name', 'Max_Char_Count',
'Min_Char_Count']

# Results
print(char_count_stats_per_author)


nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Remove stopwords
def remove_stopwords(tweet):
    words = tweet.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

data["Tweet"] = data["Tweet"].apply(remove_stopwords)
all_tweets_text = ' '.join(data["Tweet"])

### Removed the word "hppts" as it doesn´t include any meaning in the wordclouds
data["Tweet"] = data["Tweet"].str.replace("https", "", regex=False)


# Puting the names better

data["Author_Name"]=data["Author_Name"].replace('@BillGates','BillGates')
data["Author_Name"]=data["Author_Name"].replace('@CNN','CNN')
data["Author_Name"]=data["Author_Name"].replace('@neymarjr','NeymarJR')
data["Author_Name"]=data["Author_Name"].replace('@BarackObama','BarackObama')
data["Author_Name"]=data["Author_Name"].replace('@katyperry','KatyPerry')
data["Author_Name"]=data["Author_Name"].replace('@justinbieber','JustinBieber')
data["Author_Name"]=data["Author_Name"].replace('@rihanna','Rihanna')
```

```
data["Author_Name"]=data["Author_Name"].replace('@ArianaGrande','ArianaGrande')
data["Author_Name"]=data["Author_Name"].replace('@YouTube','YouTube')
data["Author_Name"]=data["Author_Name"].replace('@KimKardashian','KimKardashian')
data["Author_Name"]=data["Author_Name"].replace('@britneyspears','BritneySpears')
data["Author_Name"]=data["Author_Name"].replace('@ddlovato','DemiLovato')
data["Author_Name"]=data["Author_Name"].replace('@shakira','Shakira')
data["Author_Name"]=data["Author_Name"].replace('@jimmyfallon','JimmyFallon')
data["Author_Name"]=data["Author_Name"].replace('@Cristiano','CristianoRonaldo')


unique_authors=data["Author_Name"].unique()
print(unique_authors)

bill_gates = data[data["Author_Name"]=="BillGates"]
cnn = data[data["Author_Name"]=="CNN"]
neymar = data[data["Author_Name"]=="NeymarJR"]
barack_obama = data[data["Author_Name"]=="BarackObama"]
katy_perry = data[data["Author_Name"]=="KatyPerry"]
justin_bieber = data[data["Author_Name"]=="JustinBieber"]
rihanna = data[data["Author_Name"]=="Rihanna"]
ariana_grande = data[data["Author_Name"]=="ArianaGrande"]
youtube = data[data["Author_Name"]=="YouTube"]
kim_k = data[data["Author_Name"]=="KimKardashian"]
britney_spears = data[data["Author_Name"]=="BritneySpears"]
demi_lov = data[data["Author_Name"]=="DemiLovato"]
shakira = data[data["Author_Name"]=="Shakira"]
jimmy_fallon = data[data["Author_Name"]=="JimmyFallon"]
cristiano_ronaldo = data[data["Author_Name"]=="CristianoRonaldo"]
#### Word Clouds generated

# Bill Gates
combined_bill = ' '.join(bill_gates['Tweet'])

bill_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_bill)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(bill_wc, interpolation='bilinear')
plt.title('Bill Gates')
plt.axis('off')
```

```
# CNN
combined_cnn = ' '.join(cnn['Tweet'])

cnn_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_cnn)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(cnn_wc, interpolation='bilinear')
plt.title('CNN')
plt.axis('off')


# Neymar
combined_neymar = ' '.join(neymar['Tweet'])

neymar_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_neymar)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(neymar_wc, interpolation='bilinear')
plt.title('Neymar')
plt.axis('off')

# Barack Obama
combined_barack = ' '.join(barack_obama['Tweet'])

barack_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_barack)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(barack_wc, interpolation='bilinear')
plt.title('Barack Obama')
plt.axis('off')

# Katy Perry
combined_katy = ' '.join(katy_perry['Tweet'])

katy_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_katy)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
```

```
plt.imshow(katy_wc, interpolation='bilinear')
plt.title('Katy Perry')
plt.axis('off')

# Justin Bieber
combined_bieber = ' '.join(justin_bieber['Tweet'])

bieber_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_bieber)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(bieber_wc, interpolation='bilinear')
plt.title('Justin Bieber')
plt.axis('off')

# Rihanna
combined_rihanna = ' '.join(rihanna['Tweet'])

rihanna_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_rihanna)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(rihanna_wc, interpolation='bilinear')
plt.title('Rihanna')
plt.axis('off')

# Ariana Grande
combined_ariana = ' '.join(ariana_grande['Tweet'])

ariana_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_ariana)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(ariana_wc, interpolation='bilinear')
plt.title('Ariana Grande')
plt.axis('off')

# Youtube
combined_yt = ' '.join(youtube['Tweet'])

yt_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_yt)
```

```
plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(yt_wc, interpolation='bilinear')
plt.title('Youtube')
plt.axis('off')

# Kim Kardashian
combined_kim = ' '.join(kim_k['Tweet'])

kim_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_kim)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(kim_wc, interpolation='bilinear')
plt.title('Kim Kardashian')
plt.axis('off')

# Britney Spears
combined_bp = ' '.join(britney_spears['Tweet'])

bp_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_bp)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(bp_wc, interpolation='bilinear')
plt.title('Britney Spears')
plt.axis('off')

# Demi Lovato
combined_demi = ' '.join(demi_lov['Tweet'])

dl_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_demi)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(dl_wc, interpolation='bilinear')
plt.title('Demi Lovato')
plt.axis('off')

# Shakira
combined_shakira = ' '.join(shakira['Tweet'])
```

```
shakira_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_shakira)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(shakira_wc, interpolation='bilinear')
plt.title('Shakira')
plt.axis('off')

# Jimmy Fallon
combined_jimmy = ' '.join(jimmy_fallon['Tweet'])

jimmy_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_jimmy)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(jimmy_wc, interpolation='bilinear')
plt.title('Jimmy Fallon')
plt.axis('off')

# Cristiano Ronaldo
combined_ronaldo = ' '.join(cristiano_ronaldo['Tweet'])

ronaldo_wc = WordCloud(width=800, height=400,
background_color='white').generate(combined_ronaldo)

plt.figure(figsize=(18, 9))
plt.subplot(2, 3, 1)
plt.imshow(ronaldo_wc, interpolation='bilinear')
plt.title('Cristiano Ronaldo')
plt.axis('off')
```