PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

SAAD DAHLEB BLIDA UNIVERSITY

FACULTY OF SCIENCES

COMPUTER SCIENCE DEPARTMENT

End of studies project with a view to obtaining the

Master's degree in Computer systems and networks

Theme

Authorship Verification From Text

Presented by:

- Mahi Abdelkrim
- Menouer Ramzi

Supervisor:

- Dr. Amina Madani

Co-Supervisor:

- Dr. Fatima Boumahdi

The jury:

- Dr. Mahfoud Bala
- Dr. Khaddija Midoun

Year: 2020-2021

Acknowledgments

First of all, we would like to express our gratitude to my first supervisor Doctor Madani Amina whose expertise, understanding, patience and motivation enriched considerably in our graduating experience. For giving us precious advice and trusting our capabilities from the beginning. Her insightful comments and constructive criticism were always thought provoking and useful at the different stages of our academic attendance. We thank Doctor Boumahdi Fatima also for all her support, patience and help throughout the research. Without their guidance, this research project would never have materialized.

Furthermore, we ought to extend our gratitude to Saad Dahleb University of Blida for the support it has provided through the duration of our academic career. Our fellow students also deserve special mention for all these exchanges of knowledge and skills between us.

Moreover, we would like to express our thankfulness to our family for their unconditional support and full trust that makes it possible for us to strive for our own dreams.

Abstract

Authorship verification (AV) is one of various topics parts of authorship analysis field that deals with the problem of determine whether two texts were written by the same author or not. A combination of a similarity-based methods and relevant linguistic features are used to achieve high accuracy authorship verification. To address this problem, we proposed a new approach using the Autoencoder deep learning method. Challenges in the context of Authorship verification have greatly increased in recent years, as the challenge of PAN (series of scientific events) for three last years from 2020 to 2022. To experiment our approach, we used the data provided by PAN 2021 AV task.

Keywords: Authorship verification, PAN, Autoencoder, deep learning, similarity.

Résumé

La vérification de l'auteur est une thématique faisant partie du domaine de l'analyse de l'auteur, qui aborde le problème de déterminer si deux textes ont été écrits par le même auteur ou non. Une collection de méthodes de remplacement sur la similitude et de traits linguistiques pertinentes sont utilisées pour obtenir une vérification d'auteur de haute précision. Pour résoudre ce problème, nous avons proposé une nouvelle approche utilisant la méthode d'apprentissage en profondeur Autoencoder. Les défis dans le contexte de la vérification de l'auteur ont augmenté ces dernières années, comme le défi du PAN à travers les trois dernières années de 2020 à 2022. Pour expérimenter notre approche, nous avons utilisé les données fournies par la tâche PAN 2021 AV.

Mots Clé: Vérification de l'auteur, PAN, Autoencoder, Apprentissage en profondeur, la similitude.

ملخص

التحقق من التأليف هو جزء من مجال تحليل التأليف، والذي يعالج مشكلة تحديد ما إذا كان نصان قد كتبهما نفس المؤلف. يتم استخدام مجموعة من طرق التشابه البديلة والسمات اللغوية ذات الصلة لتحقيق التحقق من التأليف بدقة عالية. لحل هذه المشكلة، اقترحنا طريقة جديدة باستخدام طريقة التعلم العميق Autoencoder. في السنوات الأخيرة زادت التحديات بشكل كبير في سياق التحقق من التأليف، كتحدي PAN في السنوات الثلاث الماضية من 2020 إلى 2022. لتجربة طريقتنا، استخدمنا البيانات المقدمة من مهمة PAN 2021. AV.

الكلمات المفتاحية: التحقق من التأليف،PAN ،Autoencoder ، التعلم العميق، التشابه.

Contents

List of Figures

List of Tables

General Introduction

The world has changed over the past few years, «the Internet is growing exponentially with a vast amount of data every day. Such a high growth rate brings problems like fraudulent, stolen, or unidentified data. These problems can be very dangerous in places like the government sector, public websites, forensics, and schools. Because of these risks, we must do detection of truth for which it is necessary to analyze the authorship of a text»[1], to reduce the occurrence of these problems. Authorship Analysis is one such technique, which is used to find the authorship of a text.

Authorship Analysis is categorized as three classes. One of these three types is the authorship verification. It analyses two or more text to determine whether they were written by the same author or not. It has a wide range of applications in fraud and plagiarism detection problems. As known examples of authorship verification problems, we review fraud emails and unknown data. where authorship verification may help us to reduce these risks[2]. The question that we ask is how can we verify the authorship of text?

In this field of authorship verification, there are a lot of research that have been carried out using the artificial intelligence methods as machine learning and deep learning. This field has become a huge challenge recently specially in PAN from 2020 to 2022 at CLEF (Conference and Labs of the Evaluation Forum). PAN2021 provided the same dataset of the last year PAN2020, which consists of pairs of (snippets from) different fanfics, the goal of the task is to find new approaches in the authorship verification.

We propose a new approach to solve the problem of authorship verification using the Autoencoder deep learning method, which achieved good results. We start our approach by extract the features of our dataset in two steps. The first is preprocessing and the second is the vectorization using Word2vec method. After that, we start the text classification using our Autoencoder method, we have to split the dataset into three-part “training” and “validation” and “test”.

We used the dataset provided by PAN 2020 in our experimentation

The structure of this document is organized as follows:

Chapter 01: We start by presenting the artificial intelligence and machine learning and we go directly to present the deep learning and its basics, then we move on to the different deep learning techniques used in the text classification.

Chapter 02: The second chapter is the state of the art presenting the works related to the authorship verification problem.

Chapter 03: In the third chapter, we will present our new model and we dive deep explaining each of its parts that we used to build it.

Chapter 04: In the last chapter, we will present the different tools used to build our model. At the end, we share the results we have achieved and compare them to other works.

Chapter 01: Deep Learning

Introduction

In the last years, artificial intelligence (AI) has been a trend in all fields. Artificial intelligence (has many subfields and machine learning is one of them, the evolution of machine learning has led to significant advancements and improvements in the way we interact with our world. One of these exciting advancements is Deep Learning, which is driving today's AI explosion. Therefore, in this chapter we will introduce the deep learning part. Then we will discuss the different famous methods and techniques of deep learning.

1. Artificial intelligence

There is no consensus about the definition of AI. In broad terms, Ryan Calo [3] conceptualizes it as "a set of techniques aimed at approximating some aspect of human or animal cognition using machines". In colloquial terms, AI applications intend to automate human tasks through the use of machines, in a faster, more accurate, and safe manner than when the tasks performed by humans. AI applications even go as far as performing tasks that are not possible for humans to do, due to our biological limitations. AI systems range from ordinary applications (e.g., cell phone voice assistants) to very sophisticated systems capable of driving cars, performing medical diagnostics, profiling people, or even controlling entire sectors in a given industry. In fact, the term AI encompasses a wide range of techniques in many scientific areas, especially computer sciences. There are various subfields such as robotics, machine learning, neural networks, computer vision, facial and speech recognition.

Furthermore, artificial Intelligence is a field in computer science that is concerned with the automation of intelligence and the enablement of machines imitate humans behaviors and actions and to achieve complex tasks in complex environments[4].

2. Machine learning

The term "machine learning" was coined by Arthur Samuel in 1959 [5], an American pioneer in the field of computer gaming and artificial intelligence, and stated "it gives computers the ability to learn without being explicitly programmed."

So let us start to answer a few good questions: what is machine learning? In addition, what is the difference between traditional programming and machine learning? It is easy to get the difference between them as follows:

- **Traditional programming:** In traditional programming, we have a box that has two inputs (Input, Rule) and the traditional model generates the output based on the rule that we add. Figure 1 shows the model of a traditional programming diagram.

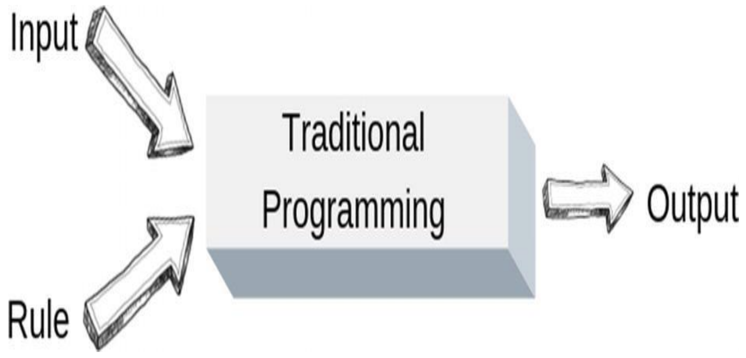


Figure 1: Traditional programming diagram[5].

- **Machine learning:** In machine learning, we have a box that has two inputs (Input, Output) and the machine-learning model trains to get the rule that generates the output from input.
  Figure 2 present the machine learning programming model that shows how it differs from traditional programming [5].

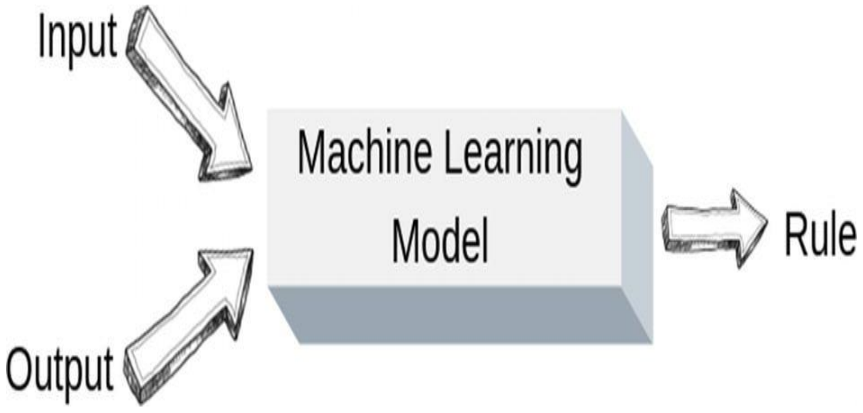


Figure 2: The machine-learning diagram [5]**.**

3. Machine learning methods

Machine learning is classified into four categories based on the kind of dataset experience. A model is allowed as follows [6]: supervised learning (SL), unsupervised learning (UL), semi-supervised learning (SSL), and reinforcement learning (RL). We briefly discuss each of these machine-learning paradigms.

3.1 Supervised learning

Supervised machine learning is the major category of machine learning that drives a lot of applications and value for businesses. In this type of learning, the model trained on the data for which we already have the correct labels or outputs. In short, we try to map the relationship between input data and output data in such a way that it can generalize well on unseen data as well, as shown in Figure 3. The training of the model takes place by comparing the actual output with the predicted output and then optimizing the function to reduce the total error between the actual and predicted [7].

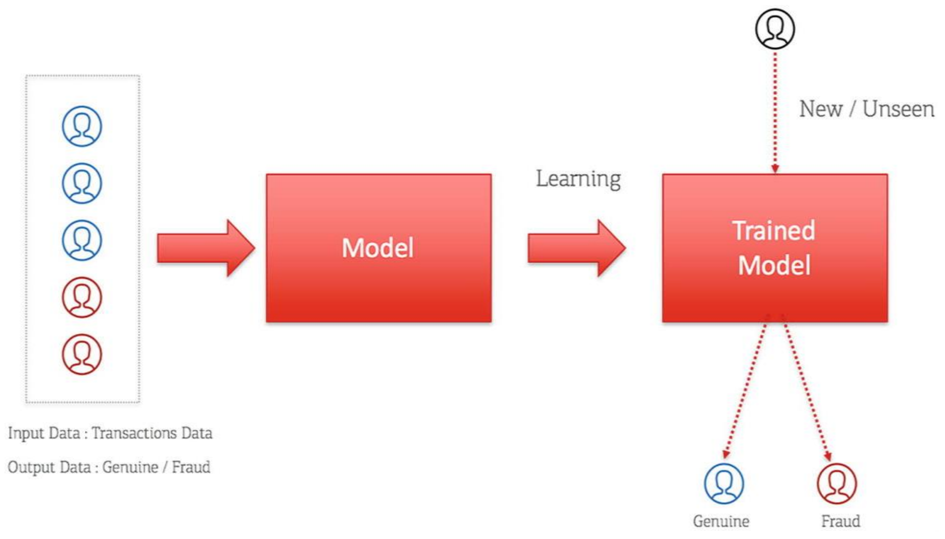


Figure 3: Supervised Learning training model [7].

3.2 Unsupervised Learning

Unsupervised Learning is a class of machine learning techniques to find the patterns in data. The data given to unsupervised algorithms are not labeled, which means only the input variables are given with no corresponding output variables. In unsupervised learning, the algorithms are left to themselves to discover interesting structures in the data, where we have only input data and no corresponding output variables. The easy definition is that in unsupervised learning we wish to learn the inherent structure of our data without using explicitly provided labels.

However, why do they call it unsupervised learning? They called it unsupervised learning because unlike supervised learning, there are no given correct answers and the machine itself finds the answers. In Figure 4, we give a representation of unsupervised learning.

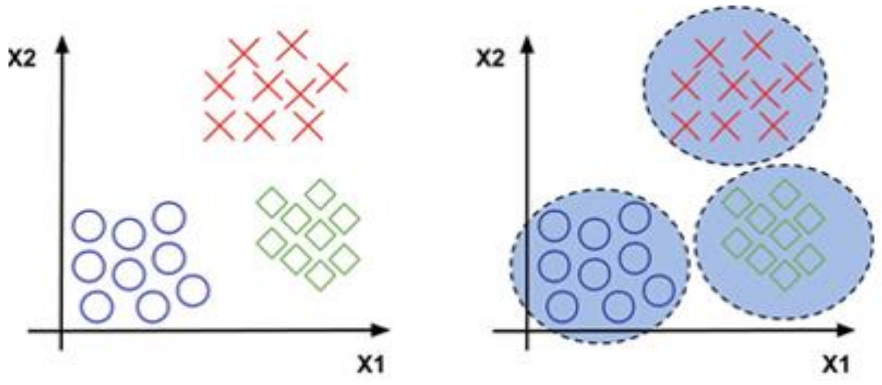


Figure 4: How an unsupervised learning algorithm clusters data into groups or zonesSemi supervised Learning.

When we have a problem with a large amount of input data and only some of the data labeled, this is called a semi-supervised learning problem. These problems sit in between supervised and unsupervised learning.

How does it work? We can use unsupervised learning techniques to discover and learn the structure in the input variables, then we use just a few labeled data as supervised learning technic to make the model get good predictions for the unlabeled data, feed that labeled data as training data. After that we test the model unknown data and make predictions[5].

In table 1, we present the difference between the three approaches of Machine Learning.

Supervised learning	Unsupervised Learning	Semi supervised Learning
The data labeled and the algorithms learn to predict the output from the input data.	The data is unlabeled and the algorithms learn the inherent structure from the input data.	Just Some data labeled but most of it is unlabeled, and a mixture of supervised and unsupervised techniques can be used.

Table 1 :The difference between the three Approaches of Machine Learning [5].

3.3 Reinforcement Learning

The reinforcement learning is a type of machine learning methods, it is a bit different from conventional supervised or unsupervised method. First, in reinforcement learning there is an agent that we want to train over a period to interact with a specific environment and improve its performance with regard to the type of actions it performs on the environment. Moreover, for interacting with the specific environment the agent starts with a set of strategies or policies. On observe the current state of the environment, it takes a particular action based on a specific rule. Based on the action, the agent gets a reward that could be beneficial or detrimental. The iterative process continues until it learns enough about its environment, and It updates its current policies and strategies if needed[8].

A suitable reinforcement learning methodology has been described in Figure 5.

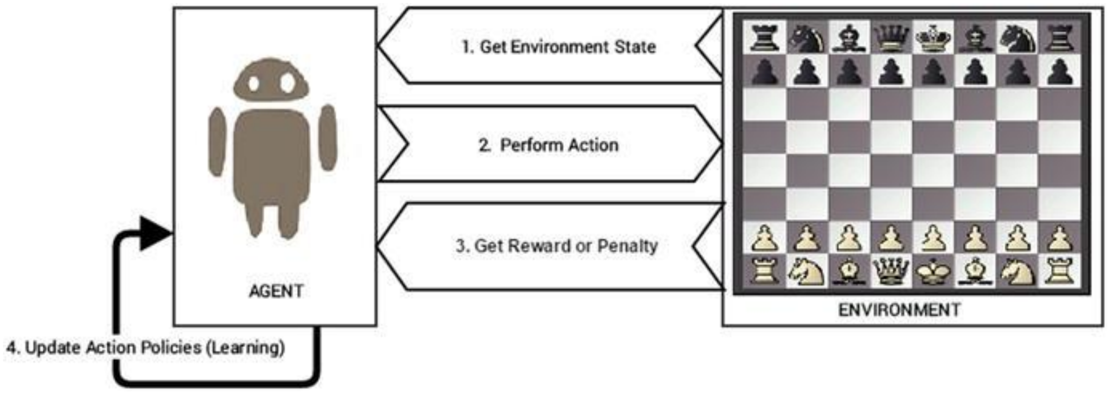


Figure 5: Reinforcement learning training a robot to play chess [8].

4. Deep learning

4.1 From Machine Learning to Deep Learning

We now know and understand that machine learning is a subset of artificial intelligence, and deep learning is a subset of machine learning. Therefore, every machine-learning program is under the category of AI programs but not vice versa. The question then is if the approaches of machine learning and AI are the same. The answer is yes, because every machine-learning problem is an AI problem and deep learning is a subset of machine learning. We should keep in mind that deep learning is nothing more than methods that enhance machine learning algorithms to be more accurate and make some stages easy, like feature extractions, etc. The easiest takeaway for understanding the difference between machine learning and deep learning is to remember that deep learning is a subset of machine learning [5].

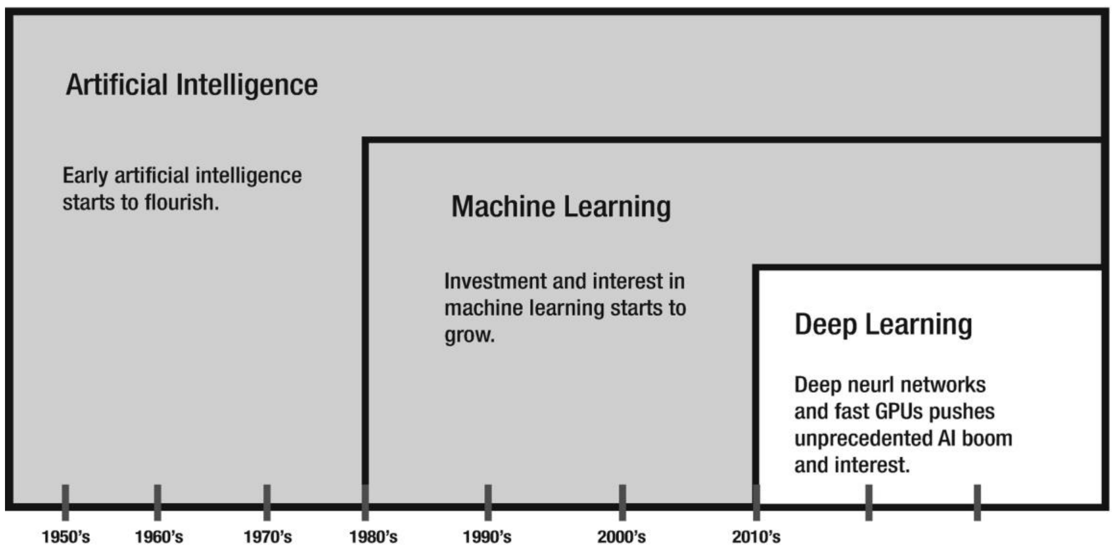

Artificial Intelligence
Early artificial intelligence starts to flourish.
Machine Learning
Investment and interest in machine learning starts to grow.
Deep Learning
Deep neurl networks and fast GPUs pushes unprecedented AI boom and interest.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Figure 6: Relationship between AI, machine learning, and deep learning [9].

4.2 What is deep learning?

The term "deep learning" is somewhat ambiguous. In many circles, deep learning is a re-branding term for neural networks or is used to refer to neural networks with many consecutive (deep) layers. Moreover, it is a system learned via neural networks without being guided by humans [10].

This type of system allows for processing huge amounts of data (big data) to get relationships and patterns that humans are often unable to detect or to observe [11]. The word "deep" refers to the number of hidden layers in the neural network. Furthermore, a deep learning architecture is a multilayer stack of simple modules with nonlinear input–output mappings, which are subject to learning. Each module in the stack transforms its input to increase both the selectivity and the invariance of the representation. Deep neural networks are multilayer networks with many hidden layers [12] (see Figure 7).

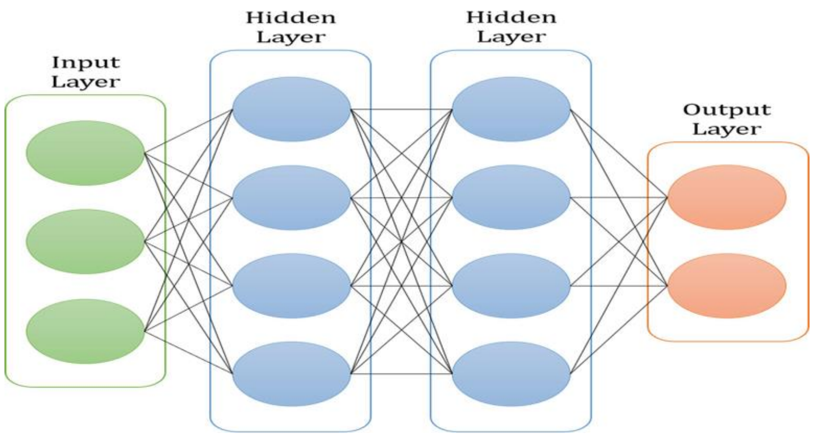


Figure 7: Neural network with two hidden layers [10].

4.3 Deep learning Technics

4.3.1 Perceptron

A Perceptron is the smallest layer in the neural network. It is a linear classifier (binary), and it is used in supervised learning. It helps to classify the given input data to get the output value. Since the inputs are fed directly to the output via the weights, the perceptron can be considered the simplest kind of feedforward network [13].

4.3.2 Convolutional Neural Networks

A convolutional neural network (convent or CNN for short) is a type of neural network that is particularly good at analyzing images (although they can be applied to audio and text data). The neurons in the layers of a convolutional neural network are arranged in three dimensions: height, width, and depth. CNNs use convolutional layers to learn local patterns in its input feature space (images) such as textures and edges. In contrast, fully connected (dense) layers learn global patterns. The neurons in a convolutional layer are only connected to a small region of the layer preceding it instead of all of the neurons, as is the case with dense layers. A dense layer’s fully connected structure can lead to an extremely large number of parameters that are inefficient and could quickly lead to overfitting.

4.3.2.1 Convolutional Neural Network Architecture

Convolutional neural networks consist of several layers, each trying to identify and learn various features. The main types of layers are convolutional layer, pooling layer, and fully connected layer. The layers can be classified into two main categories: feature detection layers and classification layers. Figure 8 display what a typical CNN architecture looks like [9].

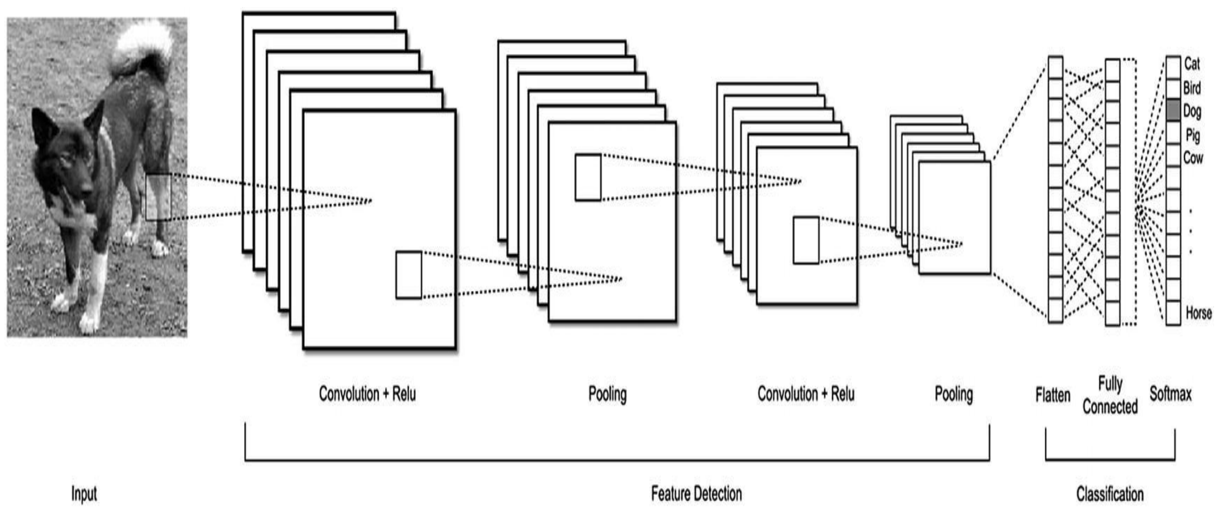


Figure 8: A CNN architecture for classifying animal images [9].

A. Feature Detection Layers

➢ Convolutional Layer

The convolutional layer is the primary building block of a convolutional neural network. The convolutional layer runs the input images or texts through a series of convolutional kernels or filters that activate certain features from the input images or texts. A convolution is a mathematical operation that performs element-wise multiplication at each location where the input element and the kernel element overlap as the kernel slides (or strides) across the input feature map.

➢ Rectified Linear Unit (ReLU) Activation Function

It is common practice to include an activation layer after each convolutional layer. The activation function can be also specified through the activation argument supported by all forward layers. Activation functions convert an input of a node to an output signal that is used as input to the next layer. Activation functions make neural networks more powerful by allowing it to learn more complex data such as images, audio, and text and introducing nonlinear properties to the neural network.

➢ Pooling Layer

Pooling layers reduce computational complexity and the parameter count by shrinking the dimension of the input image or texts. By reducing dimensionality, pooling layers also help control overfitting. It is also common to insert a pooling layer after every convolution layer.

There are two main kinds of pooling: average pooling and max pooling. Average pooling uses the average of all values from each pooling region while max pooling uses the maximum value.

B. Classification Layers

- **Flatten Layer**

The flatten layer converts a two-dimensional matrix into a one-dimensional vector before the data is fed into the fully connected dense layer.

- **Fully Connected (Dense) Layer**

The fully connected layer, also known as the dense layer, receives the data from the flatten layer and outputs a vector containing the probabilities for each class.

- **Dropout Layer**

A dropout layer randomly deactivated some of the neurons in the network during training. Dropout is a form of regularization that helps reduce model complexity and prevents overfitting.

- **Softmax and Sigmoid Functions**

The final dense layer provides the classification output. It uses either a softmax function for multiclass classification tasks or a sigmoid function for binary classification task [9].

4.3.3 Activation Functions

The activation functions are the main parts in the artificial neural networks layers, we use it in two steps for the hidden layers and for the output layer. In the hidden layers to control how well the network architecture learns the training dataset, and in the output layer to define the type of prediction results[1].

There is four activation functions types we will define them as follow:

4.3.3.1 Sigmoid

The sigmoid function is a useful activation for a variety of reasons. This function acts as a continuous squashing function those bounds its output in the range (0, 1). It is similar to the

[1] https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

step function but has a smooth, continuous derivative ideal for gradient descent methods. The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

It is also zero-centered, creating a simple decision boundary for binary classification tasks, and the derivative of the sigmoid function is mathematically convenient:

$$\sigma' = \sigma(x)\big(1 - \sigma(x)\big)$$

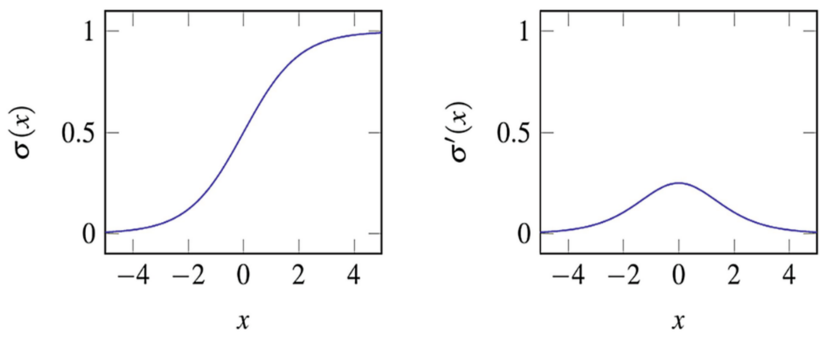


Figure 9: Sigmoid activation function and its derivative [10].

4.3.3.2 Tanh

The tanh function is very similar to the sigmoid function. The only difference is that it is symmetric around the origin. The range of output values in this case is$(-1,1)$. Therefore, the inputs to the next layers will not always be of the same sign[2]. The tanh function is defined as:

$$tanh(x) = 2 * \sigma(2x) - 1$$

In addition, neural networks with tanh functions converge faster than sigmoid functions. Moreover, the neural networks with tanh activation functions have lower classification error than those with sigmoid activation functions.

[2] https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

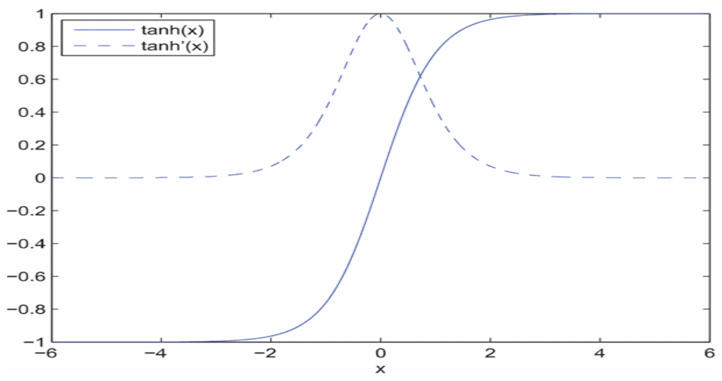


Figure 10: Tanh activation function and its derivative.

However, the calculation of the derivatives of hyperbolic tangent functions, listed as follows, is more complicated than the sigmoid function. Moreover, it has the same soft saturation as sigmoid function, which also has the vanishing gradient problem [14].

4.3.3.3 ReLU

ReLU stands for rectified linear unit and is a non-linear activation function, which is widely used in neural networks. The upper hand of using ReLU function is that not all the neurons are activated at the same time. This implies that a neuron will be deactivated only when the output of linear transformation is zero. It can be defined mathematically as:

$$f(x) = max(0, x)$$

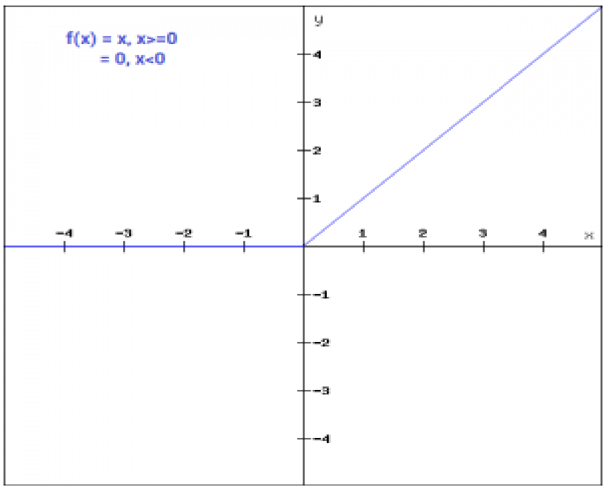


Figure 11: ReLU activation function plot.

ReLU is more efficient than other functions, because not all the neurons are activated at the same time, rather a certain number of neurons are activated at a time. In some cases, the value of gradient is zero, due to which the weights and biases are not updated during backpropagation step in neural network training [15].

4.3.3.4 SoftMax

The squashing concept of the sigmoid function extends to multiple classes by way of the softmax function. The softmax function allows us to output a categorical probability distribution over K classes.

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

We can use the softmax function to produce a vector of probabilities according to the output of that neuron. In the case of a classification problem, that has $K = 3$ classes, the final layer of our network will be a fully connected layer with an output of three neurons. If we apply the softmax function to the output of the last layer, we get a probability for each class by assigning a class to each neuron. The softmax computation is shown in Figure 12.

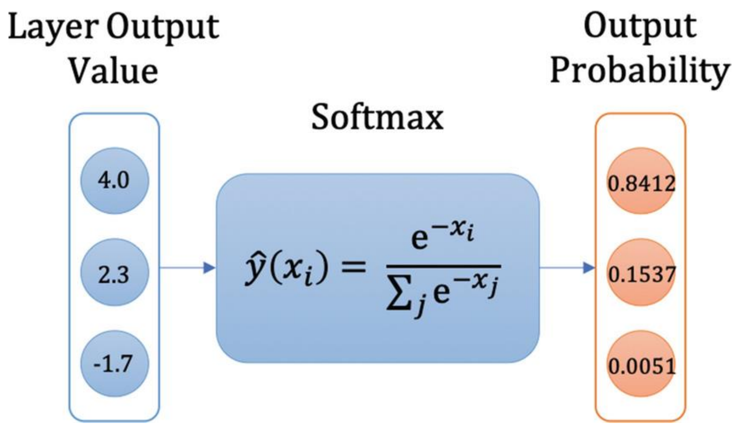


Figure 12: The softmax computation.

The softmax probabilities can become very small, especially when there are many classes and the predictions become more confident. Most of the time a log-based softmax function is used to avoid underflow errors. The softmax function is a particular case for activation functions. It is rarely seen as an activation that occurs between layers.

Therefore, the softmax is often treated as the last layer of a network for multiclass classification rather than an activation function [10].

4.3.4 Recurrent Neural Networks (RNNs)

RNNs are used in deep learning and in the development of models that simulate the activity of neurons in the human brain. They are especially powerful when it is critical to predict an outcome and are distinct from other types of artificial neural networks because they use feedback loops to process a sequence of data that informs the final output, which can also be a sequence of data. These feedback loops allow information to persist; the effect is often described as memory. The logic behind a RNN is to consider the sequence of the input [16].

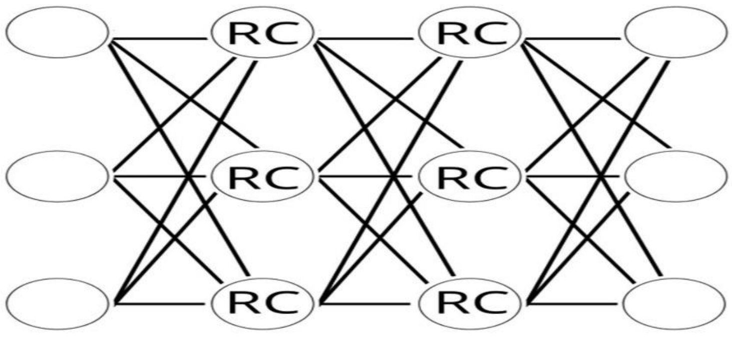


Figure 13: Recurrent neural network [16].

4.3.5 Long short-term memory (LSTM)

Long short-term memory (LSTM) is a neural network architecture, it is a type of recurrent neural network (RNN) deep learning, it is for the management of long sequential data, and LSTM has feedback connections. LSTM use an efficient, gradient-based algorithm, can make it learn a bridge time intervals in excess of 1000 steps, without loss the possibility of short time lag. [17].

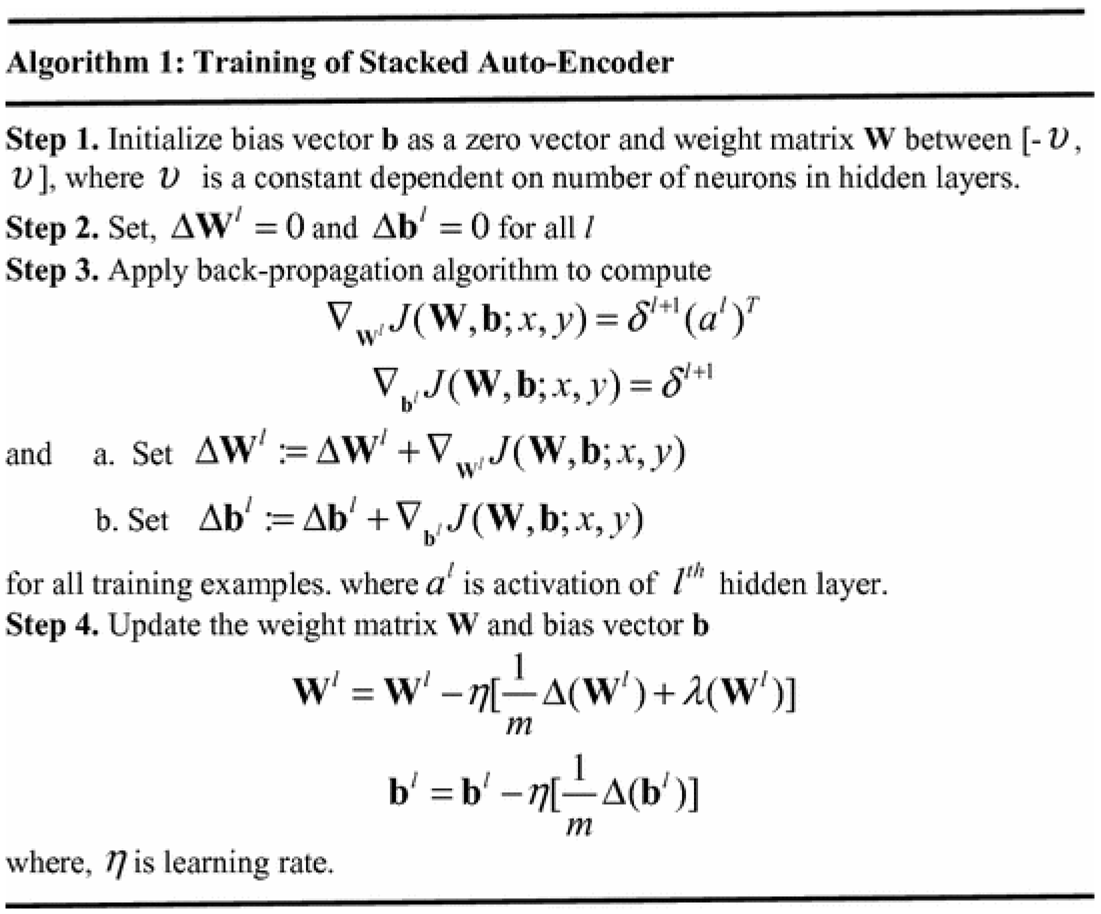
4.3.6 Auto-Encoder (AE)

Auto-encoder is a type of artificial neural network, which can be used to learn a compressed representation of data. Simplest form of an auto-encoder comprises an input, an output and one or more hidden layers connected to them. Since, it does not use labels so this process is an unsupervised learning. An auto-encoder learns the hypothesis (prediction) function $\boldsymbol{hW, b(x) = x}$ with unlabeled data, where $\boldsymbol{W}$ and $\boldsymbol{b}$ are weight matrices and bias vectors of the network respectively. Given unlabeled training samples, the cost (error) function for an auto-encoder formulated as below:

$$J(\mathbf{W},\mathbf{b}) = \frac{1}{m}\sum_{i=1}^{N}\frac{1}{2}\left\|(h_{\mathbf{W},\mathbf{b}}(x^l) - \hat{x}^l)^2\right\| + \frac{\lambda}{2}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\mathbf{W}_{ji}^2 + \beta\sum_{j=1}^{s_l} KL(\rho \parallel \hat{\rho}_j)$$

Figure 14: The cost (error) function for an auto-encoder.

In the Equation that mentioned in Figure 14, $\boldsymbol{KL}$ is the Kullback-Leibler divergence function. $\boldsymbol{\lambda}$ Is regularization parameter used to minimize the problem of overfitting, $\boldsymbol{\rho}$ is sparsity parameter used to put restriction on hidden units, which minimize the dependency between features. $\boldsymbol{\beta}$ is a parameter that controls sparsity penalty term. The parameter $\boldsymbol{\rho\^j}$ is the average activation of hidden unit $\boldsymbol{j}$ and the activation function of a hidden unit implicitly depends on $\boldsymbol{W}$ and **b**. The aim is to minimize the cost function $\boldsymbol{J}$ $(\boldsymbol{W}, \boldsymbol{b})$ with respect to W and b to train our network. The back propagation (BP) algorithm is mainly used for computing the gradients by using batch gradient descent optimization for learning the weights of the network. The intuition for using back propagation is that for given training sample$(\boldsymbol{x}, \boldsymbol{y})$**,** we first run the forward propagation to compute the activation function of every node in all the layers of the complete network. Then, for every node $\boldsymbol{i}$ in layer **l**, compute the error $\boldsymbol{\sigma_i^{(l)}}$ that shows how much corresponding units (nodes) is responsible for errors at output of every layer in the network[18]. The training of stacked auto-encoder using back propagation is describing in the Figure below:



Algorithm 1: Training of Stacked Auto-Encoder

Step 1. Initialize bias vector **b** as a zero vector and weight matrix **W** between [- υ, υ], where υ is a constant dependent on number of neurons in hidden layers.
Step 2. Set, $\Delta \mathbf{W}^l = 0$ and $\Delta \mathbf{b}^l = 0$ for all l
Step 3. Apply back-propagation algorithm to compute

$$\nabla_{\mathbf{W}^l} J(\mathbf{W}, \mathbf{b}; x, y) = \delta^{l+1}(a^l)^T$$

$$\nabla_{\mathbf{b}^l} J(\mathbf{W}, \mathbf{b}; x, y) = \delta^{l+1}$$

and a. Set $\Delta \mathbf{W}^l := \Delta \mathbf{W}^l + \nabla_{\mathbf{W}^l} J(\mathbf{W}, \mathbf{b}; x, y)$

b. Set $\Delta \mathbf{b}^l := \Delta \mathbf{b}^l + \nabla_{\mathbf{b}^l} J(\mathbf{W}, \mathbf{b}; x, y)$

for all training examples. where a^l is activation of l^{th} hidden layer.
Step 4. Update the weight matrix **W** and bias vector **b**

$$\mathbf{W}^l = \mathbf{W}^l - \eta[\frac{1}{m}\Delta(\mathbf{W}^l) + \lambda(\mathbf{W}^l)]$$

$$\mathbf{b}^l = \mathbf{b}^l - \eta[\frac{1}{m}\Delta(\mathbf{b}^l)]$$

where, η is learning rate.

Figure 15: Training of stacked auto-Encoder Algorithm [18].

Moreover, the architecture of AutoEncoder shows in the figure below.

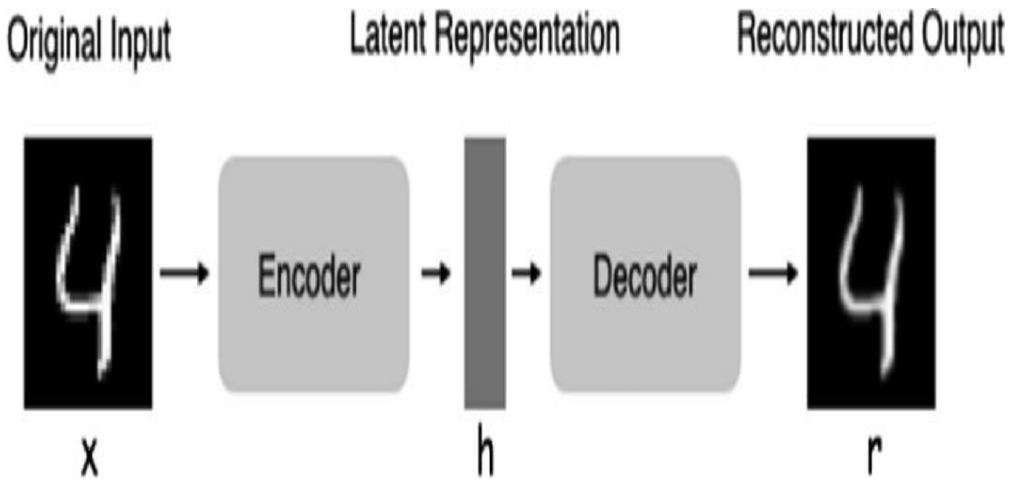


Figure 16: Architecture of Autoencoder[19]

Conclusion

In conclusion, deep learning has been and is used in many areas, including authorship, it has achieved a revolution and best performance to our area of the authorship verification. In the next chapter, we will see and discuss some works that have been done related to authorship verification using deep learning techniques.

Chapter 02: Authorship Verification: State of the art

Introduction

The Internet has being so wide with the huge of data generated every day. Such a high growth of data rate brings some problems like fraudulent or unidentified data. These problems can be dangerous. Because of these risks, the authorship analysis came to solve these problems. Authorship analysis has three basic types: authorship attribution, authorship profiling and authorship verification. In this chapter, we will define what it means authorship analysis and focus on authorship verification as a main task, and we will see some works related to the field of authorship verification that used different approaches to solve authorship problems.

1. Authorship Analysis Key Concepts

Authorship analysis is the operation of examining the features of a document or text like stories and scientific articles, in order to extract the conclusion on its authorship, to reduce scientific theft and the plagiarism. The origin of Authorship analysis return to the linguistic research field that name is stylometry, which indicate to statistical analysis of the writing literary style[20].

Authorship Analysis is a science of distinguishably between writing styles of authors by specify the features of the personality of the writers or authors and examining document and texts authored by them. Therefore, its goal is to determine biographic features of individuals and authors like age, gender, native language and based on the information that available to do the study of that individual or author[21]. The figure bellow shows that Authorship Analysis is the intersection between artificial intelligence, psychology and linguistics.

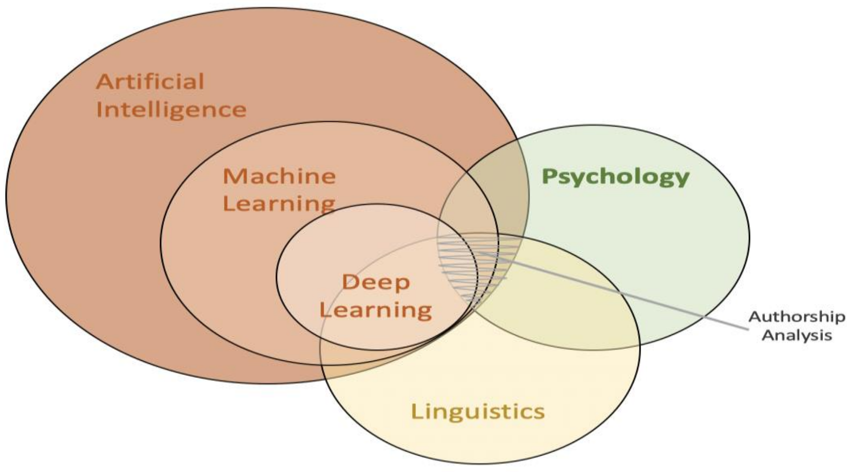


Figure 17: Authorship Analysis is a combination of Artificial Intelligence, Linguistics and Cognitive Psychology [21].

2. Types of Authorship Analysis Tasks

The three primary types involved in Authorship Analysis are Authorship Attribution, Authorship Verification and Authorship Profiling. These types are summarized as follows:

2.1 Authorship Attribution

Authorship attribution is a text classification technique used to infer the authorship of a document. By identifying features of writing style in a document and comparing it to features from other documents, a computer gets a higher performance than a human in analysis of characteristics of text, it can make a determination of stylistic similarity and thus of the plausibility of authorship by any specific person. There are many applications, including education (plagiarism detection), forensic science (identifying the author of a piece of evidence such as a threatening letter), history (resolving questions of disputed works), and journalism (identifying the true authors behind pen names) [22].

In addition, the goal of authorship attribution or authorship identification in the context of online documents is to identify the true author of a disputed anonymous document. In forensic science, his /her fingerprint can uniquely identify an individual. Likewise, in cyber forensics, an investigator seeks to identify specific writing styles, called Wordprint or Writeprint, of potential suspects, and then use them to develop a model. The Writeprint of a suspect extracted from her previously written documents. The model applied to the disputed document to identify its true author among the suspects. In forensic analysis, the investigator is required to support her findings by convincing arguments in a court of law [23].

2.2 Authorship Profiling

Authorship profiling is becoming increasingly important in the current technological development around the world. Authorship profiling is the analysis of a given set of texts in an attempt to reveal the various characteristics of an author based on both stylistic and content-based features. The characteristics analyzed generally include age and gender, although recent studies have looked at other characteristics such as personality traits and occupation [24].

The applications of authorship profiling abound in forensics, security, and commercial settings. For example, authorship profiling can assist police in identifying a perpetrator of a crime when there are too few (or too many) specific suspects to consider [25]. Authorship profiling distinguishes between categories of authors by studying their social aspect, how language is involved or how the author can be distinguished from a psychological point of view. This information helps to determine aspects of profiling such as gender, age, native language, or personality type. Authorship profiling is an increasingly important problem, for applications in forensics, security, and marketing. From a forensic linguistics perspective, for example, one would like to learn about the linguistic profile of the author of a harassing text message (the language used by a certain type of people) and identify certain characteristics (language as evidence). From a marketing point of view, companies may be interested in learning about the demographics of people who love or hate their products, looking at blogs and online product reviews as a source of analysis [26].

2.3 Authorship verification

Authorship verification (AV) is an active research area of computational linguistics that can be expressed as a fundamental question of stylometry to decide if the same person wrote two texts or not. It has a wide range of applications in forensic linguistics and fraud and plagiarism detection. Among notable examples, we can name blackmail messages, false insurance claims or online reviews and opinion statements, where authorship verification may help us to reduce these risks [2].

Studies consider the problem of authorship verification as a similarity detection problem: to determine whether two texts were written by the same person or not without knowing the real author of the document [20]. Linguists who aim to uncover the authorship of anonymously written texts by inferring author-specific characteristics from the texts traditionally perform AV. So-called linguistic features represent such characteristics. They are derived from an analysis of errors (e.g. spelling mistakes), textual idiosyncrasies (e.g. grammatical inconsistencies) and stylistic patterns [27].

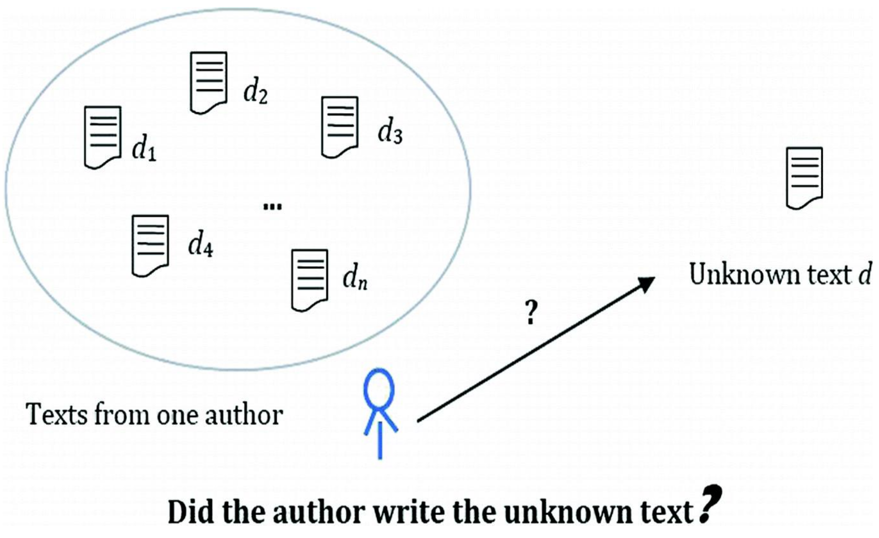


Figure 18: Authorship verification problem [1].

3. Related work

In this section, we will review some related works in the field of authorship verification using deep learning.

In PAN 2020, Ordoñez et al.,[28] proposed a neural network for learning discriminative features from the texts as well as the fandom from which the text derives. Their system uses the Longformer, a variant of transformer models that pre-trained on large amounts of text. This model combines global self-attention and local self-attention to enable efficient processing of long text inputs. Moreover, they augment the pre-trained Longformer model with additional fully connected layers and fine-tune it to learn features that are useful for author verification. Their model incorporates fandom information via the use of a multi-task loss function that optimizes for both authorship verification and topic correspondence. They used both of datasets small and large provided by PAN. They have compared their model with (CN^2) that used a convolutional stack followed by a recursive self-attention stack simultaneously. But, their Longformer-based system won and attained a 0.963 overall verification score, on a held-out subset of the PAN-provided “large training” set, but on the official PAN test set, attained a 0.685 overall score.

In the study[29], the author has proposed a simple model to solve the authorship verification problem based on a Labbé similarity. It is a simple text similarity technique used when facing with pairs of snippets. She proposed to select features by ranking them according to their frequency of occurrence in each text and taking only the most frequent ones (from 100 to 500) but including the most frequent ones in the underlying language. Such a representation strategy based on words used frequently by a given author; she used the dataset provided by PAN2020. It have shown a good performance with the small dataset of F1= 0.705 and AUC = 0.840.

Halvaniand et al.,[30] proposed in the context of the AV shared task at the PAN 2020 workshop an alternative approach, which considers only topic-agnostic features in its classification decision. They used as authorship verification the TAVeer approach, which was inspired by the methodology of biometric recognition systems. They aim to recognize individuals, based on a variety of physiological characteristics and behavioral features obtained from the hand, vein, fingerprint, face, eye, ear or voice. TAVeer employs an ensemble of distance-based classifiers, where each one aims to accept or reject the questioned authorship. Each classifier is provided with a category of stylistic features extracted from an individual linguistic layer (in each document). In this context, the Equal Error Rate (EER) algorithm serves as a thresholding mechanism. They used the small dataset, they have split the training data set into a training and validation set, where for the former only 5,000 verification cases were used (in other words, less than 10% of the entire data set). On the official test set, their approach ranked third out of all submitted approaches, attained a 0.825 overall score.

In the study [2], the author has proposed a data compression method based on the widespread Prediction by Partial Matching (PPM) algorithm extended with Context-free Grammar character preprocessing. He had chosen a Data compression algorithm CBC for his method drawing on the research by Halvani et al [31]. He used a context-free grammar preprocessing for PPM to reduce the overall length of a text and simplify the distribution of characters. He had used the EER algorithm on n text pairs with the equal number of positive (Y) and negative (N) authorship cases, to find a decision thresholdϴ, for which his data compression dissimilarity measure M gives us the score $sp = M(Tx, Ty)$. He had used the small data set that was proposed in PAN2020 in his experimentation. He used 2000 text pairs. His approach yielded better results, but could not break a "glass ceiling" of around 0.8 overall score.

In [32], the author has proposed to adapt Higher Criticism (HC) statistics for the problem of authorship verification as an unsupervised untrained discriminator of two documents. His method takes word-by-word p-values based on a binomial allocation model of words between the documents and combines these P-values to a single test statistic using HC. This method has two steps: in the first step, he performed many exact binomial tests: one test for each word in a prescribed dictionary, the result of each test is a P-value according to a binomial allocation model between the two documents. In the second step, he took the P-values resulting from the first step and combined them to a single score using the HC statistic. The performance of his method in the PAN2020 Authorship Verification shows that it serves as an effective authorship discriminator that requires very little tuning.

In their work[27], the authors have presented a hierarchical fusion of two well-known approaches into a single end-to-end learning procedure. The first is the ADHOMINEM system (Siamese network for representation learning).[33] It is used as a deep metric learning framework to measure the similarity between two text samples, to learn a pseudo-metric that maps a document of variable length onto a fixed-sized feature vector. In addition, at the top, they incorporate the second approach that is a probabilistic linear discriminant analysis (PLDA) layer[34], which functions as a pairwise discriminator to perform Bayes factor scoring in the learned metric space. They used the small and the large dataset provided by PAN. The small dataset model achieves only 0.897 overall score and the large one 0.935 overall score. Their approach ranked first out of all submitted approaches.

In PAN 2020, Araujo-Pino et al.,[35] have presented a deep learning Siamese network approach. Their neural architecture approach receives character n-grams as inputs. So in order to train their network, the first step is to transform the texts dataset into character n-grams (with n varying from 1 to 3) frequency vectors (the dimensions of the vectors correspond to the frequency of each n-gram) as input and learns to identify if these documents are written by the same author. They used the Scikit Learn Python module to extract all the n-grams. They used two datasets: large and small that were provided by The PAN 2020 authorship verification organization. They trained two models, one with the small dataset and the other with the large One, they only used the first 10000 (ten thousand) characters from each document to speed up the n-gram extraction process. Both datasets split on two training with 70 and validation with 30 percent of samples respectively. They used the parameters that achieved the best classification Performance in terms of AUC on 30% of both large and small training datasets. They achieved a good result that the AUC score is 1.0 when the Training and Validation sets come from the same dataset. On the other hand, the small dataset model achieves only 0.823 AUC on the validation set that comes from the large training set.

In the work [36], the author has proposed an approach to solve the problem of authorship verification of short messages. This approach is based on the deep sequence-to-sequence CNN model. She evaluated her approach on the Enron email dataset using the CNN classifier on a two-class training data, composed of positive (written by the "target" user) and negative (written by "someone else"). She trained her model to 52 remaining users. For each user, 1000 verified email messages were sampled, CNN model was trained for each user. 90% of this data used for training and 10% for testing. She achieved good results. The average overall accuracy is 97% for 52 users, which is significantly better than what most of the previously published works reported on the Enron dataset.

Weerasinghe et al.,[37] have proposed an approach to solve the problem of authorship verification for the PAN 2020 task. In their approach, they created two models (trained on the smaller and larger datasets). They used a Linear Regression classier for the smaller dataset and a Neural Network for the larger dataset. They extracted stylometric features from the documents and used the absolute difference between the feature vectors as input to their classier. These models achieved AUCs of 0.939 and 0.953 on the small and large datasets. Making them the second-best models on both datasets submitted to the shared task.

4. Comparison table

This table is presenting the main parts of all the previous work that are explained above based on:

- The dataset used in their work.
- The method used in the paper
- The results for the evaluation metrics used AUC, F1, c@1, F0.5u and the accuracy.

❖ **Evaluation metrics[3]:** The evaluation metrics that have been used in the comparison results define as follow:

  - **AUC (Area under the Curve):** Is the conventional area-under-the-curve score, it is an evaluation metric provided by scikit-learn library in Python language. AUC use to evaluate the deep learning models performance.

[3] https://pan.webis.de/clef21/pan21-web/author-identification.html

- **F1:** The F1-score, also called the F-score, is a measure of a model's accuracy on a dataset. It is used to evaluate binary classification systems, which classify examples into 'positive' or 'negative'[4].
- **c@1:** Is a variant of the conventional F1-score introduced by Peñas and Rodrigo (2011)[38]. It is well suited for evaluating Reading Comprehension tests, which able to reward systems that maintain correct answers and at the same time decrease the number of incorrect ones.
- **F0.5u:** Is a new proposed measure introduced by Bevendorff et al.,[39] which puts more emphasis on deciding same-author cases correctly. It want a balance between precision and recall, with more weight on precision.

The table is as follow:

Citation	Dataset	Methods	Metrics				
			AUC	c@1	F0.5u	F1	Accuracy
[28]	PAN large dataset	Longformer model	0.696	0.640	0.655	0.748	/
[32]	PAN small dataset	Higher Criticism (HC) statistics	0.866	0.801	0.815	0.809	/
[29]		The Labbé similarity	0.840	0.545	0.599	0.705	/
[30]		TAVeer (topic -agnostic feature)	0.878	0.796	0.819	0.807	/
[2]		Prediction by Partial Matching	0.786	0.786	0.809	0.800	/

[4] https://deepai.org/machine-learning-glossary-and-terms/f-score

		(PPM) algorithm					
[27]	PAN Large dataset	- Siamese network for representation learning. - Probabilistic linear discriminant analysis (PLDA)	0.969	0.928	0.907	0.936	/
	PAN small dataset		0.940	0.889	0.853	0.906	
[35]	PAN Large dataset	deep learning Siamese network for both dataset	0.859	0.751	0.745	0.800	/
	PAN small dataset		0.874	0.770	0.762	0.811	
[37]	PAN Large dataset	Neural Network for the larger dataset.	0.953	0.880	0.882	0.891	/

	PAN small dataset	Linear Regression classifier for the smaller dataset.	0.939	0.833	0.817	0.860	
[36]	Enron dataset	application of CNN to NLP	/	/	/	/	0.97

Table 2: Classification of some related works in the field of AV.

5. Discussion

From the table above we will discuss all the differences between all the methods and results of related works.

We have processed these works [28], [32], [29], [30], [2], [27], [35], [37], in the field of authorship verification, the first eight works realized in the last years in PAN 20 at CLEF (Conference and Labs of the Evaluation Forum). All of the works used the dataset provided by PAN 20, some works used both of dataset the smaller and the larger, and the others used just the smaller dataset.

We review that all the works used the machine learning methods and the artificial neural network algorithms for its models with the PAN Dataset, and got a good and acceptable results. Where this work [27], ranked the first in PAN20 challenge with the large dataset: AUC=0.969, F1_score = 0.928, c@1= 0.907, F.05=0.936, and the third with the small dataset: AUC=0.940, F1_score =0.889, c@1= 0.853, F.05=0.906.

However, it is not enough because machine learning is not effective for the massive data.

Moreover, we have processed this work [36], but it is independent about the others works, it realized with Enron dataset using the Convolutional Neural Network (CNN) deep learning method. It achieved good results with 0.97 accuracy, it ranked the first in all the work that used the Enron dataset in the field of authorship verification. This work inspired us to used deep learning to build a new model and get the best results between all the other works that used the PAN Dataset, for this reason we will propose a new method use the deep learning technic because in the last years, it has achieved a best results for the processing of massive data.

Moreover, the deep learning has the possibility to process a huge and complex data, these capabilities make it very powerful when treat with unstructured data. In other hand, deep learning has a starvation for data, for this reason it need a huge of unstructured data to be effective. For this purpose, it will help us process the data provided by PAN20 and achieved best results.

Conclusion

In this chapter, we have started by introducing the authorship analysis and focused on authorship verification as a type of it. After that, we have been presented the most relevant works related to our research in order to propose a new approach that will give better results in the field of authorship verification.

The next chapter describes our proposed model and the process we followed to design it, in detail.

Chapter 03: Conception of a new Authorship Verification model

Introduction

After reviewing the related work in this field, in this chapter we present the architecture of our model for authorship verification from text. Our model focuses on two main parts. In the first part, we extract the features of the texts in two steps: we start with the preprocessing of data after that we used word2vec for the vectorization of the text. In the second part, we split our dataset in three "train dataset", "validation dataset" and "test dataset". After that, we used the Autoencoder for the text classification part, with the "training dataset" to build our model.

1. Architecture of our model

This is the two main parts, each part with two steps. We will detail each part bellow.

- ➢ Part 1: Features Extraction.
  - Step 1: Preprocessing.
  - Step 2: Vectorization.
- ➢ Part 2: Text classification using Autoencoder.

In the previous works, researchers used many different methods of deep learning and machine learning like CNN to solve this problem of authorship verification. However, according to our research, we have worked in a method that used an Autoencoder for the authorship verification.

In the figure below (Figure 19), we present all the parts and steps of our model of authorship verification.

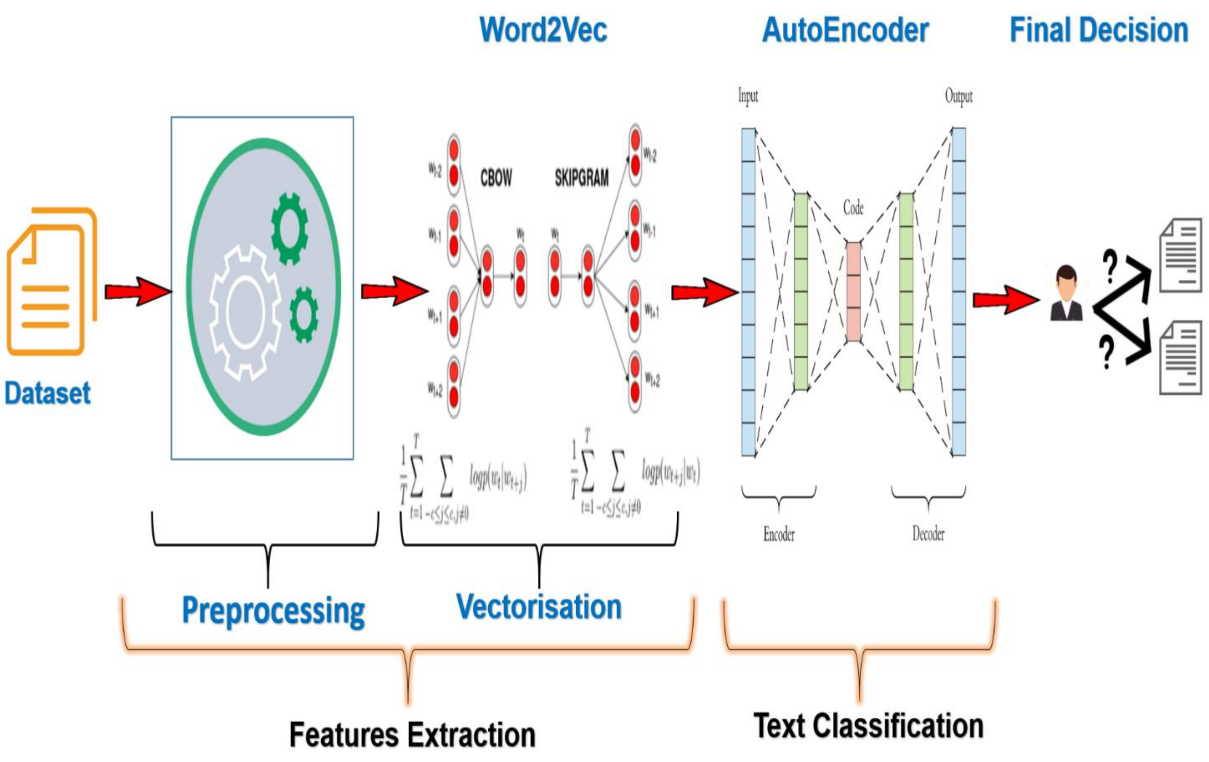


Figure 19: Architecture of our model of authorship verification.

2. Features Extraction

In this part, we want to extract all the features of the pair of text of dataset. This step present in two phases:

- Preprocessing
- Vectorization

2.1 Preprocessing

The main goal of this step is to reduce the text and cleaning it to simple words, and standardize it in order to make it easier to use. It can represent the words as keys that we can take it to give us the possibility to recognize and process the text.

There are some different tools and methods, which we used for the preprocessing of dataset, including:

- Remove the numbers and punctuation

- Remove capital letters
- Tokenization
- Remove the Stopwords
- Lemmatization

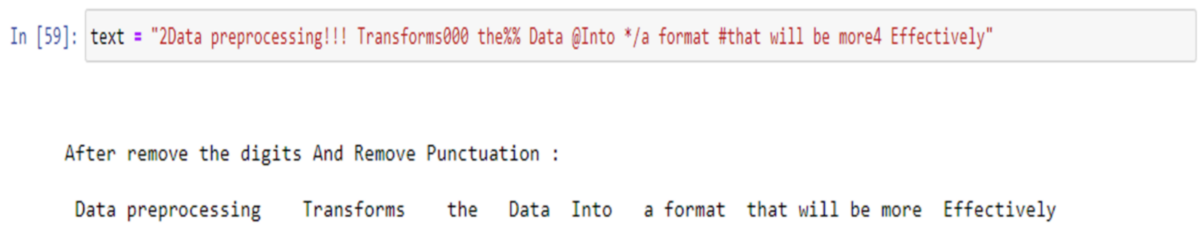
The techniques we used in this phase are as follows:

❖ Remove Punctuation

Punctuation has no effect on the text analysis so it is removed, that they offer no useful information for classification. It includes symbols words such as "/" and "@".

❖ Remove the numbers

The goal of removing numbers is to make words simple for processing.

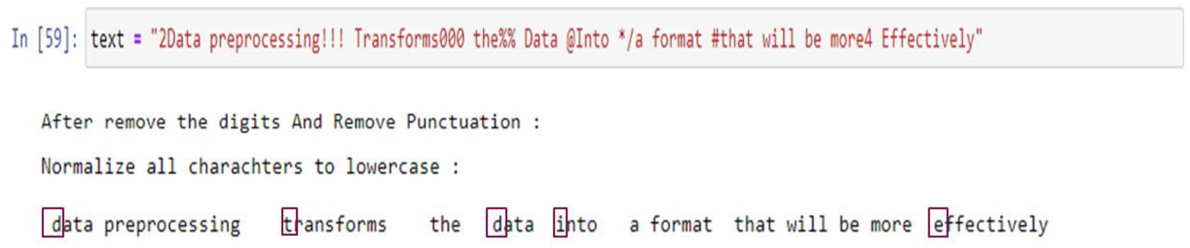


```
In [59]: text = "2Data preprocessing!!! Transforms000 the%% Data @Into */a format #that will be more4 Effectively"

After remove the digits And Remove Punctuation :

 Data preprocessing    Transforms    the   Data  Into   a format  that will be more  Effectively
```

Figure 20: An example of remove the digits and punctuation.

❖ Remove capital letters

We convert all uppercase characters to lowercase.

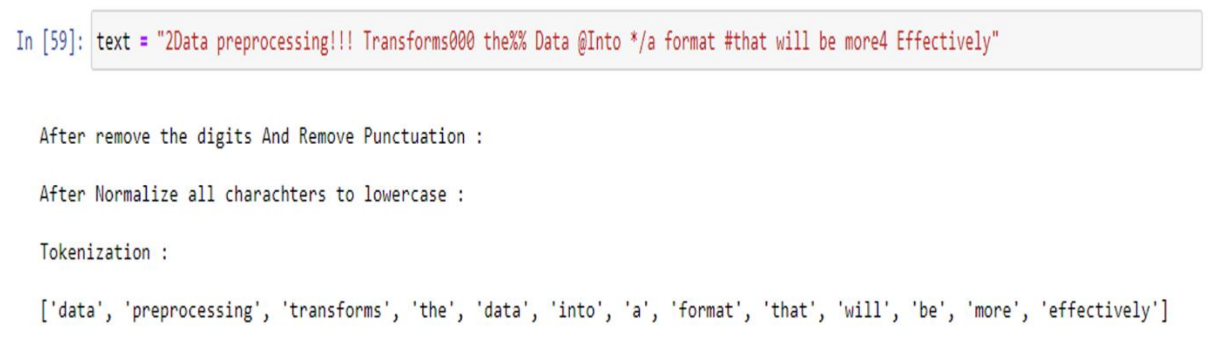


```
In [59]: text = "2Data preprocessing!!! Transforms000 the%% Data @Into */a format #that will be more4 Effectively"

After remove the digits And Remove Punctuation :
Normalize all charachters to lowercase :
data preprocessing    transforms    the   data  into   a format  that will be more  effectively
```

Figure 21: An example of removing the capital letters.

❖ Tokenization

Tokenization is a method of transforming text to a very simple, a piece of data stocked in a named list. Tokenization can be used to secure sensitive data by replacing the original data with an unrelated value of the same length and format.

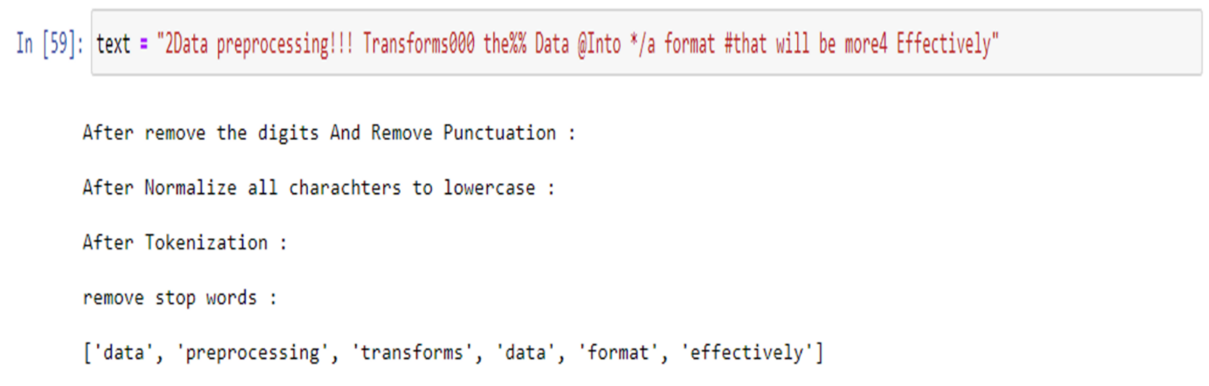


```
In [59]: text = "2Data preprocessing!!! Transforms000 the%% Data @Into */a format #that will be more4 Effectively"

  After remove the digits And Remove Punctuation :

  After Normalize all charachters to lowercase :

  Tokenization :

  ['data', 'preprocessing', 'transforms', 'the', 'data', 'into', 'a', 'format', 'that', 'will', 'be', 'more', 'effectively']
```

Figure 22: An example of tokenization of a text.

❖ Remove StopWords (meaningless words)

Stopwords is words that occur so frequently in language, they not offer useful information for classification. Contrary it is harmful for the text classification, and are generally very common and present in most posts. It is includes words such as “the” and “are”.

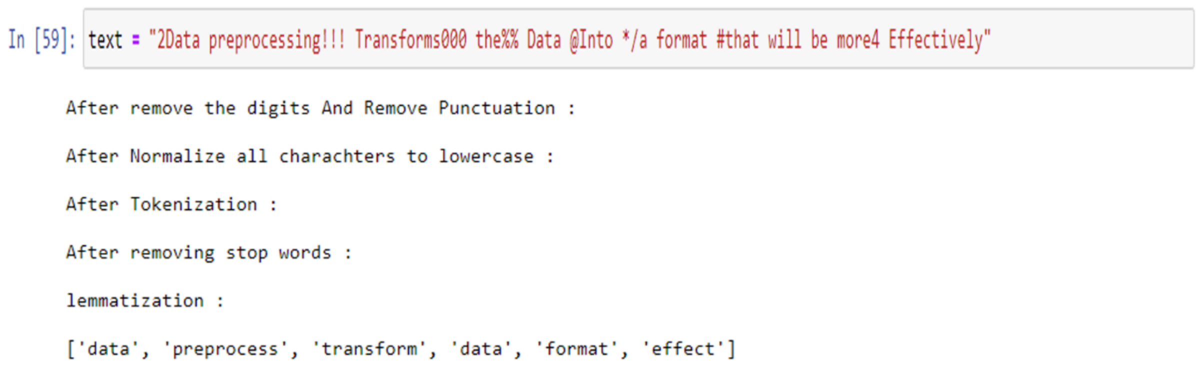


```
In [59]: text = "2Data preprocessing!!! Transforms000 the%% Data @Into */a format #that will be more4 Effectively"

      After remove the digits And Remove Punctuation :

      After Normalize all charachters to lowercase :

      After Tokenization :

      remove stop words :

      ['data', 'preprocessing', 'transforms', 'data', 'format', 'effectively']
```

Figure 23: An example of removing the StopWords

❖ Lemmatization

The goal of lemmatization is to remove inflections and map a word to its root form. The only difference is that, lemmatization tries to do it the proper way. It does not just chop things off, it actually transforms words to the actual root. For example, the word "better" would map to "good". It may use a dictionary such as WordNet for mappings or some special rule-based approaches. Here is an example of lemmatization:



```
In [59]: text = "2Data preprocessing!!! Transforms000 the%% Data @Into */a format #that will be more4 Effectively"

    After remove the digits And Remove Punctuation :

    After Normalize all charachters to lowercase :

    After Tokenization :

    After removing stop words :

    lemmatization :

    ['data', 'preprocess', 'transform', 'data', 'format', 'effect']
```

Figure 24: An example of lemmatization of text.

2.2 Vectorization

This is the second important step of features extraction part. Its goal is to transform words to vectors to finish the features extraction part and get ready to entre it into the Autoencoder model.

In this step, we used the word embedding as a method of Vectorization.

2.1.1 Word Embedding

It is a method of learning a representation of words in a document by real numbers, it used especially in automatic language processing. Moreover, it is a word representation type that allows machine learning algorithms to understand words with similar meanings. This technique allows each word in our fandom pair text in the dataset to be represented by a vector of real numbers. To do Word Embedding, we used Word2Vec algorithm.

2.1.1.1 Word2Vec

Word2vec is a word-embedding algorithm. It is a popular sequence embedding method that transforms natural language into distributed vector representations. It can capture contextual word-to-word relationships in a multidimensional space and has been widely used as a preliminary step for predictive models in semantic and information retrieval tasks. We used it in our case to transform all the pair text to vectors and we can extract the similarity between words. Word2vec is based on a two-layer neural network, it has two neural architectures called CBOW and Skip-Gram [40].

- **CBOW (Continuous Bag of Words) Model**
  The model is fed by the context, and predicts the target word. The result of the hidden layer is the new representation of the word[5].
  Figure 25 describes the Word2vec Continuous Bag of Words (CBOW) model process.

- **Skip-Gram Model**
  The model is fed by the target word, and predicts the words of the context. The result of the hidden layer is the new representation of the word[6].
  Figure 25 describes the Word2vec skip-gram model process.

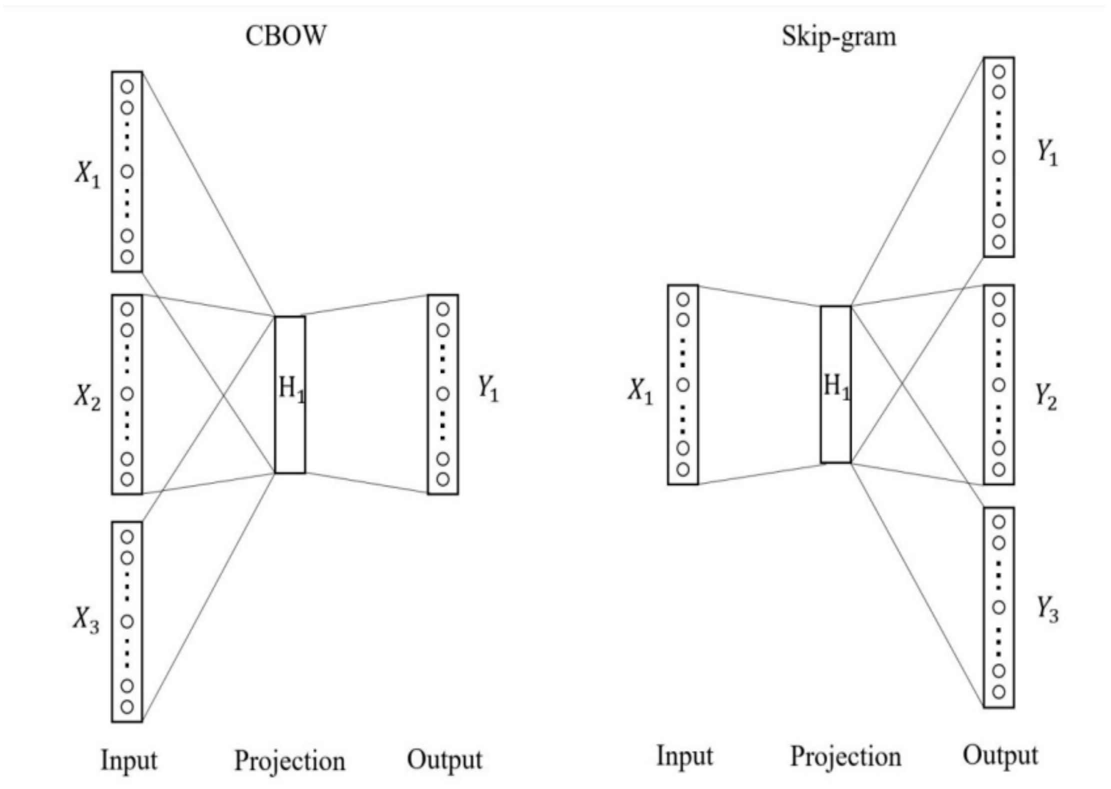


Figure 25: Word2vec model (Continuous Bag of Words (CBOW) and Skip-Ngram)[40].

[5] https://datascientest.com/nlp-word-embedding-word2vec
[6] https://datascientest.com/nlp-word-embedding-word2vec

3. Text classification

In this section, we will present our method we used to do the classification of the pairs fandoms texts. We used an Autoencoder to learn a compressed representation of the input features for a text classification predictive modeling problem.

We will used a binary (2-class) classification task with two dimension (pair of texts) and we take 2000 as number of inputs features for the two pairs (2, 2000), and we take 37,872 samples for training.

The model will take the input, then output the same input values. It will learn to recreate the input pattern correctly.

The encoder learns how to interpret the input and compress it to build a compressed representation we call it the latent layer. The decoder takes the output of the encoder (the latent layer) and attempts to recreate the same input. Before fitting the model, we will split our dataset into "training", "validation" and "testing", and scale the input data by normalizing the values to the range [0:1], a good practice with MLPs.

We will define the encoder as having 10 hidden layers, the first with the number of inputs (2, 2000) and the second with less number of inputs and so on. Finally, the last encoded hidden layer gives us the latent layer with fewer inputs than the previous layers (60). The decoder will be defined with a similar structure, it get the latent as input and extract the features to obtain the output data, it is the same data that we have been input it in the encoded layers.

The table below shown the tokens size that we used in each layers and the activation function we used.

layer	Tokens size	Activation function
Input	(2, 2000)	X
Encoded	1800	Relu
Encoded	1600	Relu
Encoded	1400	Relu

Encoded	1200	Relu
Encoded	1000	Relu
Encoded	800	Relu
Encoded	600	Relu
Encoded	200	Relu
Encoded	100	Relu
Encoded	60	Relu
Decoded	100	Relu
Decoded	200	Relu
Decoded	600	Relu
Decoded	800	Relu
Decoded	1000	Relu
Decoded	1200	Relu
Decoded	1400	Relu
Decoded	1600	Relu
Decoded	1800	Relu
Decoded	2000	Sigmoid

Table 3: All layers with its tokens size and activation function.

The output layer will have the same number of nodes as the input data and will use a linear activation function to output numeric values like RELU activation function.

After that, the model will be fit using the efficient $\boldsymbol{mean\_squared\_error}$ version of stochastic gradient descent and minimizes the mean squared error, given that reconstruction is a type of multi-output regression problem.

At the end, we can train the model to reproduce the input easier. After training, we can plot the learning curves for the training and testing to evaluate the model learned the reconstruction problem well. Finally, we can save the Autoencoder model.

Conclusion

Finally, we have walked through the different phases and layers of our new model. More importantly, we dived deep into each part to give a better understanding of the basics of all the methods used, that way everything be clear.

In the next chapter, we will describe all the tools from hardware and the software languages that we used to build our model. Moreover, we present the results that we got it from our new model.

Chapter 04: Experimentation and results

Introduction

To achieve our research and build our authorship verification model, we used a set of tools of software and materials.

We will present in this chapter the set of materials used. Then, we will discuss different evaluation metrics that will demonstrate the effectiveness of our proposed approach. At last, we will discuss the results obtained.

1. Hardware

The hardware is the most important stuff that play a very important role in our research and all research activity. This domain deal with a huge data that need a powerful hardware to achieve good results.

This project was limited by time, in order to speed up the training of our Autoencoder we took the initiative to run in three different machines.

The first is an Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz x64 processor with 4 GB RAM.

The second is an Intel (R) Core (™) i3-2310M CPU @ 2.10GHz x64 processor with 6 GB RAM.

However, these machines did not allow us to finish our training because they are weak and not gave us the performance that we need with the huge dataset we used. So we asked for a powerful machine and we got the third machine.

The third is an AMD Ryzen 5 3400G with Radeon Vega Graphics 3.70GHz x64 processor with 16 GB RAM.

Due to the difficulty imposed by this challenge, we just used just the small dataset and we had difficulties to manipulate it, for successfully complete this project.

2. Software

First, to build our model we had been work with different software. We started with integrated development environment (IDE) PyCharm Community Edition 2020.2.3, but it was difficult for us.

For this reason, we change it to used Jupyter that help us because it was simple and easy to use.

Moreover, we used also Google Colab platform provided by Google Company as cloud service. It is very helpful because all the libraries have been installed in, you just need to put your code and run it and got the result. However, it is not free, Google Colab give you just a one-month free with a less performance, to push you to pay and get all the performance that you want.

In addition, we used Python 3.8 as the programming language. We have deduced from previous work that this is the most appropriate language for a number of reasons:

- It is an interpreted language, which means that Python directly executes the code line by line.
- A high-level programming language has English-like syntax. It easier to read and understand the code.
- It is a very wide language, it is used for all domains.
- It has diverse and rich libraries, those dedicated to deep learning and those used for the management of other data structures and others.
- Its simplicity, it allows expressing complex equations and formulas and it is easy, which consumes less time for the whole development process.

3. Library

In this part, we will present all libraries used during to build our deep learning model.

- ❖ **TensorFlow[7]:** Is an open source platform and Python library, Created by researchers of the Google Brain team for the field of machine learning and deep learning technics

[7] https://www.tensorflow.org/

and methods. It has a flexible architecture of tools make the deployment so easy on various platforms like GPU, CPU and TPU.

- **Keras**[8]**:** Is an open source free Python library, used for deep learning and machine learning models, and it acts as an API of tensorflow library. Keras helps to run experiments easily and faster and a same code can be run on CPU or on GPU, seamlessly.
- **Genism**[9]**:** Is an open source library provided by Python language, it used for the machine learning and deep learning technics, and for the unsupervised topics and NLP, It has provided a several algorithms like the Word2vec algorithm for the vectorization step in the features extraction part.
- **Scikit-learn**[10]**:** is an open source free Python library, it has created by several of researchers for the field of machine learning and deep learning, and it is commercial usable. It allows the movement to efficient versions of numerous of current algorithms.
- **NumPy**[11]**:** Is one of many open source library of python, it is for the Numeric arrays. It has created especially for scientific computing, in particular matrix computing, while offering multiple functions allowing the creation and manipulation of matrices and vectors.

4. Dataset

In this section, we will discuss the dataset used in our research.

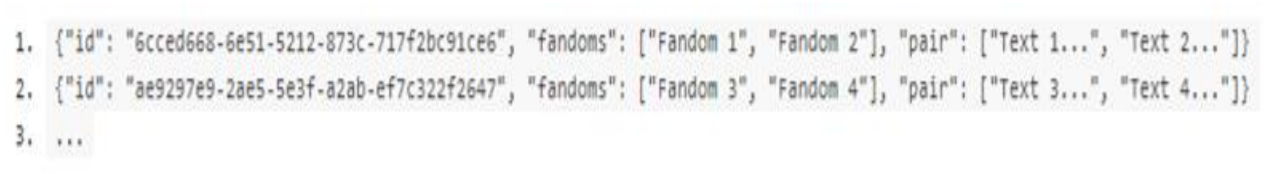
PAN2020[12], provide two datasets small and large. For our experiments, we used the smaller dataset. The datasets focuses over a collection of fanfiction texts. Fanfiction is a fan-written extension of a storyline in which a so-called fandom topic describes the principal subject of the document. The dataset consists of pairs of two different fanfics text that were obtained drawn from fanfiction.net. Each pair was assigned a unique identifier and they distinguish between same-author pairs and different-authors pairs. Additionally, PAN offer metadata on the fandom (i.e. thematic category) for each text in the pair (note that fanfic "crossovers" were not included and only single-fandom texts were considered). The fandom distribution in these dataset

[8] https://keras.io/
[9] https://pypi.org/project/gensim/
[10] https://scikit-learn.org/stable/
[11] https://numpy.org/doc/stable/user/whatisnumpy.html
[12] https://pan.webis.de/data.html#pan20-authorship-verification

maximally approximates the distribution of the fandoms in the original dataset. The test datasets is structured in the exact same way, but participants should expect a significant shift in the relation between authors and fandoms. The dataset have a collection of 52601 pair of text (105202 text) each pair with its id.

Both the small and the large datasets come with two newline delimited JSON files (.jsonl).

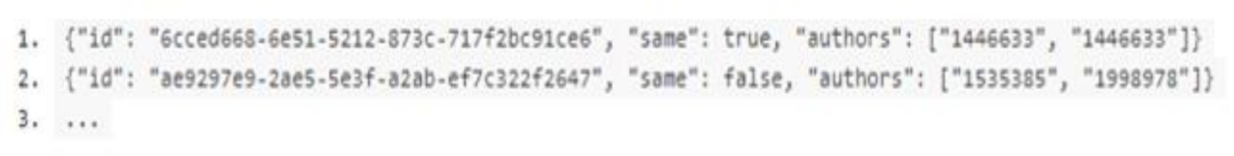
The first file is training-small.jsonl, contains pairs of texts (each pair has a unique ID) and their fandom labels:



```
{"id": "6cced668-6e51-5212-873c-717f2bc91ce6", "fandoms": ["Fandom 1", "Fandom 2"], "pair": ["Text 1...", "Text 2..."]}
{"id": "ae9297e9-2ae5-5e3f-a2ab-ef7c322f2647", "fandoms": ["Fandom 3", "Fandom 4"], "pair": ["Text 3...", "Text 4..."]}
...
```

Figure 26: The structure of the training Small file.

The second file is truth.jsonl, contains the ground truth for all pairs. The ground truth is composed of a boolean flag indicating if texts in a pair are from the same author and the numeric author IDs:



```
{"id": "6cced668-6e51-5212-873c-717f2bc91ce6", "same": true, "authors": ["1446633", "1446633"]}
{"id": "ae9297e9-2ae5-5e3f-a2ab-ef7c322f2647", "same": false, "authors": ["1535385", "1998978"]}
...
```

Figure 27: The structure of the Truth file.

The table 4 bellow represents the statistics of the small dataset we used for our experiments.

Number of Pairs	Number of Fandom texts	Number of Positive	Number of Negative	Number of the training pairs	Number of the validation pairs	Number of the test pairs
52601	105202	27834	24767	37873	4208	10520

Table 4: The statistics of the small dataset.

5. Implementation

In this section, we will start the main work to build our model using the Autoencoder. After the extraction of the features (preprocessing and vectorization) of the dataset, we used the truth dataset also, it has two classes positive writing by the same author (same = true), and negative writing by different author (same = false).

The size of each text is 2000 tokens all their value between$[0\ ;\ 1]$. If the size is greater than 2000 tokens, automatically remove the remainder; else add 0 to the text whose size is less than 2000 to fill it.

We will split the data into three parts “training”, “validation” and “testing”.

For the training, we took 80% of the data, and for the testing, we took 20% of data. Moreover, for the validation we took 10% from the training data. It is mean we took 72% for training and 8% for validation from the main data.

In addition, it will help us in reducing the chances of overfitting, as we will be validating our model on data it would not have seen in training phase.

At last, we got 42081 samples for training, and 10520 for testing. Moreover, we took 10% from the testing for the validation so the last statistics is:

Training: 37873 samples.

Validation: 4208 samples.

Test: 10520 samples.

5.1 Training and validation

After the model is created, we have to compile it using the optimizer function name is $\boldsymbol{RMSprop}()$ **.** Moreover, we must to specify the loss type via the argument $\boldsymbol{loss}$**,** its name is $\boldsymbol{mean\_squared\_error}$, since the loss after every batch will be computed between the batch of predicted output and the ground truth using mean squared error.

After that, we went to start train the model using the $\boldsymbol{fit}()$ Keras function. We have train our model for 50 epochs. The $\boldsymbol{fit}()$ function return a history object, by stored the result of this function in a variable, and we can use it later to plot the loss function plot between training and validation, which will help us to analyze our model's performance visually.

At last, we save our model using this function $\boldsymbol{save\_weights}()$ to use it in part of testing.

5.2 Testing

In this case, we use a graphical interface to do the test data predicting and classification of our model. The figure below shows the description of the graphical interface that we build it to testing our model.

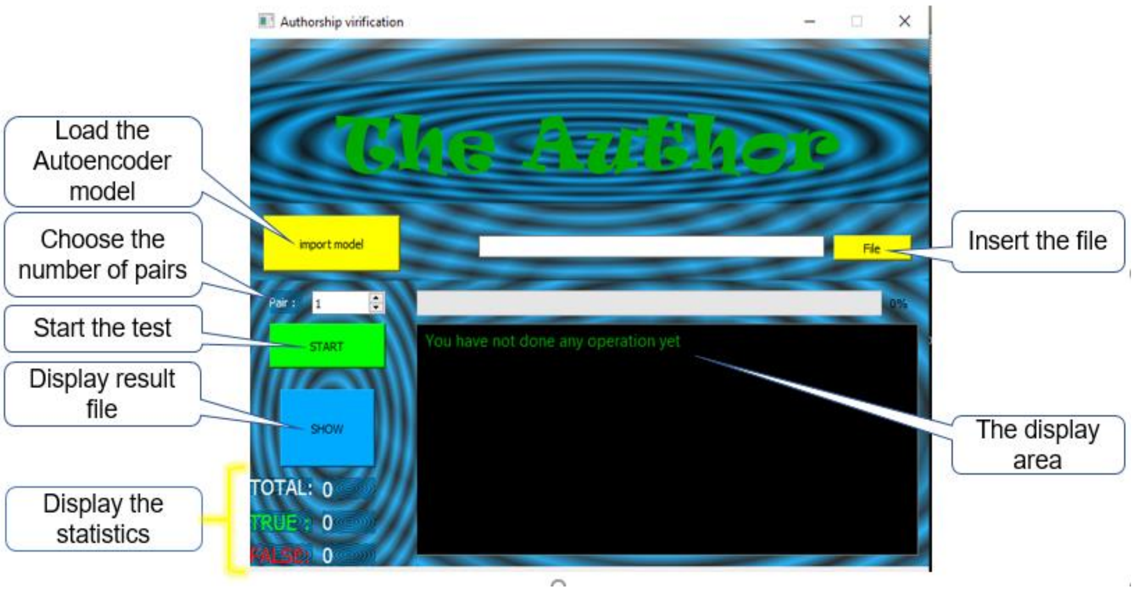
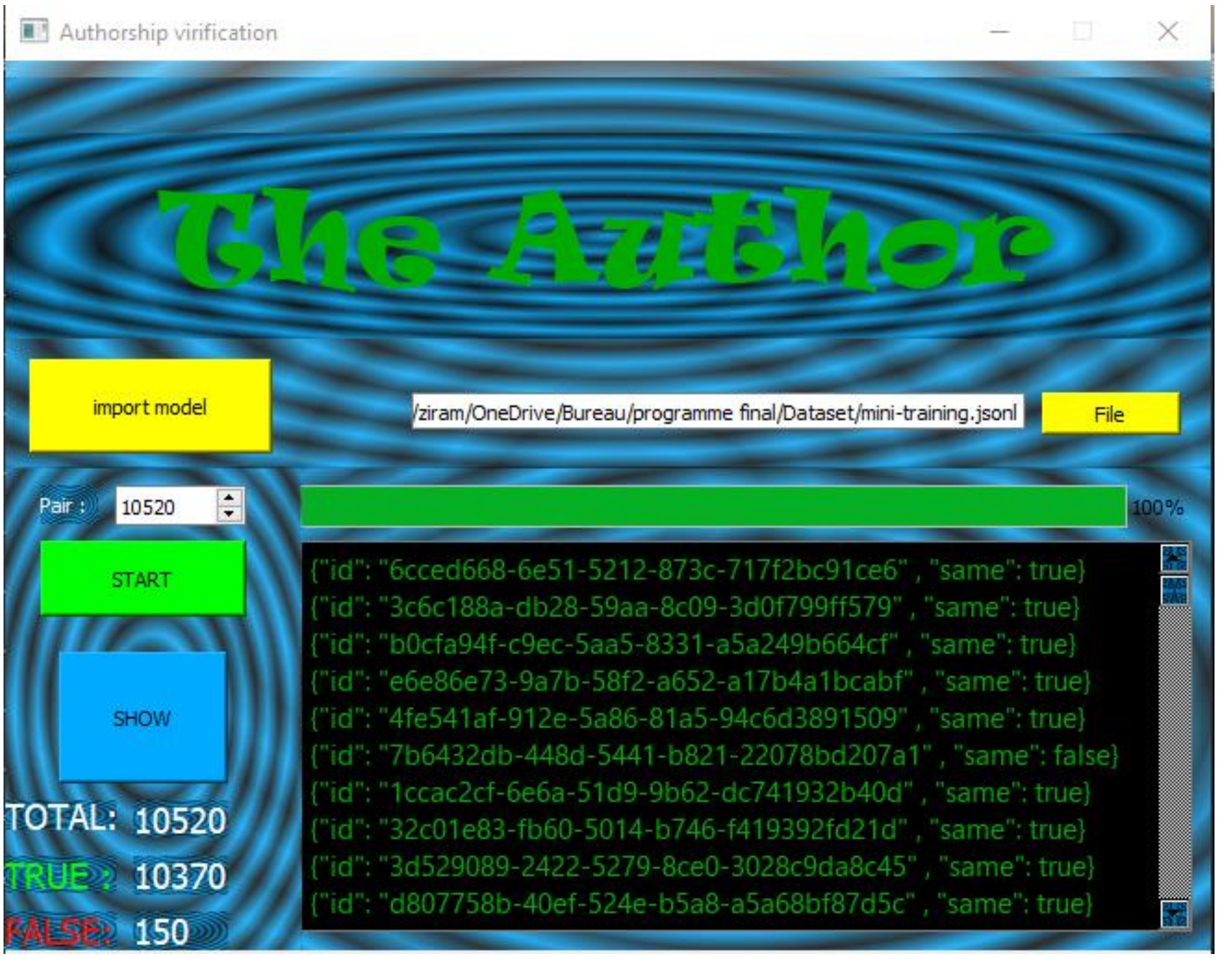


Figure 28: The description of the graphical interface

After that, we show those predictions that we get. The results are printed in a new file JSONL and we display these results in the display area.

The figure below shows the predictions results.



/ziram/OneDrive/Bureau/programme final/Dataset/mini-training.jsonl

Pair : 10520

```
{"id": "6cced668-6e51-5212-873c-717f2bc91ce6" , "same": true}
{"id": "3c6c188a-db28-59aa-8c09-3d0f799ff579" , "same": true}
{"id": "b0cfa94f-c9ec-5aa5-8331-a5a249b664cf" , "same": true}
{"id": "e6e86e73-9a7b-58f2-a652-a17b4a1bcabf" , "same": true}
{"id": "4fe541af-912e-5a86-81a5-94c6d3891509" , "same": true}
{"id": "7b6432db-448d-5441-b821-22078bd207a1" , "same": false}
{"id": "1ccac2cf-6e6a-51d9-9b62-dc741932b40d" , "same": true}
{"id": "32c01e83-fb60-5014-b746-f419392fd21d" , "same": true}
{"id": "3d529089-2422-5279-8ce0-3028c9da8c45" , "same": true}
{"id": "d807758b-40ef-524e-b5a8-a5a68bf87d5c" , "same": true}
```

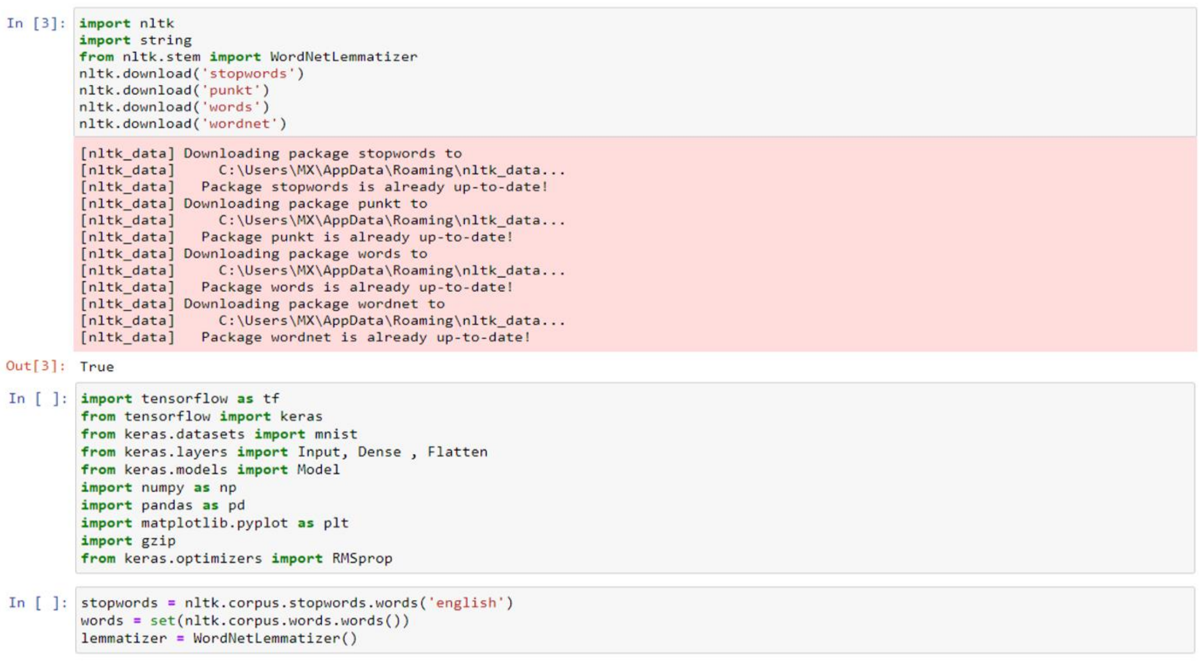
TOTAL: 10520
TRUE : 10370
FALSE: 150

Figure 29: The display of the printed results.

5.3 Source code

Now, we explain some of our source code that we used to build our model.

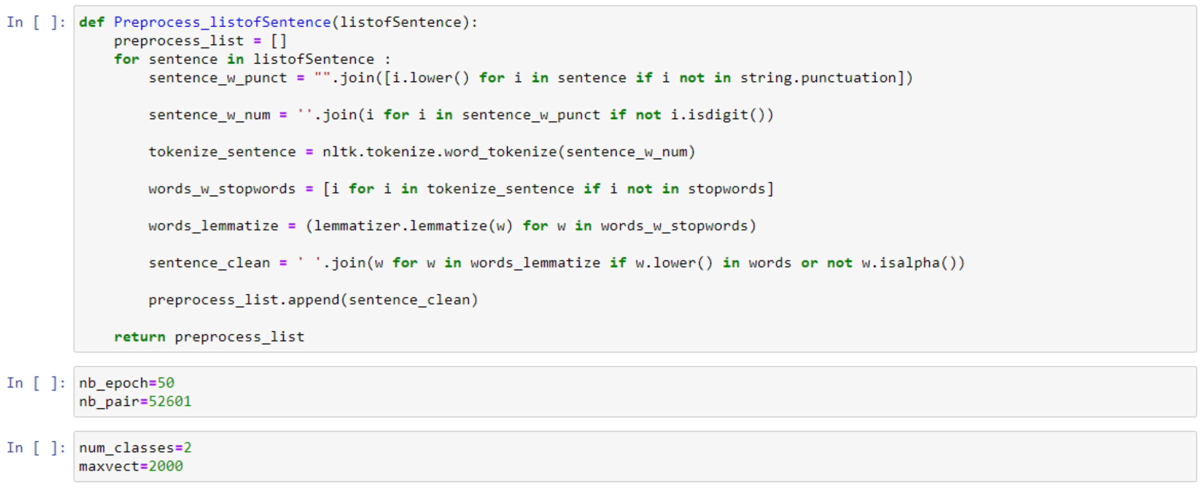
The figure below shows the source code of the data download data and the import packages used in our work.



```
In [3]: import nltk
        import string
        from nltk.stem import WordNetLemmatizer
        nltk.download('stopwords')
        nltk.download('punkt')
        nltk.download('words')
        nltk.download('wordnet')

        [nltk_data] Downloading package stopwords to
        [nltk_data]     C:\Users\MX\AppData\Roaming\nltk_data...
        [nltk_data]   Package stopwords is already up-to-date!
        [nltk_data] Downloading package punkt to
        [nltk_data]     C:\Users\MX\AppData\Roaming\nltk_data...
        [nltk_data]   Package punkt is already up-to-date!
        [nltk_data] Downloading package words to
        [nltk_data]     C:\Users\MX\AppData\Roaming\nltk_data...
        [nltk_data]   Package words is already up-to-date!
        [nltk_data] Downloading package wordnet to
        [nltk_data]     C:\Users\MX\AppData\Roaming\nltk_data...
        [nltk_data]   Package wordnet is already up-to-date!

Out[3]: True

In [ ]: import tensorflow as tf
        from tensorflow import keras
        from keras.datasets import mnist
        from keras.layers import Input, Dense , Flatten
        from keras.models import Model
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import gzip
        from keras.optimizers import RMSprop

In [ ]: stopwords = nltk.corpus.stopwords.words('english')
        words = set(nltk.corpus.words.words())
        lemmatizer = WordNetLemmatizer()
```

Figure 30: The download data and the import packages.

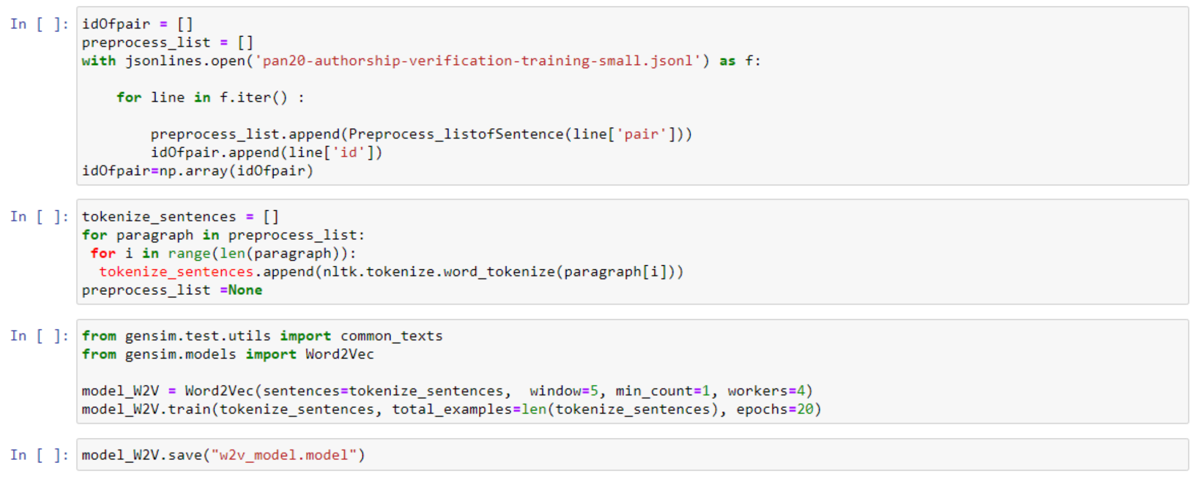
Figure 31 presents the source code of the preprocessing function we created and some initialization of variable.



```
In [ ]: def Preprocess_listofSentence(listofSentence):
            preprocess_list = []
            for sentence in listofSentence :
                sentence_w_punct = "".join([i.lower() for i in sentence if i not in string.punctuation])

                sentence_w_num = ''.join(i for i in sentence_w_punct if not i.isdigit())

                tokenize_sentence = nltk.tokenize.word_tokenize(sentence_w_num)

                words_w_stopwords = [i for i in tokenize_sentence if i not in stopwords]

                words_lemmatize = (lemmatizer.lemmatize(w) for w in words_w_stopwords)

                sentence_clean = ' '.join(w for w in words_lemmatize if w.lower() in words or not w.isalpha())

                preprocess_list.append(sentence_clean)

            return preprocess_list

In [ ]: nb_epoch=50
        nb_pair=52601

In [ ]: num_classes=2
        maxvect=2000
```

Figure 31: The preprocessing function we created and some initialization of variable.

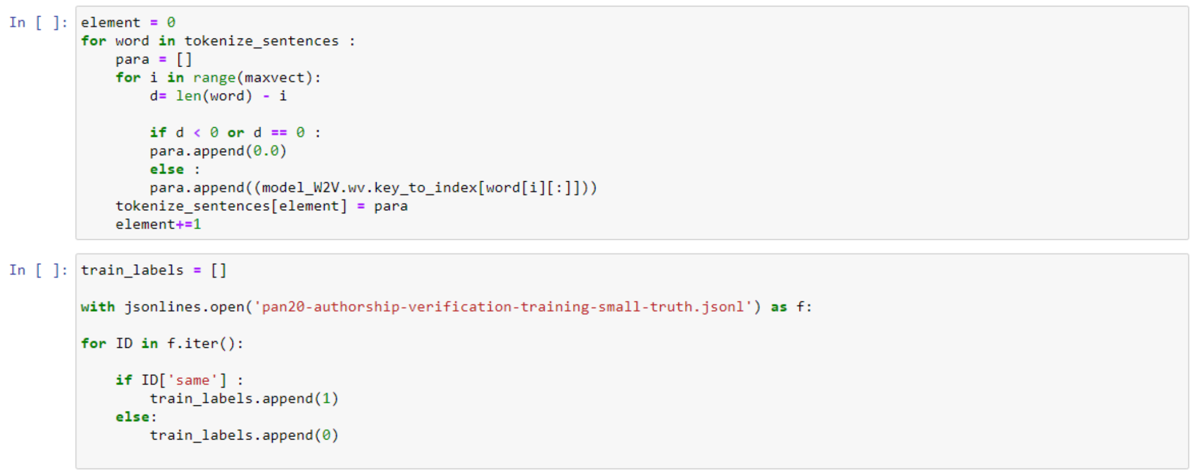
In the figure below, we present the source code of the features extraction of our dataset using Word2vec for the vectorization step in our model.



```
In [ ]: idOfpair = []
        preprocess_list = []
        with jsonlines.open('pan20-authorship-verification-training-small.jsonl') as f:

            for line in f.iter() :

                preprocess_list.append(Preprocess_listofSentence(line['pair']))
                idOfpair.append(line['id'])
        idOfpair=np.array(idOfpair)

In [ ]: tokenize_sentences = []
        for paragraph in preprocess_list:
         for i in range(len(paragraph)):
          tokenize_sentences.append(nltk.tokenize.word_tokenize(paragraph[i]))
        preprocess_list =None

In [ ]: from gensim.test.utils import common_texts
        from gensim.models import Word2Vec

        model_W2V = Word2Vec(sentences=tokenize_sentences,  window=5, min_count=1, workers=4)
        model_W2V.train(tokenize_sentences, total_examples=len(tokenize_sentences), epochs=20)

In [ ]: model_W2V.save("w2v_model.model")
```

Figure 32: The features extraction of our dataset using Word2vec for the vectorization step.

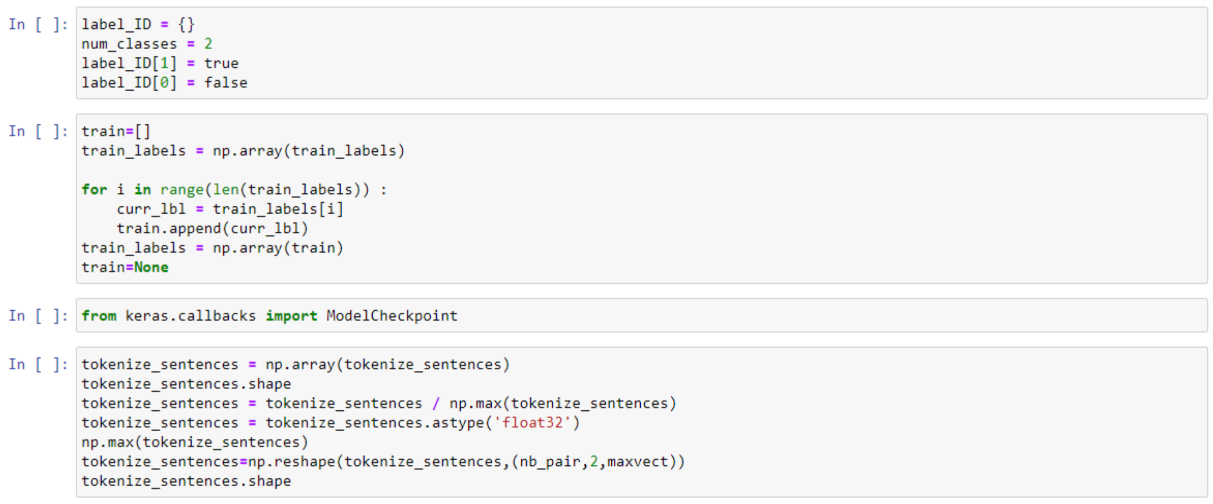
The figure below shows the source code of the management of texts and the loading of the truth data classes that we used in the classification part.



```
In [ ]: element = 0
        for word in tokenize_sentences :
            para = []
            for i in range(maxvect):
                d= len(word) - i

                if d < 0 or d == 0 :
                para.append(0.0)
                else :
                para.append((model_W2V.wv.key_to_index[word[i][:]]))
            tokenize_sentences[element] = para
            element+=1

In [ ]: train_labels = []

        with jsonlines.open('pan20-authorship-verification-training-small-truth.jsonl') as f:

        for ID in f.iter():

            if ID['same'] :
                train_labels.append(1)
            else:
                train_labels.append(0)
```

Figure 33: The process of the truth data.

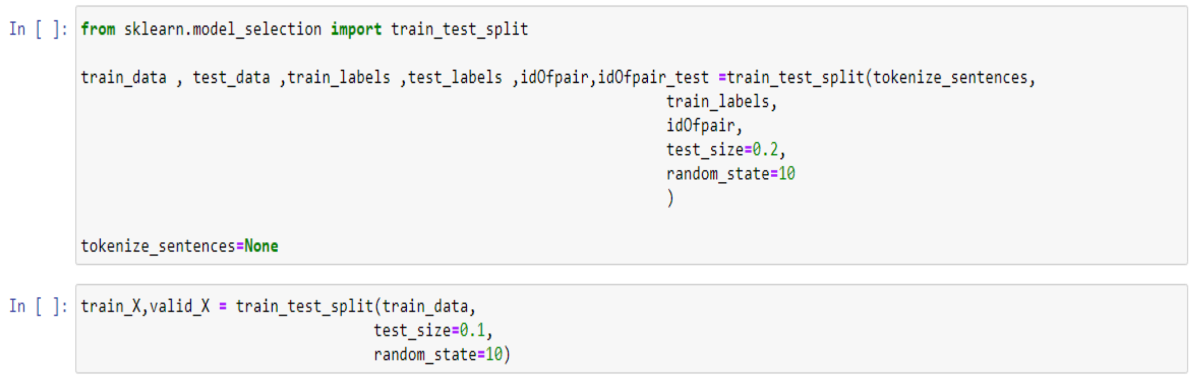
Figure 34 shows the source code of the initialization of the two classes that we used in the classification part.



```
In [ ]: label_ID = {}
        num_classes = 2
        label_ID[1] = true
        label_ID[0] = false

In [ ]: train=[]
        train_labels = np.array(train_labels)

        for i in range(len(train_labels)) :
            curr_lbl = train_labels[i]
            train.append(curr_lbl)
        train_labels = np.array(train)
        train=None

In [ ]: from keras.callbacks import ModelCheckpoint

In [ ]: tokenize_sentences = np.array(tokenize_sentences)
        tokenize_sentences.shape
        tokenize_sentences = tokenize_sentences / np.max(tokenize_sentences)
        tokenize_sentences = tokenize_sentences.astype('float32')
        np.max(tokenize_sentences)
        tokenize_sentences=np.reshape(tokenize_sentences,(nb_pair,2,maxvect))
        tokenize_sentences.shape
```

Figure 34: The initialization of the two classes of the classification.

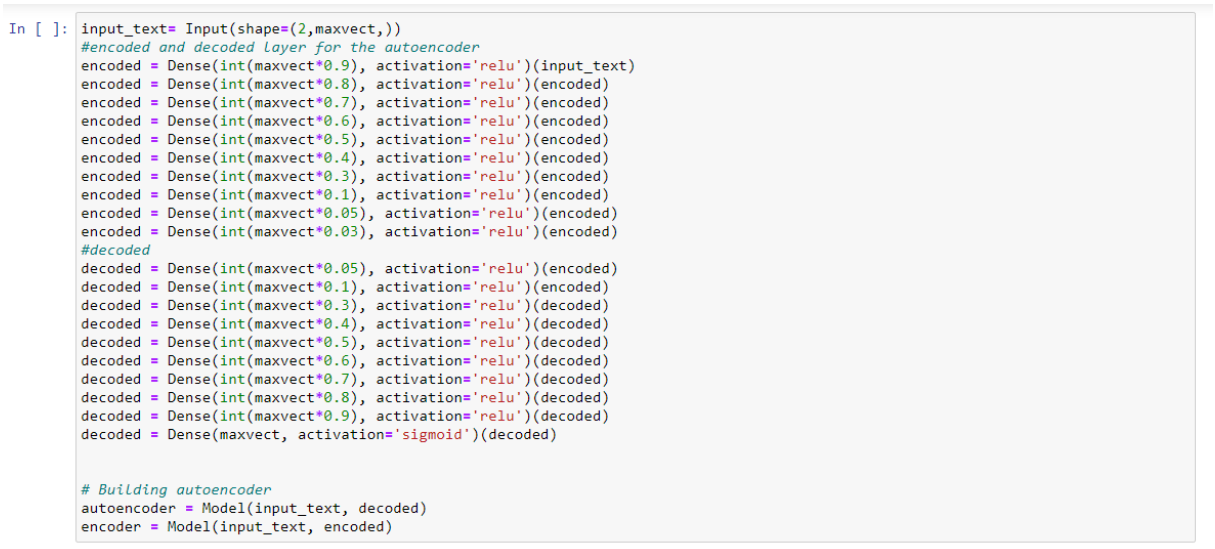
The figure below illustrates the source code of the split of our dataset.



```
In [ ]: from sklearn.model_selection import train_test_split

        train_data , test_data ,train_labels ,test_labels ,idOfpair,idOfpair_test =train_test_split(tokenize_sentences,
                                                                                    train_labels,
                                                                                    idOfpair,
                                                                                    test_size=0.2,
                                                                                    random_state=10
                                                                                    )

        tokenize_sentences=None

In [ ]: train_X,valid_X = train_test_split(train_data,
                                           test_size=0.1,
                                           random_state=10)
```

Figure 35: The split of our dataset.

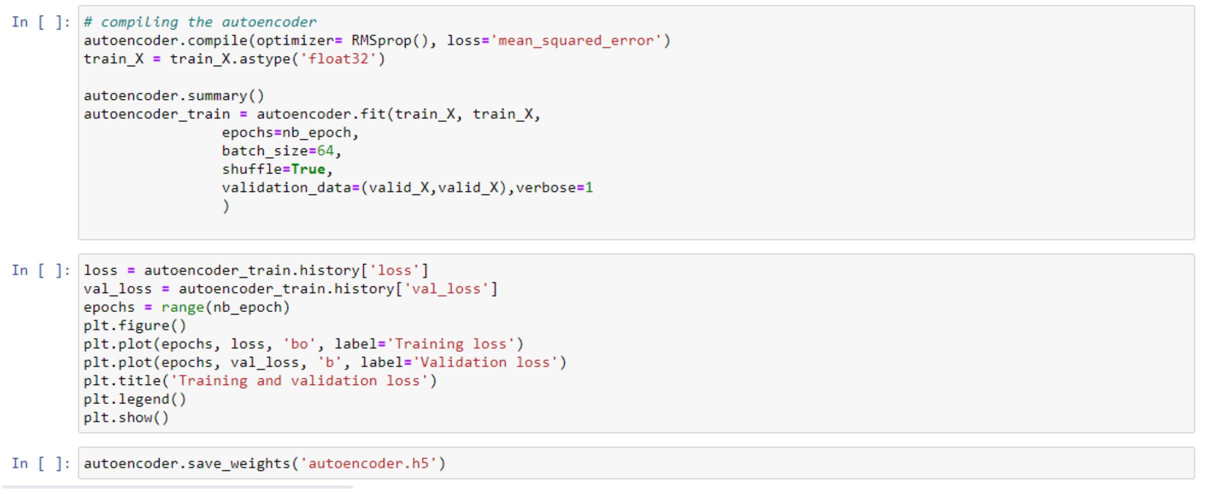
The figure 36 shows the source code of the Autoencoder features extraction for classification and the encoded and decoded layers with their characteristics.



```
In [ ]: input_text= Input(shape=(2,maxvect,))
        #encoded and decoded layer for the autoencoder
        encoded = Dense(int(maxvect*0.9), activation='relu')(input_text)
        encoded = Dense(int(maxvect*0.8), activation='relu')(encoded)
        encoded = Dense(int(maxvect*0.7), activation='relu')(encoded)
        encoded = Dense(int(maxvect*0.6), activation='relu')(encoded)
        encoded = Dense(int(maxvect*0.5), activation='relu')(encoded)
        encoded = Dense(int(maxvect*0.4), activation='relu')(encoded)
        encoded = Dense(int(maxvect*0.3), activation='relu')(encoded)
        encoded = Dense(int(maxvect*0.1), activation='relu')(encoded)
        encoded = Dense(int(maxvect*0.05), activation='relu')(encoded)
        encoded = Dense(int(maxvect*0.03), activation='relu')(encoded)
        #decoded
        decoded = Dense(int(maxvect*0.05), activation='relu')(encoded)
        decoded = Dense(int(maxvect*0.1), activation='relu')(encoded)
        decoded = Dense(int(maxvect*0.3), activation='relu')(decoded)
        decoded = Dense(int(maxvect*0.4), activation='relu')(decoded)
        decoded = Dense(int(maxvect*0.5), activation='relu')(decoded)
        decoded = Dense(int(maxvect*0.6), activation='relu')(decoded)
        decoded = Dense(int(maxvect*0.7), activation='relu')(decoded)
        decoded = Dense(int(maxvect*0.8), activation='relu')(decoded)
        decoded = Dense(int(maxvect*0.9), activation='relu')(decoded)
        decoded = Dense(maxvect, activation='sigmoid')(decoded)


        # Building autoencoder
        autoencoder = Model(input_text, decoded)
        encoder = Model(input_text, encoded)
```

Figure 36: The Autoencoder features extraction for classification.

The figure below represent the source code of the compiling the Autoencoder model and how to save it and the process of the prediction loss.



```
In [ ]: # compiling the autoencoder
        autoencoder.compile(optimizer= RMSprop(), loss='mean_squared_error')
        train_X = train_X.astype('float32')

        autoencoder.summary()
        autoencoder_train = autoencoder.fit(train_X, train_X,
                        epochs=nb_epoch,
                        batch_size=64,
                        shuffle=True,
                        validation_data=(valid_X,valid_X),verbose=1
                        )

In [ ]: loss = autoencoder_train.history['loss']
        val_loss = autoencoder_train.history['val_loss']
        epochs = range(nb_epoch)
        plt.figure()
        plt.plot(epochs, loss, 'bo', label='Training loss')
        plt.plot(epochs, val_loss, 'b', label='Validation loss')
        plt.title('Training and validation loss')
        plt.legend()
        plt.show()

In [ ]: autoencoder.save_weights('autoencoder.h5')
```

Figure 37: The compiling of the Autoencoder model.

6. Evaluation metrics

In this section, we will present the evaluation metrics that we used to evaluate our new model in the field of authorship verification. We evaluate the results of the predictions using the evaluation metrics like F1-score and Accuracy, and calculate the predictions loss using the loss function.

- **Recall[13]:** Recall (R) is defined as the number of true positives (T_p) over the number of true positives plus the number of false negatives (F_n) as follows:

$$\boldsymbol{R = \frac{Tp}{Tp + Fn}}$$

- **Precision[14]:** Precision (P) is defined as the number of true positives (T_p) over the number of true positives plus the number of false positives (F_p) as follows:

$$\boldsymbol{P = \frac{Tp}{Tp + Fp}}$$

- **Accuracy[15]:** Accuracy is used for evaluating classification models.

  Informally, accuracy is the fraction of predictions of our model got right. Formally, accuracy has the following definition:

$$\boldsymbol{Accuracy = \frac{Number\ of\ correct\ predictions}{Totalnumber\ of\ predictions}}$$

  In addition, for binary classification, accuracy can be calculated in terms of positives and negatives as follows:

$$\boldsymbol{Accuracy = \frac{TP + TN}{TP + TN + FP + FN}}$$

  Where *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.

[13] https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
[14] https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
[15] https://developers.google.com/machine-learning/crash-course/classification/accuracy

- **F1-score:** The F1-score, also called the F-score, is a measure of a model's accuracy on a dataset. It is used to evaluate binary classification systems, which classify examples into 'positive' or 'negative'[16].
  The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is[17]:

$$\boldsymbol{F1 = 2 * (precision * recall) / (precision + recall)}$$

- **Loss[18]:** Loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. The goal of training a model is to find a set of weights and biases that have low loss, on average, across all examples.

7. Results

Finally, this is the last section in our research; we will present our results after the evaluation.

We achieved good result using the Autoencoder deep learning method. We are got a 0.98 for the accuracy and F1-score metrics. Table 5 shows the accuracy and F1-score evaluation results that we got it. In addition, the precision we are got 0.98 in the two classes 0 for the two texts that are writing by different authors, and class 1 for the two texts that are writing by the same author. Moreover, in the recall, we got 0.97 in the class 0 and we got 0.99 in the class 1. Table 6 shows the precision and recall evaluation results that we got it.

accuracy	**F1-score**
0.98	0.98

Table 5: The results of our model.

[16] https://deepai.org/machine-learning-glossary-and-terms/f-score
[17] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
[18] https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

	Precision	**Recall**
Class 0	0.98	0.97
Class 1	0.98	0.99

Table 6: The results of the recall and precision.

The figure below shows the results of the evaluation of our model using the accuracy metric.

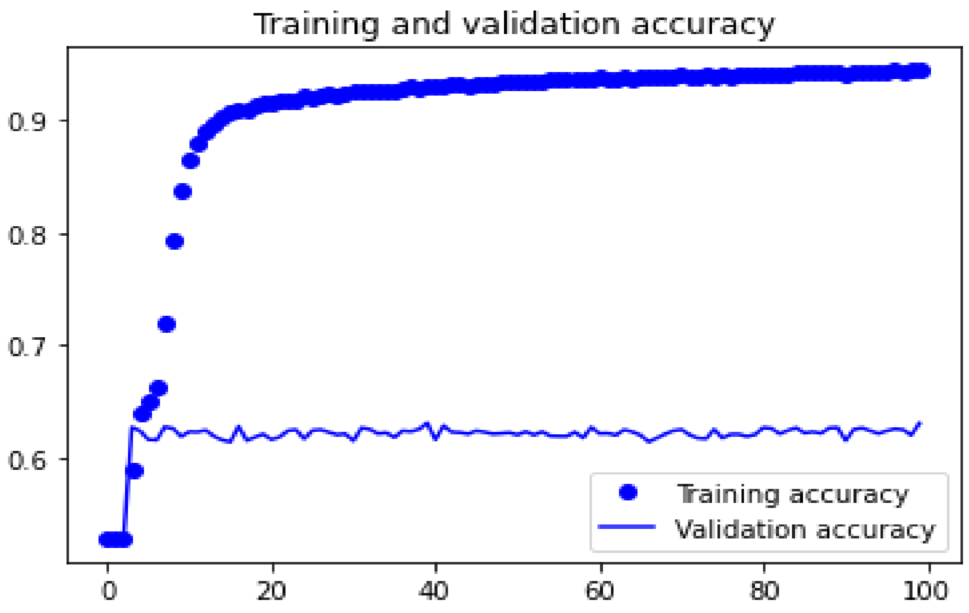


Figure 38: The training and validation accuracy.

Moreover, we present how much we get the loss using the loss function to calculate the penalty of the bad prediction, that we got it during the evaluation of our model. The figure bellow shows how much is the loss.

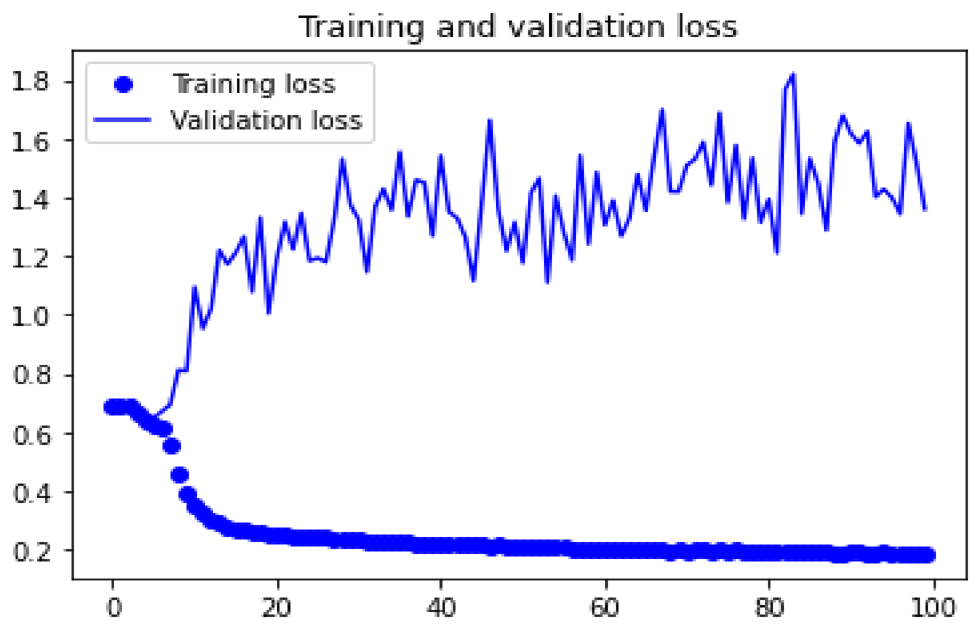


Figure 39: The loss volume of our model.

Finally, we will compare our results with the others previous works in the field of authorship verification. As raison, we evaluated our model just with F1-score because we did not participate at the competition of PAN 2021 due to time constraints. The table below shows the difference between all the works results that used the small dataset.

The work	**F1-score**
Our model	0.98
boenninghoff20-small [27]	0.906
weerasinghe20-small [37]	0.860
araujo20-small [35]	0.811
kipnis20-small [32]	0.809
halvani20b-small [30]	0.807

gagala20-small [2]	0.800
Ordoñez20-small[28]	0.748
Ikae20-small [29]	0.705

Table 7: The F1-score of all the works that used the small dataset provided by PAN.

The figure below shows a histogram that represents the comparison of our results with related works using F1-score.

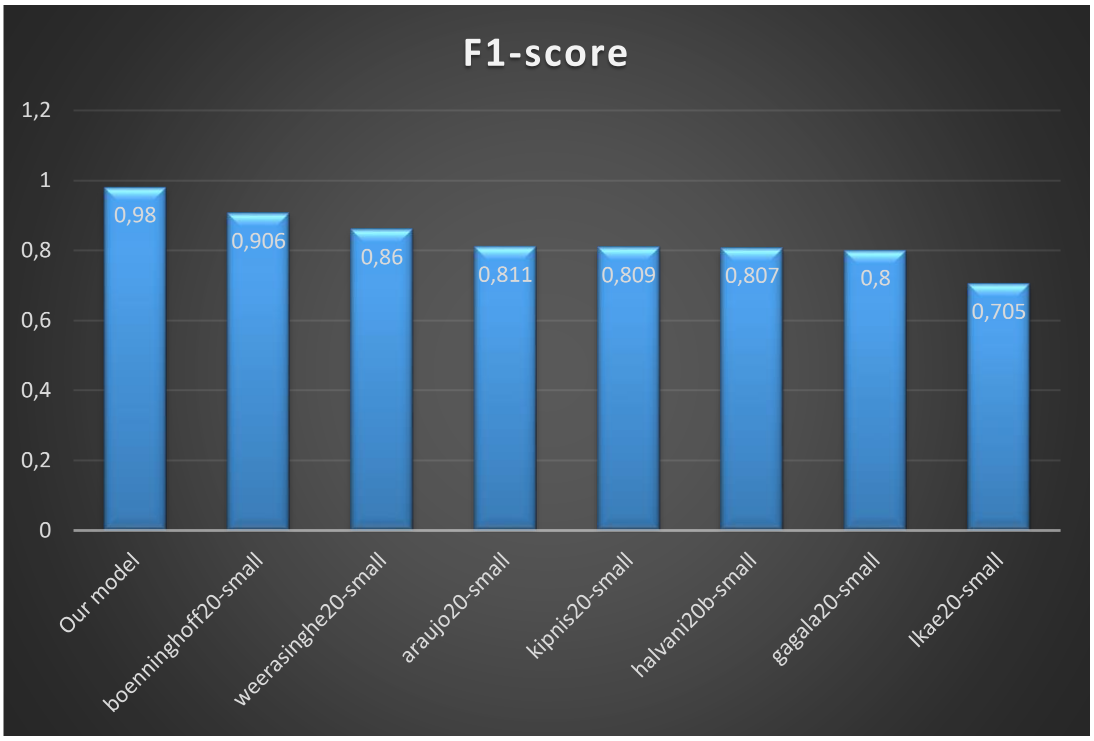


Figure 40: A histogram of our results with the other works using F1-score.

Our model achieved the best results for the F1-score metric.

Conclusion

Finally, this experimentation chapter was the most interesting part of our research study, because we presented the different tools used, and the results obtained from our new model. Moreover, we even went so far as to measure the performance of our model with the challenge of last year. So now, we move on to the general conclusion.

General Conclusion

Finally, we presented a new type of authorship verification (AV) system that focuses in the Autoencoder deep learning method to solve the authorship verification problem, where the task was to determine for a pair of texts if both texts were written by the same author or not. Our approach is unsupervised method in which we use neural networks for the mission of compressed learning, it is composed of two main layers the encoded layer and the decoded layer, who complement each other. Before we start using the Autoencoder, we prepare the dataset provided by PAN 2020 at CLEF. The dataset focuses over a collection of fanfiction texts. First, we extract the features of our dataset in two steps the first is preprocessing and the second is the vectorization using Word2vec method. After that, we start the text classification using our proposed method, we have to split the dataset into tree part “training”, “validation” and “test”. For the training, we took 80% of the data, and for the testing, we took 20% of data. Moreover, for the validation we took 10% from the training data. It is mean we took 72% for training and 8% for validation from the main data.

The proposed method achieved excellent overall performance scores, outperforming all other models that participated in the last year in the PAN 2020 Authorship Verification Task for the small dataset challenge.

Nevertheless, our AV method leaves room for further improvements. As future work, we would like to optimize our model for the test or the evaluation metrics. We believe it is possible to optimize our model and run it with powerful computer that have a good performance to make our optimization easier when we use more encoded and decoded layers to get good results and doesn't take more time. In addition, we would also like to perform the features analysis of our model to see which features become important in determining if two documents are written by the same person. Moreover, we would like to make our new model of authorship verification as an online service to detect the plagiarism and use it for detection of fraudulent, stolen, or unidentified data.

References

1. Kumar, S., et al., *A New Approach for Authorship Verification Using Information Retrieval Features*, in *Innovations in Computer Science and Engineering*. 2019, Springer. p. 23-29.
2. Gagała, Ł.J.W.N.o.C., *Authorship verification with prediction by partial matching and context-free grammar.* 2020.
3. Parentoni, L., *Artificial Intelligence*, in *Encyclopedia of the Philosophy of Law and Social Philosophy*, M. Sellers and S. Kirste, Editors. 2020, Springer Netherlands: Dordrecht. p. 1-4.
4. Batarseh, F.A., *Artificial Intelligence*, in *Encyclopedia of Big Data*, L.A. Schintler and C.L. McNeely, Editors. 2018, Springer International Publishing: Cham. p. 1-3.
5. El-Amir, H. and M. Hamdy, *A Gentle Introduction*, in *Deep Learning Pipeline: Building a Deep Learning Model with TensorFlow*. 2020, Apress: Berkeley, CA. p. 3-36.
6. Bhalley, R., *Machine Learning Basics*, in *Deep Learning with Swift for TensorFlow: Differentiable Programming with Swift*. 2021, Apress: Berkeley, CA. p. 1-35.
7. Singh, P., *Introduction to Machine Learning*, in *Deploy Machine Learning Models to Production: With Flask, Streamlit, Docker, and Kubernetes on Google Cloud Platform*. 2021, Apress: Berkeley, CA. p. 1-54.
8. Sarkar, D., R. Bali, and T. Sharma, *Machine Learning Basics*, in *Practical Machine Learning with Python: A Problem-Solver's Guide to Building Real-World Intelligent Systems*. 2018, Apress: Berkeley, CA. p. 3-65.
9. Quinto, B., *Deep Learning*, in *Next-Generation Machine Learning with Spark: Covers XGBoost, LightGBM, Spark NLP, Distributed Deep Learning with Keras, and More*. 2020, Apress: Berkeley, CA. p. 289-348.
10. Kamath, U., J. Liu, and J. Whitaker, *Basics of Deep Learning*, in *Deep Learning for NLP and Speech Recognition*. 2019, Springer International Publishing: Cham. p. 141-201.
11. Taulli, T., *Deep Learning*, in *Artificial Intelligence Basics: A Non-Technical Introduction*. 2019, Apress: Berkeley, CA. p. 69-90.
12. Du, K.-L. and M.N.S. Swamy, *Deep Learning*, in *Neural Networks and Statistical Learning*. 2019, Springer London: London. p. 717-736.
13. El-Amir, H. and M. Hamdy, *Deep Learning Fundamentals*, in *Deep Learning Pipeline: Building a Deep Learning Model with TensorFlow*. 2020, Apress: Berkeley, CA. p. 279-343.
14. Ding, B., H. Qian, and J. Zhou. *Activation functions and their characteristics in deep neural networks*. in *2018 Chinese Control And Decision Conference (CCDC)*. 2018.
15. Sharma, S. and S.J.T.D.S. Sharma, *Activation functions in neural networks.* 2017. **6**(12): p. 310-316.
16. Silaparasetty, V., *Neural Network Collection*, in *Deep Learning Projects Using TensorFlow 2: Neural Network Development with Python and Keras*. 2020, Apress: Berkeley, CA. p. 249-347.
17. Hochreiter, S. and J. Schmidhuber, *Long Short-Term Memory.* Neural Computation, 1997. **9**(8): p. 1735-1780.
18. Singh, V. and N.K. Verma, *Deep Learning Architecture for High-Level Feature Generation Using Stacked Auto Encoder for Business Intelligence*, in *Complex Systems: Solutions and Challenges in Economics, Management and Engineering: Dedicated to Professor Jaime Gil Aluja*, C. Berger-Vachon, et al., Editors. 2018, Springer International Publishing: Cham. p. 269-283.
19. Bohra, A. and N.C. Barwar. *Deep Learning Architectures, Methods, and Frameworks: A Review*. 2021. Singapore: Springer Singapore.
20. El, S.E.M. and I.J.I.J.o.C.A. Kassou, *Authorship analysis studies: A survey.* 2014. **86**(12).
21. Job, I., *Authorship Analysis as a Text Classification or Clustering Problem.*
22. Juola, P., *Authorship Analysis and Attribution*, in *Encyclopedia of Big Data*, L.A. Schintler and C.L. McNeely, Editors. 2020, Springer International Publishing: Cham. p. 1-3.

23. Iqbal, F., M. Debbabi, and B.C.M. Fung, *Authorship Analysis Approaches*, in *Machine Learning for Authorship Attribution and Cyber Forensics*. 2020, Springer International Publishing: Cham. p. 45-56.
24. Wiegmann, M., B. Stein, and M. Potthast. *Overview of the Celebrity Profiling Task at PAN 2019*. in *CLEF (Working Notes)*. 2019.
25. Argamon, S., et al., *Automatically profiling the author of an anonymous text.* 2009. **52**(2): p. 119-123.
26. Rangel, F., et al. *Overview of the 3rd Author Profiling Task at PAN 2015*. in *CLEF*. 2015. sn.
27. Boenninghoff, B., et al., *Deep bayes factor scoring for authorship verification.* 2020.
28. Ordoñez, J., R.R. Soto, and B.Y.J.W.N.o.C. Chen, *Will longformers PAN out for authorship verification.* 2020.
29. Ikae, C.J.W.N.o.C., *UniNE at PAN-CLEF 2020: Author verification.* 2020.
30. Halvani, O., L. Graner, and R.J.W.N.o.C. Regev, *Cross-domain authorship verification based on topic agnostic features.* 2020.
31. Halvani, O., C. Winter, and L. Graner. *On the usefulness of compression models for authorship verification*. in *Proceedings of the 12th international conference on availability, reliability and security*. 2017.
32. Kipnis, A.J.W.N.o.C., *Higher criticism as an unsupervised authorship discriminator.* 2020.
33. Boenninghoff, B., et al. *Explainable authorship verification in social media via attention-based similarity learning*. in *2019 IEEE International Conference on Big Data (Big Data)*. 2019. IEEE.
34. Cumani, S., et al., *Pairwise Discriminative Speaker Verification in the ${\rm I} $-Vector Space.* 2013. **21**(6): p. 1217-1227.
35. Araujo-Pino, E., H. Gómez-Adorno, and G.J.W.N.o.C. Fuentes-Pineda, *Siamese network applied to authorship verification.* 2020.
36. Litvak, M. *Deep dive into authorship verification of email messages with convolutional neural network*. in *Annual International Symposium on Information Management and Big Data*. 2018. Springer.
37. Weerasinghe, J. and R. Greenstadt. *Feature Vector Difference based Neural Network and Logistic Regression Models for Authorship Verification*. in *Working Notes of CLEF 2020-Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*. 2020. CEUR-WS. org.
38. Peñas, A. and A. Rodrigo, *A simple measure to assess non-response.* 2011.
39. Bevendorff, J., et al. *Generalizing unmasking for short texts*. in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019.
40. Jang, B., et al., *Bi-LSTM model to increase accuracy in text classification: Combining Word2vec CNN and attention mechanism.* 2020. **10**(17): p. 5841.