Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Verteilte Systeme
Fachgebiet Wissensbasierte Systeme (KBS)

Debunking Medical Misinformation - Rootcauses, Benchmarks, and Explanations

Bachelorarbeit

im Studiengang Informatik

von

Baraa Ragab Tbab

Prüfer: Prof. Wolfgang Nejdl
Zweitprüfer: Prof. Niloy Ganguly
Betreuer: Thanh Tam Nguyen

Hannover, 22.02.2022

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 22.02.2022

Baraa Ragab Tbab

Summary

According to IBM, Machine learning is a branch of artificial intelligence that focuses on using algorithms and data to simulate the ways in which humans learn [1]. Machine learning is used to create models capable of solving some problems that are sometimes relatively difficult for humans to solve, such as classification and prediction. In this thesis, machine learning models were created to analyze medical data (textual data) related to COVID-19, and the goal is to categorize this news between true and false news. Several experiments were also conducted on created models, and their performance was tested and measured in terms of accuracy to see if such models can be relied upon in practical life.

Contents

Chapter 1

Introduction

1.1 Motivation

In recent decades, machine learning (ML) has enabled humanity to achieve great and rapid progress in all areas of life, spanning simple everyday tasks to complex topics. For example, machine learning technology is embedded in social media programs and search engines, such as Google, to determine appropriate content for each user. Machine learning is also used in targeted advertising, whereby relevant advertisements are directed at individuals following the identification of their interests via analyses of their purchasing behavior; the benefits being an easier electronic shopping experience for the user and reduced advertisement costs for the firm. Machine learning is also used to make highly accurate predictions based on available information in areas such as medicine, economics and literature. In medicine, machine learning is used to classify data, predict diseases once symptoms are known, manufacture and develop medicines, fight epidemics and predict where they will spread. Additionally, machine learning is an integral component of various ongoing health projects, such as the Human Genome Editing Project (Casper), which is intended to improve species and combat diseases. Machine learning models have outperformed humans in many classification and prediction tasks, as well as in mind games such as Chess and Go. To design machine learning programs that perform the above-mentioned functions with high accuracy, a large dataset to teach and test the machine is necessary as the process is largely based on analyzing data and extracting patterns that are humanly understandable.

1.2 Problem Definition

Humankind, represented by the World Health Organization, is continuing to undertake great efforts to fight epidemics such as Covid-19 and its variants. These efforts face multiple obstacles which make combating the epidemic

more difficult, one of which is false news and pervasive rumors on social media and the Internet.

This thesis sheds light on the vast spread of false news surrounding Covid-19, which in turn hinders the fight against the epidemic, causing further economic and health devastation globally. Confronting false news is essential to eradicating the epidemic. Various medical reports highlight a number of the catastrophic results stemming from the spread of false news related to the Covid-19 virus, such as heightened feelings of panic, fear, anxiety and depression, along with attacks on medical personnel, and, in some countries, deaths that could have been otherwise avoided. Users blindly trusting false news and failing to verify the authenticity of their claims with official sources further exacerbate the issue and contribute to the rapid spread of misinformation.

1.3 Thesis Objective

In this thesis, I will focus on building machine learning models that are capable of analyzing and understanding texts related to the Covid-19 virus and its modifiers, allowing them to accurately classify the texts as true or false. To achieve this, I will first build four models with different algorithms such as KNN, Naive-Bayes and SVM. Next, I will train and test each model separately and subsequently evaluate and measure their results and accuracy using Evaluation Metrics and Confusion Matrix. I will present the results in the form of tables and graphs. Finally, I will create an ensemble model using the previously created models and apply the aforementioned steps to compare its results and accuracy with the results of the previous models when they were tested separately from each other. Each model requires a large dataset for training and testing therefore obtaining this set will be the initial step in this thesis. I will be collecting a sizeable dataset consisting of articles and tweets that contain news related to Covid-19, some of which are false news and others which are true. To achieve better results, I will utilize data preprocessing and data cleaning, ensuring that the dataset is balanced. Each model will be applied to the available dataset before and after the preprocessing and each model will be first trained on the dataset consisting of tweets only and later on the dataset consisting of articles and tweets together with the aim of comparing the results in all cases.

I will use the explainer (LIME) to illustrate how each word in the article or tweet affected the model in making its decision during the classification process. In conclusion, I will use the word cloud to show which words are most frequently used in both false and true news.

1.4 Thesis Structure

In Chapter 2, I explain the theoretical and mathematical basics relevant to this thesis. First, I explain what natural language processing (NLP) is and how to use it in machine learning models, along with a simple example. Next, I delve into one of the branches of natural language processing, text classification, where I explain the meaning of the term and how to use it within an illustrative example. I also demonstrate the need to use Tokenizer to convert words and text into vectors and explain the conversion mechanism. Finally, I explain the algorithms that will be used in the machine learning models that I create along with the difference between their structures and the mechanism for evaluating the results of each model using different techniques.

In Chapter 3, I explore how I preprocessed the dataset, found the right tokenizer for each of the models used and created each form individually.

In Chapter 4, I declare which experiments have been performed and which setup and data sets have been used. I subsequently present the results of these experiments. Then I used the LIME Explainer with the Naive-Bayes and SVM Models to interpret it. In the conclusion of Chapter 4, I evaluate and interpret these results.

In Chapter 5, I briefly summarize what I have done in this thesis and draw a conclusion surrounding natural language processing (NLP) in machine learning and whether the results of these models are always trustworthy. I also provide an outlook on further research questions and suggestions that have been raised from the results of this thesis.

Chapter 2

Fundamentals

2.1 What is Natural Language Processing?

According to IBM, natural language processing (NLP) is a branch of artificial intelligence (AI) that strives to build machines that understand and respond to text or voice data in a similar way to humans [2]. NLP is at the core of various tasks, such as text translation, spam filters, grammar correction, missing text prediction, virtual assistants and text classification. In other words, we can say that NLP is utilized in ML models to make their success possible as it allows the ML model to understand the structure and meaning of human language through syntax analysis.

2.2 What is Text Classification?

Two types of data exist in computer science. The first type is structured data which contains information that adheres to a predetermined model and is tabular so that there is a relationship between columns and rows [26]. The second type is unstructured data, which does not contain a predetermined data model and is neither structured nor tabular [26]. Unstructured data can often contain a mixture of useful and not useful information for machine learning models, making it difficult to analyze such data with them [25].

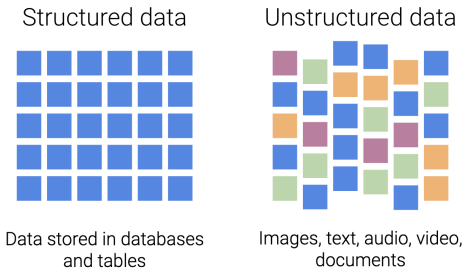

Figure 2.1: Types of data. source: Grace Kim, accern [3]

Texts are an example of unstructured data, as a set of data consisting of text does not follow a single, predefined structure. This type of dataset can be handled using NLP. As mentioned in section 2.1, NLP is at the core of many tasks and, if we consider each of these tasks an NLP problem, we can determine that most NLP problems can be formulated as classification problems [29].

Text classification is is one of the NLP problem. In other words, text classification is the process of structuring a textual dataset based on its content according to predefined, structured classes [37]. Using a movie review dataset as an example, reviews could be classified into either positive or negative with the help of machine learning models as the model analyzes each review and makes it understandable by the machine.

Review	Sentiment
no such thing is sort of a minimalist beauty and the beast , but in this case th ˙·.	negative
not for everyone , but for those with whom it will connect , it's a nice departu ˙·.	positive
it virtually defines a comedy that's strongly mediocre , with funny bits surfaci ˙·.	negative
meandering and confusing .	negative
if you can keep your eyes open amid all the blood and gore , you'll see del toro ˙·.	positive

Figure 2.2: Sample Data: Movie Review Sentence Polarity. source: Bo Pang, Lillian Lee, Year: 2005, Wolfram Data Repository [4]

2.3 Text Embedding

Automatic interpretation of the semantics of natural languages is not an easy task. It is necessary to convert or encode the words in such unstructured textual dataset into numerical values that can be linked to a structured dataset. Doing this makes it possible for the machine to analyze using data mining techniques and algorithms, a process that is called tokenization [36].

For example, using the text "I love my mother and I love my father" we can begin to encode the words in the text by assigning the word "I" a numerical value of "001" and the word "love" a numerical value of "002" allowing us to tokenize the text word by word.

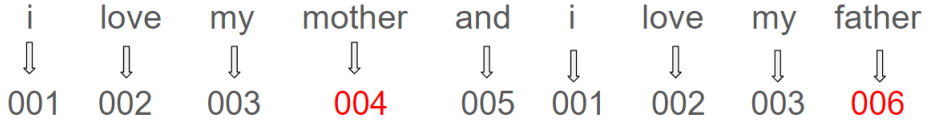

Figure 2.3: Tokenization example.

2.3.1 Tensorflow Tokenizer

There are numerous tools available for text tokenization, one of them being the Tensorflow Tokenizer [5]. The key objective of the Tensorflow Tokenizer is to simply create a dictionary, adding each new word in the text or textual dataset to it as a new numeric token, then representing the text in a numerical sequence in the correct order. Tensorflow tokenizer is flexible as it can identify the presence of a punctuation mark at the end of a word and delete it, adding the word to the dictionary if it does not exist without creating a new token.

How to create the tokenizer and represent a text in a numeric sequence, as well as what hyperparameters are needed for better performance, will now be explained. The Tensorflow Tokenizer can be created using the class "tf.keras.preprocessing.text.Tokenizer" [5]. First a dictionary of words must be created and then the text can be represented in a numerical sequence [5]. The hyperparameters of Tensorflow tokenizer to create a dictionary and to represent the text in a numerical sequence are as following:

- num_words: the maximum number of words to keep, based on word frequency. Only the most common num_words-1 words will be kept [5].
- oov_token = "<OOV>": if given, it will be added to "word_index" (dictionary) and used to replace the words which do not exist in the dictionary "out-of-vocabulary" <OOV> during representing the text in a numerical sequence [5].

There are other hyperparameters for the Tensorflow Tokenizer, however, the above are the only parameters needed for this thesis (for more details [5]).

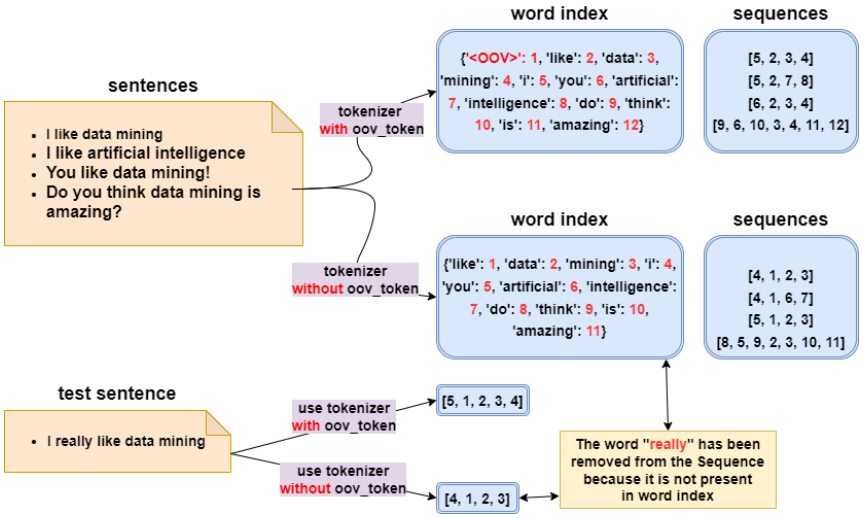

Figure 2.4: Tensorflow Tokenizer example.

As we can see in Figure 2.4, the list returned from the "text-to-sequence" method includes sequences that do not have the same length. Therefore, constructing the features of the machine learning model with the result is not possible. Instead, the sequence lengths must be standardized by either adding zeros to the shorter sequences or truncating tokens from the longer sequences, resulting in all sequences being uniform in length. To perform this process, the following parameters are required [6]:

- sequences: List of sequences (each sequence is a list of integers) [6].
- maxlen: Optional, maximum length of all sequences. If not provided, sequences will be padded to the length of the longest individual sequence [6].
- padding: 'pre' or 'post' (optional, defaults to 'pre') pad either before or after each sequence [6].
- truncating: 'pre' or 'post' (optional, defaults to 'pre') remove values from sequences larger than maxlen, either at the beginning or at the end of the sequences [6].

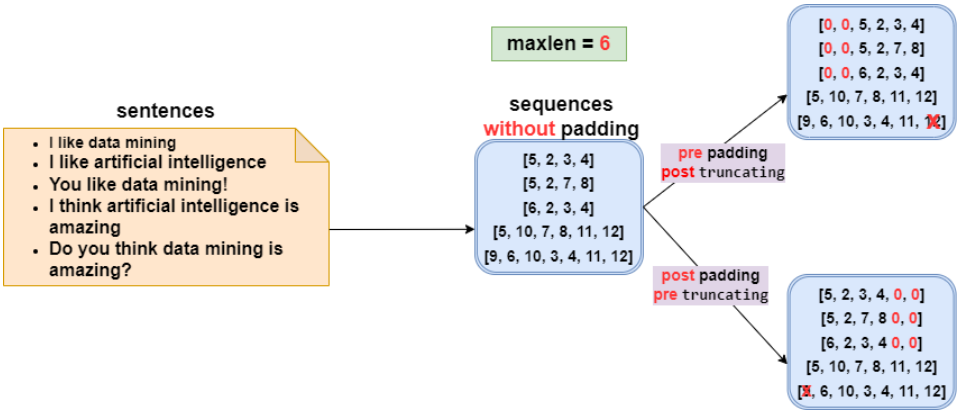

Figure 2.5: padding and truncating sequences example.

2.3.2 BERT

According to Google AI Language, BERT stands for Bidirectional Encoder Representations from Transformers. BERT advances the state of the art for many NLP tasks [30]. There are various types of BERT models, one of them being the 'paraphrase-MiniLM-L6-v2' (SentenceTransformer). SentenceTransformer belongs to the "SBERT" family [7], which is considered a sophisticated model for sentence, text and image embedding [41]. The 'paraphrase-MiniLM-L6-v2' model will be utilized within this thesis.

The 'paraphrase-MiniLM-L6-v2' model can be employed for many tasks like clustering and classifying [8], and has two different embedding modes. The first embedding mode is called 'token embedding' as it maps each individual token in a sentence to a 384-dimensional vector space whereas the second embedding mode is called 'sentence embedding' as it maps sentences and texts to a 384-dimensional vector space.

The 'paraphrase-MiniLM-L6-v2' model can be created using the class " SentenceTransformer ('paraphrase-MiniLM-L6-v2') ", after which the method "encode" can be used to make "token_embeddings" or "sentence_embeddings" of datasets. The method "encode" draws upon the following parameters:

- sentences: the sentences to embed [9].
- output_value: Default sentence_embedding, to get sentence embeddings. Can be set to token_embeddings to get wordpiece token embeddings. Set to None, to get all output values [9].
- convert_to_numpy: If true, the output is a list of numpy vectors. Otherwise, it is a list of pytorch tensors [9].

There are additional parameters for the encode method, however, only the parameters required for this experiment have been detailed (for more details [9]).

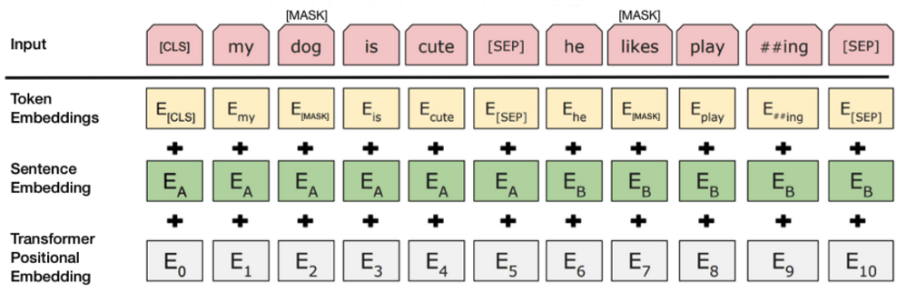

Figure 2.6: BERT input representation. Author :Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. Journal: arXiv preprint arXiv:1810.04805 Year: 2018 [10]

2.4 Keras Models

According to François Chollet, author of "Deep Learning with Python" book: "Keras is a deep learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model" [27]. There are two methods of composing models in Keras:

- sequential composition [34].

- functional composition [34].

Sequential composition will now be further explored as it is the model utilized throughout this thesis. The core idea of sequential composition is to stack different Keras layers in sequential order. Each layer contains exactly one input and one output tensor [11]. In the sequential Keras model, the outputs of each layer are considered inputs to the next layer, with the exception of the first and last layers in which the dataset is considered an input within the first layer and the output of the last layer is considered the final result of the sequential model.

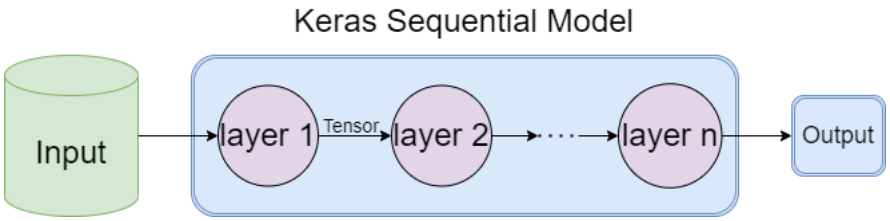

Figure 2.7: Keras sequential model with n layers example.

Keras layers are the core building blocks of Keras sequential models. There are numerous types of Keras layers, such as fully connected layers, convolutional layers, pooling layers, recurrent layers and normalization layers [12].

The dense layer is a type of core layer that belongs to fully connected layers as this layer connects each input to each output within its layer [12]. There are different parameters that define the properties of this layer, such as:

- units: Positive integer, dimensionality of the output space. [13].

- activation: Activation function to use. If nothing is specified, no activation is applied (more details in the following paragraph) [13].

There are additional parameters for the Dense layer, however, the above are the only required for this experiment (for more details [13]).

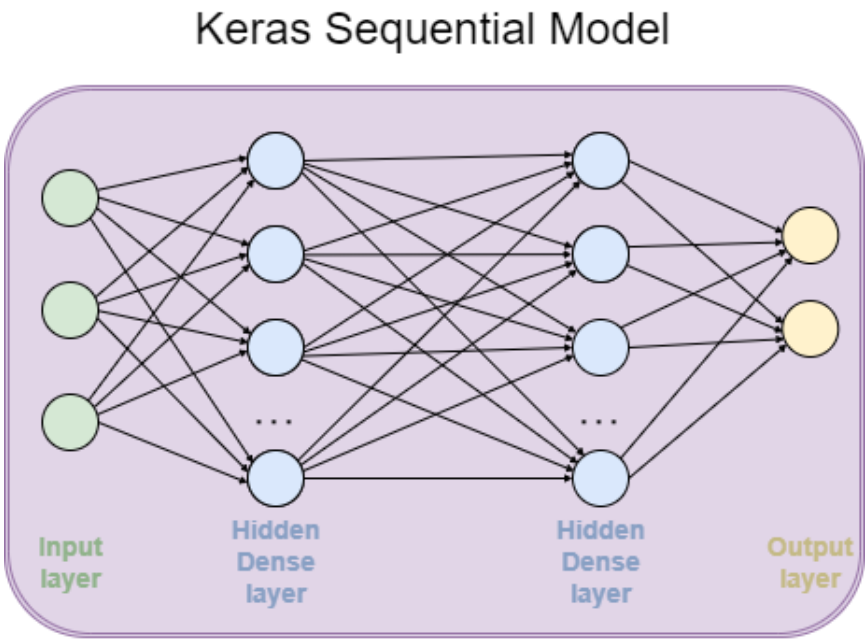

Figure 2.8: Keras sequential model with 2 hidden Dense layers example.

When creating a dense layer, an activation function can be added using the activation parameter. There are various values that determine which activation function is most desirable for use, with typical activation functions being the following [14]:

- relu function: relu function is abbreviation for the rectified linear unit activation function. relu uses the following parameters [14]:
  - x: Input tensor or variable.

$$\text{x} = [\text{-7, -4, 0, 5, 12}] \quad \longrightarrow \quad \text{output} = [\text{0, 0, 0, 5, 12}] \tag{2.1}$$

  - alpha: A float that governs the slope for values lower than the threshold.

$$\text{input} = [\text{-7, -4, 0, 5, 12}] \xrightarrow{\text{alpha=0.5}} \text{output} = [\text{-3.5, -2, 0, 5, 12}] \tag{2.2}$$

  - max_value: A float that sets the saturation threshold (the largest value the function will return).

$$\text{input} = [\text{-7, -4, 0, 5, 12}] \xrightarrow{\text{max\_value=4.0}} \text{output} = [\text{0, 0, 0, 4, 4}] \tag{2.3}$$

– threshold: A float giving the threshold value of the activation function below which values will be damped or set to zero.

$$\text{input} = [-7, -4, 0, 5, 12] \xrightarrow{\text{threshold}=5.0} \text{output} = [0, 0, 0, 0, 12] \tag{2.4}$$

- sigmoid function:

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

$$\text{input} = [-20, -1.0, 0.0] \longrightarrow \text{output} = [2.0611537e\text{-}09, 2.6894143e\text{-}01, 5.0000000e\text{-}01]$$

There are other activation functions, however, the previously mentioned functions are the only required for this experiment (for additional details, reference [14]).

The embedding layer is an additional type of core layer [15] that is often used for text analysis. The embedding layer must first integrated into a model where it performs embedding operations in the input layer. The embedding layer takes a 2D tensor of integers of shape as the input, where each entry is a sequence of integers that must have the same length, and then converts the input into dense vectors of fixed size [27]. When creating this layer, three different values must be specified, which are the following [15]:

- input_dim: Integer. the size of the vocabulary.
- output_dim: Integer. The output size of the vector.
- input_length: The length of the input sequences to the model.

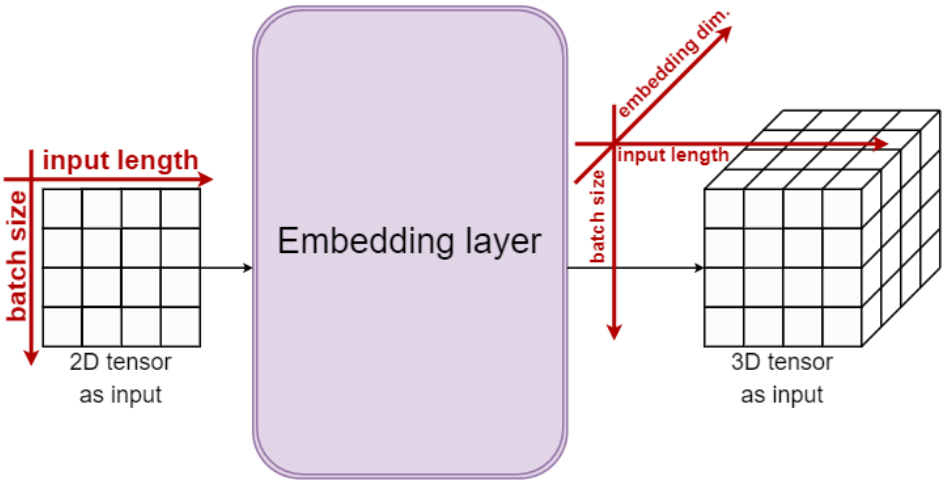

Figure 2.9: Embedding layers.

Pooling layers are another form of Keras layers which include different layers, one of them being the GlobalAveragePooling1D layer. The GlobalAveragePooling1D layer takes 3D tensor with shape (batch_size, steps, features) as input and computes the average of all time steps for each feature dimension, the output being a 2D tensor with shape (batch_size, features), therefore reducing the number of dimensions [16].

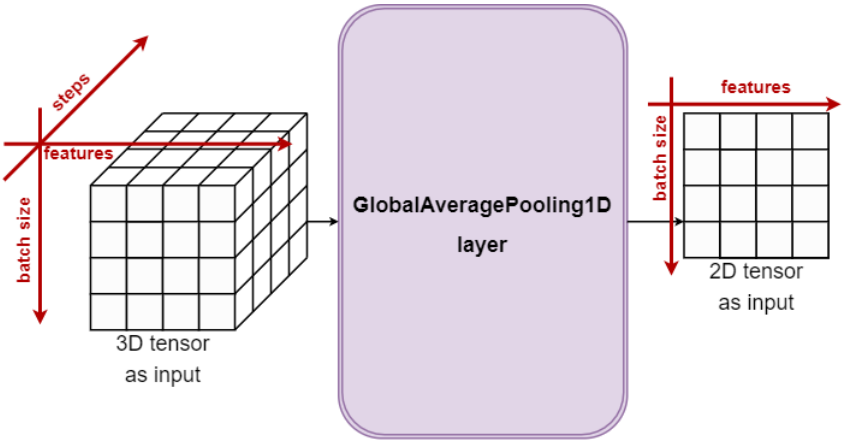

Figure 2.10: GlobalAveragePooling1D layers.

When training a keras model, the number of epochs must be set. This number is used to separate training into distinct phases. It is considered useful for periodic evaluation at the end of each epoch [17].

2.5 k-Nearest Neighbors Algorithm

The K-Nearest Neighbors Algorithm (KNN) is one of the simplest and most widely used supervised machine learning algorithms. The algorithm can be applied to solve both classification and regression problems [39]. The dataset is typically tabular as it consists of rows and columns, with the rows called "instances" and the columns called "features." The KNN algorithm requires a dataset with a label for each instance and can predict the class (label) of a new instance by calculating the distance between the new instance and each instance in the dataset. The K nearest neighbors to the new instance is subsequently identified. The final step is computing the mode of the K labels, which will be the predicted class for the new instance [39]. To compute the distance, the following is used:

- Supremum distance.
- Manhattan distance.
- Euclidean distance.

- Minkowski distance.

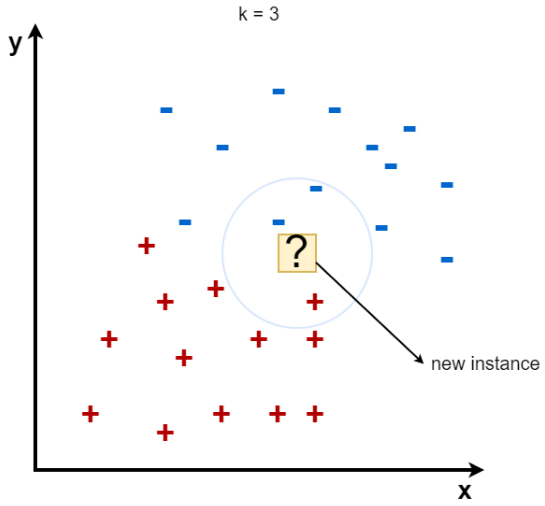

Figure 2.11: in this example for k = 3 the result should be "-".

Choosing the correct value for K is dependent on the type and size of the dataset. To choose the correct K, the KNN algorithm must be run several times with different values in order to choose the K that increases the accuracy of the model.

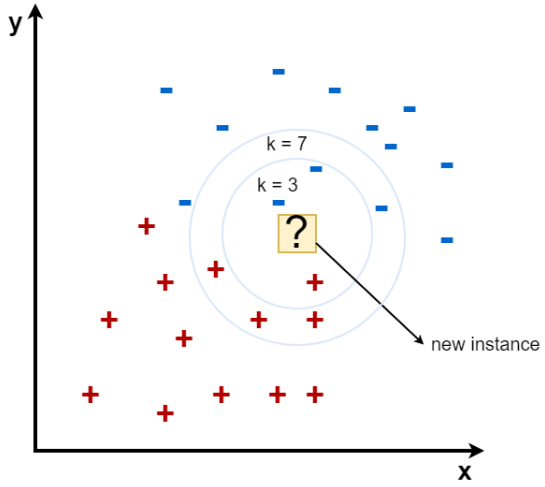

Figure 2.12: Value of k can change the result.

As shown in Figure 2.12, the value of K can change the result of the prediction. Usually, the K should not be so small as to cause the result

to be sensitive to outliers, nor should it be too large so that other classes negatively affect the result.

2.6 Naive-Bayes Algorithm

The Naive-Bayes algorithm is a probabilistic algorithm as it mainly depends on applying Bayes' theorem. Bayes' theorem can be used to calculate the posterior probability P(c|x) by calculating P(c), P(x), and P(x|c) [43].

$$P(c|x) = \frac{P(x|c)p(c)}{p(x)} \tag{2.6}$$

Height	Color of Hair	Color of Eyes	Gender	Class
Low	Blond	Blue	F	yes
Low	Blond	Blue	M	no
Low	Brown	Hazel	F	no
Tall	Brown	Blue	M	yes
Tall	Brown	Hazel	M	no
Low	Red	Blue	F	no
Tall	Blond	Hazel	M	yes
Tall	Red	Hazel	F	yes

Figure 2.13

Suppose, for example, that there is a new instance x : [Height = "Low", Color of Hair = "Blond", Color of Eyes = "Hazel", Gender = "M"] and it is desired to predict its class based on the data set in Figure 2.13. To classify the new instance x, it is necessary to estimate P(yes|X) and P(no|X), compare the results and select the highest of them.

$$p(yes|x) = \frac{p(x|yes)p(yes)}{p(x)}$$
$$p(yes) = \frac{4}{8} = \frac{1}{2}$$
$$p(x|yes) = p(Low|yes) * p(Blond|yes) * p(Hazel|yes) * p(M|yes)$$
$$p(Low|yes) = \frac{1}{4}, p(Blond|yes) = p(Hazel|yes) = p(M|yes) = \frac{2}{4}$$
$$p(x|yes) = \frac{1}{4} * \frac{2}{4} * \frac{2}{4} * \frac{2}{4} = \frac{1}{32}$$
$$p(x|yes)p(yes) = \frac{1}{64} = \frac{2}{128}$$

$$p(no|x) = \frac{p(x|no)p(no)}{p(x)}$$
$$p(no) = \frac{4}{8} = \frac{1}{2}$$
$$p(x|no) = p(Low|no) * p(Blond|no) * p(Hazel|no) * p(M|no)$$
$$p(Low|no) = \frac{3}{4}, p(Blond|no) = \frac{1}{4}, p(Hazel|no) = p(M|no) = \frac{2}{4}$$
$$p(x|no) = \frac{3}{4} * \frac{1}{4} * \frac{2}{4} * \frac{2}{4} = \frac{3}{64}$$
$$p(x|no)p(no) = \frac{3}{128}$$

$$p(x|yes)p(yes) < p(x|no)p(no) \Rightarrow \text{x will be classified in the class "no"}$$

Figure 2.14

2.7 Support Vector Machine

Support Vector Machine (SVM) is a classification algorithm that used in ML. In this algorithm instances are represented as points on the coordinate axis in n dimensions, where n is the number of features for each instance. The goal is to find a hyperplane that will separate data based on its class [42]. The following figure shows an example of data distribution on a 2D coordinate axis and gives a set of possible solutions:

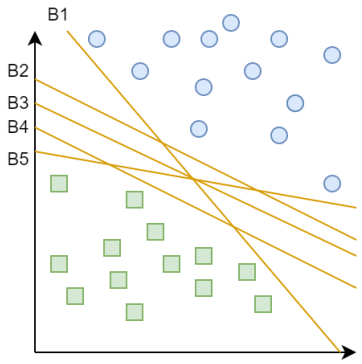

Figure 2.15: In this example the best solution is B3.

From the previous figure 2.15, we can see that there is more than one solution (B1, B2, ...), but the goal is to find the best solution that results in the largest minimum distance for training instances. Twice this distance is called margin.In another word the best solution is the hyperplane that maximizes the margin. There are two main types of SVM, Linear SVM and Non-linear SVM, where the selection of the appropriate hyperplane depends on the distribution of data on the coordinate axis [42]. The following figure shows an example of SVM Types:

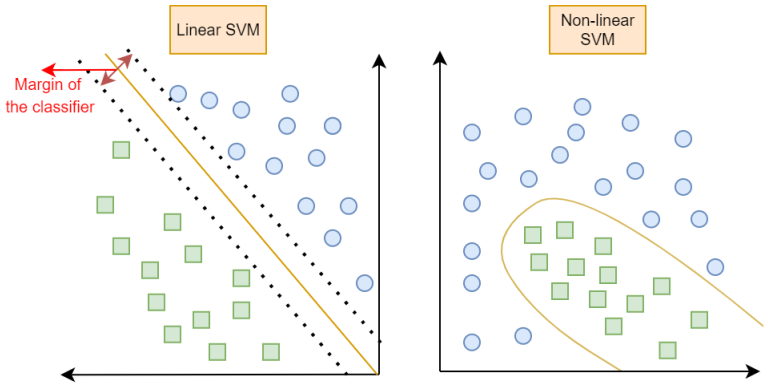

Figure 2.16: SVM example.

2.8 Evaluating machine learning models

Evaluating machine learning models is a process that is as equally important as creating the model itself. The model cannot be utilized in real life without first evaluating its results and testing its performance, especially if the purpose of the model is to solve a sensitive problem. Usually, the model is tested following its training where there is a part of the dataset dedicated to testing as this part of the dataset has not been presented to the model to be tested before. There are several methods and tools that can be used to analyze the results after testing the model, such as confusion matrix and classifier evaluation metrics [38].

2.8.1 Confusion Matrix

A confusion matrix is a tool used to analyze the results of a classification model test with which the classifier's performance can be measured. The confusion matrix is a 2D matrix (m x m) where m is the number of classification classes in the dataset. The confusion matrix is built from the test dataset where columns express the predicted class and the rows express the actual class [32].

	Predicted "true"	Predicted "false"	Totals
Actual "true"	TP (true positive)	FN (false negative)	TP + FN
Actual "false"	FP(false positive)	TN (true negative)	FP + TN
Totals	TP + FP	FN + TN	

Figure 2.17: confusion matrix structure.

- (TP): instances correctly predicted as true (predict = "true" and real class = "true") [28].
- (FN): instances predicted as false, but they belong to the true class (predict = "false" and real class = "true") [28].
- (FP): instances predicted as true , but they belong to the false class (predict = "true" and real class = "false") [28].
- (TN): instances correctly predicted as false(predict = "false" and real class = "false") [28].

The following is an example:

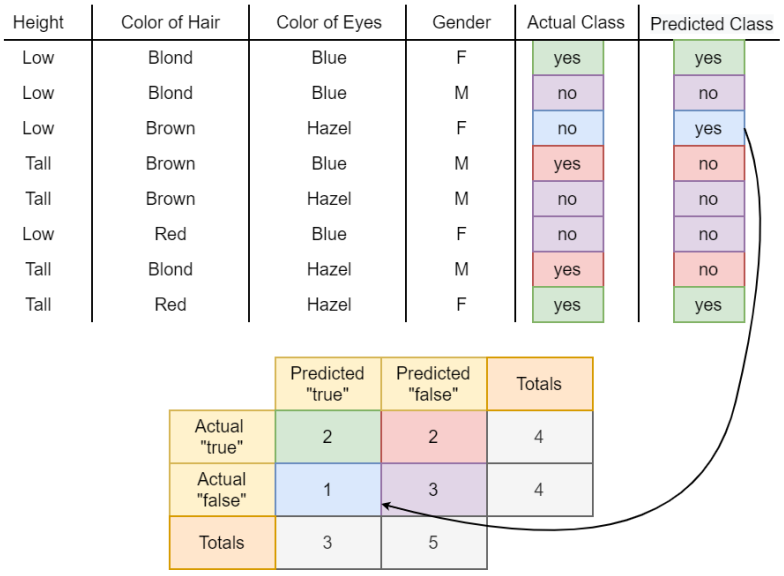

Figure 2.18: confusion matrix example.

2.8.2 Classifier evaluation metrics

There are a variety of classifier evaluation metrics that can be calculated based on the results of the confusion matrix. These metrics assist in better understanding the model's test results [28]. The most famous of these metrics are:

- Accuracy: accuracy is a metric that describes the result of dividing the number of correctly predicted results by the number of total prediction cases and is often used with classification problems [33].

$$accuracy = \frac{TP+TN}{TP+FP+FN+TN} \tag{2.7}$$

- precision: precision is a metric that describes the ratio of true positive (TP) to all positive predictions, that are predicted by the model (TP+FP) [33].

$$precision = \frac{TP}{TP+FP} \tag{2.8}$$

- recall: recall is a metric that describes the ratio of true positive (TP) to all positive instances in the dataset (TP+FN) [33].

$$recall = \frac{TP}{TP+FN} \tag{2.9}$$

- f1-score: f1-score is a metric that describes the harmonic mean of precision and recall. The model will receive a high f-score if precision and recall are high [33].

$$f1 - score = 2 * \frac{precision * recall}{precision + recall} \tag{2.10}$$

2.9 Machine Learning Models Explanation

When creating a machine learning model, training and testing the model alone is insufficient – even in instances where the evaluation metrics of the model are high. The ability to explain the decisions and predictions made by machine learning models is as important of an element as creating the model itself, the goal of which is to enable humans to understand the operations that the model performs to make its decisions, leading to performance improvement [24].

2.9.1 LIME

LIME is one of the most important of the various tools that help interpret the behavior of machine learning models. In this thesis, the general interaction between LIME and machine learning models will be explained and an illustrative example of text classification models will be provided. After the model is created, trained, and tested, LIME analyzes the results of the model and informs users to what degree each feature in the instance affects the model's decision while solving a problem such as prediction or classification. LIME assigns each feature a weight between -1 and 1. Features with weights close to 0 do not have a significant effect on the model when it makes its decision. Conversely, features close to 1 or -1 in weight have a stronger effect. After determining the weight of each feature, the positive and negative weights are summed to determine prediction probabilities for each class [31]. Using the classification of x or y as an example, the texts are instances and the words are features. After giving each word in the text a weight, words with a weight close to 1 have a strong effect on the model

to classify the text in the category x and words with a weight close to -1 have a strong effect on the model to classify the text in the category y. The following figure illustrates an example of LIME being used to interpret the results of the random forest model as the model classifies the passengers of the Titanic:

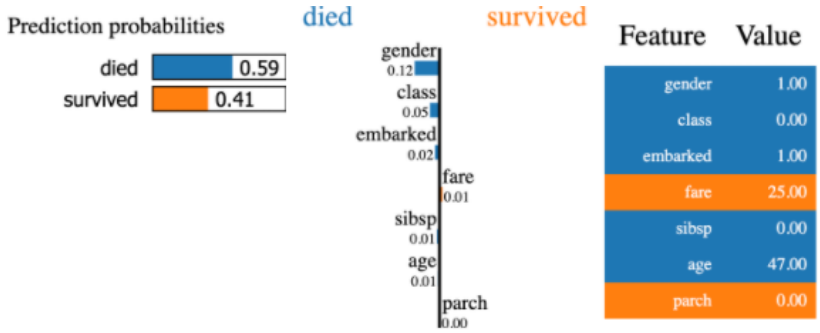

Figure 2.19: A plot of LIME model values for the random forest model and passenger Henry for the Titanic data. source: drwhy [18]

2.10 Word Cloud

The goal of a word cloud is to analyze text and identify the most frequently used words. Word clouds are used in machine learning for the purpose of analyzing the performance of the model and knowing which words affected the performance of the model [35]. Word clouds come in various shapes and colors and font sizes can be controlled. The following figure illustrates an example of a word cloud:

Figure 2.20: word cloud example . source: petitessen [19]

Chapter 3

Methodology and Implementation

In this chapter, I show how I created the dataset and how I performed the data preprocessing steps. In addition, I explain what the hyperparameters values are used when creating the tokenizer and why I chose these values. I will also explain how i created machine learning models and what are the hyperparameters values used when creating each model. I will also explain the mechanism used to choose the appropriate tokenizer for each model.

In this chapter, the method in which the dataset was created will be shared along with how the data preprocessing steps were performed. In addition, the hyperparameters values used when creating the tokenizer and why they were chosen will be explained. Finally, how the machine learning models were created and which hyperparameters values were used in doing so along with the mechanism used to choose the appropriate tokenizer for each model will be explored.

3.1 Creating and Preprocessing The Data

The process of creating a clean and effective dataset is highly important as the dataset will be used to train and test the machine learning model, thus affecting its ultimate performance. A model can be poorly trained and tested because the dataset used is ineffective, leading to unfavorable results when the model is well designed.

In this thesis, several datasets related to Covid-19 news have been collected while taking into account the avoidance of duplication, the source of these datasets will be mentioned in the next chapter. Some of these datasets contain tweets and others contain articles. Out of the four datasets were created, the first one contains only short tweets without data preprocessing process, the second contains the same short tweets but with data preprocessing, the third contains short tweets and relatively long articles

without data preprocessing process, and the fourth contains the same data as the third dataset but with data preprocessing. The goal of using multiple and differing datasets is to compare the performance of the model when the dataset used changes. For the purpose of this thesis, the performance of the classification model will be compared in two cases: one with short text only and the other with short and long texts together. Both cases will be tested with and without data preprocessing. Figure 3.1 explains the content of each data set and the purpose of creating several datasets.

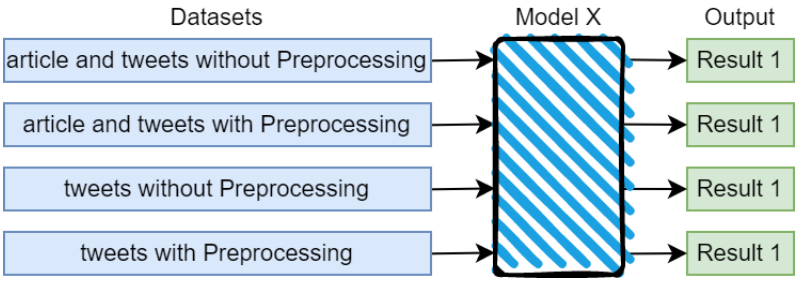

Figure 3.1: goal is to compare the results when the dataset changes.

The following figure shows the data distribution according to text-length and type:

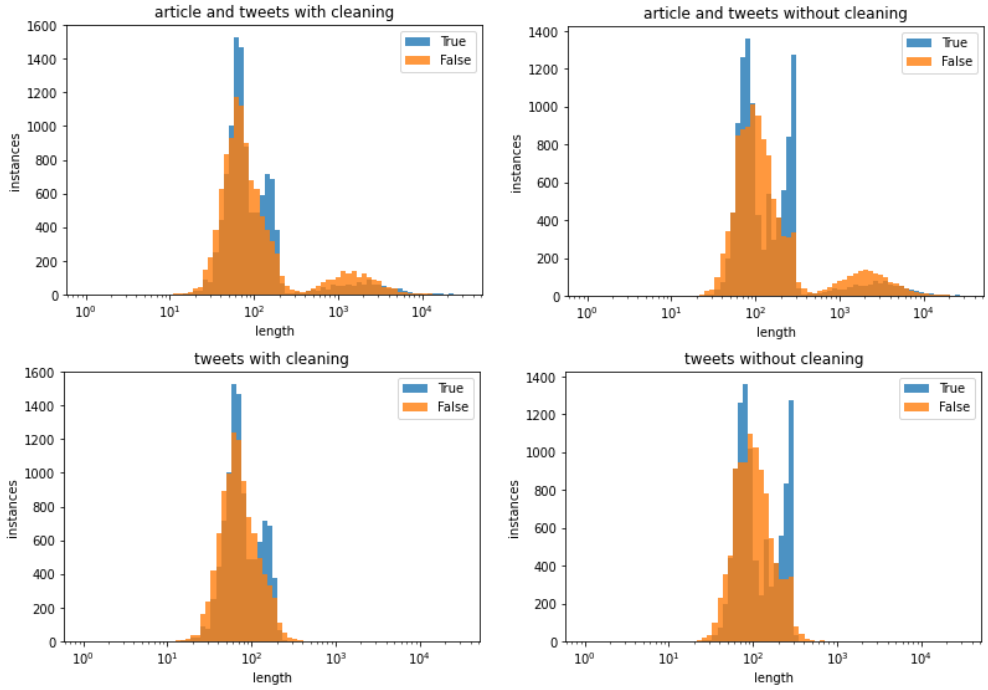

Figure 3.2: Comparing between the datasets distribution.

The dataset used consists of two features, namely the text and the label. The text contains news related to Covid-19 and the label is used to classify this news {real, fake}. As mentioned in Section 2.2, texts are unstructured data and can often contain a mixture of useful and not useful information. To avoid this negativity, the process of cleaning the data was carried out in this thesis according to the following steps for each instance in the dataset:

1. remove the web links("https:\\").
2. Remove Mention(@).
3. Remove Hashtag(#).
4. Remove tab(\t) and newline(\n) characters.
5. Remove the digits.
6. Remove every character, that is not in the ascii table.
7. Remove the duplicate whitespaces.
8. Convert all letters to lowercase.
9. Remove the punctuation marks.
10. Remove stop words(below is a detailed explanation).

In the English language, there exists a group of common words that are used in most texts and sentences and which do not have a significant impact on the meaning of the text, such as "the, or, for, a, of, and in," among others. These words are referred to as "stop words." In the event that such words are present in the false and real news, it may mislead the model while classifying the text. To avoid this, deleting these words from the texts before starting to train and test the model is the best solution [40].

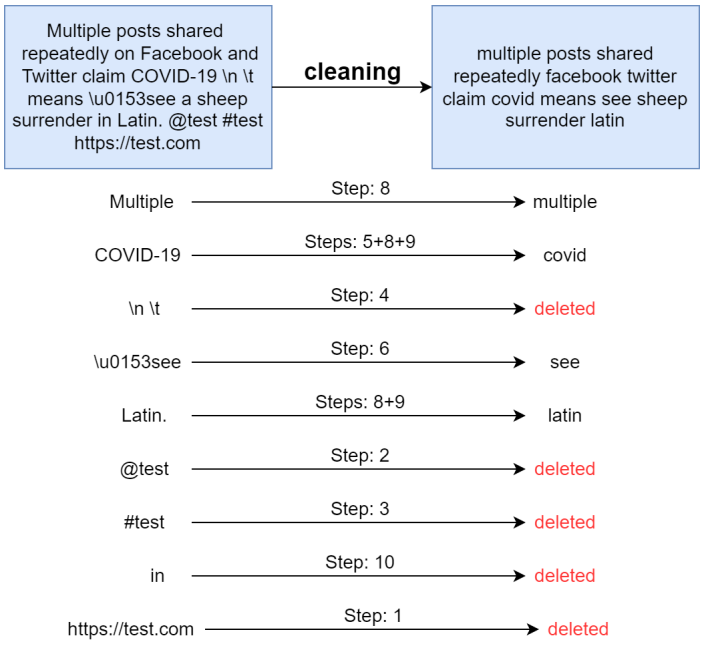

Figure 3.3: Illustrative example of data cleaning steps.

3.2 Create and Find the Right Tokenizer for each Model

As mentioned in Section 2.3.1, a tokenizer must be used to deal with text data, therefore this thesis utilizes Tensorflow Tokenizer and Bert Tokenizer. To create Tensorflow Tokenizer, the hyperparameters were set as follows:

1. num_words:To determine the value of this parameter, the words in the available dataset were counted. The dataset contained 46,149 words, therefore 45,000 as a value for the size of the dictionary is sufficient for this experiment.

2. oov_token: This parameter was set to "<OOV>" to preserve the original length of the text in case it contains new words that are not stored in the Tokenizer dictionary, resulting in the text losing part of its meaning.

After creating the Tensorflow tokenizer, the text was represented in a numeric sequence. The numerical sequences were of different lengths, making it difficult to construct the features for the machine learning model. To remedy this, the lengths of these sequences were standardized to a maximum length of 250 for a dataset containing tweets and 2,000 for a dataset containing tweets and articles. In sequences longer than the standardized

length, the values from the end were removed. Conversely, in sequences shorter than the standardized length, a quantity of zeros was added at the end of the sequence until it was equal in length to the maximum length.

Following several manual tests of the tokenizer using the available dataset and models that created it, the performance and accuracy of some models were found to be unfavorable, such as was the case for the KNN model. The reason for this is that the Tensorflow tokenizer, while in the process of padding, was, in some cases, adding a sizeable number of zeros to the short sequences, leading to a large similarity between sequences that were fundamentally different from each other and sometimes made the model perform poorly. Due to this, 'paraphrase- MiniLM-L6-v2' (SentenceTransformer) model as encoder was used. This type of model is able to perform several tasks, one of which is encoding the text into a sequence of 384 dimensions. As a final step, the results of the Tensorflow tokenizer and SentenceTransformer encoder were compared using the models previously created. In each model, the tokenizer/encoder that was found to have the better results was used.

3.3 Models Implementation

Four machine learning models using several algorithms were created. Below, the method in creating each model and the hyperparameters values used are explained. It should be noted that the details of the Naive-Bayes and SVM models are not discussed in this chapter, because both models did not require hyperparameters and initial settings for the experiments and the dataset did not also require tokenization for these models.

3.3.1 Keras Sequential Model

The first model created was the Keras sequential model. To determine the number of epochs for this model, the greedy search method within the range [10,30] was used to reach the highest accuracy value and avoid overfitting. This model consisted of 4 layers in the following order:

1. Embedding layer: Within this layer, the model can compute the embedding vector for each word in training dataset. In this layer, the hyperparameters values were set as follows:

    - input_dim: the same value of the num_words in Tensorflow tokenizer (45000).
    - output_dim: After several manual experiments, the value of this parameter to 18 was set due to the keras sequential model producing the best results (accuracy) with this value.

- input_length: It depends on the dataset used (only tweets dataset the value will be 250, article and tweets dataset the value will be 2000).

2. GlobalAveragePooling1D layer: This layer was used to compute the average of all time steps for each feature dimension. No parameters were used for this layer.

3. Dense layer: This layer was used to compute the 'relu' activation function for the output from the GlobalAveragePooling1D layer. After several manual experiments, the dimensionality of the output space was set to 24 because the keras sequential model outputs the best results (accuracy) at this value.

4. Dense layer: The output layer, this layer computes the 'sigmoid' activation function from the output from the first Dense layer. The output from the 'sigmoid' activation function is an integer between [0, 1]. This is the final result, therefore if the result is closer to 1 then the predicted label will be true, if the result is closer to 0 then the predicted label will be false.

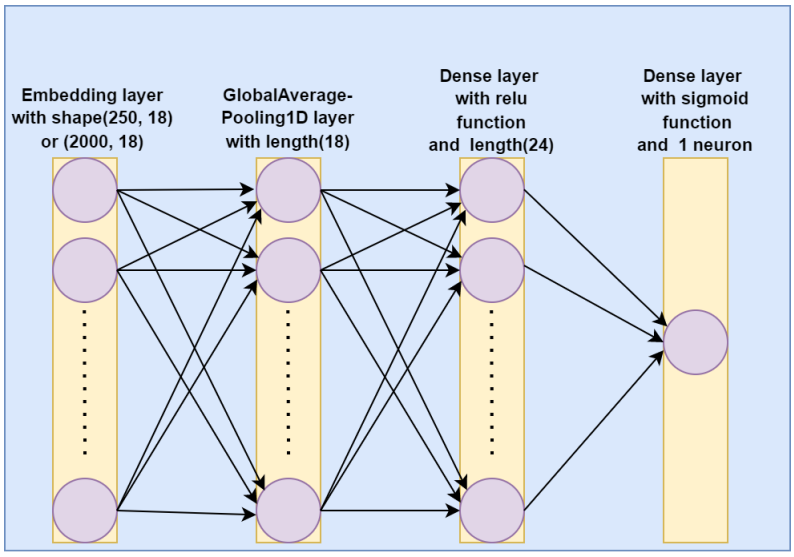

Figure 3.4: The structure of the Keras model used in this thesis.

3.3.2 k-Nearest Neighbors Model

As mentioned in Section 2.5, the performance of the KNN algorithm depends on the appropriate selection of k, making the selection of the value of k in this algorithm of essential importance. There are several methods to select

the appropriate value for k. One of these methods is the greedy search, which was utilized to search for the best k value within the range [1, 50]. A limit of 50 was set in the search for the best value of k because it was noticed through experience that the performance of the algorithm does not improve with values greater than such. With this model, the BERT encoder 2.3.2 was used due to the model's performance faring better with this encoder than with the Tensorflow tokenizer.

Chapter 4

Experiments and Evaluation

The following chapter will explore the experiments performed and which datasets were used for them. The results of the different models will also be compared, including the results with and without preprocessing the data, the goal of which is to understand if preprocessed data improves the accuracy and performance of the models used. The results of the LIME interpreter and the word cloud will also be shown, as they were used with 2 models out of 4. Finally, an ensemble model will be created consisting of three models which are Keras model, SVM model, and Naive-Bayes model. This model classifies the tweet or article as real news only if at least two out of the three models mentioned above classify the tweet or article as real news, otherwise it is classified as false news.

4.1 Experiments

Four datasets related to news about Covid-19 were used in these experiments, some which contain tweets and others which contain articles. Articles and tweets were also combined together to form the datasets introduced in Chapter 3.1. Table 4.1 displays the datasets used in this thesis. It was found that some of the tweets and articles were repeated in more than one dataset, consequently, duplicates were deleted, therefore the number of tweets and articles in the datasets created is not equal to the sum of the tweets and articles in the datasets.

dataset name	number of instances	type	source
COVID-19 Fake News Dataset	3002	article	Mendeley[20]
Covid-19 News Dataset	16990	tweets	zenodo[21]
COVID-19 Fake News Dataset	8560	tweets	github[22]
COVID Fake News Dataset	10202	tweets	zenodo[23]

Table 4.1: The source of the datasets that were used in this thesis

The following table displays the datasets created using the existing datasets in the previous Table 4.1, where 't' symbolizes tweets and 'a & t' symbolizes articles and tweets:

dataset type	preprocessed	number of instances
t	yes	19850
t	no	19852
a & t	yes	21940
a & t	no	21944

Table 4.2: the created dataset for the experiments

After the datasets were completed, a word cloud was utilized to identify which words were more frequently used in fake news and real news. Figure 4.1 displays the word clouds for real news and fake news.

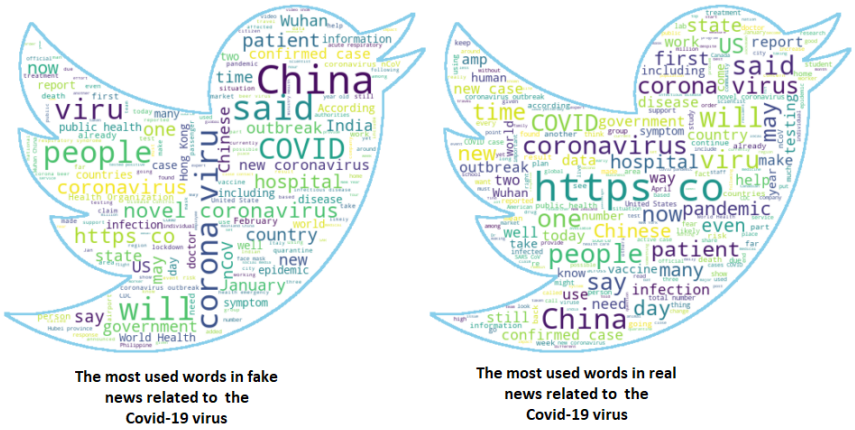

Figure 4.1: The dataset used in this figure is a & t without preprocessing.

there are Obvious common words between real and fake news such as "COVID" and "People", this words will not have much influence on the model during the classification process.

As many experiments were conducted which was a time intensive process, more than one computer were utilized to enable conducting the experiments in parallel. The computers used for the experiments are the following:

number	CPU	GPU	RAM
1	Intel(R) Core i5-10400F	RTX 3060 Ti 8GB	16GB
2	Intel Core i7-7700HQ	GTX 1050 Ti 4GB	16 GB

Table 4.3: the used computers for the experiments

The experiments progress

In this thesis, four different experiments were conducted using four different models (keras sequential model, KNN, Naive-Bayes, SVM). Each experiment was divided into four sub-experiments, with each using a different dataset from the datasets in Table 4.2. A confusion matrix and classifier evaluation metrics (accuracy, precision, recall, f1-score) were used in each sub-experiment to measure the performance of each model. Upon completion of each experiment, the average of the classifier evaluation metrics for the four sub-experiments were calculated as shown in the following table:

sub-experiments	used dataset	preprocessed	accuracy
1	t	yes	x
2	t	no	y
3	a & t	yes	z
4	a & t	no	u
average			$\frac{x+y+z+u}{4}$

Table 4.4: Calculate the average of the classifier evaluation metrics

Finally, the averages calculated in each experiment were compared as shown in Table 4.5. Accordingly, the performance of the different models using several datasets was compared.

experiments	used model	accuracy average
1	keras sequential	x
2	KNN	y
3	Naive-Bayes	z
4	SVM	u

Table 4.5: Calculate the average of the classifier evaluation metrics

Experiments were performed with different running times. For example, each sub-experiment of the Keras model lasted 30 minutes (including word tokenization). Thus, this experiment took 15 * 4 = 60 minutes, which is equivalent to one hour. The following table displays the time taken for each experiment:

experiments	used model	running time
1	keras sequential	60 minutes
2	KNN	90 minutes
3	Naive-Bayes	48 seconds
4	SVM	9.5 hours

Table 4.6: running time for each experiment

4.1.1 Compare Between Results From Different Models

First, the components and results of each experiment will be presented separately. After, the results of the experiments will be compared with each

other.

The first experiment:

In the first experiment, the keras sequential model and Tensorflow tokenizer were used as explained in Sections 3.3.1 and 2.3.1. The Tensorflow tokenizer corresponds to the keras embedding layer and activation functions 'relu' 2.4 more than the BERT encoder 2.3.2 because the BERT encoder contains both positive and negative values, which affects the performance of the embedding. In addition, the function 'relu' replaces the negative values generated by the BERT encoder with zeros. The following table displays the results of the first experiment:

sub-exp.	dataset	preprocessed	accuracy	precision	recall	f1-score
1	a & t	yes	0.86	0.85	0.88	0.86
2	a & t	no	0.88	0.90	0.85	0.87
3	t	yes	0.91	0.91	0.91	0.91
4	t	no	0.93	0.93	0.93	0.93
average			**0.89**	**0.90**	**0.89**	**0.89**

Table 4.7: Calculate the average of the Keras classifier evaluation metrics

The following is another representation of the results, namely the confusion matrix:

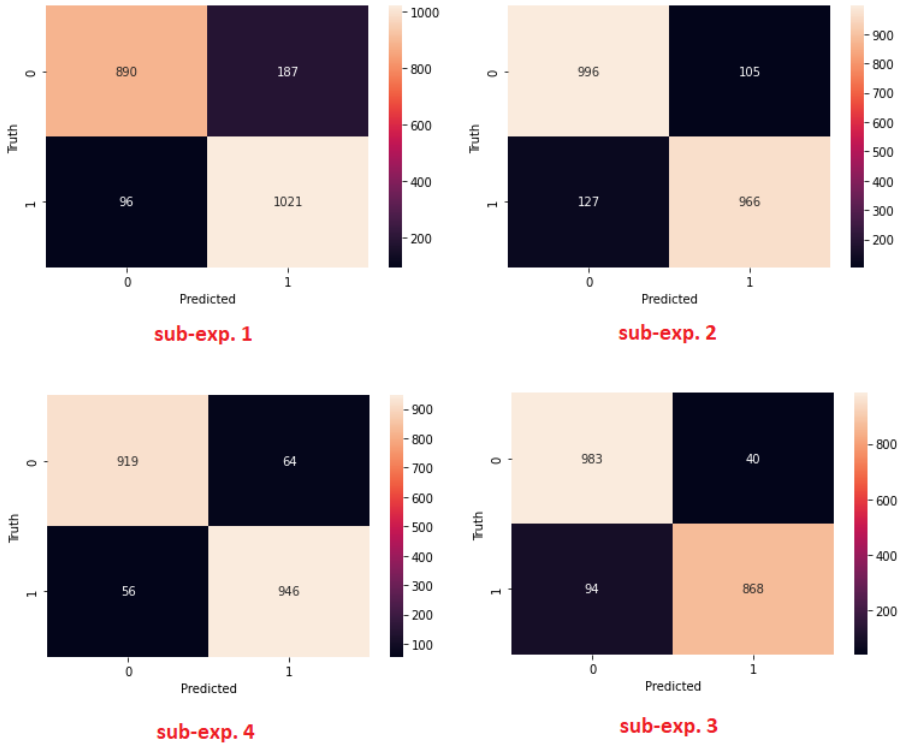

Figure 4.2: Confusion matrix for Keras model.

Drawing on the results presented in Table 4.7, the accuracy of the model in each sub-experiment was always higher than 86% and the accuracy varies according to the dataset used for training and testing. In the first and second sub-experiments, the accuracy of the model was relatively less compared to the third and fourth sub-experiments, the reason for this is that the datasets used in the first two sub-experiments contain long and short texts (articles and tweets) as shown in Figure 3.2. Therefore, it was necessary to enlarge the input_length to 2,000 characters to fit the length of the long texts. Since the length of the texts in the dataset must be standardized, the tokenizer added multiple zeros to the short texts (tweets) to make their length equal to the input_length (2,000), thus leading to a greater similarity between the short texts and a decrease in the accuracy of the model. From Table 4.7, it can be concluded that the accuracy of the model in each sub-experiment becomes less when using the preprocessed dataset. The reason for this is that, through data preprocessing, the Mention (@) and Hashtag (#) are omitted. However, it is important to note that the hashtag often carries an indication as to whether the news is true or false and, by deleting it, the percentage of similarity between tweets increases, leading to a decrease in the accuracy of the model. In this experiment, each sub-experiment consisted of fifteen epochs. The following chart displays the accuracy and val_accuracy evolved in every epoch:

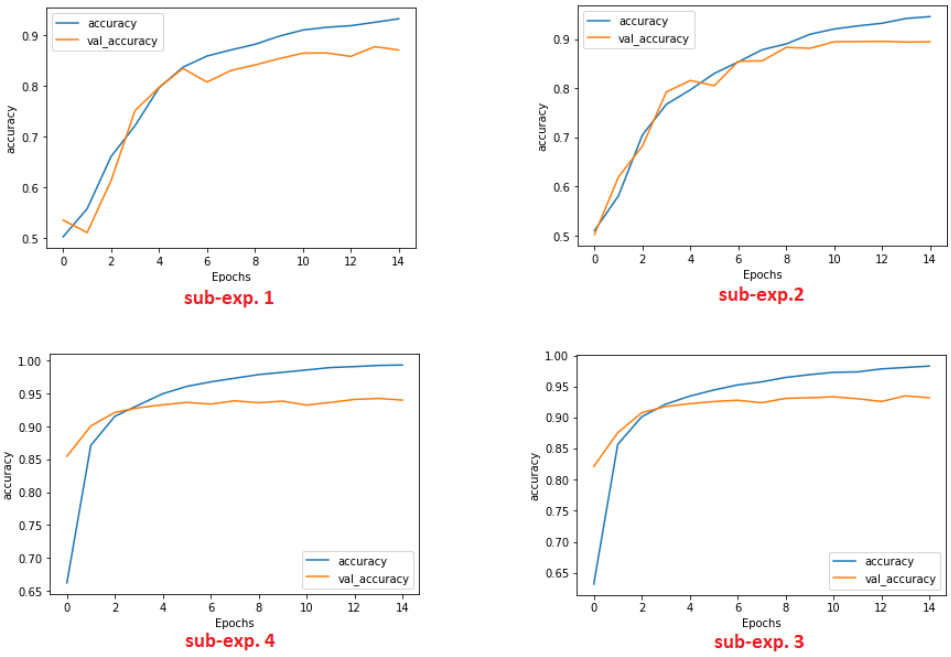

Figure 4.3: Comparing the accuracy in each epoch.

The second experiment:

In the second experiment, the KNN model explained in Section 3.3.2 was utilized. As mentioned in Section 3.2, the KNN model provides poor accuracy when using the Tensorflow tokenizer. Therefore, as a first step in the experiment, the BERT encoder 2.3.2 was used to encode the texts (Covid-19 news and tweets) that exist in the dataset. Next, a search for the best k value was conducted. In this experiment, cross-validation with ten splits was used. Thus, the best value of k was searched for in each sub-experiment ten times (for each split in cross-validation). The following figure displays a number of the k values in the sub-experiments:

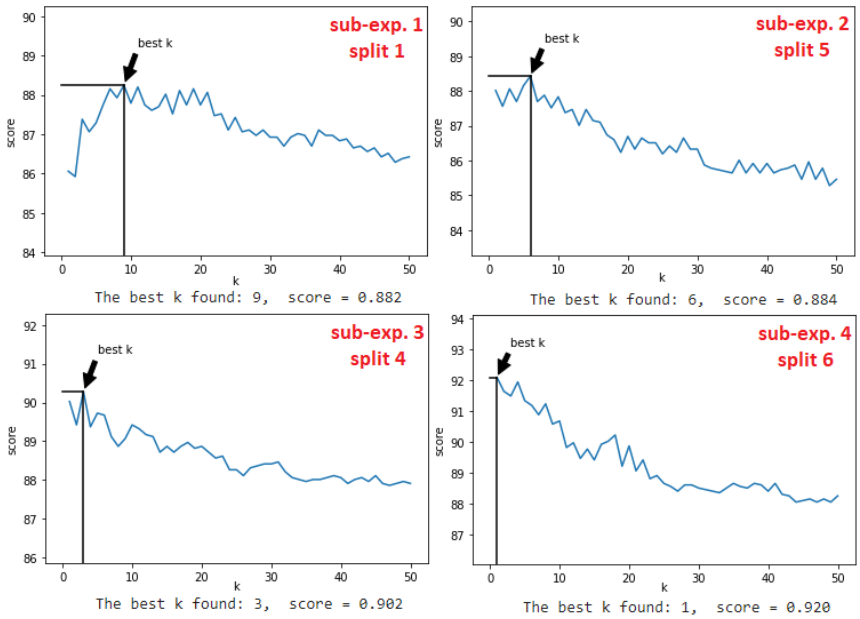

Figure 4.4: The best k between 1 and 50.

The following table displays the results of the second experiment:

sub-exp.	dataset	preprocessed	accuracy	precision	recall	f1-score
1	a & t	yes	0.88	0.89	0.86	0.87
2	a & t	no	0.88	0.89	0.87	0.88
3	t	yes	0.90	0.91	0.89	0.90
4	t	no	0.91	0.92	0.91	0.91
average			**0.89**	**0.90**	**0.88**	**0.89**

Table 4.8: Calculate the average of the KNN classifier evaluation metrics

Table 4.8 exhibits that the results of the KNN model in this experiment do not differ greatly from the results of the first experiment (Keras model). Despite the use of the BERT encoder, the results of this experiment can be interpreted due to the same reasons explained in the previous experiment. As the discrepancy between the long and short texts led to a similarity between

the texts that are supposed differ from one another, the data preprocessing had a negative effect for the same reasons as the first experiment.

The following figure displays another representation for the results, namely the confusion matrix:

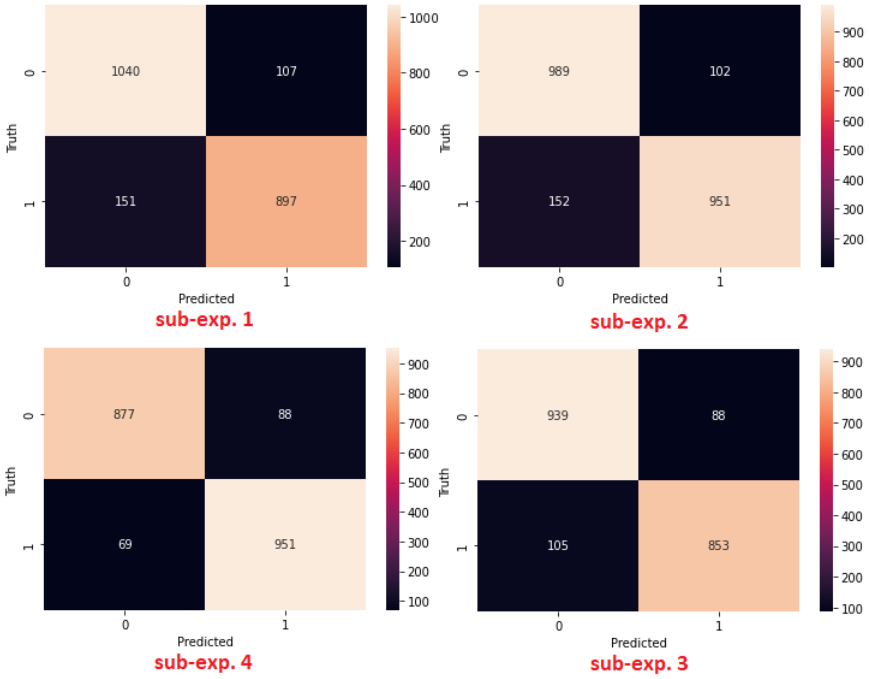

Figure 4.5: Confusion matrix for KNN model.

The third experiment:
The third experiment utilized the Naive-Bayes model. This experiment is distinguished from the others in terms of speed as the model in each sub-experiment took only twelve seconds to train and test. The reason for this short length of time is that the Naive-Bayes model relies only on calculating the probabilities of each class and does not require iteration, epoch or complex mathematical operations based on matrices which require longer processing times. The following table displays the results of the third experiment:

sub-exp.	dataset	preprocessed	accuracy	precision	recall	f1-score
1	a & t	yes	0.88	0.89	0.86	0.88
2	a & t	no	0.88	0.90	0.86	0.88
3	t	yes	0.91	0.90	0.92	0.91
4	t	no	0.92	0.91	0.93	0.92
average			**0.90**	**0.90**	**0.89**	**0.90**

Table 4.9: Calculate the average of the Naive-Bayes classifier evaluation metrics.

The following figure displays another representation for the results, namely the confusion matrix:

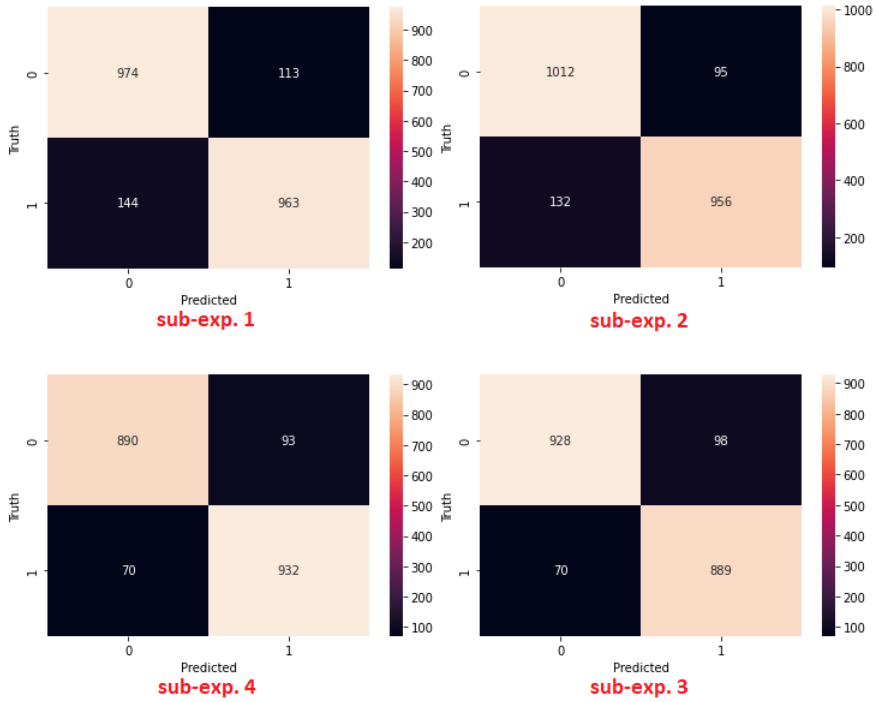

Figure 4.6: Confusion matrix for Naive-Bayes model.

Drawing on Table 4.9, it is evident that the results of the third experiment did not differ from the first and second experiment. This result was expected due to the dataset used being the same. Therefore, it can be concluded that the Naive-Bayes model performed better with the homogeneous dataset (comprised only of tweets).

The Fourth experiment:

In the fourth experiment, the support vector machine (SVM) model was used. The training process for this model is relatively slow compared to other models however the model is distinguished by its high levels of accuracy and performance. The following table displays the results of the fourth experiment:

sub-exp.	dataset	preprocessed	accuracy	precision	recall	f1-score
1	a & t	yes	0.90	0.91	0.89	0.90
2	a & t	no	0.91	0.91	0.90	0.91
3	t	yes	0.93	0.93	0.92	0.93
4	t	no	0.93	0.93	0.94	0.93
average			**0.92**	**0.92**	**0.91**	**0.92**

Table 4.10: Calculate the average of the SVM classifier evaluation metrics.

The following figure displays another representation for the results, namely the confusion matrix:

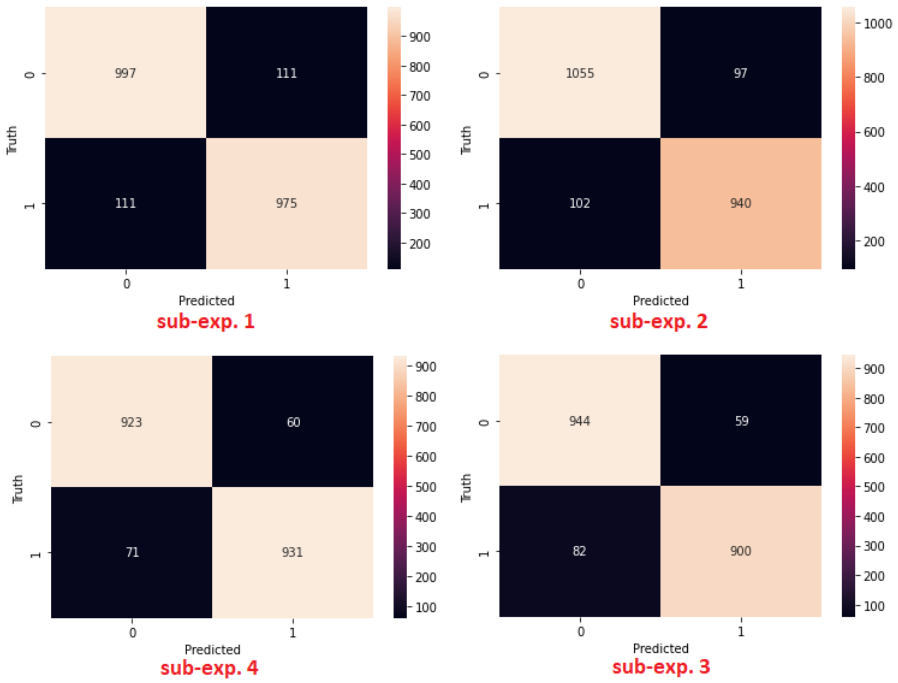

Figure 4.7: Confusion matrix for SVM model.

Drawing on Table 4.10, it is evident that no significant difference exists between the results of the sub-experiments as in the previous experiments. The model performed greater than or equal to 90% in all sub-experiments.

Comparison of the results of the four experiments:
As next step, the results of the four experiments will be compared according to the accuracy of the model in each experiment. The following table displays which model and tokenizer) were used in each experiment along with the experiments' results:

experiment	used model	tokenizer/encoder	running time	accuracy
1	Keras (sequential)	Tensorflow tokenizer	60 minutes	0.89
2	KNN	BERT encoder	90 minutes	0.89
3	Naive-Bayes	-	48 seconds	0.90
4	SVM	-	9.5 hours	0.92
average				**0.90**

Table 4.11: SVM has the highest accuracy among the four models.

Drawing on the previous table, we note that there is no significant difference between the accuracy of the performance of the four models as the average accuracy was 90%. The best model in terms of running time

was Naive-Bayes whilst the best model in terms of performance accuracy was SVM which recorded an accuracy of 92%.

4.1.2 LIME Explainer and Word Cloud

Upon completing the four experiments, a LIME interpreter and word cloud were utilized to explain the results of the SVM and Naive-Bayes models. The following example shows how the SVM and Naive-Bayes models dealt with the tweet "Google Hyderabad begins screening employees," and how each word affected the model while making the classification decision.

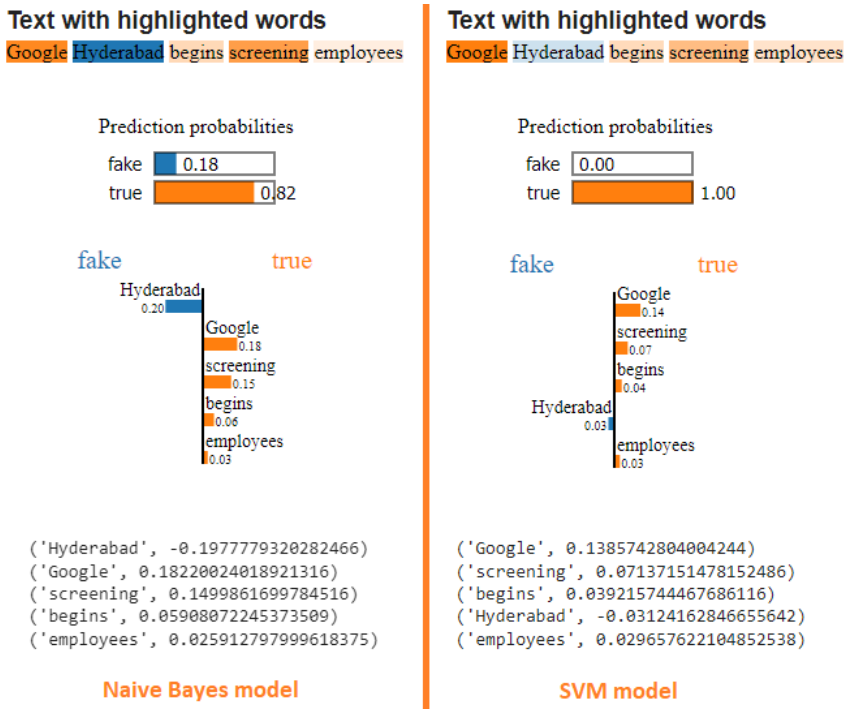

Figure 4.8: An example to explain the performance of the SVM and Naive-Bayes models in the fourth sub-experiment for each model.

As shown in the previous example, each word has an effect on the model while making the classification decision. This effect varies from one word to another and from one model to another. For example, the word "Google" has a different effect than the word "Hyderabad".

Following the creation of the explainer LIME, the weight of each word in each instance was calculated. The four words that had the most effect on the model from each instance were selected. Subsequently, two different word clouds were created from the selections. The first word cloud contains the most affected words in the model for classifying the instance in the class "true". The second word cloud contains the most affected words in the

model for classifying the instance in the class "false". Figure 4.9 shows the two different word clouds of the SVM and Naive-Bayes models:

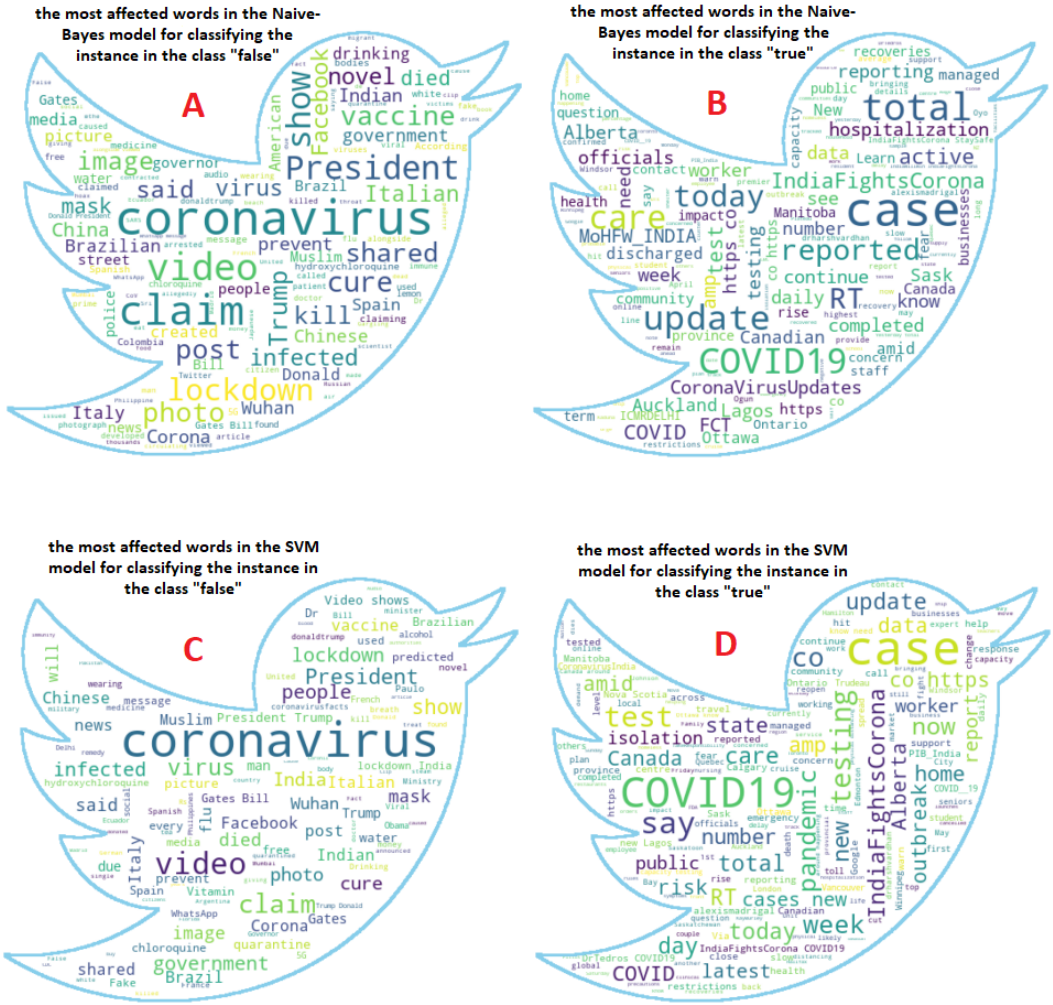

Figure 4.9: The dataset used in this figure is a & t without preprocessing.

It can be seen that there is a significant similarity between word clouds (B and D) that contain the words used to predict real news in the SVM and Naive-Bayes models. It can also be seen that the word clouds (A and C) that contain the words used to predict fake news are very similar in the SVM and Naive-Bayes models. It can also be seen the great contrast between the word clouds (A and B).

4.1.3 Create the Ensemble Model and Compare its Results With the Results of Previous Experiments

As a final step in this thesis, an ensemble mode was created from three models namely Keras model, SVM model, and Naive-Bayes model. This model was trained and tested only on the dataset consisting of tweets without preprocessing (the dataset used in the fourth sub-experiment for the four models), and accordingly the results of the fourth sub-experiment for each of the four models compared with the results of the ensemble mode. The following table displays the result of this comparison:

used model	used dataset	accuracy
Keras (sequential)	Tweets without preprocessing	0.93
KNN	Tweets without preprocessing	0.91
Naive-Bayes	Tweets without preprocessing	0.92
SVM	Tweets without preprocessing	0.93
ensemble mode	Tweets without preprocessing	0.94

Table 4.12: ensemble model has the highest accuracy.

From the table 4.12 it can be seen that the performance of the ensemble mode was the best among all the other models used in this thesis, the reason for this is that it relies on three models to make its decision, which reduces the error rate, but despite this good performance, the difference is not large between it and the other models.

4.2 Results Summary

From the results of the experiments conducted in this thesis, it can be concluded that:

- Machine learning models that are capable of understanding and analyzing texts can be created with the aim of solving problems such as classification with accuracy sometimes exceeding 90%(+- 3%). In this experiment there are no significant differences in the performance of the models used. However, there are significant differences between the models in terms of the time needed to create, train and test the model.
- Data preprocessing does not necessarily improve the performance of the machine learning model. In some cases, it can have an adverse effect on the performance of the model, and this is what happened in the four experiments conducted in this thesis.
- The performance of the model, its decisions, and the effect of each feature on the model's performance can be explained through several

tools. In this thesis, two tools are explained, namely LIME and word cloud.

- In the context of these experiments, it was found that the performance of the Ensemble model in classifying texts is better than the performance of each model separately, as shown in the section 4.1.3.

Chapter 5

Summary and Outlook

The purpose of this chapter is to give a final overview of the content and the most important results of the work.

5.1 Summary

In this thesis, emphasis is placed on natural language processing methods in order to classify texts by machine learning models into categories. The problem was that there is a lot of false news about the Covid-19 virus, which in turn leads to the spread of rumors and hinders the world from facing this pandemic. So the solution was to provide a tool capable of classifying this news between fake and real using machine learning techniques. Several models were created and several experiments were carried out using a different datasets, then the results of the experiments were analyzed and the performance of the models was measured and compared among them. In addition, an attempt was made to interpret the performance of the models using several tools (LIME, word cloud). After completing the experiments, the conclusion was reached that the machine learning models are capable of solving text classification problems with high accuracy.

5.2 Outlook

The results of this thesis leave open the question whether the performance of the models used in this thesis will differ positively or negatively, or whether the performance will remain similar to the same case in this thesis.

Based on the results presented in this thesis, it can be asked whether it is possible to improve the performance of the models used in this thesis in some way, or create different models (such as LSTM, Decision Tree, CNN, etc.) and how its performance in terms of accuracy will be.

Bibliography

[1] What is Machine Learning?
https://www.ibm.com/cloud/learn/machine-learning.

[2] What is Natural Language Processing?
https://www.ibm.com/cloud/learn/natural-language-processing.

[3] What is the difference between structured and unstructured data?
https://blog.accern.com/post/the-difference-between-structured-and-unstructured-data.

[4] Sample Data: Movie Review Sentence Polarity: Wolfram Data Repository
https://datarepository.wolframcloud.com/resources/Sample-Data-Movie-Review-Sentence-Polarity.

[5] Tf.keras.preprocessing.text.tokenizer :   Tensorflow core v2.8.0
https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer.

[6] tf.keras.preprocessing.sequence.pad_sequences : TensorFlow Core v2.8.0
https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences.

[7] Pretrained models
https://www.sbert.net/docs/pretrained_models.html#sentence-embedding-models.

[8] sentence-transformers/all-MiniLM-L6-v2 · Hugging Face
https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

[9] SentenceTransformer
https://www.sbert.net/docs/package_reference/SentenceTransformer.html#sentence_transformers.SentenceTransformer.encode.

[10] Mit BERT automatisiert Fragen beantworten (Teil 1)
https://dida.do/de/blog/mit-bert-automatisiert-fragen-beantworten-teil-1.

[11] Keras documentation: The Sequential model
https://keras.io/guides/sequential_model/.

[12] Keras documentation: Keras layers API
https://keras.io/api/layers/.

[13] Keras documentation: Dense layer
https://keras.io/api/layers/core_layers/dense/.

[14] Keras documentation: Layer activation functions
https://keras.io/api/layers/activations/#relu-function.

[15] Keras documentation: Embedding layer
https://keras.io/api/layers/core_layers/embedding/.

[16] Keras documentation: GlobalAveragePooling1D layer
https://keras.io/api/layers/pooling_layers/global_average_pooling1d/.

[17] Keras documentation: Keras FAQ
https://keras.io/getting_started/faq/#:~:text=Epoch%3A%20an%20arbitrary%20cutoff%2C%20generally,the%20end%20of%20every%20epoch.

[18] Explanatory Model Analysis
https://ema.drwhy.ai/LIME.html.

[19] Word Cloud
https://www.petitessen.net/wordcloud/.

[20] COVID-19 Fake News Dataset
https://data.mendeley.com/datasets/zwfdmp5syg/1.

[21] Covid-19 News Dataset Both Fake and Real
https://zenodo.org/record/4722484#.YgX6RN_MJD9.

[22] covid_fake_news/data at main · diptamath/covid_fake_news
https://github.com/diptamath/covid_fake_news/tree/main/data.

[23] COVID Fake News Dataset
https://zenodo.org/record/4282522#.YNJ1r-gzZPY.

[24] O. Biran and C. Cotton. Explanation and justification in machine learning: A survey. In *IJCAI-17 workshop on explainable AI (XAI)*, volume 8, pages 8–13, 2017.

[25] R. Blumberg and S. Atre. The problem with unstructured data. *Dm Review*, 13(42-49):62, 2003.

[26] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *International Conference on Database Theory*, pages 336–350. Springer, 1997.

[27] F. Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[28] E. Costa, A. Lorena, A. Carvalho, and A. Freitas. A review of performance evaluation measures for hierarchical classifiers. In *Evaluation methods for machine learning II: Papers from the AAAI-2007 workshop*, pages 1–6, 2007.

[29] W. Daelemans and V. Hoste. Evaluation of machine learning methods for natural language processing tasks. In *3rd International conference on Language Resources and Evaluation (LREC 2002)*. European Language Resources Association (ELRA), 2002.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[31] J. Dieber and S. Kirrane. Why model why? assessing the strengths and limitations of lime. *arXiv preprint arXiv:2012.00093*, 2020.

[32] I. Düntsch and G. Gediga. Confusion matrices and rough set data analysis. In *Journal of Physics: Conference Series*. IOP Publishing, 2019.

[33] M. Grandini, E. Bagli, and G. Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.

[34] A. Gulli and S. Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[35] F. Heimerl, S. Lohmann, S. Lange, and T. Ertl. Word cloud explorer: Text analytics based on word clouds. In *2014 47th Hawaii international conference on system sciences*, pages 1833–1842. IEEE, 2014.

[36] S. Kannan, V. Gurusamy, S. Vijayarani, J. Ilamathi, M. Nithya, S. Kannan, and V. Gurusamy. Preprocessing techniques for text mining. *International Journal of Computer Science & Communication Networks*, 5(1):7–16, 2014.

[37] L. Khreisat. Arabic text classification using n-gram frequency statistics a comparative study. *DMIN*, 2006:78–82, 2006.

[38] J. D. Novaković, A. Veljović, S. S. Ilić, Ž. Papić, and T. Milica. Evaluation of classification models in machine learning. *Theory and Applications of Mathematics & Computer Science*, 2017.

[39] H. Rajaguru and S. K. Prabhakar. *KNN classifier and K-means clustering for robust classification of epilepsy from EEG signals. A detailed analysis.* diplom. de, 2017.

[40] R. Rani and D. Lobiyal. Performance evaluation of text-mining models with hindi stopwords lists. *Journal of King Saud University-Computer and Information Sciences*, 2020.

[41] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, 11 2019.

[42] P. Tan, M. Steinbach, A. Karpatne, and V. Kumar. *Introduction to Data Mining.* What's New in Computer Science Series. Pearson, 2019.

[43] K. Vembandasamy, R. Sasipriya, and E. Deepa. Heart diseases detection using naive bayes algorithm. *International Journal of Innovative Science, Engineering & Technology*, 2(9):441–444, 2015.