2023

Cross-Lingual model for hate speech detection on Twitter: a case of Swahili and Swahili-English slang.

Kariuki, Andrew Oduor
School of Computing and Engineering Sciences
Strathmore University

Cross-Lingual Model for Hate Speech Detection on Twitter: A Case of Swahili and Swahili-English Slang

By

Andrew Oduor Kariuki

145244

Master of Science in Information Technology

2023

Cross-Lingual Model for Hate Speech Detection on Twitter: A Case of Swahili and Swahili-English Slang

By

Andrew Oduor Kariuki

145244

Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Science in Information Technology at Strathmore University

School of Computing & Engineering Sciences

Strathmore University

Nairobi, Kenya

June 2023

Declaration and Approval

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Student Name: **Andrew Oduor Kariuki**

Student Number: **145244**

Signature: Date: 25th May 2023

Approval

The thesis of Kariuki Andrew Oduor was reviewed and approved for examination by the following:

Dr Vincent Omwenga

School of Computing & Engineering Sciences,

Strathmore University.

Dr Julius Butime,

Dean, School of Computing & Engineering Sciences,

Strathmore University.

Dr Bernard Shibwabo,

Director of Graduate Studies,

Strathmore University.

Abstract

The prevalence and entrenchment of online hate, hate crimes and hate speech in contemporary society concern organisations and governments. Detecting online hate, especially on social media, has proven daunting as offensive languages have multifaceted behaviours, and most training data are topic specific. On top of that, available solutions and research are geared towards the English language; thus, detecting online hate in lower-level languages like Swahili and Indigenous African Languages is much more difficult. This has worsened because social platforms such as Twitter, Facebook, Instagram, Rumble and YouTube enable consumers to converse and participate in their native dialects. This research proposed using cross-lingual transfer learning for hate detection to overcome these challenges. A Cross-Lingual model built on a BERT pre-trained model was developed as part of the research's experimental methodology, and its performance was compared to those of more established text classifiers like SVM, NB, and LR. Through the Twitter API, more than 300K tweets with a Kenyan focus were collected. These tweets focused on Kenya's most divisive moments in history, namely the 2013, 2017, and 2022 general elections. A set of predetermined criteria, including user location, tweet location, hashtags, pro-hate accounts, hate patterns, and racial epithets, were used to collect the data. For usage in the model development, training and validation, a random sample of over 20K tweets was annotated as hate or non-hate. The developed Cross-Lingual model achieved a ROC curve area under the curve of 0.77 and an accuracy of 77 per cent. The following are the contributions made by this study. Primarily, the research established an empirical framework and methodology for utilising transfer learning to identify the offensive language in low-resource languages. Additionally, this strategy was crucial in creating a text classification framework that could be broadly applied to different types of abusive language on online platforms. The model's results may thus be used to inform data-driven legislation regarding the detection of online hate as well as evidence-based decisions by pertinent intelligence agencies.

Keywords: *deep learning, free speech, freedom of speech, hate detection, hate speech, machine learning, natural language processing, social media, Twitter.*

Acknowledgements

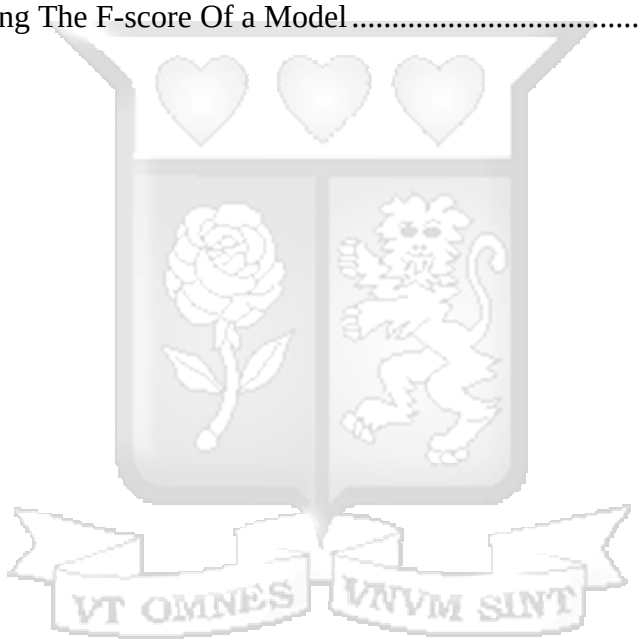
I thank the Mighty Creator for His mercies, guidance, understanding, and insight during my education. I also demonstrate my appreciation for my study and supervising teachers, Professor Ismail Ateya and Dr Vincent Omwenga, for their tireless efforts, teachings, encouragement, and assistance in ensuring this study's success and the unit's success. My classmates deserve special acknowledgement for their contributions to the research. Their energy, vision, sincerity, and motivation have all left an indelible mark on me. Thank you so much for your prayers, support, and encouragement. Finally, thank everyone who assisted me in completing this research project, whether actively or passively.

Table of Contents

List of Figures

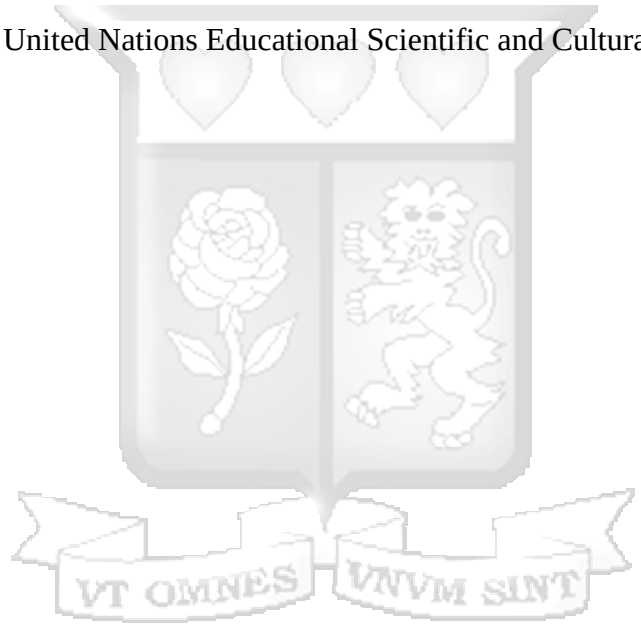
List of Tables

List of Equations



List of Abbreviations

AI	-	Artificial Intelligence
BERT	-	Bidirectional Encoder Representations from Transformers
CNN	-	Convolutional Neural Networks
ML	-	Machine Learning
NGOs	-	Non-Governmental Organizations
SVM	-	Support Vector Machine
UN	-	United Nations
UNESCO	-	United Nations Educational Scientific and Cultural Organization



Definition of Terms

Term	Definition	Source
Coded language	The use of words to include layers of meaning. The use of coded language allows the speaker to relate meaning without being specific.	Henry & Tator, 2000
Deep Learning	A branch of artificial intelligence and machine learning that mimics how humans acquire knowledge	Brush & Burns, 2021
Freedom of speech	Also referred to as free speech, is the ability of individuals to express themselves without interference or regulation	UN, 2019
Hate Speech	Any form of communication in speech, writing or behaviour that attacks or uses discriminatory language concerning a person or group of people	UN, 2019
Natural Language Processing	The capacity of computer software to comprehend natural language, whether it be spoken or written	Burns & Lutkevich, 2021
Stereotyping	A deeply ingrained generalized idea held by a people on the typical behaviours, qualities, attitudes, skills, and flaws of others, such as members of various ethnic communities	NCIC, 2013
Twitter	A social networking and microblogging platform where users can send “tweets”, respond to “replies”, and resend “retweet” brief posts known as "tweets," which are typically 280 characters long	Twitter, 2005

Chapter One: Introduction

1.1 Background of Study

UNESCO (2015) describes hate speech as the use of obscene, derogatory, or threatening language or behaviour, as well as the public exhibition of any written or printed material intended to create hate towards a certain group or individual because of their ethnicity, or skin tone. According to Waldron (2012), hate speech undermines the rightful expectation that none of us will face prejudice or violence just because we belong to a particular race, religion, or another vulnerable group. Several events have shown the dire implications of hate speech on the social order, one in Kenya and another in the USA.

2007 brought forth the darkest time in Kenya's history, an election marred by hostility, hatred, and persecution. Kenya experienced post-election violence where Over 600,000 people were evicted from their houses, and over 1,100 individuals lost their lives (Roberts, 2009). Kenya's 2007 elections brought hostile disparities in which neighbours turned against neighbours and friends against friends. A time when politicians divide people based on their ethnicity and foster tribalism in the process (Roberts, 2009). Due to the violence in 2007–2008, it was realised how detrimental hate speech was to the country, and as a result, steps were taken to combat it.

Recently, the world was surprised to watch the sworn custodian of democracy and protector of human rights, the USA, experience the effects of hateful speech. On January 6th, 2021, the U.S. saw a mob of their former president Donald Trump's supporters storm and violently attack the Capitol Building in Washington, D.C. (Sinnar, 2020). This resulted from radical hate speech and propaganda fanned by Donald Trump regarding the outcome of the 2020 general election. The attack caused approximately $1.5 million worth of damage to the U.S. Capitol building and the assault of 140 police officers (Sinnar, 2020). The impacts of hate speech were evident to everyone, especially when it came from authorities or people in positions of power.

In the wake of such events, democracies struggled to implement measures for tackling hate speech. European countries and the USA have imposed compulsory laws that force social media platforms to block traces of hate on their platforms. Kenya formed the f National Cohesion and Integration Commission (NCIC) to achieve its quest for peace and cohesion (Omondi, 2013). However, all these are still far-fetched dreams; due to the widespread adoption of social media and the volume of published posts, it is becoming more and more

difficult for humans to moderate hate speech thoroughly and promptly. These factors have led to a rise in curiosity about automatic hate speech detection. To combat online hate, researchers and developers have begun leveraging the power of machine learning, especially natural language processing. These new techniques use traditional classifiers, deep learning, and transfer learning to detect patterns within datasets(Kumar, 2022).

These classifiers are trained by extracting features from text into meaningful vector representations using analytical methods such as bag-of-words, TF-IDF, and word embedding (Kumar, 2022). Some classifiers use more advanced techniques such as word2vec, Glove, FastText, and transformer-based analysis to obtain more graphic representations. To analyse instances of objectionable language, a cleaned data set is passed via machine learning methods such as Support Vector Machines, Convolutional Neural Networks, Multi-Layer Perception, Extreme Gradient Boosting, Long Short-Term Memory networks, and Naive Bayes (Kumar, 2022).

With all these advancements, hate speech classification is still challenging (de Gibert et al., 2018). Data scarcity is one factor; most public hate speech datasets are available only in English. Building models for languages with fewer resources is significantly more complex (Vidgen & Derczynski, 2020). The necessity to identify hateful content adds to the task's difficulties. Although many people generally know what hateful language is, this concept only sometimes transfers into a concise list of criteria for building hate detection systems.

1.2 Problem Statement

Social Media platforms have enabled users to connect in their native languages (Pamungkas & Patti, 2019). The integration of indigenous languages such as Swahili into online content gives offensive content a multifaceted behaviour (Pamungkas & Patti, 2019). Such a phenomenon has made detecting online hate a domain-specific task as existing models are curated towards detecting hate in high-resource languages like English (Pamungkas & Patti, 2019). Making nations like Kenya dependent on manual techniques of online hate detection or leveraging social media companies to do their due diligence (Mugambi, 2017).

Such attempts always prove futile due to the sheer volume of online content and the reluctance of these companies to do their part (Sharpe, 2022). These challenges prove the need for a cross-lingual classifier that employs inter-task and inter-language transfer learning (Ranasinghe & Zampieri, 2020). In inter-language transfer learning, the classifier is trained using English

OLID, and then its weights and softmax layer are saved and used to train a new language (Zampieri et al., 2019). On the other hand, inter-task transfer learning, only the weights are saved and used to initialise the weights of a new language (Kumar et al., 2020).

1.3 Research Objectives

1.3.1 General Objective

This study aimed to develop a cross-lingual model for Twitter hate speech detection through transferred learning.

1.3.2 Specific Objectives

i. To identify the existing approaches used in detecting hate speech on social media.
ii. To review the shortcomings of the existing automated hate speech detection approaches.
iii. To examine how cross-lingual content filtering and transfer learning can be used to increase the efficacy of detecting hateful content.
iv. To develop a model that employs the concept of cross-lingual and cross-domain fine-tuning to detect hate speech.
v. To validate the model using the various evaluation and classification matrices.

1.4 Research Questions

i. What approaches are used to detect hate speech on social media?
ii. What are the shortcomings of the existing approaches to automated hate speech detection?
iii. How can cross-lingual content filtering and transfer learning be used to increase the efficacy of detecting hateful content?
iv. How can we develop a cross-lingual model that should consider low-resource languages through inter-language transferred learning?
v. How should the proposed model be validated for precision and accuracy?

1.5 Justification

Africa has suffered from ethnic conflict for an extended period. From the conflict involving the southern Igbo and the northern Hausa in the Biafra War to the ethnic cleansing in Darfur and Rwanda and the never-ending cycle of violence following elections in Kenya, Africa has

experienced ethnic turmoil (Varshney, 2009). All these conflicts have been fuelled using offensive language and hate funning.

This research is essential for developing nations, especially African nations like Kenya, Uganda, and Tanzania, that constantly experience the impact of hate speech. These nations have continuously experienced widespread hate, hate crimes and violence in their multicultural societies. These nations need to be addressed regarding research on and implementation of techniques to detect online hate, as most communicate in low-resource languages. The outcomes of this study provided a machine learning model that can perform cross-domain detection of offensive content online and foster cohesion and peace in the developing world. Through this process, these nations can focus most of their resources on developing their economies rather than on bringing people together.

1.6 Scope and Limitations

This study focused only on the Swahili language and the Kenyan population. To increase the effectiveness and efficiency of detecting offensive content in African languages, it should specifically focus on how the existing models and datasets used in detecting harmful content in the English language can be alleviated into the Swahili language and Swahili-English slang. It should touch on overcoming the biggest obstacle in online hate speech detection, which has been the multifaceted nature of offensive language and its dependence on high-resource languages such as English.

Chapter Two: Literature Review

2.1 Introduction

Twitter users take advantage of its publicity and reach to disseminate hateful content and messages. Contents aimed at frightening, intimidating, or silencing other users of the platform, such content can be a motivational factor for others to commit acts of violence, otherwise referred to as hate crimes (Saleem, Dillon, Benesch & Ruth, 2017). For this reason, there is a need to identify, analyse and curb the spread of hateful messages on social media sites, specifically Twitter. But categorizing hate speech is the main challenge in detecting it. Legislatures, communities, and associations all have varying definitions of what constitutes a hateful comment, which is arbitrary (Sellars, 2016).

2.1.1 The Concept of Hate Speech.

Maintaining free expression is crucial for democratic societies that uphold civil liberties. When offensive speech puts people at risk or in danger, it is prohibited by human rights treaties. *Hate speech* is "any type of expression which spreads, provoke, promote or excuses racial hatred, xenophobia, anti-Semitism or other forms of hatred based on intolerance" (Committee of Ministers of the Council of Europe, 1997). Hate speech frequently targets the socially marginalised or "other." Such acts are shown by the "othering" of minority groups like women, the LGBTQI+ community, and racial, ethnic, religious, and cultural minorities.

Hate speech is defined as speech that "attacks or uses pejorative or discriminatory language concerning a person or a group based on whom they are, in other words, based on their religion, ethnicity, nationality, race, colour, descent, gender, or other identity factors," (U.N. Strategy and Plan of Action on Hate Speech, 2019). Disputed interpretations of offensive language by the annotators as a result led to low-quality data, which negatively impacted the effectiveness of their models. (Idris, Akinola, & Olalekan, 2020).

The Kenyan anti-hate speech bill suggested with the creation of the Kenya National Cohesion and Integration Commission (NCIC) Act, 2008, also refers to this problem. (N.C. for law, 2008). The Act defines hate speech as *"Words of incitement and hatred against individuals based on certain group characteristics they share. It includes speech that advocates or encourages violent acts against a specific group and creates a climate of hate or prejudice which may, in turn, foster the commission of hate crimes."* This Act incorrectly classifies remarks that are offensive as hate speech, punishes those found guilty of uttering offensive

utterances as hate speech perpetrators, and lumps cyberbullying and obscene language together under a single banner. (Idris, Akinola, & Olalekan, 2020).

Figure 2.1 Pyramid of Hate

2.1.2 Hate Speech on Twitter

Twitter is currently one of the most significant data sources for academics, new mongers, and governmental communications, making it the top platform among the several extant social networks (Twitter, 2020). Twitter is where news frequently breaks before it does on mainstream news outlets. With an average of five hundred million tweets generated daily, Twitter's usage has rapidly increased, especially during events. Figure 2.2 shows Elon Musk's tweet describing Twitter as the e de facto public town square.

Its defining features are Twitter's short message restriction (currently 280 characters) and unfiltered stream. In addition to its effective communication capabilities, it has made it easier

for malicious individuals to spread violent behaviour, including online harassment, cyberbullying, and hate speech. Between July and December of 2020, 1,126,990 distinct accounts were "actioned" by Twitter for violating its hateful conduct policy, a 77% increase from the previous six months. A tweet could be deleted, or an account could be banned, among other things. As a result, this suggests that hateful remarks on the site continue to be on the rise.

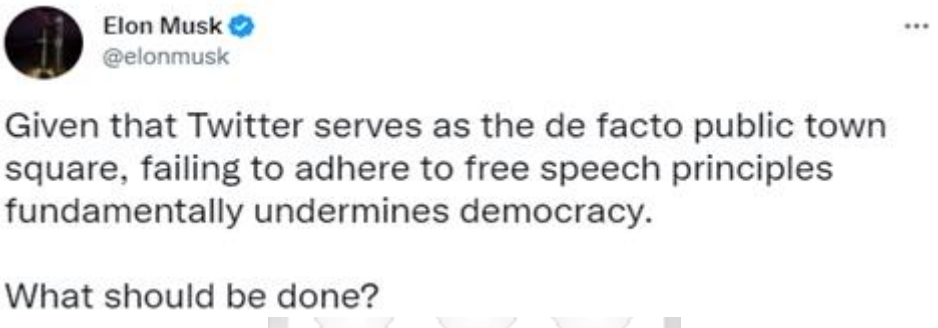


Figure 2.2 Elon Musk's Tweet Describing Twitter as The De Facto Town Hall (Twitter, 2022)

2.2 Hate Speech Detection Using Machine Learning

2.2.1 The Problem of Text Classification

Text classification is a supervised learning problem which categorises text/tokens into organised groups. Given a train set D with classified properties (d, c), wherein (d, c) ϵ X × C (Manning, 2008). For example:

$$(\mathrm{d}, \mathrm{c}) = \{\text{Nairobi defends its rights to it's Lamu strip, Kenya}\}$$

Equation 2.1 The Problem of Text Classification

In Equation 2.1, the one-sentence document Nairobi defends its rights to its Lamu strip and the class (or label) Kenya. In this sense, a machine learning algorithm utilizes a classifier method γ to assign features to classifications. As shown in Equation 2.2, the learning algorithm takes the training set D as input and returns the learned classification function γ, wherein X is the feature space; and a limited class size C = { c1,c2…,chi} (Manning, 2008).

$$\gamma: \mathrm{X} \rightarrow \mathrm{C}$$

Equation 2.2 Text Classification Function

2.2.2 Text Pre-processing

Most collected datasets are always unstructured; hence text pre-processing plays a crucial role in preparing data for classification. Pre-processing includes removing punctuation and special characters, lowercasing, eliminating stop words and repetitious words, lemmatization, tokenization, and stemming. (Manning, 2008). Stop words are frequently used in any language (Vijayarani et al., 2015). Stop words in English include "the," "is," and "and", for instance. These words always fall under articles, prepositions, and pronouns. Stop words are commonly used in NLP and text mining applications to filter unnecessary words so that the keywords can be the focus (Mugambi, 2017).

The variety of suffixes available for words in natural speech increases the factors that must be considered during assessment. Stripping a word of its various suffixes may be crucial to match words accurately, saving processing time and resources (Mugambi, 2017). Another important pre-processing function is stemming, where all variations of words are reduced to their roots. It is crucial to keep phrases with distinct connotations apart while stemming. The underlying premise is that words with a joint base can be converted to that base since they share the same meanings regardless of their morphology.

2.2.3 Feature Selection

The process of selecting a specific group of phrases from the training set only for utilizing those phrases as characteristics for the classification of texts is known as feature selection (Manning, 2008). The fundamental goals of feature selection are twofold. First, reducing the quantity of adequate vocabulary increases the efficiency of training and using a classifier. This is especially crucial for classifiers because, in contrast to N.B., they are more expensive to train (Manning, 2008). Second, by removing noisy characteristics, feature selection frequently improves classification accuracy. When introduced to document representation, a noise feature raises the classification error for brand-new data (Manning, 2008).

2.2.4 Document Representation

Document identification tries to express document data as a vector of a fixed size that can explain what's inside of the document to reduce complexity and make documents simpler to manipulate (Liu et al., 2020). BOW and TF-IDF are two popular techniques that can be applied, according to Mugambi (2017). According to Mugambi, the BOW technique sees documents as an assortment of phrases that include many but are not necessarily in order. The dictionary is made up of all the different words found in the corpus. Next, a vector of word frequencies is

used to represent each document. This model assumes that words are independent of one another, and that word order is irrelevant. Furthermore, this model does not provide term weighting in particular documents (Mugambi, 2017).

An algebraic paradigm called the vector space model treats items (like text) as vectors. This simplifies determining word similarity or the relevancy of a search query and a document (Salton, Wong, and Yang, 1975). Cosine similarity is frequently utilised to compare two vectors. The linear algebraic vector space model makes use of non-binary term weights. As a result, it is possible to calculate the continuous degree of similarity between two items, such as a query and documents, while still allowing for partial matching (Salton, Wong, and Yang, 1975).

2.2.5 Feature Weighting

An essential step in text classification is feature weighting, which determines the feature weight for each aspect of documents (Xue & Xu, 2010). The standard method of feature weighting ad Term Frequency/Inverse Document Frequency (TF-IDF) and sentiment weighting.

In TF-IDF, term frequency measures how frequently a term occurs within a document. The most straightforward calculation is to count the instances of each word as shown in Equation 2.3. However, there are ways to change that value according to the length of the document or the occurrence of the term that appears the most often (Mugambi, 2017). Raw word frequency has the drawback that relevance does not increase with usage. The term's log frequency weight is frequently applied as a result.

$$\mathrm{TF} = \frac{\text{Number of times a keyword is found in a document}}{\text{Number of words in a document}}$$

Equation 2.3 Term Frequency Formula

The inverse document frequency as shown in Equation 2.4 quantifies how frequently a phrase appears in a corpus of documents. It is calculated by dividing the total number of documents by the proportion of documents in the corpus that contain the phrase.

$$IDF = \log\left(\frac{Number\ of\ documents}{Number\ of\ documents\ containing\ keyword}\right)$$

Equation 2.4 Inverse Document Frequency Formula

The term frequency-the inverse document frequency (TF-IDF) can be obtained by multiplying these two metrics once they have been computed as shown in Equation 2.5. This value

demonstrates the significance of a term concerning a specific document within a corpus of texts.

$$TF - IDF = TF * IDF$$

$$TF - IDF = (\frac{Number\ of\ keyword\ in\ document}{Number\ of\ words\ in\ document}) * \log(\frac{Number\ of\ documents}{Number\ of\ documents\ with\ keyword})$$

Equation 2.5 TF-IDF Formula

In sentiment weighting, the weighting of terms is based on how positive or negative they are. This method makes use of SentiWordNet, which is a subset of the WordNet technique. SentiWordNet assigns a numerical number to each set of synonyms in WordNet to demonstrate the dataset's objectivity, positivity, and negativity (Mugambi, 2017).

2.2.6 Hateful Speech Identification Issues Considering Machine Learning

Machine learning models are being applied in detecting hate speech. These models run on algorithms that analyse text using training data, which are pre-defined datasets. Such algorithms can statistically analyse and evaluate the tones of writing and text collected online. As such, they can identify hate speech disguised as humour or sarcasm through methodologies like sentimental analysis. Making them the perfect tool for combating it, especially on social media (Abro et al., 2020). According to (Abro et al., 2020), the development of such tools faces challenges such as:

i. **Dataset Acquisition**: the first significant problem is the availability of datasets on hate speech in various parts of the world, particularly in Kenya. Since there is a crucial need to extend the hate speech deterrent effort to other non-Western regions of the world, a sizable dataset is needed to undertake internet analysis. Thus, culture and tradition affect the efforts made to monitor hate speech.

ii. **Data Sparsity**: a modern issue is the feature set of a dataset. On Twitter, for example, each tweet is limited to 140 characters. In this situation, the information included in a single tweet may not be adequate to draw broad conclusions about a certain subject (Mullah & Zainon, 2021). This issue could arise in any task that involves the processing of brief communications.

iii. **Lack of Dataset Balance**: since it naturally occurs in most real concerns, the problem of an uneven class dispersion in the dataset frequently arises in hate speech identification. The algorithm will get more information from the majority class (non-

hate) than from the minority class (hate speech). In most cases, the average (non-hateful) comment outweighs the abnormal (hateful) message (Mullah & Zainon, 2021).

iv. **Variations In Cultures and Languages**: differences in culture and tradition affect the concept of hate speech and what qualifies as hate speech. People's culture and traditions significantly influence how they determine whether something is undesirable. For example, what is acceptable speech in Kenya that may be considered hate speech in the U.S.? Analysts contend that for social media corporations to deal with this issue on their platforms, research on hate speech in non-western regions of the world needs to be conducted (Tulkens, 2013).

v. **Stereotyping**, i.e., during the current pandemic: pandemic or natural catastrophe victims may be stereotyped. Figure 2.3 depicts stereotyping on Twitter, where Donald Trump calls the Corona Virus Chinese Virus. A notable example of how Chinese people encountered stereotypes worldwide is the COVID-19 outbreak.

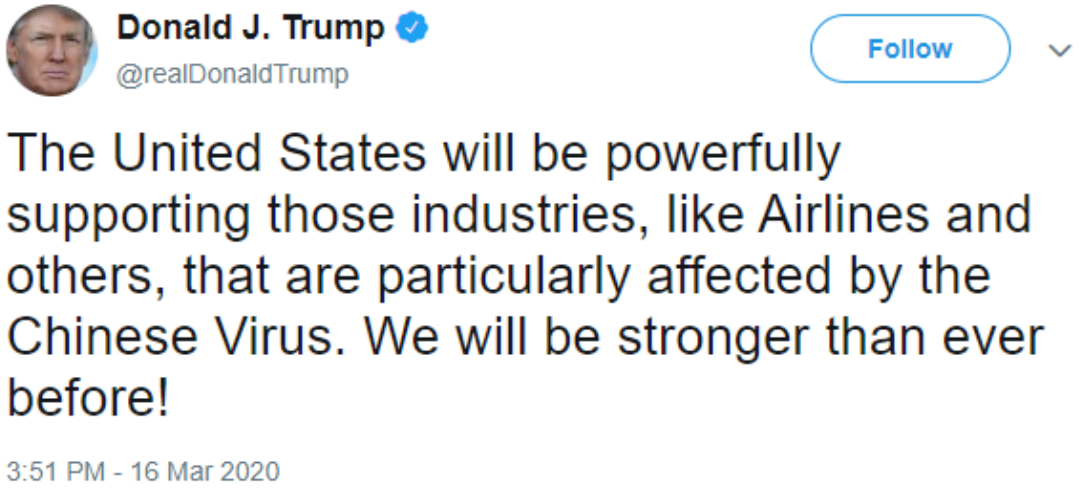


Figure 2.3 Donal Trump Stereotyping the Chinese for COVID (Twitter, 2020)

2.3 Empirical Literature

Numerous research projects on classifiers for hate speech detection use supervised learning techniques (classical methods). Support Vector Machines (SVM) are among the most popular (Schmidt & Wiegand, 2017). The works of Malmasi and Zampieri (2017), Davidson et al. (2017), and Robinson et al. (2018) are a few instances where SVMs were employed.

2.3.1 Most Relevant Papers

Table 2.1 Relevant Papers on Hate Speech Detection.

Paper	**Dataset(tweets)**	**Best Methods**	**Results**	**Limitations**
Vidgen & Yasseri (2020)	109488 containing islamophobias	Support Vector Machine	77% accuracy	The data collection was for the UK setting, and the word context had not been considered.
Davidson et.al (2017)	25000	Support Vector Machine	91% F1-score	For each class, the one-versus-rest model undergoes training, and the class label is given to the classifier with the best probability scores among all models.
Burnap & Williams (2014).	5593	Support Vector Machine	91% F1-score	The hate class for sexual orientation received only a 0.51 F1-score.
Watanabe, Bouazizi & Ohtsuki (2018).	14000	J48 graft	78% F1-score	Hatred speech is classified into three types: clean, insensitive, and hateful (three types of hatred coupled with hostile hate).
Bouazizi & Ohtsuki (2016).	2228 with sarcasm	Random Forest	83% accuracy	Most sarcastic tweets cannot be classified into the classification of sarcasm that opposes a happy mood with an adverse scenario. Sarcasm was not

				recognised as slander by certain writers.
Siino et.al (2021)	120000	Fuzzy ensemble	80% accuracy	Profile detection was prioritised over content detection.

2.4 Theoretical Literature

2.4.1 Information Theory

Bayin (2019) established a framework to illustrate a typical methodology for the information process as show in Figure 2.4. The methodology displayed data, information, knowledge, and objective, which are all fundamental parts of the information theory (Bayin, 2019). Information, in contrast to data, is a collection of rules that specify how and in what order the various records are related. Data is the raw information provided in natural languages. The aim marks the highest output, the ultimate reason for data production, while knowledge represents the capacity to comprehend and implement the established rules.

Figure 2.4 Information Theory Bayin (2019)

Figure 2.5 indicates how information theory is applied in machine learning (Stone, 2015). It shows that before data is fed to a machine learning mode, it must be pre-processed to give it meaning and context. Moreover, even after data has gone through the model's engine, it must be validated for precision and accuracy to ensure that the information is helpful to its end user (Stone, 2015).

Figure 2.5 Detailed Flow of Data into Information

2.4.2 Applied Optimization Theory

Information is now more easily accessible for creating and simulating scientific advancements and advances thanks to big data. To handle data-driven issues in research and engineering, it is now more important than ever to create large-scale analysis and augmentation models (Rao, 2019). Optimisation theory uses mathematics to determine the optimal course of action in various situations. An objective function is created, given conditions, and given the task of finding a solution that fits all the requirements. When creating a multifunctional heuristic algorithm, all contributing aspects must be considered as potential slow-changing scales.

Whenever non-linear issues have a minimum of a single non-linear functionality, optimisation models that address linear and non-linear issues are frequently developed (Chaovalitwongse et al., 2017). Chaovalitwongse et al. (2017) provided an approach to addressing an optimisation problem, as shown in Figure 2.6, where a model's development begins with formulating an empirical query that captures a challenging procedure. The process is simplified by delimiters

created by identifying the most crucial characteristics and eliminating the undesirable ones. A model with guidelines, constraints, and formulas is provided and constructed at this point.

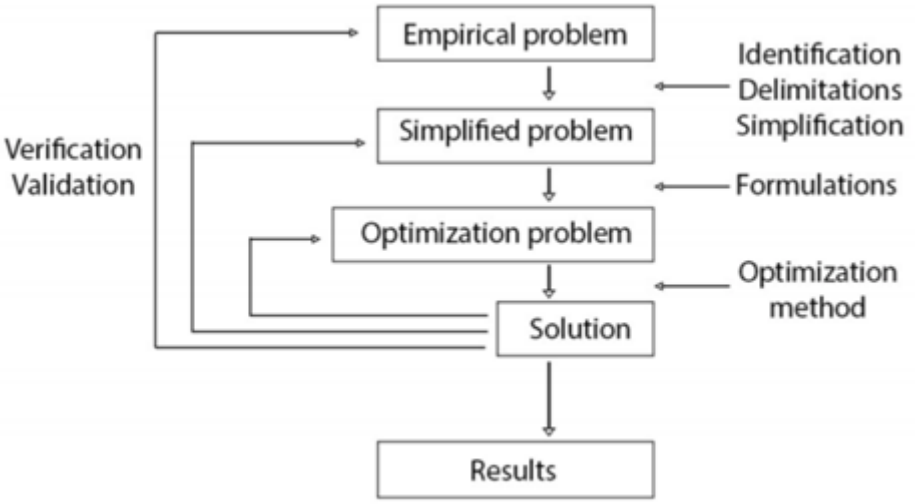


Figure 2.6 Applied Optimization Theory (Chaovalitwongse et.al, 2017)

2.4.3 Bayes' Theorem

The Bayes theorem is frequently referred to as the probability of "causes" formula. The Bayes theorem is based on predictive analysis, which can predict the likelihood that an event connected to any condition would occur. It is also considered in the situation of conditional probability. Conditional probability is when other factors determine the likelihood of an event occurring. For instance, suppose we need to determine the likelihood of selecting a blue ball from the second of three separate bags of balls, each containing three balls of a different colour: red, blue, and black.

Equation 2.6 Formula for The Bayes' Theorem

$$p(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

From Equation 2.6, if B is true, the likelihood of A being true is equal to that of B if A is indeed multiplied by the likelihood of A being true divided by B being true. Any given use case may have several tags attached, but they are all determined separately. We could use tags like Pricing, Usability, Features, etc., for categorising customer evaluations; however, each text entry would only be evaluated against one tag at a time:

Table 2.2 Using Bayes' Formula to Determine Whether Words or Phrases Fall into A Set of Specific "Tags".

Text	**Tag**
“Too expensive”	Pricing
“Super-fast installation”	Not Pricing
“Wish I would have known about the monthly fees”	Pricing
“Great product for the price”	Pricing
“There aren’t a lot of integrations”	Not Pricing

2.4.4 Regression Theory

A statistical theory analyses the relationship between the target or dependent variable and the independent variable in a dataset. Various regression analysis techniques are applied when the target and independent variables exhibit a linear or non-linear relationship, and the target variable has continuous values (Sur & Candès, 2019). Regression analysis is frequently used to identify cause and effect relationships, forecast trends, time series, and predictor strength. Regression analysis is the primary method for resolving regression issues in machine learning with data modelling. This approach involves locating the best-fit line, which minimizes the separation from every data point while traversing all the data points (Sur & Candès, 2019).

2.4.5 Space Transition Theory

Humanity has historically comprehended the impact of slanderous language. Abayomi (2020) states that crimes moving from one domain to another happens throughout the space shift. In this situation, conventional space gives way to the internet and vice versa. As we move to the digital space, some will comply with what is expected. On the other hand, some will not. However, with cyberspace's anonymity, individuals might get lost in the intrigue and end up indulging in hate and hate crimes as they believe the reparations are slightly reduced (Sharief, Hawasara, & Sinaulan, 2021).

2.4.6 Related Studies

Gitari et al. (2015) generated a list of "hate verbs"—wordings that support or suggest hatred—using a lexical-based methodology. Banning terms as the system's primary goal was made possible using this corpus of hate verbs in naive/straightforward systems. Because trying to adapt a model to a dataset with big dimensions (containing multiple features in question) renders a model susceptible to overfitting and requires more training time, Sorzano et al.

(2014)'s work on Dimensionality Reduction helps to mitigate a model's difficulty with generalizations and overfitting.

Zebari et al. (2020) removed unnecessary or redundant characteristics from a given dataset using feature selection. It looks for a segment of the initial feature set where the most important features predominate. He also built a new, more targeted collection of features using feature extraction that successfully catches most of the dataset's essential data. Waseem et al. (2016) used three distinct groups of annotators to label statements of hatred using the same machine-learning approaches on multiple annotated data sets, yielding noticeably different results. This revealed the impact of using annotators from a variety of backgrounds.

A pre-trained text classification model, BERT, was used in Mozafari, Farahbakhsh, and Crespi's (2019) transfer learning strategy to improve toxic content identification on evaluation datasets that are publicly provided. When using the pre-trained BERT model and adjusting it on the downstream objective using syntactical and context-sensitive information, they discovered that all BERT's transformers performed better, but they were incapable of "debasing" utterances of hate.

Using machine learning and deep learning techniques, Parikh, Desai, and Bisht (2019) investigated the identification of hate speech in social media messages. To find hate speech and objectionable language in tweets and Facebook postings, research was undertaken. They employed Deep Learning-based methods that make use of Convolutional Neural Networks, as well as text classification techniques including LR, SVM and NB classifiers. They conducted their studies using two methods; in the first, they employed CountVectorizer and TFI-DF to represent the input features. Both logistic regression and Nave-Bayes classifiers were trained using the features. Their second strategy employed hierarchical CNN with three I.D. Convolutional layers. Their research involved no data pre-processing. The efficacy of the techniques they employed was below 70% in any of the sub-tasks used in their trial.

Other researchers that used machine learning approaches like Gaydhani et al. (2018), Fauzi to increase the reliability of their classifier, Yuniarti (2018), Burnap and Williams (2015), Olteanu, Talamadupula and Varshney (2017), and even Davidson et al. (2017) did not apply an ensemble or multistep technique. Despite using an ensemble technique, Zhang et al. (2018) and Badjatiya et al. (2017)'s final classifiers nevertheless had to be capable to recognize the creative language, alternate grammar, and out-of-vocabulary words (OVW).

The requirement for strong comparative reports against the efforts of other researchers was another area in which this field needed to be improved. This could be because of various datasets acquired by various researchers. All the freely available internet datasets utilized for this study showed a significant class mismatch among the hateful language distinction and other classifications within the identical dataset. This suggests that most recent research was conducted under these unequal conditions, and it has been demonstrated that unequal conditions can lead to biases and machine learning models that overfit the prevalent class labels.

Aluru et al. (2020) experimented with a cross-lingual technique centred on zero-shot and few-shot learning by translating their model across nine distinct languages. They tested a variety of different machine learning methods, such as MUSE3 and LASER4 embeddings, and vector representations. In most testing configurations, LASER plus a Logistic Regression (L.R.) model turned out as the most efficient combination. This shows that traditional machine-learning techniques should still be considered. Stappen et al. (2020) employed a method that entailed extracting traits from models that had already been trained. They proposed a system in which tokenized text is first propagated by a pre-trained model, which then extracts vector representations. These representations were loaded into a categorization model. They also used an end-to-end classification model to fine-tune pre-trained multilingual models like BERT over the training data.

Meta-information from the text's authors and the network dynamics of the tweet could be seen as having "multilingual" features because they have nothing to do with the language in which the text is created. In the cross-lingual analysis of data in English and Spanish, Arango et al. (2020) used these characteristics and combined them with traditional machine-learning models.

Pamungkas and Patti (2020) investigated the idea of converting each of their datasets into a single language. Using datasets in English, Spanish, Italian, and German, they investigated this tactic. Once they got all their data in one language, they utilized a monolingual approach to categorize it and forecast the presence of hate in the text.

2.5 Frameworks

2.5.1 TensorFlow

A complete, open-source framework for creating machine learning models is called TensorFlow. It is a representation math toolbox that uses dataflow and distinct computing to

perform several operations geared toward deep neural network inference and training. It provides engineers with the necessary tools, frameworks, and public resources to construct machine-learning solutions. TensorFlow allows you to design dataflow graphs and structures to determine how data traverses a graph by accepting inputs as a multi-dimensional array called Tensor. You can use it to make a flowchart of the possible processes on these components, with the output displayed at the other end.

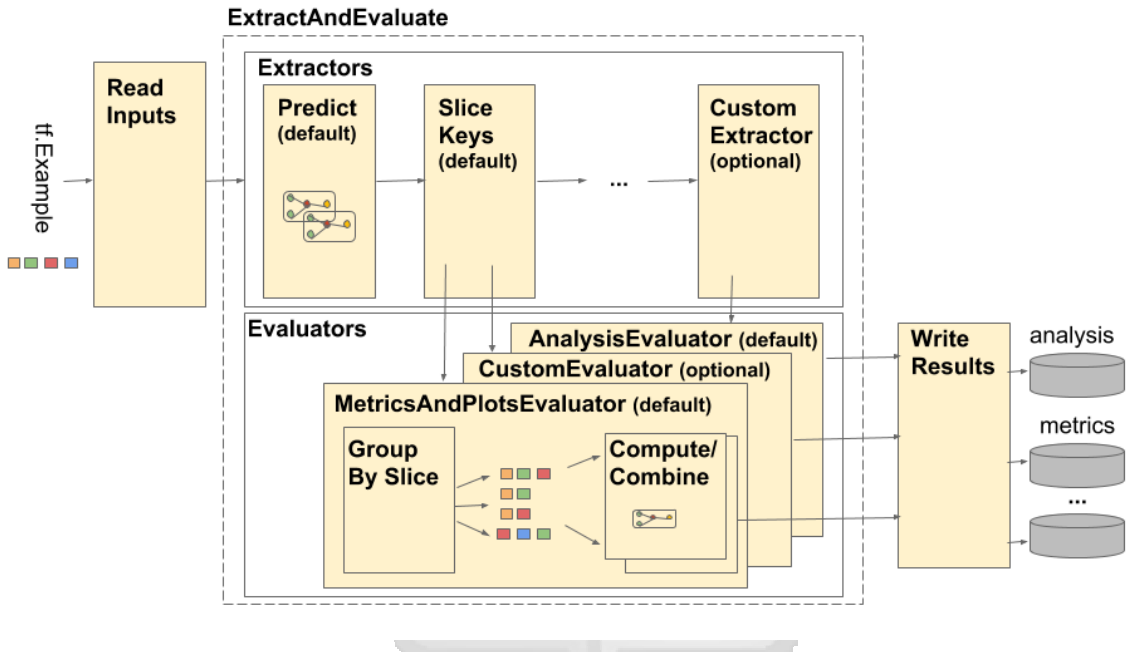


Figure 2.7 Depiction of The TensorFlow Model Analysis (TFMA) Pipeline (tensorflow.org, 2022)

2.5.2 PyTorch

PyTorch is an open-source machine learning and deep learning framework built by Facebook on top of the Torch library. It was created using CUDA, C++, and Python. It is well-known among novices in data science and machine learning because it manages challenging procedures effortlessly. It has broad applications in natural language processes, research, and computer vision. PyTorch uses standard debuggers like PDB and PyCharm.

2.5.3 Keras

Keras is an open-source framework developed on top of TensorFlow. It boasts of efficiently run-on GPUs and CPUs and has become the choice of high-level neural networks. Keras is fast, easy to implement, and modular by nature. It is not domain specific and can be applied to

data-centric domains like healthcare, corporate insights, sales predictions, customer support, virtual assistants, etc.

2.5.4 Comparing Deep Learning Frameworks

Table 2.3 Comparison of Deep Learning Frameworks (domino data la b.com,20222)

Element	**TensorFlow**	**PyTorch**	**Keras**
Interface	C++, Python, CUDA	Python, C/C++, Julia	Python
Architecture	Difficult to use	Simple to use, less readable	Simple to use
Performance	High Performance, suitable for large datasets	High Performance, suitable for large datasets	Best suited for smaller datasets
Learning Curve	Steep	Gentle	Easy
License	Apache 2.0	BSD	MIT
Debugging	Complex to debug	Good debugging capabilities	Simple architecture requires less debugging
Community support	High	Medium	Moderately High
Speed	Fast	Fast	Slow
Visualization	Excellent	Limited	Backend dependent

2.6 Architectures/Designs

2.6.1 CCNL-Ex

Jiang and Zubiaga (2021) present a cross-lingual capsule network learning model (CCNL-Ex) that incorporates multilingual word embeddings and semantic lexical analysis. As seen in Figure 2.8, the architecture is made up of six layers..

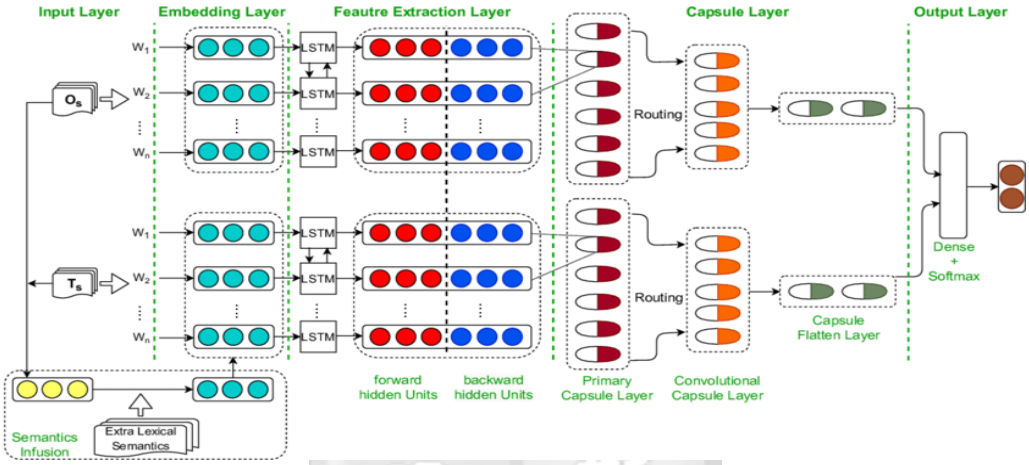


Figure 2.8 CCNL-Ex Architecture (Jiang & Zubiaga, 2021)

2.6.2 MLLIB and RDD Architecture

To detect remarks that are hateful in Amharic, Mossie, and Wang (2018), created a design. Data pre-processing and the creation of machine learning classification models, which are ideal for processing massive data like Facebook data, were done on an Apache Spark Standalone ensemble. As illustrated in Figure 2.10, Spark is built to provide effective MAP, Reduce, and filter transformation processes as well as execute actions for subsequent processes. To offer several tokenization strategies, the Spark ML pipeline is used. Furthermore, Spark provides modules for selecting features and the MLLIB library for deep learning. Data frame file type was chosen as the back end for preserving sluggish processes, which is excellent for big data, and the language of programming, Python, was employed for dataset preparation and learning algorithms with RDD (Mossie & Wang, 2018).

To appropriately categorize Facebook data into the two categories indicated above, produced by the specifications of the Naive Bayes and Random Forest techniques, the model was trained using 4,882 posts (Mossie & Wang, 2018). These classifiers were picked according to the outcomes of similar past research for English as well as for other languages.

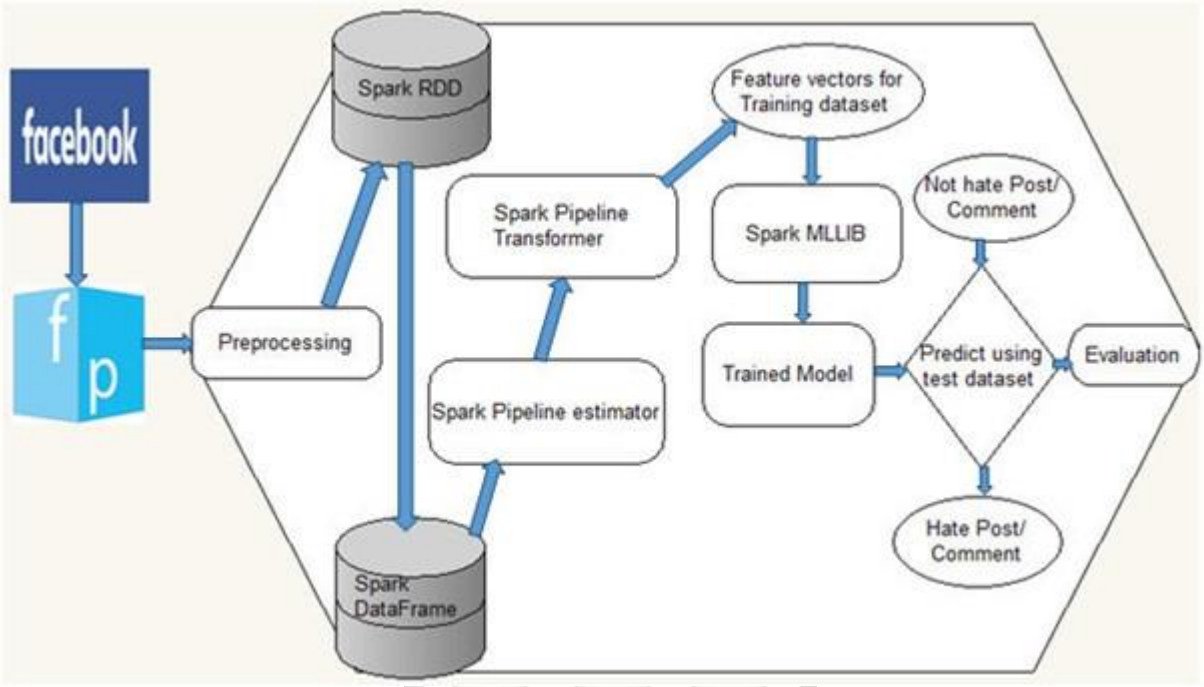


Figure 2.9 MLLIB Architecture

2.6.3 BERT and XLM Architecture

In this architectural design, as in Figure 2.11, the frozen BERT model acts as the feature extractor – brown frame. For example, a module chooses one or more output representations by pooling over the pink-indicated sequence (Stappen, Brunn, & Schuller, 2020). As depicted in green, a deep feed-forward layer is one example of the representation enhancement module that uses the selection output as input (Stappen, Brunn, & Schuller, 2020).

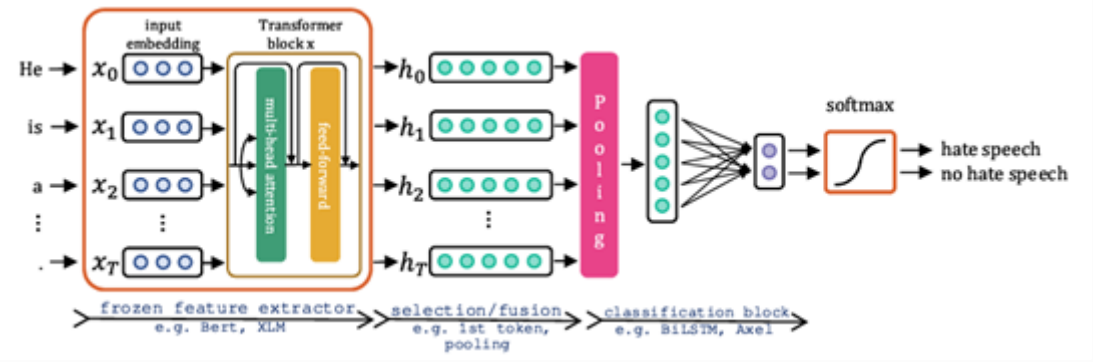


Figure 2.10 BERT Architecture

2.7 Algorithms

2.7.1 Naive Bayes

The Naive Bayes Classifier is a member of the family of Bayesian theorem-based probability classifiers. Its name, "Naive," refers to the strict independence assumptions that must be made between the input variables. Naive Bayes determines whether words or phrases fall into a set of specified "tags" (categories) or not when applied to text analysis (Pedregosa et al., 2011). Naive Bayes applies to emails, general papers, news stories, and customer feedback.

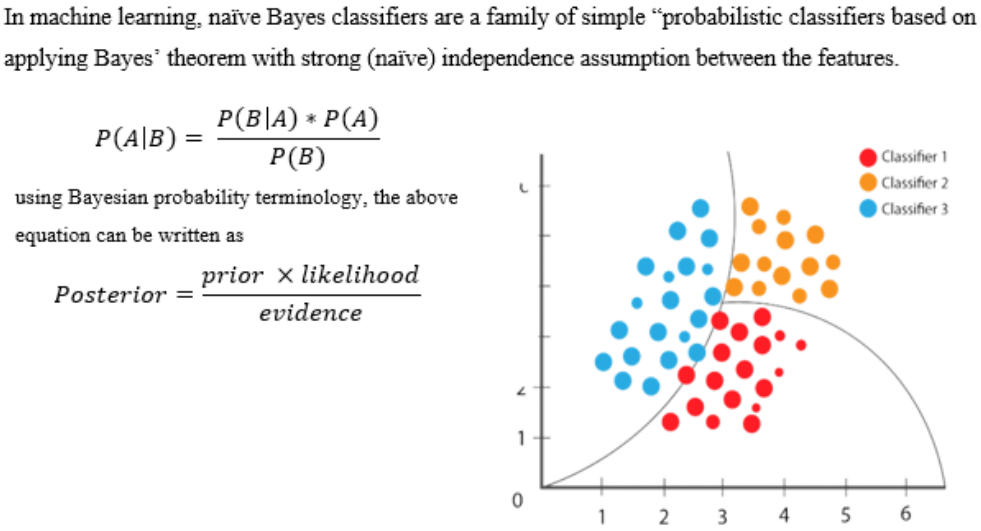


Figure 2.11 Underlying Concepts of Naive Bayes Classifiers (geeksforgeeks.com, 2022)

2.7.2 Support Vector Machines

The primary task carried out by SVM is the creation of a single linear plane in an x-dimensional space where each of the x features in each feature set corresponds to one dimension (Pedregosa et al., 2011). SVM works exceptionally well in situations where there are more dimensions than observations. A supervised learning approach used for difficulties with classification is a support vector machine. A very small percentage of samples should be on the incorrect side of the plane when the plane has been set up correctly. SVC, LinearSVC, and NuSVC are just a few of the support vector machine classes that are provided in the scikit-learn toolkit, which was employed in this work for machine learning challenges. The Stochastic Gradient Descent Classifier (SGDClassifier) with its standard loss factor (hinge) was utilized to generate the Linear SVM that was used in our investigation (Pedregosa et al., 2011).

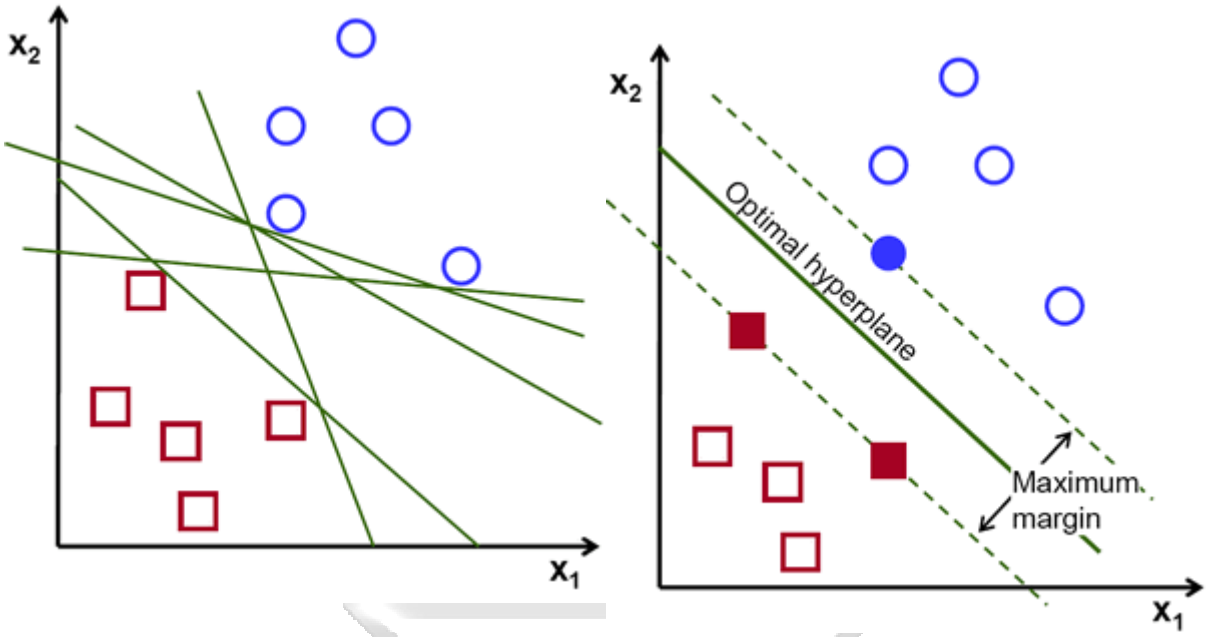


Figure 2.12 Support Vector Machines algorithm (geeksforgeeks.com, 2022)

2.7.3 Logistic Regression

This model or algorithm can analyse a data set and estimate a result using a variety of independent variables. A regression model with a categorical dependent variable is another name for it (Pedregosa et al., 2011). Since our objective is a binary classification exercise (features), the probability of a binary response based on several separate variables was calculated using a binary logistic model. The fundamental purpose of logistic regression is to determine the model that most accurately portrays the connection across the dependent and independent variables (Pedregosa et al., 2011).

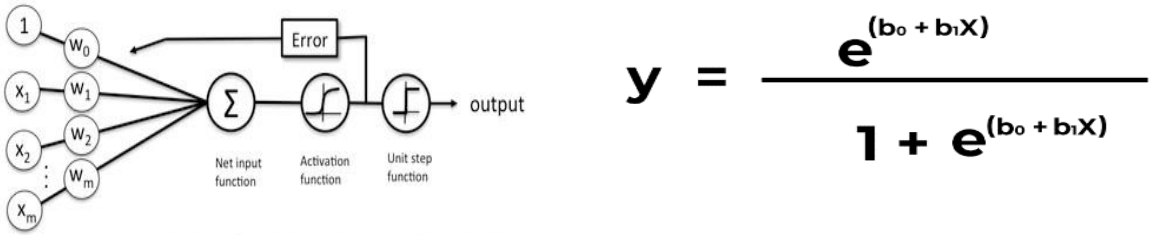


Figure 2.13 Schematics of Logistic Regression

2.8 Conceptual Framework

The study used the conceptual framework presented below to blend methodologies and develop a model that might meet the research goals based on this literature evaluation and the numerous research gaps highlighted. The framework offers a workable solution to the issue. With various

texts and current conditions, the model developed using this framework exhibited versatility and attained notable levels of precision. Arbitrary datasets and hate speech recognition scores were selected as the study's independent and dependent variables.

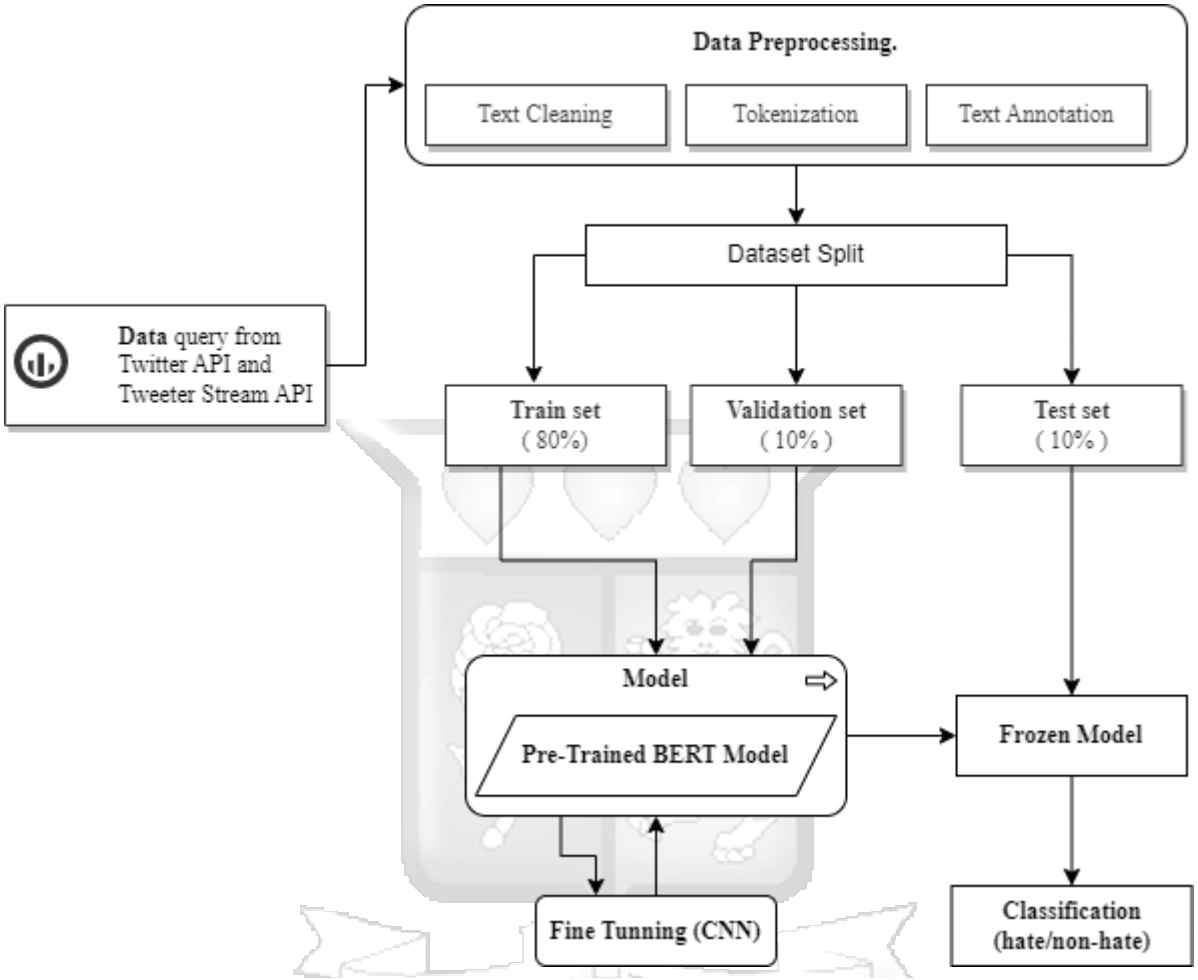


Figure 2.14 Conceptual Framework

Chapter Three: Research Design and Methodology

3.1 Introduction

Goddard and Melville (2004) outlined research as gathering information and answering unanswered questions or creating what does not currently exist. Meaning it is a process of expanding the boundaries of ignorance. Every research applies a research methodology, which is a way to solve the research problem systematically (Kothari, 2004).

The perception of hate speech in Kenya and the research methodologies used in the related studies reviewed in the literature review had a significant impact on the study. As part of the study, an applied methodology was used to build, carry out, and assess a cross-lingual model for identifying hate speech. To train the model and develop the research, primary data from previously published tweets believed to include hate speech were used. The model was tested for efficacy in detecting slanderous comments on Twitter, and the most effective machine learning approaches were chosen.

3.2 Research Design and Philosophy

3.2.1 Research Design

To answer the questions of what, where, when, how much, and by what, there is a need for the most appropriate and decisive research design (Kothari, 2004). According to Kothari (2004), a research design is the arranging of parameters for gathering and analysing information in a way that tries to balance the importance of the study's goal with efficiency in technique.

The experimental design method was employed in this study, which included creating a cross-lingual machine learning model based on transfer learning as a proof of premise and testing the model using multiple assessment metrics to determine its effectiveness (Kothari, 2004).

3.2.2 Research Philosophy

A research philosophy provides a guiding framework for research based on ideas about reality and the nature of knowledge (Collis & Hussey, 2014, p.43). This research study is underpinned by the positivists' research philosophy as it focused on scientific testing of the hypothesis and finding logical or mathematical proof that derives from statistical analysis (Collis & Hussey, 2014). It also used large sample sizes to produce precise, objective, and quantitative data (Collis & Hussey, 2014). Positivism holds that reality exists independently of us. As a result,

researchers can objectively observe reality, unlike interpretivism, where reality is seen as highly subjective because it is shaped by our perceptions (Collis & Hussey, 2014).

3.3 Population and Sampling

The entire corpus in a research study from which a sample can be drawn is referred to as a population, according to Bryman (2012). The population of this study was selected from Kenyan hate speech-related Twitter messages, or "tweets." The research used purposeful sampling, where the sample was chosen based on the researcher's assessment of the traits of tweets that qualify as hate speech. Such information was based on guidelines provided by the United States Department of Justice, UNESCO, and the Kenyan NCIC.

3.4 Data Collection Methods/Analysis

First, we procured a labelled dataset from Davidson et al. (2017) containing 24783 tweeter text, with 94.2% labelled as hate speech and 5.8% as non-hate speech. For improved accuracy and balance of the dataset, we combined the Davidson dataset with data from the University of Copenhagen, Aristotle University, the HASOC 2019 Shared Task and hastespeechdata.com.

This study also collected more recent datasets through text mining of Twitter API. We needed to apply for a developer account to access Twitter API for querying the tweet text. However, since the process of obtaining the API Key is long and will not accommodate the length of this process, we used third-party tools such as snscrape and Jefferson's Twitter API, which helped us bypass some limitations of Twitter API.

3.4.1 Data Cleaning and Preprocessing

NLP requires the conversion of datasets into lexicons that can be recognised and processed by its algorithms. Unlike regular text, tweets contain grammatically incorrect words, jointed words, abbreviations, joined words, repetitive words and special characters like hashtags and emoticons. Our goal is to use regular expressions to remove any special characters, usernames, URL links, retweets, or anything that does not conform to the semantics of a grammatically correct sentence.

The first process of Data pre-processing is lemmatisation, where we reconstruct the root form of each word. The second tokenisation uses stop words and removing words less than three characters. Lastly, we convert the text to normal or lowercase.

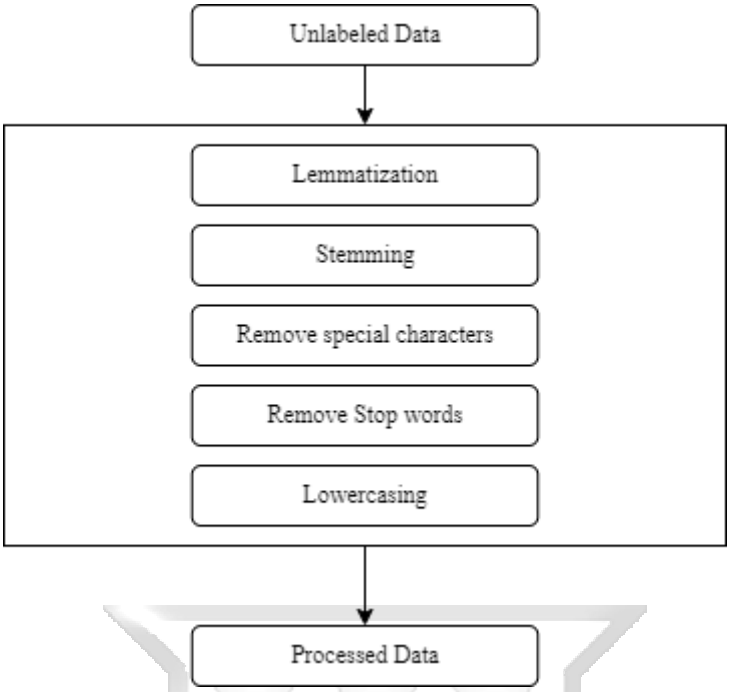


Figure 3.1 Data Preprocessing

3.4.2 Data Analysis

A TF-IDF analysis and a bag-of-words analysis were both employed in this work to analyse and extract pertinent lexicons. The TF-IDF analysis was used to help find pertinent words in the datasets. By comparing the keywords' frequencies with each other, the two-analysis helped determine how everyday hate speech was in the dataset.

3.5 Research Quality and Reliability

In this study, we employed a typical train-test-validation split; 80% of the data was used for training, 10% for validation, and 10% for testing (Xu et al., 2018). Shuffling the data before producing these splits is crucial to ensure that each split accurately represents the dataset. For precision and accuracy, we used classification and evaluation metrics.

There are four outcomes when making classification predictions Xu et al. (2018).

i. You have a true positive when you correctly predict that an observation belongs to a class, and it does.
ii. You have a genuine negative when you predict that observation does not belong to a class and does not belong to that class.

iii. You get false positives when you predict that an observation belongs to a class when it doesn't.

iv. You get false negatives when you predict that observation does not belong to a class when it does.

An illustration of how to classify texts that contain hate speech is in the confusion matrix in Figure 3.2. We would create this matrix by making predictions according to our test results and then categorizing each prediction as one of the four outcomes described above. These four results are typically plotted in this confusion matrix.

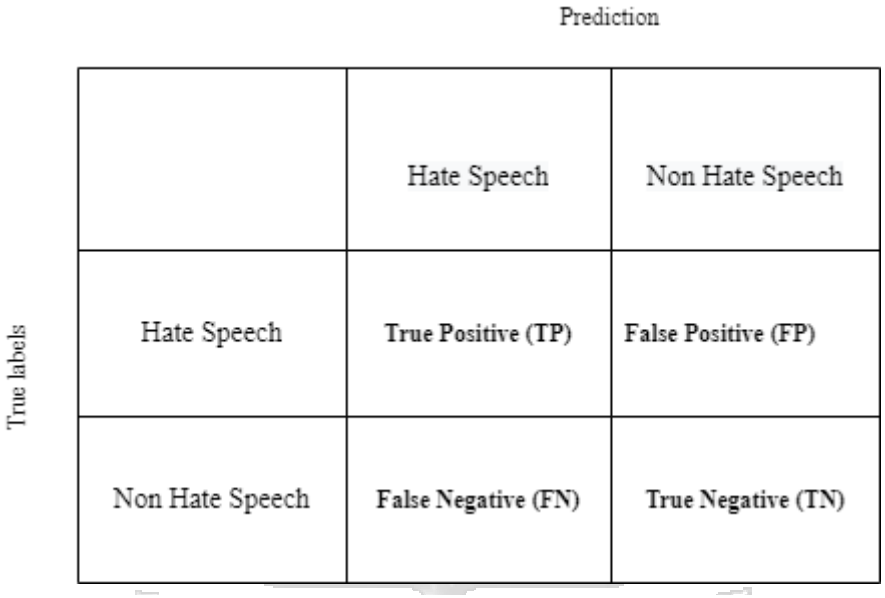


Figure 3.2 Confusion Matrix for Hate Speech Text Labels and Prediction

The four types of classification metrics are the counts of true and false positives and the counts of false and true negatives. This is the foundation for the measurements for evaluating text classifiers, such as model recall, model accuracy, model precision, and model f-scores (Feldman & Sanger, 2007).

Accuracy is the proportion of accurate projections for the test sample. It is simple to estimate by reducing the number of right guesses by the entire number of projections, as shown in Equation 3.1 (Feldman & Sanger, 2007).

Equation 3.1 Calculating the Accuracy of a Model's Prediction

$$accuracy = \frac{correct\ predictions}{all\ predictions}$$

Precision is the fraction of relevant examples (true positives) among all the examples predicted to belong in a particular class, as shown in Equation 3.2 (Feldman & Sanger, 2007).

Equation 3.2 Calculating the Precision of a Model's Prediction

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

A recall is the fraction of examples predicted to belong to a class concerning all the examples that genuinely belong in the class, as shown in Equation 3.3 (Feldman & Sanger, 2007).

Equation 3.3 Calculating Recall of a Model's Prediction

$$recall = \frac{true\ positives}{true\ positives + false\ positives}$$

F-1 Score is a calculated adjusted average of a combined recall with precision, as shown in Equation 3.4 (Feldman & Sanger, 2007).

Equation 3.4 Calculating The F-score Of a Model

$$f - score = 2 * \frac{precision * recall}{precision + recall}$$

3.6 System Development Methodology

Rapid Application Development was chosen as the main methodology of development for this research. We spent less on software development and upkeep because of RAD. The RAD methodology enables the development team to respond quickly and cost-effectively to the constantly changing system requirements. (Naz & Khan, 2015). We concentrated on a quick planning and development process in this study, which included development, testing, and feedback.

RAD is a better development method; however, it comes with risks. RAD reduces system-level scalability, reduces power efficiency and time in the system development process. The short development time impacts the system's quality (Naz & Khan, 2015). To overcome these shortcomings in our study, we employed a combination of RAD with other system development methods accompanied by valuable feedback and development tools such as DevOps that can monitor and track our development process.

In the RAD model, our cross-lingual model underwent four main stages of development, as in Figure 3.3: Planning for requirements, designing for users, building quickly, and transitioning.

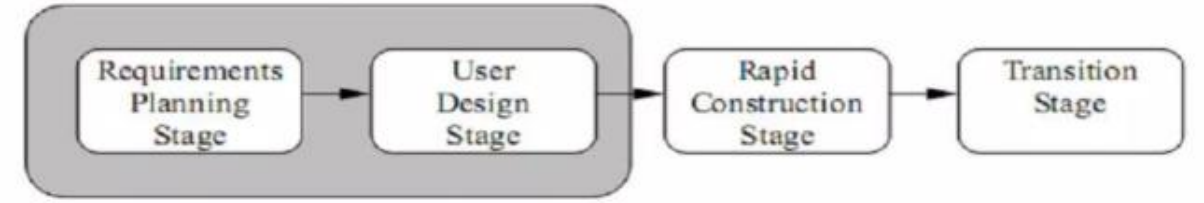


Figure 3.3 Development Stages In RAD

During the requirement collection stage, we used data mining to collect the requirements for the cross-lingual model and developed a general understanding of the models, algorithms, architectures, processes, and difficulties encountered when identifying hate speech online as well as the potential future applications of the model being developed.

In terms of the user design phase, we created the model's structure and architecture using diagrams created using the Unified Modelling Language (UML). Data diagrams, context diagrams and data flow diagrams were all included in these diagrams. We were able to represent the model's many parts and features thanks to these diagrams.

We used the Python programming language during the quick development stage to build the model by the specifications and user designs. The machine learning techniques were implemented using the TensorFlow framework. To make managing and analysing our dataset easier, we designed and built simple data structures using Python's pandas package. Panda’s being put into use, the model was assessed and verified. Then, utilizing several evaluation matrices, we decided on a mix of features and algorithms.

In the last stage, the transition stage, we deployed the developed and thoroughly evaluated model to detect and monitor any unreported occurrences of hate speech on Twitter. The model entered the unsupervised face, where it was fed with data and learned independently.

3.7 Utilization and Dissemination of Results

The results of this study were to help both governmental and non-governmental institutions that face the problem of online hate speech detection and uphold social cohesion in the dynamic internet age. The results were to form a basis for future studies in cross-lingual hate speech detection in low-resource languages like Indigenous African languages that still need to catch

up in offensive language detection. The dissemination of these findings was through publication on online academic repositories and research portals like the Strathmore University Library. On top of that, the dissemination was through word of mouth to its intended audience.

3.8 Ethical Considerations

All research depended on data. This data came from real-world users. In this way, there was a need for ethical guidelines and codes of conduct to guide research. This study handled tweets collected using the publicly available Twitter API. For this reason, this study strived to maintain the confidentiality and anonymity of these users and avoid potential harm to them. On top of that, this research employed mechanisms to enforce an honest, dependable, and credible communication of its finding. We also ensured that the research is free of plagiarism and misconduct. This research paper and topic went through an ethical clearance by Strathmore University's SU-ISERC giving it a green light of being ethically viable.

Chapter Four: System Design and Architecture

4.1 Introduction

This section gives a summary of the functionality and objectives of the cross-lingual model as well as the main elements of the recommended framework, such as its requirements analysis and architecture. UML diagrams were employed to design the model's overall architecture and intricate design. defining the system's architectural framework. The Diagrams provide an in-depth representation of the model's modules; and demonstrate how the end user would interact with the system and how the model's components interact.

4.2 Requirement Analysis

This study aimed to create a model for detecting hateful speech on Twitter within the Kenyan context. It argued the fact that Kenyan Twitter has multifaceted features due to the broad languages used in communication. This chapter highlights the different conditions that the preferred approach must meet depending on this goal.

4.2.1 Functional Requirements

i. Users should enter keywords and/or geolocations for streaming tweets using a web interface.
ii. The model should leverage user-inputted geolocations and/or keywords to stream tweets from the Twitter API.
iii. The model should clean the streamed tweets to get rid of extraneous characters and background noise.
iv. The model should preprocess the tweet by tokenizing them into input ids appropriate for natural language processing is a task. Tokenization should be carried out similarly to how the cross-lingual model was trained.
v. The model should categorize the tweet as hateful text or not using the cross-lingual inference engine that has already been trained through supervised learning.
vi. The user should see the model's categorised tweet.

4.2.2 Non-Functional Requirements

i. Availability

Since the model stream tweets, the model should be readily available for use as much as possible with very minimal downtimes. Deploying the model on a cloud instance should drive it towards this requirement.

ii. **Scalability**

The model should have the capacity to adapt to the sheer scale of requests and processes required by the user without any loss in performance and accuracy. For this, the model should be able to scale up and down using distributed celery workers.

iii. **Portability**

The model should have the capability of running on any computer no matter the operating system. The model is written in Python by default and supports all the available operating systems.

iv. **Usability**

The end-users and stakeholders should find it easy to make use of the model. It should perform predictions with few or no user errors. On top of that, the user should be able to understand its use case without strain.

4.3 Model Architecture

The model architecture employed in the present study is depicted in Figure 4.1. The model and the end user are described in the architecture as two separate entities that communicate with one another. The cross-lingual model of architecture is seen in the image above. (BERT). Consequently, the user can have excellent interpretations of languages with limited or no resources because there is only one model for every single language.

The model's functionality begins with a user request to get the prediction of tweets using specific keywords and optional geofencing. The requests trigger the Twitter Stream Service which makes use of the snscrape python package. The Twitter Stream Services streams tweets per the user-specified inputs. Streamed tweets are automatically passed through the Tweet Pre-Processor which removes noise such as the Twitter username(@), hashtags(#), contractions, misspellings and stop words.

The pre-processor continues the pipeline by sending the clean twitted to the Transformer module. The transformer is imported from a pre-trained model's tokenizer that performs tokenization, encoding and decoding and zero short translation. With tokenization, the model

creates a sequence of a word that can be used to independently train the model to help in the transcription or transliteration of the word considering the forms and sounds of words in different languages. The model the does encoding and decoding of words into sub-word tokens to increase the tokenization of Unicode or diverse language feature. The zero-short translation helps in translating between two languages even without prior data, i.e., we can translate English to Japanese and the model can have the capabilities of translating Japanese to Korean even without prior data on Korean.

Tokenized tweets pass through a pre-trained Cross-lingual Model, in this case, BERT. The model is loaded from Google's open-source torch model repository. The model is already trained in over one hundred languages. The mode of predetermined opinion units such as the English train set is used to train the classifier. The Swahili dataset is mapped to the model to test for accuracy and precision.

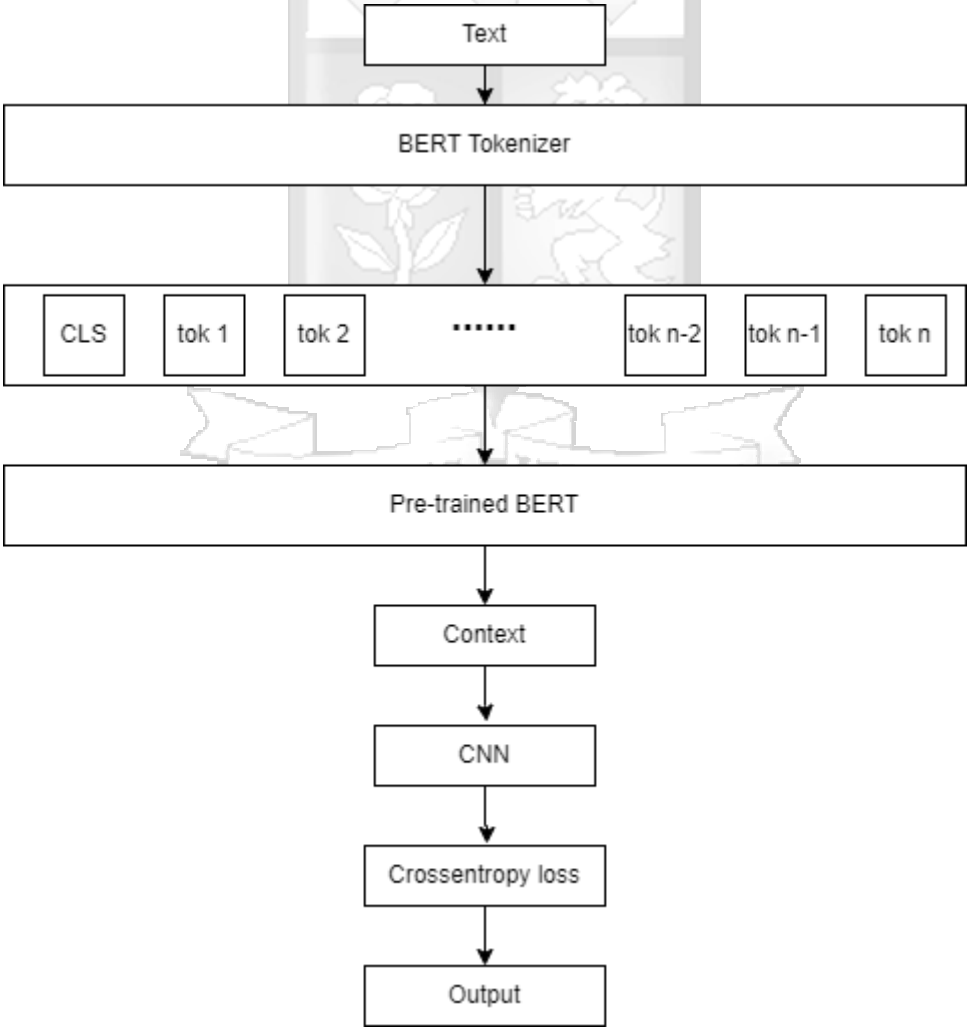


Figure 4.1 Architecture for This Model

4.4 Data Dictionary

The data dictionary in the figure contains metadata i.e., data about the tweets to be streamed from Twitter API. The data dictionary helps in the general analysis and access to both streamed and classified tweets objects.

Table 4.1 Data Dictionary Representing the Structure of Streamed Tweets

Name	Data Type	Description
Id	Text	The id of the tweet from Twitter
Text	Text	The tweet's raw content
Username	Text	The Twitter handle of the individual who tweeted
Place	Text	The geographical location of a tweet
Date	Date	Tweet's posting date

4.5 Context Diagram

Figure 4.2 depicts a high-level data flow diagram containing the proposed cross-lingual model for hate detection as the only process node. It outlines the model's context to its external entities like the user and the Twitter API. In this context diagram, the user requests the model process for the classification of tweets based on specific keywords and or geo-location. The model's inference engine triggers a streaming request to the Twitter API using the parameters provided by the user. The Twitter API responds to the model's request with a collection of tweets. The model classifies the tweet as hate and then responds to the user's initial request with classified tweets.

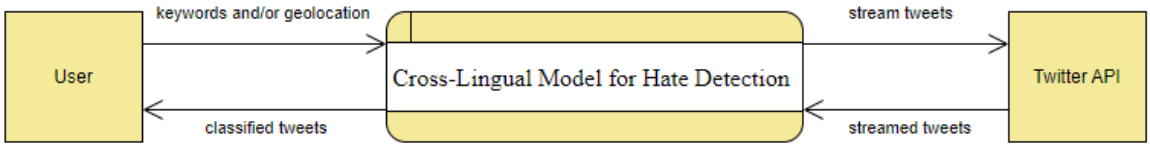


Figure 4.2 The Proposed Model's Contextual Diagram

4.6 Data Flow Diagram

The model is more precisely depicted in Figure 4.3's Data Flow Diagram (DFD), which lists all the operations and associated entities, processes, and data warehouses. The movement of

data between the DFD entities and processes is indicated by arrows. The user submits a classified tweets request to Process 1, "Stream Tweets," which makes a request to the Twitter entity. According to the user-provided download parameters and the stream tweets procedure, the Twitter API returns streamed tweets. Prepossess Tweets, Process 2's counterpart, takes tweets from Process 1 and purifies them for more accurate categorization. Sending the pre-processed tweets to Process 3's Extract Features creates a document by tokenizing, extracting, and weighing the tweet. The tweets that have been tokenized are then forwarded to process 4, also known as classify tweets. A user's web interface receives the labelled tweets because of step 5, which is known as Display Labelled Tweets.

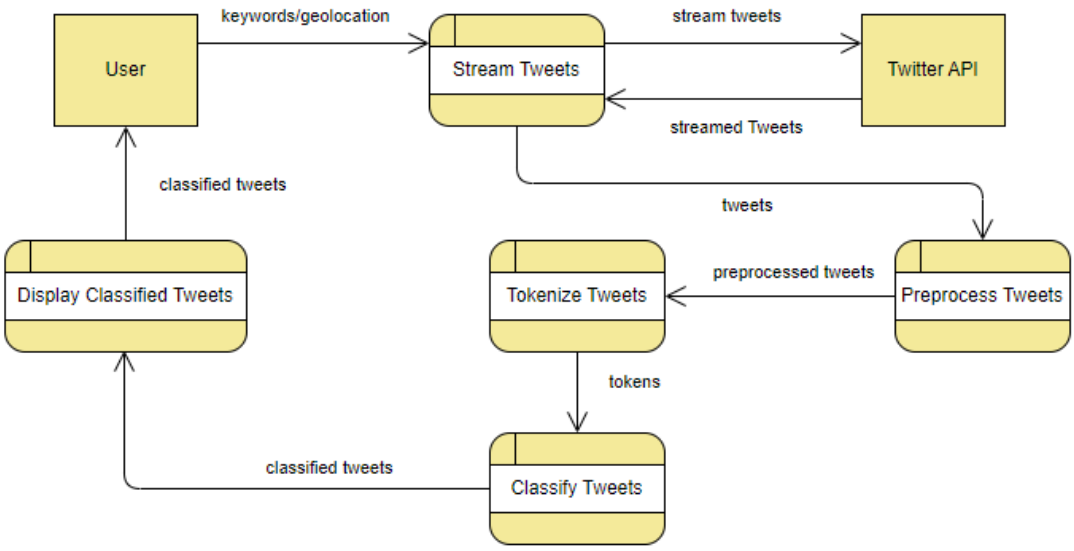


Figure 4.3 Data Flow Diagram for The Proposed Model

4.7 Wireframes

Wireframes are depictions of the visual representation of a system as seen by the use. Figure 4.4 shows the wireframe of how a user should interact with the inference engine of this study's Cross-Lingual Model. To get predictions, a user must input single or multiple keywords within the keywords input text. These keywords are the ones to be supplied to the Twitter Stream Service that communicates with the Twitter API. The user has an option to enter a geo-location for the geographic limitation of tweets to be streamed. But this geofencing is limited to the top

or most populous towns in Kenya. Figure 4.5 shows how the predictions produced by the classifier should be displayed on the user's screen.

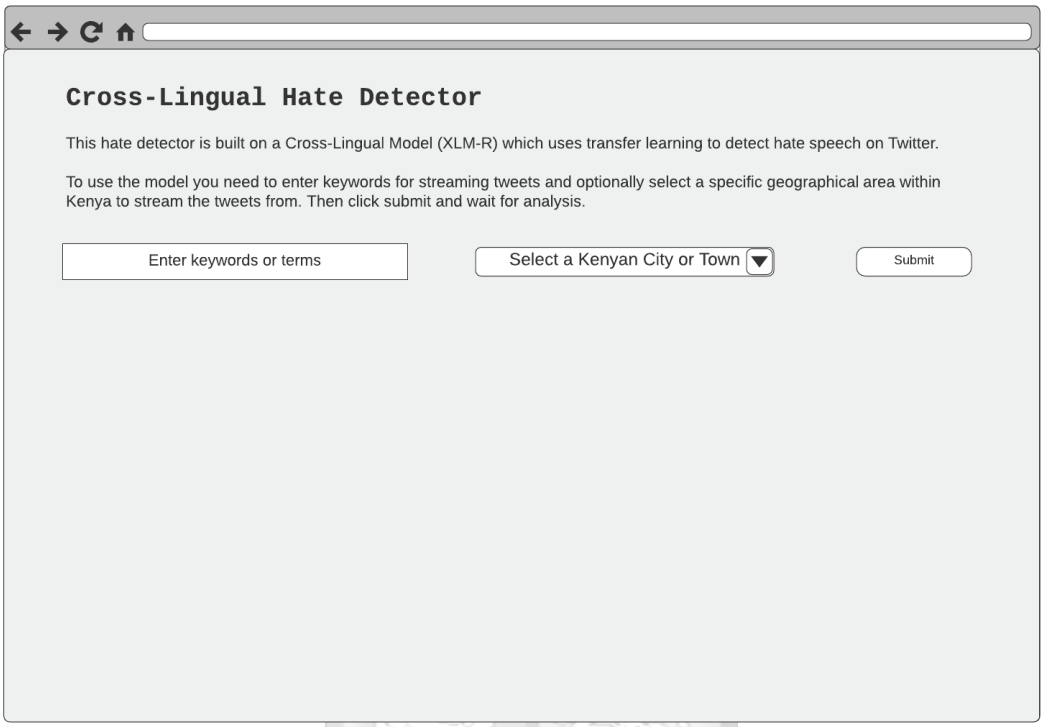


Figure 4.4 User Keywords and Geo-location Inputs

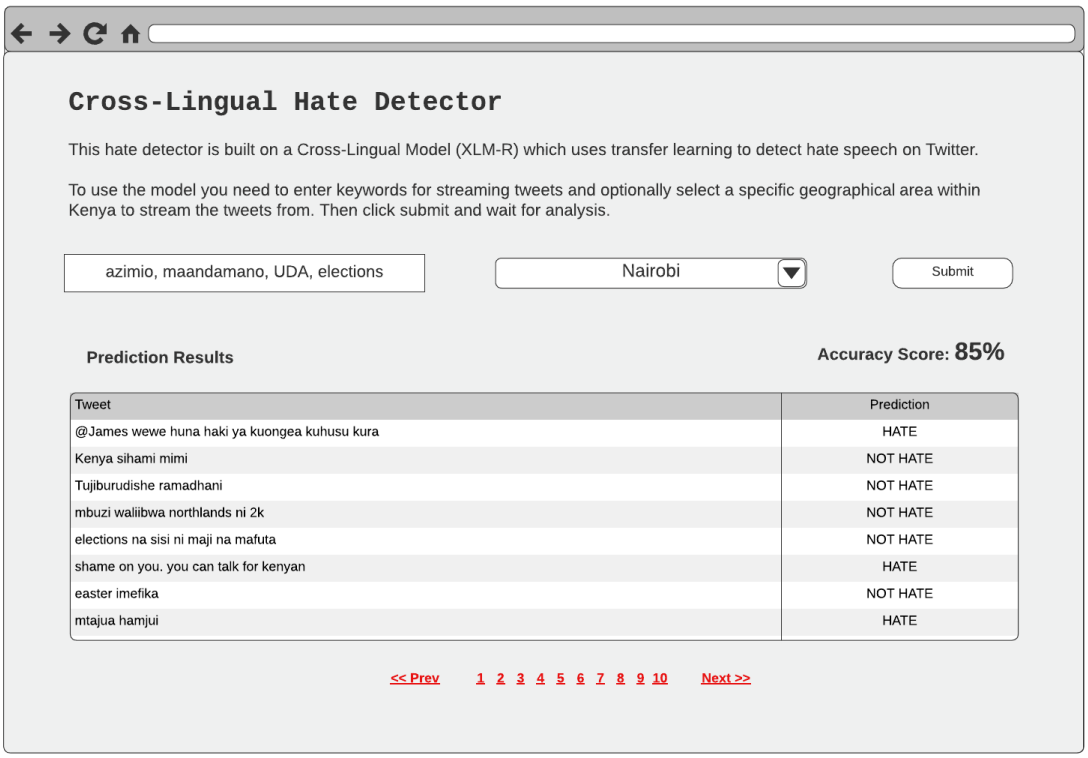


Figure 4.5 Classification Display After Model Inference

Chapter Five: System Implementation and Testing

5.1 Introduction

After the model's design phase, it was implemented, tested, and validated. The pre-trained BERT model was used to create, fine-tune, train and test the proposed model against a test set for accuracy. The model was frozen and used to detect hateful content in a fresh set of tweeter data. Experiments were performed on the prediction outcomes of the model and how well it performed against other machine learning models and algorithms like Naive Bayes, SVM and Ensemble Models. As discussed in Chapter Three, RAD methodology was used to develop the proposed model.

5.2 Implementation Environments

The design, implementation, and test of this model were carried out on an HP-ProBook Core i7 laptop with 8GB of RAM and 512GB SSD ROM. For simpler visualization, workflow, and debugging, the model was created using the Python programming language and Jupyter Notebooks.

The implementation process was moved to a Google Colab Notebook, which is powered by a Google Cloud instance with 15GB RAM, 17GB GPU RAM, and 166GB SSD ROM, due to the high cost of the model's training. The model was adapted on top a pre-trained BERT model availed by PyTorch and Google. This enabled us to benefit from transfer learning fully.

5.3 Data Collection

Python's snscrape module, which scrapes data from social media without the need for an API Key or a cap on the amount of data was used to collect data from the Twitter API. The unprocessed tweets totalling about 300,000 were to a CSV file for further analysis. The tweets were gathered using certain keywords, such as "Luos," "elections," "Kikuyus," and "persons of interest," such as Raila Odinga, William Ruto, and Uhuru Kenyatta. Most of the tweets gathered related to Kenya's most divisive historical events, namely the general elections of 2013 and 2017.

The dataset included a mix of Swahili and English text, but the majority was code-switched material, such as "King of Poverty have tuambia kitu," which was a combination of both languages.

The data collecting function in Figure 5.1 receives four core parameters: keyword, the maximum limit, geo-fence or geo-location, and date range. The keyword is the search term that the user enters for streaming tweets, such as elections 2013, maandamano, the maximum limit is the number of tweets to be streamed, and the geo-fence is the geographical region where the streamed tweets are posted, such as Nairobi and Mombasa, and the date range is the period over which the streamed tweets were posted.

```
def data_collector(
    keyword: str,max_limit: int = 5000,outfile: str = output_file,
    geo_fence: dict = None,date_range: dict = None,by_user: dict = None,exclude_retweets: bool = False, language: str = None):
    query = f"{keyword}"
    if by_user:
        query = f"{TwitterUsername(**by_user).get_username()}"
    if geo_fence:
        query = f"{query} {GeoFence(**geo_fence).get_geo_fence()}"
    if date_range:
        query = f"{query} {TweetsDateRange(**date_range).get_date_range()}"
    else:
        query = f"{query} since:2018-01-01"
    if exclude_retweets:
        query = f"{query} exclude:retweets"
    if language:
        query = f"{query} lang:{language}"
    with open(outfile, "a", encoding="utf-8") as csv_file:
        writer = csv.writer(csv_file)
        writer.writerow(
            [ "id", "tweet_date", "tweet_text", "tweet_username", "tweet_user_display_name", "tweet_user_is_verified", "tweet_user_location",
             "tweet_user_followers_count", "tweet_user_friends_count", "tweet_user_favourites_count", "tweet_user_listed_count", "tweet_replies_Count",
             "tweet_retweet_count", "tweet_likes_count", "tweet_quote_count", "tweet_language", "tweet_hashtags", "tweet_cashtags", "tweet_view_count", "tweet_place",
             "tweet_retweeted", "tweet_retweeted_id", "tweet_quoted", "tweet_quoted_id", "tweet_in_reply_to_username", "tweet_in_reply_to_user_display_name", "tweet_url", ]
        )

        for i, tweet in enumerate(sntwitter.TwitterSearchScraper(query).get_items()):
            print("Tweet: ", tweet)
            writer.writerow(
                [ tweet.id, tweet.date.strftime("%Y-%m-%d %H:%M:%S"), tweet.rawContent, tweet.user.username, tweet.user.displayname, tweet.user.verified,
                 tweet.user.location, tweet.user.followersCount, tweet.user.friendsCount, tweet.user.favouritesCount, tweet.user.listedCount, tweet.replyCount,
                 tweet.retweetCount, tweet.likeCount, tweet.quoteCount, tweet.lang, _get_hashtags_cashtags(tweet.hashtags), _get_hashtags_cashtags(tweet.cashtags),
                 tweet.viewCount, _get_tweet_place(tweet.place), _get_retweeted_quoted_tweet(tweet.retweetedTweet), _get_retweeted_quoted_tweet_id(tweet.retweetedTweet),
                 _get_retweeted_quoted_tweet(tweet.quotedTweet),
                 _get_retweeted_quoted_tweet_id(tweet.quotedTweet), _get_reply_username(tweet.inReplyToUser), _get_reply_user_fullname(tweet.inReplyToUser), tweet.url]
            )
            # Stop after collecting 10,000 tweets for each keyword
            if i >= max_limit:
                break
```

Figure 5.1 Twitter Data Collection Function

5.4 Data Annotation

Tweets are to be first classified into negative and positive before they are used to train the Cross-Lingual Model. This process is referred to as data annotation. We would have opted for manual annotation as it would be more accurate but due to the little time available, we opted for automated classification techniques such as VADER, TextBlob and NLTK Sentiment Analyzer.

Because VADER requires hardly any training data and can comprehend a text's sentiments even when it contains uppercase letters, punctuation marks, conjunctions, as well as slang phrases it was chosen as the tool of choice for pre-model training text classification. In comparison to other lexical sentiment analysis methods, such as TextBlob (76%), and NLTK (60%), VADER achieves an accuracy of 78%. The effectiveness of VADER in comparison to

other lexical techniques for sentiment analysis is shown in the table below (Bonta, 2019). The efficacy of VADER in comparison to other lexical sentiment analysis methods is shown in Figure 5.2.

Lexicon	Classification Accuracy metrics			
	Precision%	Recall%	F1 score%	Accuracy%
VADER	78.46	85.0	81.60	77.0
Textblob	76.92	81.96	79.37	74.0
NLTK	60	55.0	57	62.0

Figure 5.2 Lexicon Classifiers' Performance

The code in Figure 5.3 below shows how the VADER lexicon classifier was used to automate the annotation process. In the code, any text that had a compounded sentiment of over 0 was positive sentiment or non-hate while those below were considered negative sentiment or hate.

```
def vader_lexicon_classifier(df, data_dir, text_key):
    nltk.download("vader_lexicon")
    sid = SentimentIntensityAnalyzer()
    REMOVE NaN VALUES AND EMPTY STRINGS:
    df.dropna(inplace=True)

    blanks = []  # start with an empty list

    for i, lb, rv in df.itertuples():
        if type(rv) == str:
            if rv.isspace():
                blanks.append(i)

    df.drop(blanks, inplace=True)

    df["scores"] = df[text_key].apply(lambda text: sid.polarity_scores(text))
    df["compound"] = df["scores"].apply(lambda score_dict: score_dict["compound"])
    df["label"] = df["compound"].apply(lambda c: "positive" if c >= 0 else "negative")

    text_cal = df[["label", text_key]]

    text_cal.to_csv(f"{data_dir}/vader_text_labelled_text.csv", index=False)
    df.to_csv(f"{data_dir}/vader_labelled_text.csv", index=False)

    return df
```

Figure 5.3 VADER Lexicon Classification Function

5.5 Data cleaning

Just like the people on Twitter, tweets contain a lot of noise. Noise is the unwanted characters such as URLs, HTML characters, Unicode, hashtags(#), user mentions(@), misspellings, contractions, abbreviations, and acronyms as shown in Figure 5.2. Such entities within tweets make it difficult for machine learning techniques such as NLP to analyse, process and perform predictions on such text.

Figure 5.4 shows a code snippet for the tweet-cleaning process. This code converts the tweets into much more digestible text in Figure 5.5.

```
def pre_process_dataset(values):
    new_values = list()
    # Emoticons
    emoticons = [':-)', ':)', '(:', '(-:', ':))', '((:', ':-D', ':D', 'X-D', 'XD', 'xD', 'xD', '<3', '</3', ':\*',
                 ';-)',
                 ';)', ';-D', ';D', '(;', '(-;', ':-(', ':(', '(:', '(-:', ':,(', ':\'(', ':"(', ':((', ':D', '=D',
                 '=)',
                 '(=', '=(', ')=', '=-O', 'O-=', ':o', 'o:', 'O:', 'O:', ':-o', 'o-:', ':P', ':p', ':S', ':s', ':@',
                 ':>',
                 ':<', '^_^', '^.^', '>.>', 'T_T', 'T-T', '-.-', '*.*', '~.~', ':*', ':-*', 'xP', 'XP', 'XP', 'Xp',
                 ':-|',
                 ':->', ':-<', '$_$', '8-)', ':-P', ':-p', '=P', '=p', ':*)', '*-*', 'B-)', 'O.o', 'X-(', ')-X']

    for value in values:
        # Remove dots
        text = value.replace(".", "").lower()
        text = re.sub(r"[^a-zA-Z?.!,¿]+", " ", text)
        users = re.findall("[@]\w+", text)
        for user in users:
            text = text.replace(user, "<user>")
        urls = re.findall(r'(https?://[^\s]+)', text)
        if len(urls) != 0:
            for url in urls:
                text = text.replace(url, "<url >")
        for emo in text:
            if emo in emoji.EMOJI_DATA:
                text = text.replace(emo, "<emoticon >")
        for emo in emoticons:
            text = text.replace(emo, "<emoticon >")
        numbers = re.findall('[0-9]+', text)
        for number in numbers:
            text = text.replace(number, "<number >")
        text = text.replace('#', "<hashtag >")
        text = re.sub(r"([?.!,¿])", r" ", text)
        text = "".join(l for l in text if l not in string.punctuation)
        text = re.sub(r'[" "]+', " ", text)
        new_values.append(text)
    return new_values
```

Figure 5.4 Twitter Data Clean Up Function

Additionally, all tweets are converted to lowercase. Figure 5.5 below shows a sample of the pre-processed tweets.

id	clean_text
#	he cannot comprehend such kazi ni raila this and that
#	feigning ignorance you own no company and jealousy makes you pretend that you do not know the name of raila s company condemn what is wrong the journa
#	but daktari i thought what raila is doing is enshrined in the constitution whereas what trump did was an illigality
#	please h e ruto arrest raila
#	monday no missing lakini mnatubeba aje hiyo gari ya raila ilikuwa imejaa watu nje wamehang hakuna mtu ata mmoja hizi seven bullets zilipata ata kwa mguu a
#	these demonstrations would make more sense if raila and his fellow clowns would be giving solutions to ending high cost of living how exactly does demonstra
#	raila anakunyima usingizi bwana
#	who are they to decide wen raila retires
#	they cannot protect because the time they have it is to kill kenyans do you know ig inspector general of police he have to follow the article kulida wanaichi but

Figure 5.5 Sample Clean Tweets After Cleaning Up

5.6 Data Pre-processing

The preprocessing of data involved passing texts into a transformers tokenization pipeline as shown in Figure 5.6 below. In this pipeline, the text is split into tokens which are words, parts of words or punctuation symbols. The tokenizer adds potential special symbols and converts those symbols into unique input ids which are defined by the tokenizer's parameters. These input ids mask the text making them easily digested by the model's inference pipeline.

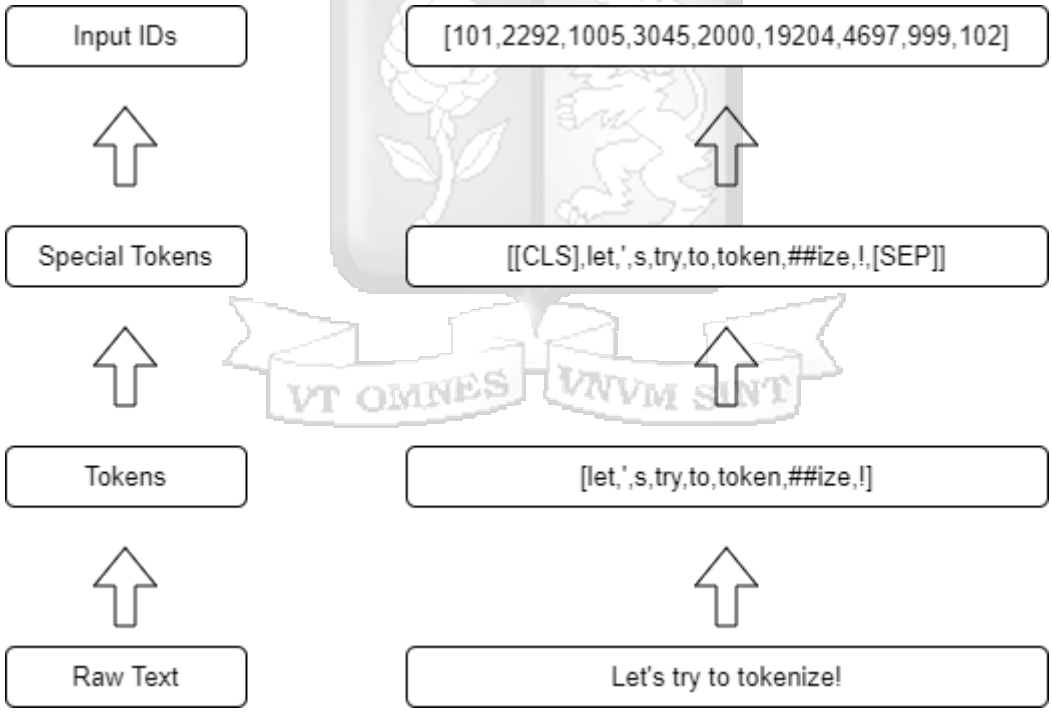


Figure 5.6 Text Tokenization with Transformers

Data pre-processing through the transformers tokenization pipeline was achieved using the function in Figure 5.7.

```
def data_process(data, labels):
    input_ids = []
    attention_masks = []
    bert_tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-uncased")
    for sentence in data:
        bert_inp = bert_tokenizer.__call__(sentence, max_length=36, padding='max_length',
                                           pad_to_max_length=True, truncation=True,
                                           return_token_type_ids=False)
        input_ids.append(bert_inp['input_ids'])
        attention_masks.append(bert_inp['attention_mask'])
    #del bert_tokenizer
    #gc.collect()
    #torch.cuda.empty_cache()
    input_ids = np.asarray(input_ids)
    attention_masks = np.array(attention_masks)
    labels = np.array(labels)
    return input_ids, attention_masks, labels
```

Figure 5.7 Tweets Preprocessing Using Tokenization

5.7 Model Generation

To build the cross-lingual model as shown in Figure 5.8, we first created the model's class, BERT_CNN, which takes the neural networks module from Python as input. The class contains two significant methods: unit and forward.

```
class BERT_CNN(nn.Module):

    def __init__(self):
        super(BERT_CNN, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-multilingual-uncased')
        self.conv = nn.Conv2d(in_channels=13, out_channels=13, kernel_size=(3, 768), padding='valid')
        self.relu = nn.ReLU()
        # change the kernel size either to (3,1), e.g. 1D max pooling
        # or remove it altogether
        self.pool = nn.MaxPool2d(kernel_size=(3, 1), stride=1)
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(416, 3)
        self.flat = nn.Flatten()
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, sent_id, mask):
        _, _, all_layers = self.bert(sent_id, attention_mask=mask, output_hidden_states=True, return_dict=False)
        # all_layers  = [13, 32, 64, 768]
        x = torch.transpose(torch.cat(tuple([t.unsqueeze(0) for t in all_layers]), 0), 0, 1)
        del all_layers
        gc.collect()
        torch.cuda.empty_cache()
        x = self.pool(self.dropout(self.relu(self.conv(self.dropout(x)))))
        x = self.fc(self.dropout(self.flat(self.dropout(x))))
        return self.softmax(x)
```

Figure 5.8 BERT Model Class

The infit method initialises the model, while the forward method defines how the model should operate from inputs to outputs. The unit method initialises the BERT model by loading the pre-

trained bert-base-multilingual-uncased model. BERT multilingual is a self-supervised transformers model pre-trained on vast corpora of bilingual datasets. It was trained solely on unprocessed texts, with no human annotation, using an artificial mechanism to produce inputs and classifications derived from these texts. The type of tokenizer used to preprocess the dataset defines the pre-trained model of choice. In addition, the rectified linear unit (ReLu), 2D convolution layer, 2D max pooling, Dropout, Linear, Flatten, and Log SoftMax are all initialised.

The forward method takes two parameters: the sent id and the mask. The sent id corresponds to the input id, which is a unique identifier for the tokens generated during the preprocessing stage. The mask, also known as the attention mask, signals to the model which tokens should and should not be attended to.

Figure 5.9 shows the model class and its push to the device that would run it.

```
# pass the pre-trained BERT to our define architecture
model = BERT_CNN()

# push the model to GPU
model = model.to(device)
```

Figure 5.9 Model Initialisation by Calling The BERT_CNN Class.

5.8 Model Training

We used 80 per cent of the annotated sample for training purposes to fine-tune the parameters, 10 per cent for validation to assess the classifier's out-of-sample accuracy while training, and 10 per cent for test purposes to assess the model's out-of-sample efficacy after learning. Statistical sampling was used to choose 0.8, 0.1, and 0.1 sections of texts from every category, negative and positive, for train, validation, and testing, respectively. This prevents overfitting. The code in Figure 5.8 shows the test, valid and train set splinting.

```
# ~~~~~~~~~~ Split train data-set into train, validation and test sets  ~~~~~~~~~~#
train_text, temp_text, train_labels, temp_labels = train_test_split(df, labels, random_state=2018, test_size=0.2, stratify=labels)

val_text, test_text, val_labels, test_labels = train_test_split(temp_text, temp_labels, random_state=2018, test_size=0.5, stratify=temp_labels)
```

Figure 5.10 Code Splitting Function - Test, Valid and Train Sets

The implementation of the proposed cross-lingual model used PyTorch BERT pre-trained model and transformers' Text Tokenizer with a pre-trained WordPiece. The model's training process used a batch size of 32 over 3 epochs. Tweets were tokenized into unique input ids using a BERT tokenizer. These tweets had their texts lowercase, punctuation separated, and erroneous characters removed. WordPiece was used to split the text into smaller words and finally even smaller units as per the provision of the BERT tokenizer. Since tweets are brief words, we set the optimum string length to 64 only for those texts with 64 characters or less otherwise the texts should be padded with zeros or trimmed to the best length. The function in Figure 5.9 below shows how the training process is implemented. The complete model training code is available in Appendix B at the bottom of this paper.

```
# loss function
#cross_entropy = nn.NLLLoss(weight=weights)
cross_entropy = nn.NLLLoss()

# set initial loss to infinite
best_valid_loss = float('inf')
# number of training epochs
epochs = 3
current = 1
# for each epoch
while current <= epochs:

    print(f'\nEpoch {current} / {epochs}:')

    # train model
    train_loss, _ = train()

    # evaluate model
    valid_loss, _ = evaluate()

    # save the best model
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        #torch.save(model.state_dict(), 'saved_weights.pth')

    # append training and validation loss
    #train_losses.append(train_loss)
    #valid_losses.append(valid_loss)

    print(f'\n\nTraining Loss: {train_loss:.3f}')
    print(f'Validation Loss: {valid_loss:.3f}')

    current = current + 1
```

Figure 5.11 Functions for Train And Evaluate Loop

5.9 Model Testing

The test dataset and two distinct metrics, the confusion matrix and evaluation matrix, are used to describe the outcomes of the cross-lingual model's evaluation. The model achieved an

astonishing accuracy score of 77%. To compute the values of the performance and confusion matrices, we used the ROC, AUC, Confusion, and Confusion Report functions provided by the scikit-learn library. First, we used the trained model to make predictions against the test dataset then pass then passed those predictions and the actual values of the labels are parameters of the scikit-learn performance calculators as show in Figure 5.12.

The precision, recall, and f1-score of the model are listed in Table 5.1, while the confusion matrix in Table 5.2 contrasts the real classifications with the projected classifications.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
                            confusion_matrix,classification_report,roc_curve, auc

# get predictions for test data
gc.collect()
torch.cuda.empty_cache()

with torch.no_grad():
    preds = model(test_seq.to(device), test_mask.to(device))
    #preds = model(test_seq, test_mask)
    preds = preds.detach().cpu().numpy()


print("Performance:")
# model's performance
preds = np.argmax(preds, axis=1)


# Calculate Classification Report
cr = classification_report(test_y, preds)

print("Accuracy: " + str(accuracy_score(test_y, preds)))

# Calculate confusion matrix
cm = confusion_matrix(test_y, preds)

# Get the false positive rate, true positive rate, and threshold values
fpr, tpr, thresholds = roc_curve(test_y, preds)

# Calculate the AUC score
roc_auc = auc(fpr, tpr)
```

Figure 5.12 Model Testing Function

Table 5.1 Cross-Lingual Confusion Matrices

	Predicted label – 0	Predicted label – 1
True Label – 0	143	52
True Label – 1	37	164

Table 5.2 Cross-Lingual Evaluation Matrices

	Precision	Recall	F-1	Support
0	0.79	0.73	0.76	195
1	0.76	0.82	0.79	201
Weighted average	0.77	0.78	0.77	396

On top of those matrices, a ROC curve shown in Figure 5.10 was also created to depict the efficacy of the cross-lingual model.

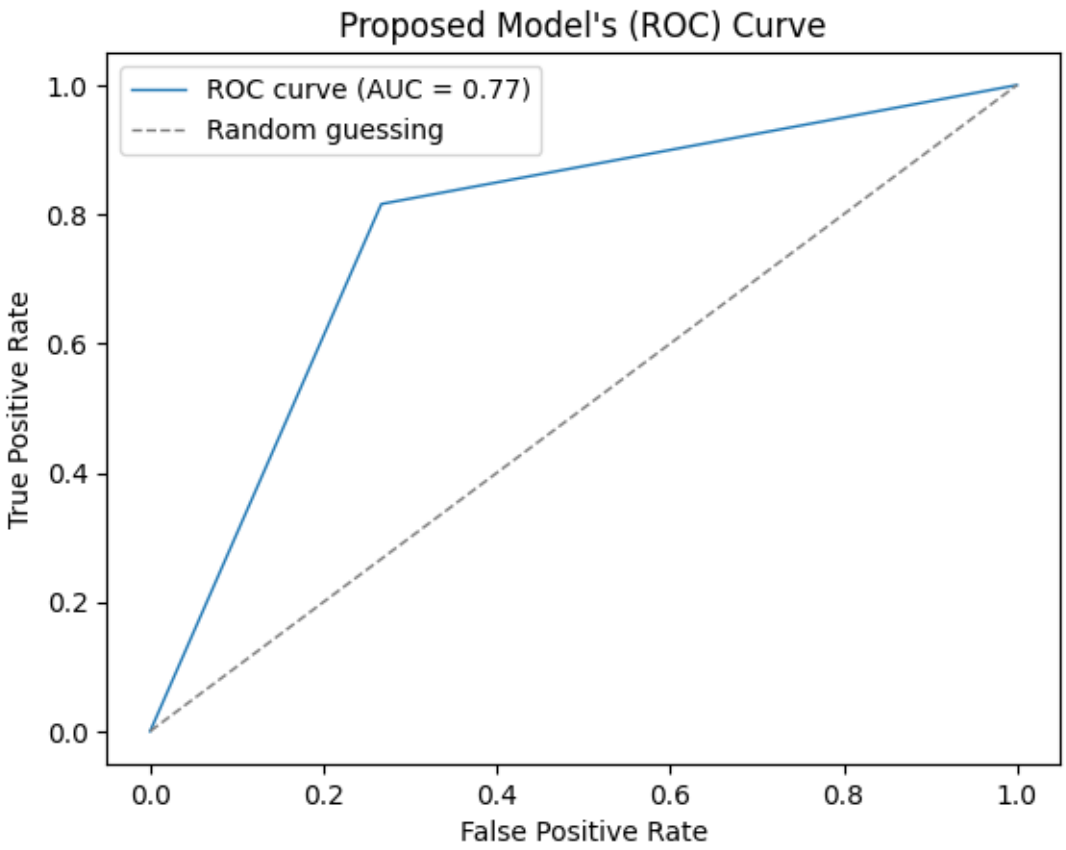


Figure 5.13 Cross-Lingual ROC curve

5.10 Saving The Model

To avoid needing to be retrained, the model requires being persistent. The code in Figure 5.11 was used to save the model locally. To perform prediction brand-new tweets, need to be streamed via the Twitter API and the model loaded as necessary.

```
output_model = "/content/drive/My Drive/Thesis/models/SwaBert.pth"

# Save model
def save(model, optimizer):
    # save
    torch.save({
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict()
    }, output_model)

save(model, optimizer)
```

Figure 5.14 Function for Persisting The Model

5.11 Using the Model in Prediction

Since the model was previously locally persisted, it is simpler to generate predictions on fresh data by simply importing the model and feeding it data as shown in Figure 5.12. To stream tweets from the Twitter API, the user provides keywords, and the streamed tweets are then sent to the built-in model for prediction.

```
# load Model
checkpoint = torch.load(output_model, map_location='cpu')
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
```

Figure 5.15 Load Persisted Model for Predictions

To stream tweets from the Twitter API, a user must enter the terms that should be used. Once the user has entered their keywords, Appendix B's stream tweets method receives them. The user enters keywords through the user interface in Figure 5.13.

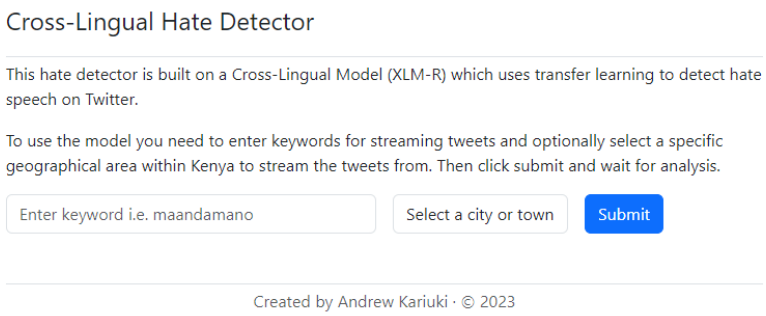


Figure 5.16 UI For End-users to Input Keywords And Geo-location

Several pre-processing procedures must be completed on the streamed tweets, including the removal of hashtags, mentions, URLs, and non-UTF8 characters. The model is then fed with the pre-processed tweets for classification.

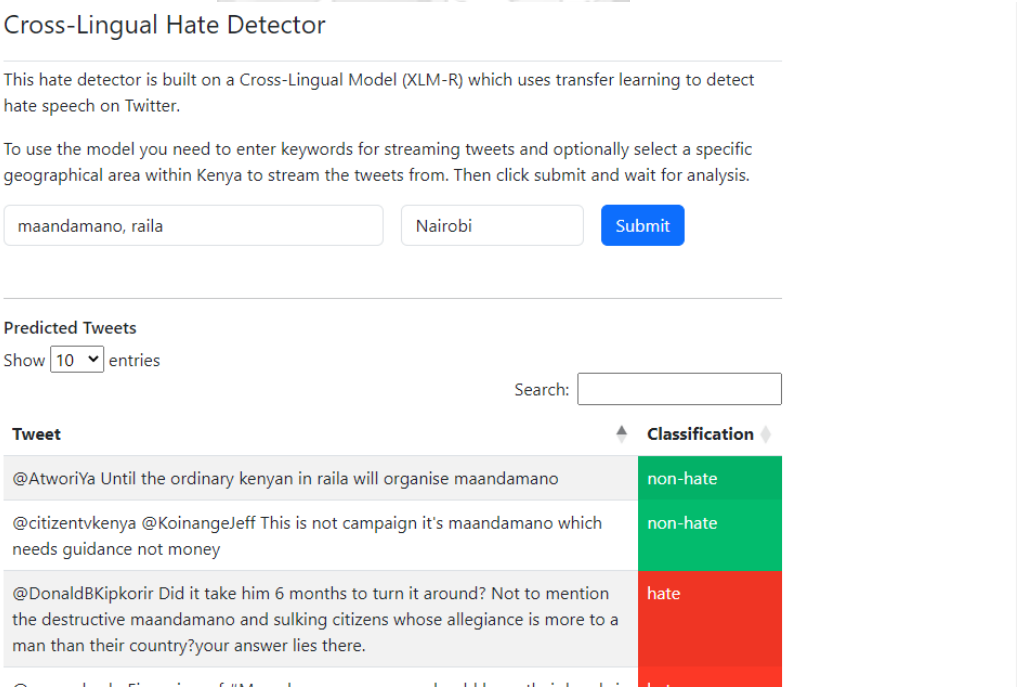


Figure 5.17 Output of Classified Tweets To End-users

5.12 Experiments with other algorithms

This experiment aimed to evaluate the efficacy of the researcher's suggested cross-lingual model with that of regularly used text categorization such as NB, SVM, LR, and an Ensemble model that brings together a combination of all three techniques. These models were built on the identical train and test set using TF-IDF feature weighting.

With the premise that the features have no relationship with each other, NB models identify probabilities straight away. SVM and LR are linear classifiers that identify classifications using an array of feature ratings. An ensemble classifier predicts a class label by casting a vote for the class label which reflects most of the class labels identified by every single classifier. The performance of matrices was calculated using the confusion matrix, Area Under the Curve, Receiver Operating Characteristic, and Confusion Report modules provided by the python's scikit-learn library. The functions are fed with the labels from the test data and the probabilistic predictions of each of the classifiers as shown in Figure.

Appendix B contains the code for carrying out the experiments, and the findings are summarized here.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix,classification_report,roc_curve, auc

# Make predictions on the test set
ensemble_test_predictions = ensemble_model.predict(X_tfidf_test)
# Make predictions on the validation set
ensemble_valid_predictions = ensemble_model.predict(X_tfidf_valid)

cm = confusion_matrix(y_test, ensemble_test_predictions)
# Plot the confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Ensemble Model')
plt.show()

# Get the classification report
report = classification_report(y_test, ensemble_test_predictions)
# Print the report
print(report)

# Make predictions on the testing data
y_pred = ensemble_model.predict_proba(X_tfidf_test)[:, 1]
# Get the false positive rate, true positive rate, and threshold values
fpr, tpr, thresholds = roc_curve(y_test, ensemble_test_predictions)
# Calculate the AUC score
roc_auc = auc(fpr, tpr)
# Plot the ROC curve
plt.plot(fpr, tpr, lw=1, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=1, color='gray', label='Random guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Ensemble Classifier')
plt.legend()
plt.show()
```

Figure 5.18 Calculating of Experiment Matrices

Table 5.3 below shows the confusion report for the SVM Classifier.

Table 5.3 Report from The Support Vector Machine Experiment

	Precision	Recall	F-1 Score	Support
0	0.74	0.79	0.77	189
1	0.80	0.74	0.77	207
Weighted Average	0.77	0.77	0.77	396

Figure 5.15 shows the ROC curve for the experiments on the SVM Classifier with an accuracy of 77%.

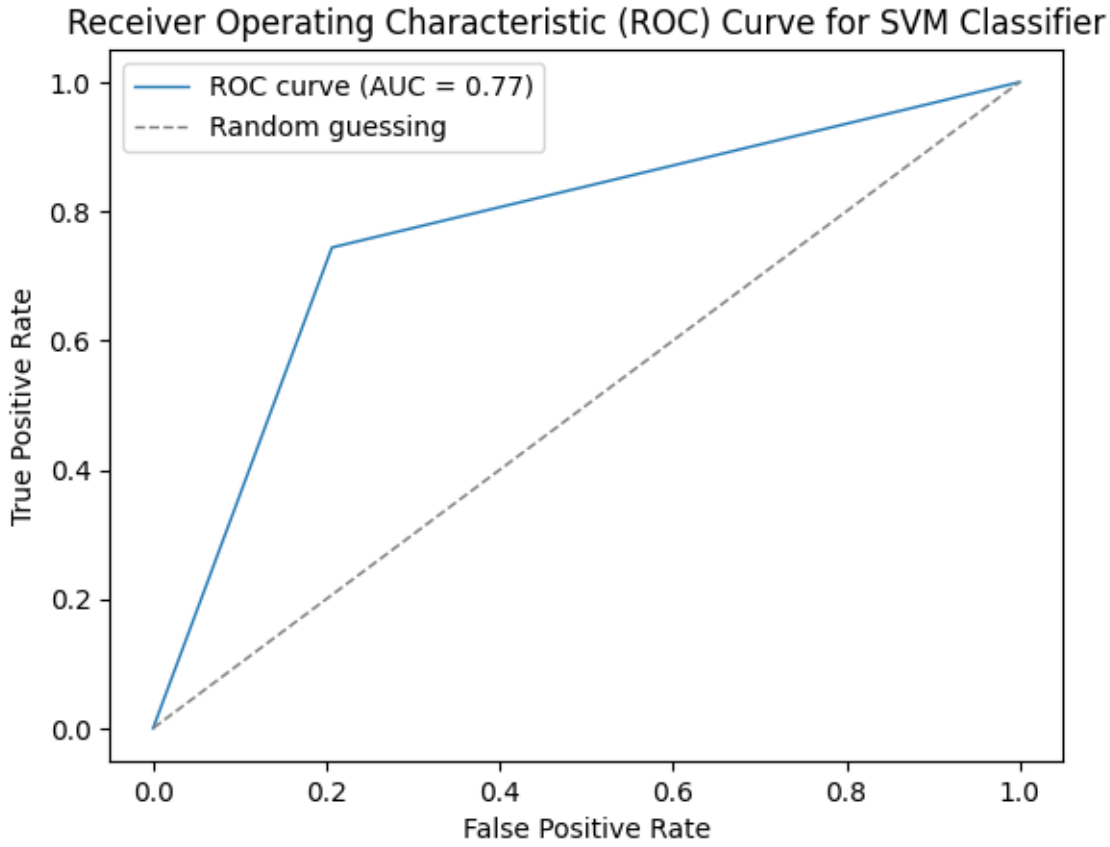


Figure 5.19 ROC Curve from The SVM Experiment

Table 5.4 below shows the confusion report for the Nave Bayes Classifier.

Table 5.4 Report from The Naive Bayes Experiment

	Precision	Recall	F-1 Score	Support
0	0.82	0.73	0.77	189
1	0.78	0.85	0.81	207
Weighted Average	0.80	0.79	0.79	396

Figure 5.16 shows the ROC curve for experiments on the Nave Bayes Classifier with an accuracy of 79%.

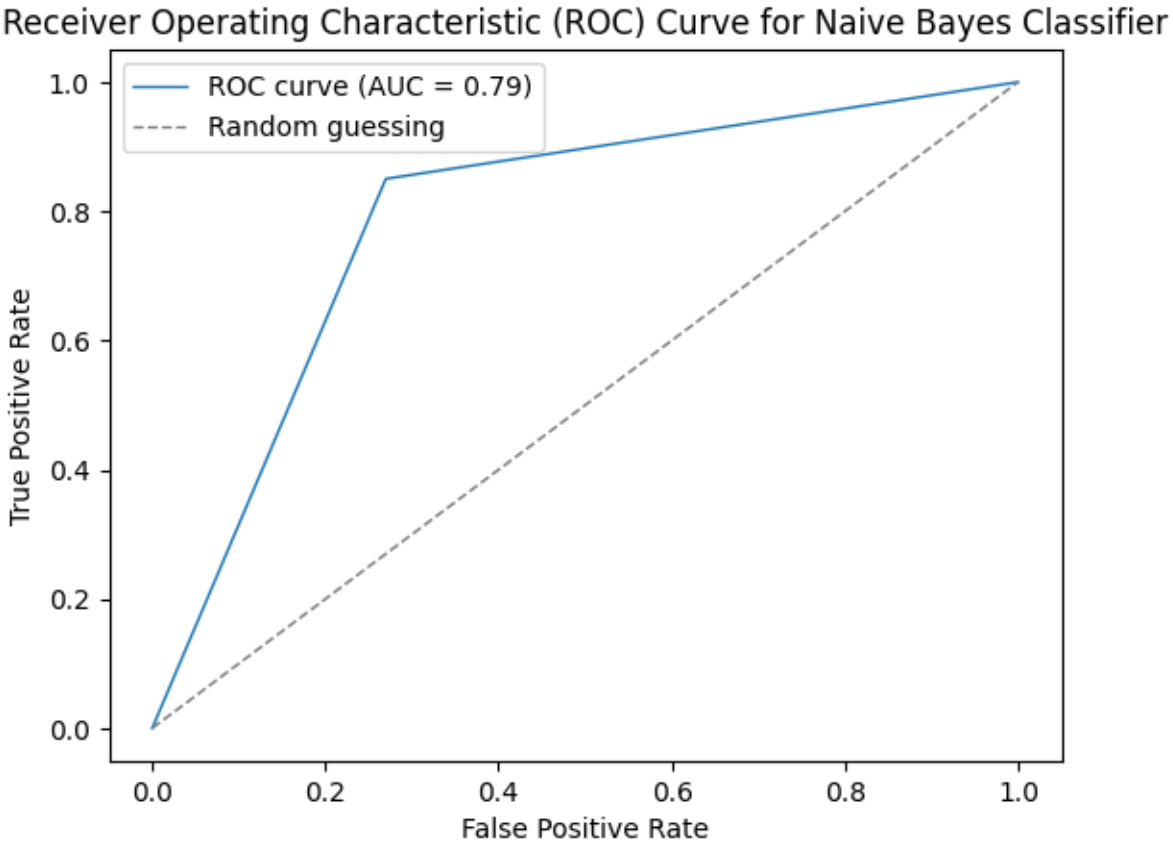


Figure 5.20 ROC Curve from The Naive Bayes Experiment

Table 5.5 below shows the confusion report for the Logistic Regression Classifier

Table 5.5 Performance Report from The Logistic Regression Experiment

	Precision	Recall	F-1 Score	Support
0	0.74	0.76	0.75	189
1	0.78	0.76	0.77	207
Weighted Average	0.76	0.76	0.76	396

Figure 5.17 shows the ROC curve for experiments on the Logistic Regression Classifier with an accuracy of 76%.

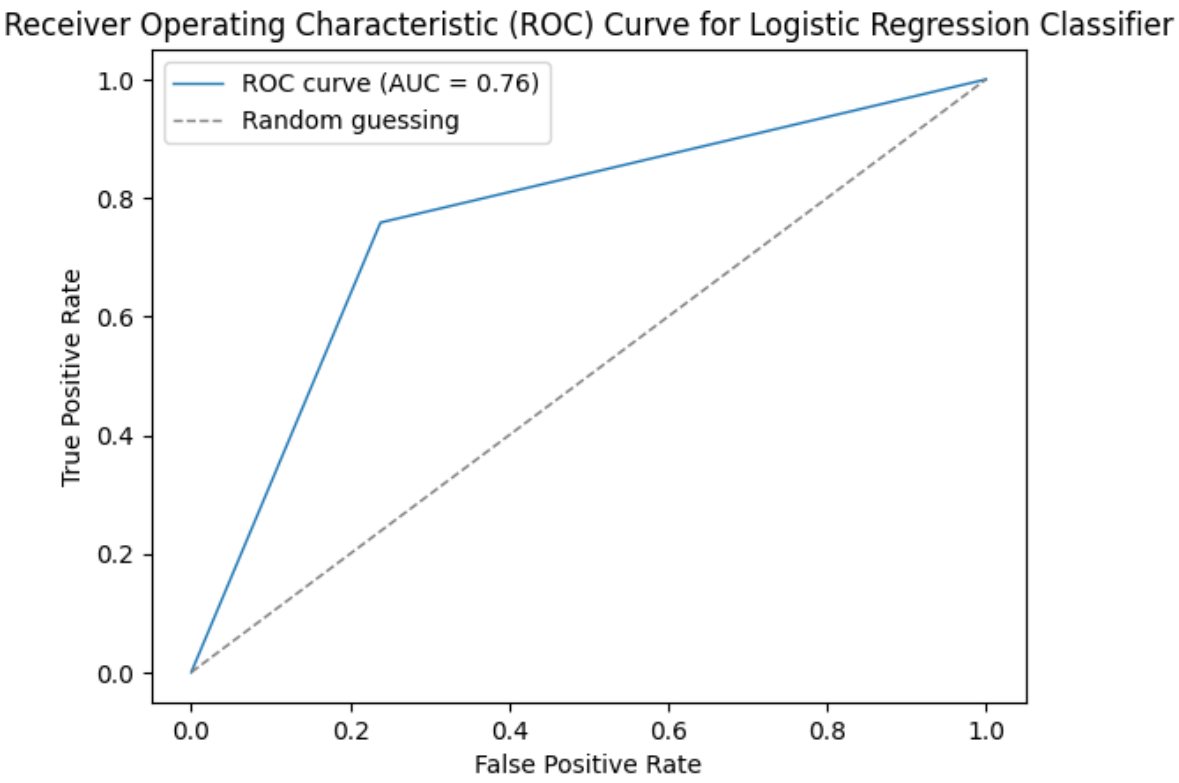


Figure 5.21 ROC Curve from The Logistic Regression Experiment

The Table 5.6 below shows the confusion report for the Ensemble Classifier.

Table 5.6 Performance Report from The Ensemble Experiment

	Precision	Recall	F-1 Score	Support
0	0.74	0.79	0.77	189
1	0.80	0.74	0.77	207
Weighted Average	0.77	0.77	0.77	396

Figure 5.18 shows the ROC curve for experiments on the Ensemble Classifier with an accuracy of 78%.

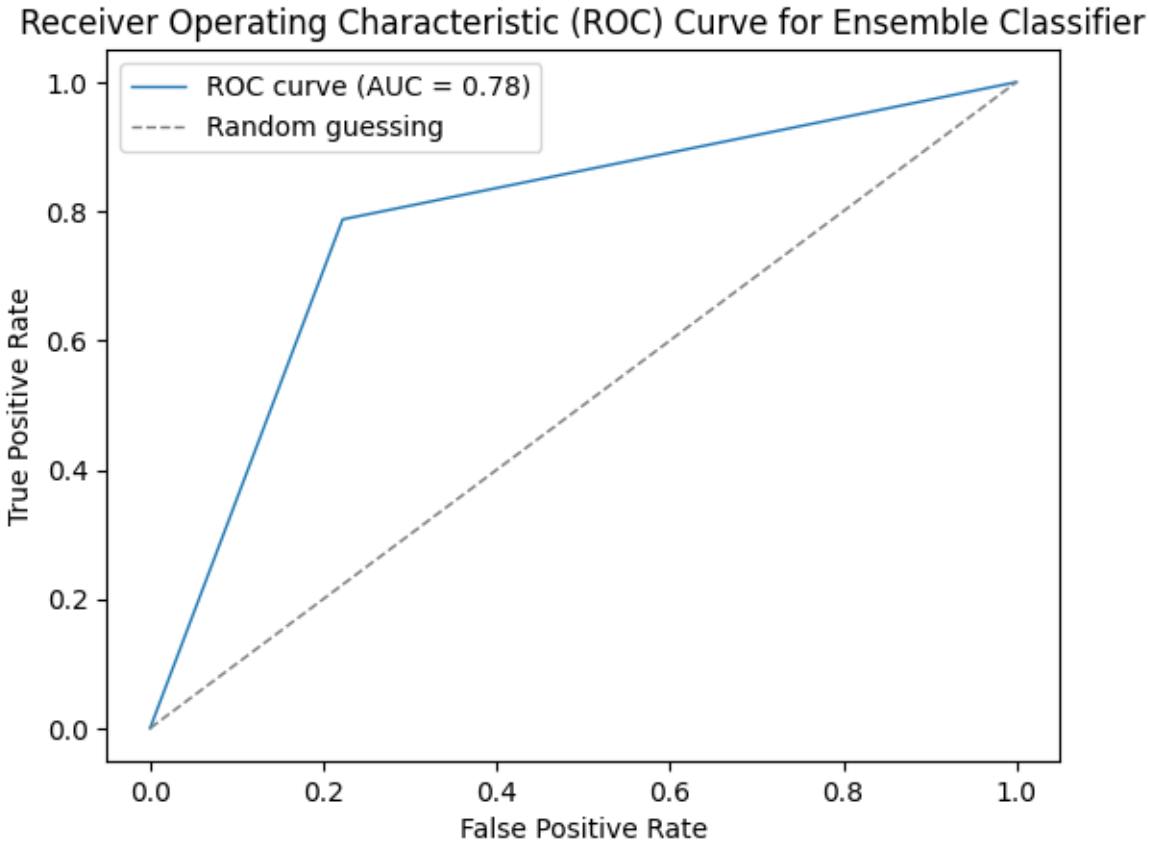


Figure 5.22 ROC Curve from The Ensemble Classifier Experiment

Chapter Six: Discussions

6.1 Introduction

This chapter includes observations focused on the research's objectives, approach, and findings. A summary of the research's main constraints opens the section with an outlook according to its results.

6.2 Study Limitations

This research has several restrictions and limitations, just like most machine learning research studies. The current research paper's empirical findings highlight a portion of constraints that can aid in advancing the study. The primary objective of this research was to create a cross-lingual model leveraging data that mainly was gathered throughout the most contentious times in Kenyan history, which included 2013, 2017, and 2022 national elections, which were held in both Swahili and Swahili-English codeswitched languages. Regardless of its success in classifying this dataset, the research omitted incidents and times in Kenyan history that had a likelihood to be hate-driven. Furthermore, the study neglected other types of hatred such as cyberbullying, gender hatred, and many others. This study solely looked at text data, leaving out other types of human interaction like emoticons and speech. Future studies ought to gravitate towards categorizing hate in these communication channels.

The research excluded other more widely used social networks such as Instagram, Facebook, TikTok, and YouTube. This constraint was caused by data access permissions, as most of these platforms restrict the type of data that an individual can view, even if it is for research purposes. Twitter is the easiest medium to scrape data from because all communications and applications are open to the public unless otherwise stated by the user. Twitter also offers a developer API that permits data searching with no legal or copyright considerations. Even though this study is limited to Twitter, the model produced can be used to categorize hate on datasets from other social media platforms.

Another obstacle to the applicability of these results was the absence of freely accessible Swahili datasets for evaluation. However, the research used automated lexicon classifiers like VADER to classify the training dataset providing a starting point for analysis. Future research will benefit greatly from this kind of research as the model is already trained in Swahili datasets

and they can leverage the capabilities of transfer learning. Yet further research ought to explore alternative methods for obtaining labelled datasets to improve the inference of the model.

6.3 Discussions

The study's overarching goal is to create a cross-lingual model for categorizing hate in social media content by way of a process known as transfer learning. Five objectives have been drawn from the core goal to fulfil it. Recognizing current techniques to detect hateful remarks, appreciating the drawbacks of these approaches, learning how cross-linguistic models can be utilized to enhance hate identification, developing a cross-lingual textual classifier for hate recognition, and training and evaluating the classifier were all part of this. The strategy for achieving these goals and their results is explained in the subsection below.

6.3.1 Identifying Existing Approaches to Detect Hate

The goal of this research was to investigate some of the approaches used to identify internet hate. Its goal was to answer the question, "What methods are used to detect hate speech on social media?"

In the process of hate speech identification, three ways are used: manual approach, keyword approach, and automated (machine learning) approach. The most basic approach to hate speech detection is the manual approach. This approach involves human intervention where the human reads a text or listens to a part of speech and uses his or her judgement to annotate that sentiment as hate or non-hate. A keyword-based approach is another fundamental method for identifying inflammatory language. Text containing potentially hostile keywords is recognized using an ontology or dictionary. HateBase, for example, keeps a collection of pejorative names for various groups in ninety-five languages. Such predetermined phrases can be used to relate words to words and be utilized in matching phrases to determine whether or not they are hate speech. This technique is quick and simple to grasp. It nevertheless has significant limits. This approach suffers low recall and high precision in that a system that is primarily based on phrases fails to identify hateful text that does not include these terms. This method cannot recognize hate speech that lacks hostile terms, and the classifier cannot deduce a word's context, as in slang. In a normal context, slang expressions like *"Hatupangwingwi"* figuratively suggest we cannot be controlled; yet, in a political setting, it signifies something quite different.

Today, the most widely used approach is automation where machine learning techniques are applied. Machine learning algorithms use tagged text data to create a classifier that can identify

hateful language according to classifications provided by content annotators. Some of the frequently utilised techniques for text classification are Logistic Regression, Support Vector Machine, and Naive Bayes. With the premise that attributes have no relationship with one another, naive Bayes algorithms label possibilities explicitly. Support Vector Machines and Logistic Regression are two linear text classification algorithms that identify classes using an array of feature counts.

6.3.2 Shortcomings of Automated Hate Detection

The goal of this study was to analyse the difficulties encountered in automating the method of identifying hateful speech on the web and, as a result, to respond to the question: What are the flaws of present techniques to automated toxic language detection?

Challenges of automatically recognizing unacceptable language, especially online, has multiple dimensions. A few of these challenges are specifically linked to the drawbacks of search term techniques. For instance, phrases can be obscured in a variety of forms, either intentionally to circumvent automated text monitoring or because of the usage of social media for interaction (Nobata et al., 2016). For example, in certain entries, characters are replaced by digits that look identical, for instance substituting the initial E with the value 3. One suggestion for reducing prejudice is to actively prepare annotators on it (Sap et al., 2019).

Another challenge comes with the scarcity and unreliability of annotation-rich datasets. An aspect contributing to this issue is the lack of a broadly acknowledged meaning for hate speech, particularly one which is constructive (Alkiviadou, 2018). This difficulty stems from cultural differences between different parts of the world; one term that is deemed hate in one language may have a different connotation in another. Furthermore, human speech is changing because of codeswitching, sarcasm, and context, and most annotations are subjective to the human perspective of whatever is transmitted or their understanding of the text itself.

A further problem to think about is unbalanced data. Although disseminating hostile and obscene content is a big issue on social platforms, it is valid that this sort of content accounts for just a tiny portion of all online communications. Much of that mismatch can also be seen in abusive language corpora (Zhang and Luo, 2019). For instance, in the HASOC 2019 training dataset, fewer than forty per cent of tweets were classified as abusive or derogatory (Mandl et al., 2019). To address this difficulty, researchers must use established methodologies.

6.3.3 Use of Cross-Lingual Models to Improve Hate Classification

The goal of this research was to provide a detailed review of how cross-lingual models can improve the entire hate classification process. This objective addressed the subject of how cross-lingual content filtering and transfer learning can be utilized to improve the efficacy of detecting hostile content.

Zampieri et al. (2020) employed the transformers architecture in their paper to detect hate speech using a cross-lingual transfer learning approach. They trained the model in English first, then used the learned weights to train and predict in another language. They concluded that cross-lingual models outperformed other models such as SVMs, RNNs, and CNNs for hate speech identification.

Cross-lingual models base on models like BERT and mBERT, for example, are pre-trained on a vast corpus of textual data and can be fine-tuned for tasks with a minimal quantity of labelled data. As a result, the model can be employed for an extensive variety of tasks related to natural language processing while requiring little domain-specific training data. Researchers can load the models, tweak their output layers, do a little fine-tuning, and apply it in our custom classifier because cross-lingual models rely on the principle of transfer learning leveraging pre-training. This gives researchers greater performance and outcomes than training a smaller model from scratch.

6.3.4 Developing a Cross-Lingual Model for Textual Hate Classification

The goal of this study was to create a cross-lingual classification algorithm that could distinguish sophisticated types of hate social media content delivered in Swahili which is a low-resource language. The major goal was to see how a cross-lingual model might be utilized to precisely identify hate speech when compared to traditional hate identification algorithms.

Collecting hate speech data was the initial stage in constructing the cross-lingual model. We used the Twitter crawler snscrape to grab tweets from the Twitter API programmatically. Twitter data from earlier studies on automatic hateful speech classification were utilized to create a baseline for a corpus for generating the cross-lingual model. Because Twitter marks each tweet with geolocation and hashtags, this data was useful in categorizing content into geographical regions and/or sensitive themes of interest.

The next step was data cleaning. All other parts of a tweet, such as hashtags, links, and account mentions, were eliminated for data cleansing. The reasoning was based on the remaining

variables rarely providing much to the details required for evaluating a tweet. The next step was to annotate the corpora to provide train, test, and validation data for the model. Due to time constraints, the researchers obtained annotated Swahili data from sources such as the Jinamizi GitHub repository and Makerere University's translated Swahili data. The VADER lexical classifier also annotated a fraction of the collected tweets. The annotation procedure resulted in approximately 12k annotated tweets, 40% of which contained hate speech and the remainder was not.

For the model, the researcher opted for a mBERT pretrained model or rather the bert-base-multilingual-uncased model. The model was loaded from the PyTorch's model library through the transformers pipeline consequently the BertTokenizer was also downloaded to encode the text for the model's consumption. The model was fine-tuned using PyTorch's neutral networks module for better performance. The model was then fed several hyperparameters such as a batch size of 32, a learning rate of 2e-5 and an epoch of 3. The model was then trained with the train sample and validated using a validation set.

6.3.5 Validation of The Cross-lingual Model

The study's purpose sought to evaluate the efficacy of the cross-lingual classifier. The major question was whether our transfer learning could be utilized to forecast other types of hate remarks from social media platforms.

To see how well the trained classifier performed, 10% of the labelled corpora was used as a test dataset. This dataset was predicted using the model that was constructed. The confusion matrix was computed to provide the counts of true and false negative and true and false positive, as shown in Table 5.1. In addition, a classification report was prepared to obtain the model's support, precision, f-1 score, and recall values, as shown in Table 5.2.

The model's accuracy during validation was 77%, which was lower than planned. This low accuracy score could be attributed to the low quality of the training sample in terms of annotation process quality and data size. Furthermore, improving the model's fine-tuning would increase its efficacy by continuously changing its hyperparameters according to their effect on an objective, in this case, accuracy in classification.

Chapter Seven: Conclusion and Recommendation

7.1 Conclusion

The multifaceted nature of communications online especially considering the widespread codeswitching has made the detection of hate or rather classification of text very difficult. This phenomenon calls for a more advanced way of detecting hate that considers words, sentiments, sarcasm, and most of the natural language of human expressions. This project was set out to achieve this objective and provide a solution to the problem at hand.

Understanding the characteristics of hate speech, as well as its prevalence and modes of expression on social media platforms, was crucial for achieving this goal. By analysing pertinent literature, the researcher also investigated alternative machine-learning strategies for text categorization and hate speech detection. This study was motivated by the diversity of today's online content, particularly on Twitter, where an increasing number of users can converse, participate, and express their opinions in their native tongues.

The researchers uncovered the flaws in existing hateful speech detection systems on social platforms after performing an in-depth examination of relevant literature and interacting with experts. Data on hate speech was collected, labelled, and preprocessed by the researchers. The labelled data had to be separated into three sets: the training, validation, and test sets. A cross-lingual model based on the BERT model, or more specifically, the BERT multilingual pre-trained model was developed using the training set. The model was then evaluated using the test set and measure to have an accuracy of over 77 per cent.

7.2 Recommendation

In contrast to the existing practice of most governmental and non-governmental organisations, this article demonstrated that a cross-lingual model may be utilized to quickly identify hateful rhetoric on Twitter. The volume of tweets that had to be read by human monitors and the amount of time needed to detect hate speech both significantly decreased because of this work. According to the study, using a broad data collection would have led to more accurate predictions. Due to the high costs associated with the labelling, training, and prediction processes, only 5520 of the 350000 total tweets gathered were utilised. The model was trained using 3312 (or 60%) of the 5520 tweets with tags; the remaining 2208 tweets were used to test and validate the model.

7.3 Future Works

Future research could look at some of the most polarizing topics like cyberbullying, depression, brand recognition and awareness and fake news. cyberbullying, depression, brand attrition and fake news have been spreading at an alarming rate over the past few years, especially on online platforms. Furthermore, additional studies might explore the possibility of detecting hostile speech in several data types like multimedia, emojis, and emoticons by applying the theoretical framework developed for this research. This can be done by retraining this cross-lingual model with a bit of better fine-tuning using audio-visual data.

Also, future research could investigate identifying hate on not only static data but also data in motion such as radio broadcastings as they currently play a huge part in the spread of hate, especially those native language-speaking radio stations. Future researchers could also employ a graphical approach to displaying hate speech on a map of topics. This could make the detection of hate speech more interactive and robust.

References

Abayomi, A. A. (2020). Applying Space Transition Theory to Cyber Crime: A Theoretical Analysis of Revenge Pornography in the 21st Century. International Journal of Innovative Science and Research Technology, 5, 11.

Abro, S., Sarang Shaikh, Z. A., Khan, S., Mujtaba, G., & Khand, Z. H. (2020). Automatic hate speech detection using machine learning: A comparative study. Machine Learning, 10(6).

Al-Garadi, M. A., Hussain, M. R., Khan, N., Murtaza, G., Nweke, H. F., Ali, I., ... & Gani, A. (2019). Predicting cyberbullying on social media in the big data era using machine learning algorithms: a review of the literature and open challenges. IEEE Access, 7, 70701-70718.

Alkiviadou, N. (2018). The legal regulation of hate speech: The international and European frameworks. Politička misao, 55(04), 203-229.

Alsafari, S., Sadaoui, S., & Mouhoub, M. (2020). Hate and offensive speech detection on Arabic social media. Online Social Networks and Media, 19, 100096.

Aluru, S. S., Mathew, B., Saha, P., & Mukherjee, A. (2020). Deep learning models for multilingual hate speech detection. arXiv preprint arXiv:2004.06465.

Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. (2017, April). Deep learning for hate speech detection in tweets. In Proceedings of the 26th international conference on World Wide Web Companion (pp. 759-760).

Bayin, S. (2013). Essentials of Mathematical Methods in Science and Engineering.

Bell, E., Bryman, A., & Harley, B. (2022). Business research methods. Oxford university press.

Biere, S., Bhulai, S., & Analytics, M. B. (2018). Hate speech detection using natural language processing techniques. Master Business AnalyticsDepartment of Mathematics Faculty of Science.

Bodrunova, S. S., Litvinenko, A., Blekanov, I., & Nepiyushchikh, D. (2021). Constructive aggression? Multiple roles of aggressive content in political discourse on Russian YouTube. Media and Communication, 9, 181-194.

Bouazizi, M., & Ohtsuki, T. O. (2016). A pattern-based approach for sarcasm detection on Twitter. IEEE Access, 4, 5477-5488.

Briliani, A., Irawan, B., & Setianingsih, C. (2019). Hate speech detection in the Indonesian language on Instagram comment section using the K-nearest neighbour classification method. In 2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS) (pp. 98-104). IEEE.

Brillouin, L. (2013). Science and information theory. Courier Corporation. Burnap, P., & Williams, M. L. (2015). Cyber hate speech on Twitter: An application of machine classification and statistical modelling for policy and decision making. Policy & Internet, 7(2), 223-242.

Burnap, P., & Williams, M. L. (2014). Hate speech, machine classification and statistical modelling of information flow on Twitter: Interpretation and communication for policy decision making.

Chaovalitwongse, W., Chou, C. A., Liang, Z., & Wang, S. (2017). Applied optimization and data mining. Annals of Operations Research, 249(1), 1-3

Cohen-Almagor, R. (2011). Fighting hate and bigotry on the Internet. Policy & Internet, 3(3), 1-26.

Collis, J., & Hussey, R. (2014). Business research: A practical guide for undergraduate and postgraduate students.

Council of Europe. Committee of Ministers. (1997). Recommendation 97 (20) on "hate speech".

Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017, May). Automated hate speech detection and the problem of offensive language. In Proceedings of the international AAAI conference on web and social media (Vol. 11, No. 1, pp. 512-515).

De Gibert, O., Perez, N., García-Pablos, A., & Cuadros, M. (2018). Hate speech dataset from a white supremacy forum. arXiv preprint arXiv:1809.04444.

Di Capua, M., Di Nardo, E., & Petrosino, A. (2016). Unsupervised cyberbullying detection in social networks. In 2016 23rd International Conference on pattern recognition (ICPR) (pp. 432-437). IEEE.

Fatahillah, N. R., Suryati, P., & Haryawan, C. (2017). Implementation of Naive Bayes classifier algorithm on social media (Twitter) to the teaching of Indonesian hate speech. In 2017 International Conference on Sustainable Information Engineering and Technology (SIET) (pp. 128-131). IEEE.

Fauzi, M. A., & Yuniarti, A. (2018). Ensemble method for Indonesian Twitter hate speech detection. Indonesian Journal of Electrical Engineering and Computer Science, 11(1), 294-299.

Founta, A. M., Djouvas, C., Chatzakou, D., Leontiadis, I., Blackburn, J., Stringhini, G., ... & Kourtellis, N. (2018). Large-scale crowdsourcing and characterization of Twitter abusive behaviour. In Twelfth International AAAI Conference on Web and social media.

Gambäck, B., & Sikdar, U. K. (2017). Using convolutional neural networks to classify hate-speech. In Proceedings of the first workshop on abusive language online (pp. 85-90).

Gaydhani, A., Doma, V., Kendre, S., & Bhagwat, L. (2018). Detecting hate speech and offensive language on Twitter using machine learning: An n-gram and tfidf based approach. arXiv preprint arXiv:1809.08651.

Gitari, N. D., Zuping, Z., Damien, H., & Long, J. (2015). A lexicon-based approach for hate speech detection. International Journal of Multimedia and Ubiquitous Engineering, 10(4), 215-230.

yaGoddard, W., & Melville, S. (2004). Research methodology: An introduction. Juta and Company Ltd.

Henry, F., & Tator, C. (2000). Racist discourse in Canada's English print media. Toronto: Canadian Race Relations Foundation.

Hutto, C., & Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In Proceedings of the International AAAI Conference on Web and social media (Vol. 8, No. 1, pp. 216-225).

Ibrohim, M. O., & Budi, I. (2019). Multi-label hate speech and abusive language detection in Indonesian Twitter. In Proceedings of the Third Workshop on Abusive Language Online (pp. 46-57).

ICAR-ATARI III, Z. (2018). How to Increase Utilization and Dissemination of Evaluation Research Findings. Int. J. Curr. Microbiol. App. Sci, 7(9), 2484-2489.

Idris, D. O., Akinola, E. O., & Olalekan, S. (2020). Detecting Hate Speech on social media Using Deep Learning Techniques.

Jaki, S., & De Smedt, T. (2019). Right-wing German hate speech on Twitter: Analysis and automatic detection. arXiv preprint arXiv:1910.07518.

Jiang, A., & Zubiaga, A. (2021, August). Cross-lingual Capsule Network for Hate Speech Detection in Social Media. In Proceedings of the 32nd ACM Conference on Hypertext and Social Media (pp. 217-223).

Kothari, C. R. (2004). Research methodology.

Kshirsagar, R., Cukuvac, T., McKeown, K., & McGregor, S. (2018). Predictive embeddings for hate speech detection on Twitter. arXiv preprint arXiv:1809.10644.

Kwok, I., & Wang, Y. (2013, June). Locate the hate: Detecting tweets against blacks. In the Twenty-seventh AAAI conference on artificial intelligence.

Liu, P., Li, W., & Zou, L. (2019). NULI at SemEval-2019 Task 6: Transfer Learning for Offensive Language Detection using Bidirectional Transformers. In SemEval@ NAACL-HLT (pp. 87-91).

Liu, Z., Lin, Y., Sun, M., Liu, Z., Lin, Y., & Sun, M. (2020). Document Representation. Representation Learning for Natural Language Processing, 91-123.

MacAvaney, S., Yao, H. R., Yang, E., Russell, K., Goharian, N., & Frieder, O. (2019). Hate speech detection: Challenges and solutions. PloS one, 14(8), e0221152. https://doi.org/10.1371/journal.pone.0221152.

Malmasi, S., & Zampieri, M. (2017). Detecting hate speech in social media. arXiv preprint arXiv:1712.06427.

Manning, C. D. (2008). Introduction to information retrieval. Syngress Publishing.

Mohiyaddeen, M., & Siddiqi, S. (2021). Automatic hate speech detection: A literature review. Available at SSRN 3887383.

Mossie, Z., & Wang, J. H. (2018). Social network hate speech detection for the Amharic language. Computer Science & Information Technology, 41-55.

Mozafari, M., Farahbakhsh, R., & Crespi, N. (2019, December). A BERT-based transfer learning approach for hate speech detection in online social media. In International Conference on Complex Networks and Their Applications (pp. 928-940). Springer, Cham.

Mugambi, S. K. (2017). Sentiment Analysis for Hate Speech Detection on social media: Tf-Idf Weighted N-Grams Based Approach (Doctoral dissertation, Strathmore University).

Mullah, N. S., & Zainon, W. M. N. W. (2021). Advances in Machine Learning Algorithms for Hate Speech Detection in social media: A Review. IEEE Access.

Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y., & Chang, Y. (2016, April). Abusive language detection in online user content. In Proceedings of the 25th international conference on world wide web (pp. 145-153).

Olteanu, A., Talamadupula, K., & Varshney, K. R. (2017, June). The limits of abstract evaluation metrics: The case of hate speech detection. In Proceedings of the 2017 ACM on web science conference (pp. 405-406).

Omondi Owino, S. (2013). The language factor in the search for national cohesion and integration in Kenya. Les Cahiers d'Afrique de l'Est/The East African Review, (47), 57-69.

Oriola, O., & Kotzé, E. (2020). Evaluating machine learning techniques for detecting offensive and hate speech in South African tweets. IEEE Access, 8, 21496-21509.

Orawit, T. (2006). Rapid Application Development.

Pamungkas, E. W., & Patti, V. (2019). Cross-domain and cross-lingual abusive language detection: A hybrid approach with deep learning and a multilingual lexicon. In Proceedings of the 57th annual meeting of the Association for computational linguistics: Student research workshop (pp. 363-370).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of Machine Learning Research, 12, 2825-2830.

Parikh, A., Desai, H., & Bisht, A. S. (2019). DA Master at HASOC 2019: Identification of Hate Speech using Machine Learning and Deep Learning Approaches for social media Posts. In FIRE (Working Notes) (pp. 315-319).

Pitsilis, G. K., Ramampiaro, H., & Langseth, H. (2018). Detecting offensive language in tweets using deep learning. arXiv preprint arXiv:1801.04433.

Rao, S. S. (2019). Engineering optimization: theory and practice. John Wiley & Sons.

Ribeiro, M. H., Calais, P. H., Santos, Y. A., Almeida, V. A., & Meira Jr, W. (2018). Characterizing and detecting hateful users on Twitter. At the Twelfth international AAAI conference on web and social media.

Ring, C. E. (2013). Hate speech in social media: An exploration of the problem and its proposed solutions (Doctoral dissertation, University of Colorado at Boulder).

Roberts, M. J. (2009). Conflict analysis of the 2007 post-election violence in Kenya. Managing conflicts in Africa's democratic transitions, 141-155.

Ross, B., Rist, M., Carbonell, G., Cabrera, B., Kurowsky, N., & Wojatzki, M. (2017). Measuring the reliability of hate speech annotations: The case of the European refugee crisis. arXiv preprint arXiv:1701.08118.

Saleem, H. M., Dillon, K. P., Benesch, S., & Ruths, D. (2017). A web of hate: Tackling hateful speech in online social spaces. arXiv preprint arXiv:1709.10159.

Sambuli, N., Morara, F., & Mahihu, C. (2013). Umati: Monitoring online dangerous speech.

Sap, M., Card, D., Gabriel, S., Choi, Y., & Smith, N. A. (2019, July). The risk of racial bias in hate speech detection. In Proceedings of the 57th annual meeting of the association for computational linguistics (pp. 1668-1678).

Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. Communications of the ACM, 18(11), 613-620.

Schmidt, A., & Wiegand, M. (2017, April). A survey on hate speech detection using natural language processing. In Proceedings of the fifth international workshop on natural language processing for social media (pp. 1-10).

Sellars, A. (2016). Defining hate speech. Berkman Klein Centre Research Publication, (2016-20), 16-48.

Sharief, E. H., Hawasara, W., & Sinaulan, R. L. (2021, December). Hate Speech through social media in Indonesia: Based on Space Transition Theory in Cyber Criminology. In Proceeding (pp. 337-345).

Sharpe, R. (2022). BRIEFING July 2022 FACEBOOK UNABLE TO DETECT HATE.

Siino, M., Di Nuovo, E., Tinnirello, I., & La Cascia, M. (2021). Detection of hate speech spreaders using convolutional neural networks. In CLEF (Working Notes) (pp. 2126-2136).

Sinnar, S. (2020). The Capitol Invasion and The Framing of Political Violence. Journal of National Security Law & Policy, 11, 1-3.

Slonje, R., Smith, P. K., & Frisén, A. (2013). The nature of cyberbullying, and strategies for prevention. Computers in human behaviour, 29(1), 26-32.

Sood, S., Antin, J., & Churchill, E. (2012, May). Profanity uses in online communities. In Proceedings of the SIGCHI conference on human factors in computing systems (pp. 1481-1490).

Sorzano, C. O. S., Vargas, J., & Montano, A. P. (2014). A survey of dimensionality reduction techniques. arXiv preprint arXiv:1403.2877.

Spertus, E. (1997). Smokey: Automatic recognition of hostile messages. In Aaai/iaai (pp. 1058-1065).

Stappen, L., Brunn, F., & Schuller, B. (2020). Cross-Lingual Zero-and Few-Shot Hate Speech Detection Utilising Frozen Transformer Language Models and AXEL.

Stone, J. V. (2015). Information Theory: A Tutorial Introduction.

Tulkens, F. (2013). The Hate Factor in Political Speech. Where Do Responsibilities Lie? In Report of the Council of Europe Conference.

Van Huynh, T., Nguyen, V. D., Van Nguyen, K., Nguyen, N. L. T., & Nguyen, A. G. T. (2019). Hate speech detection on Vietnamese social media text using the bi-gru-lstm-cnn model. arXiv preprint arXiv:1911.03644.

Varshney, A. (2009). Ethnicity and ethnic conflict.

Vidgen, B., & Yasseri, T. (2020). Detecting weak and strong Islamophobic hate speech on social media. Journal of Information Technology & Politics, 17(1), 66-78.

Vijayarani, S., Ilamathi, M. J., & Nithya, M. (2015). Preprocessing techniques for text mining-an overview. International Journal of Computer Science & Communication Networks, 5(1), 7-16.

Waseem, Z., & Hovy, D. (2016). Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter. In Proceedings of the NAACL Student Research Workshop (pp. 88-93).

Watanabe, H., Bouazizi, M., & Ohtsuki, T. (2018). Hate Speech on Twitter: A Pragmatic Approach to Collect Hateful and Offensive Expressions and Perform Hate Speech Detection. IEEE Access, 6, 13825-13835.

Xu, T., Guo, Z., Liu, S., He, X., Meng, Y., Xu, Z., ... & Song, L. (2018). Evaluating different machine learning methods for upscaling evapotranspiration from flux towers to the regional scale. Journal of Geophysical Research: Atmospheres, 123(16), 8674-8690.

Xue, W., & Xu, X. (2010). Three new feature weighting methods for text categorization. In Web Information Systems and Mining: International Conference, WISM 2010, Sanya, China, October 23-24, 2010. Proceedings (pp. 352-359). Springer Berlin Heidelberg.

Yu, L. (2020). Ethical considerations in case studies. In 2020 1st Workshop on Ethics in Requirements Engineering Research and Practice (REthics) (pp. 15-21). IEEE.

Zebari, R., Abdulazeez, A., Zeebaree, D., Zebari, D., & Saeed, J. (2020). A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. Journal of Applied Science and Technology Trends, 1(2), 56-70.

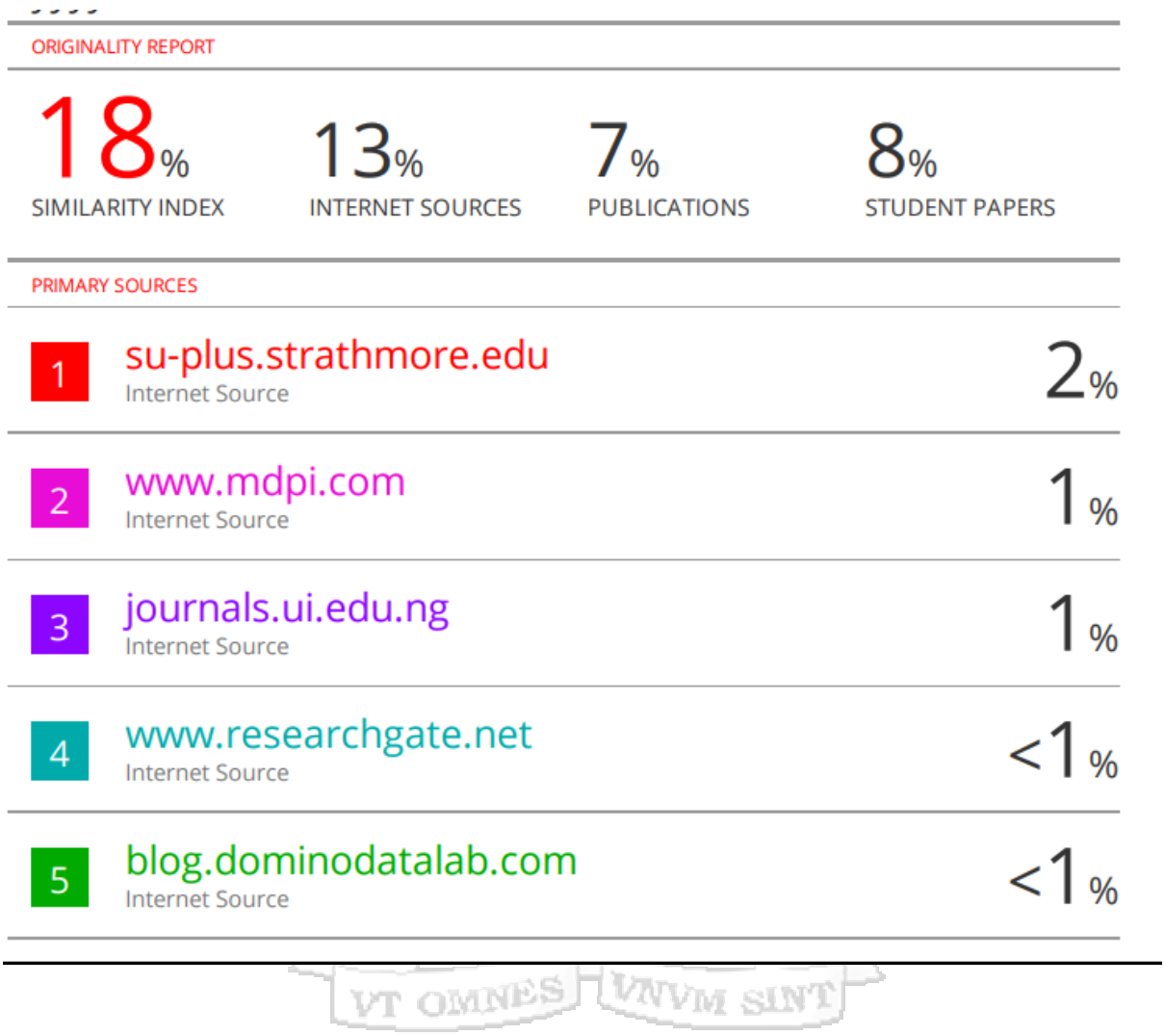
Zhang, Z., & Luo, L. (2019). Hate speech detection: A solved problem? the challenging case of long tail on twitter. Semantic Web, 10(5), 925-945.

Zhang, Z., Robinson, D., & Tepper, J. (2018, June). Detecting hate speech on Twitter using a convolution-gru-based deep neural network. In the European semantic web conference (pp. 745-760). Springer, Cham.

Zhao, R., & Mao, K. (2016). Cyberbullying detection based on semantic-enhanced marginalized denoising auto-encoder. IEEE Transactions on Affective Computing, 8(3), 328-339.

Appendices

Appendix A: Similarity Report



ORIGINALITY REPORT

18%	13%	7%	8%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	su-plus.strathmore.edu Internet Source	2%
2	www.mdpi.com Internet Source	1%
3	journals.ui.edu.ng Internet Source	1%
4	www.researchgate.net Internet Source	<1%
5	blog.dominodatalab.com Internet Source	<1%

Appendix B: Ethical Clearance Confirmation

Strathmore UNIVERSITY

22nd February 2023

Mr Kariuki Andrew Oduor,
andrew.kariuki@strathmore.edu

Dear Mr Kariuki,

RE: Cross-Lingual Model for Hate Speech Detection on Twitter: A Case of Swahili and Swahili-English Slang

This is to inform you that SU-ISERC has reviewed and **approved** your above **SU- master's** research proposal. Your application reference number is **SU-ISERC1583/23.** The approval period is from **22nd February 2023 to 21st February 2024.**

This approval is subject to compliance with the following requirements:

i. Only approved documents including (informed consents, study instruments, and MTA) will be used
ii. All changes including (amendments, deviations, and violations) are submitted for review and approval by SU-ISERC.
iii. Death and life-threatening problems and serious adverse events or unexpected adverse events whether related or unrelated to the study must be reported to SU-ISERC within 48 hours of notification
iv. Any changes, anticipated or otherwise, that may increase the risks or affect the safety or welfare of study participants and others or affect the integrity of the research must be reported to SU-ISERC within 48 hours
v. Clearance for the export of biological specimens must be obtained from relevant institutions.
vi. Submission of a request for renewal of approval at least 60 days prior to the expiry of the approval period. Attach a comprehensive progress report to support the renewal.
vii. Submission of an executive summary report within 90 days of completion of the study to SU-ISERC.

Before commencing your study, you will be expected to obtain a research license from National Commission for Science, Technology, and Innovation (NACOSTI) https://research-portal.nacosti.go.ke/ and obtain other clearances needed.

Yours sincerely,

for: **Dr Ben Ngoye,**
Secretary; SU-ISERC

Cc: Mr Ambrose Rachier,
Chairperson; SU-ISERC

STRATHMORE UNIVERSITY INSTITUTIONAL SCIENTIFIC AND ETHICAL REVIEW COMMITTEE
(SU-ISERC)
22-Feb-2023
Email:ethicsreview@strathmore.edu
P.O BOX 59857-00200
NAIROBI-KENYA

Appendix C: System Implementation Code

Dataset Collection Function

This function was used to collect data or corpora used for training as testing the proposed model.

```
import csv

import snscrape.modules.twitter as sntwitter

from SwaRoberta.corpora.model import GeoFence, TweetsDateRange, TwitterUsername
from SwaRoberta.corpora.utils import (
     get hashtags cashtags,
     get reply user fullname,
     get reply username,
     get retweeted quoted tweet,
     get retweeted quoted tweet id,
     get tweet place,
)

 file = "data/corpora.csv"
def corpora(
    keyword: str,
    max limit: int = 10000,
    outfile: str =  file,
    location: dict = None,
    dates: dict = None,
    username: dict = None,
    retweets: bool = False,
    language: str = None,
):
    """
    Function to:
        1. collect for tweets corpora with a given keyword, location, dates.
        2. write to a csv file
    """
    query = f"{keyword}"

    if username:
        query = f"{TwitterUsername(**username).get_username()}"
    if location:
        query = f"{query} {GeoFence(**location).get_geo_fence()}"
    if dates:
        query = f"{query} {TweetsDateRange(**dates).get_date_range()}"
    else:
        query = f"{query} since:2018-12-01"
    if retweets:
        query = f"{query} exclude:retweets"
    if language:
        query = f"{query} lang:{language}"

    with open(outfile, "a", encoding="utf-8") as csv_file:
        writer = csv.writer(csv_file)
        writer.writerow(
            [
```

```
        "id",
        "tweet_date",
        "tweet_text",
        "tweet_username",
        "tweet_user_display_name",
        "tweet_user_is_verified",
        "tweet_user_location",
        "tweet_user_followers_count",
        "tweet_user_friends_count",
        "tweet_user_favourites_count",
        "tweet_user_listed_count",
        "tweet_replies_Count",
        "tweet_retweet_count",
        "tweet_likes_count",
        "tweet_quote_count",
        "tweet_language",
        "tweet_hashtags",
        "tweet_cashtags",
        "tweet_view_count",
        "tweet_place",
        "tweet_retweeted",
        "tweet_retweeted_id",
        "tweet_quoted",
        "tweet_quoted_id",
        "tweet_in_reply_to_username",
        "tweet_in_reply_to_user_display_name",
        "tweet_url",
    ]
)

for i, tweet in enumerate(sntwitter.TwitterSearchScraper(query).get_items()):
    writer.writerow(
        [
            tweet.id,
            tweet.date.strftime("%Y-%m-%d %H:%M:%S"),
            tweet.rawContent,
            tweet.user.username,
            tweet.user.displayname,
            tweet.user.verified,
            tweet.user.location,
            tweet.user.followersCount,
            tweet.user.friendsCount,
            tweet.user.favouritesCount,
            tweet.user.listedCount,
            tweet.replyCount,
            tweet.retweetCount,
            tweet.likeCount,
            tweet.quoteCount,
            tweet.lang,
             get hashtags cashtags(tweet.hashtags),
             get hashtags cashtags(tweet.cashtags),
            tweet.viewCount,
             get tweet place(tweet.place),
             get retweeted quoted tweet(tweet.retweetedTweet),
             get retweeted quoted tweet id(tweet.retweetedTweet),
             get retweeted quoted tweet(tweet.quotedTweet),
             get retweeted quoted tweet id(tweet.quotedTweet),
            _get_reply_username(tweet.inReplyToUser),
```

```
                    _get_reply_user_fullname(tweet.inReplyToUser),
                    tweet.url,
                ]
            )
            if i >= max_limit:
                break
```

Dataset loading function.

Code to load the annotated dataset for preprocessing.

```
def load_dataset():
    dataset = pd.read_csv("labeled_tweets.csv")
    return dataset['tweet'].tolist(), dataset['class']
```

Dataset Cleaning function

This code helped in the cleaning of the tweets collected to remove unwanted characters and noise.

```
def dataset_cleaning(dataset):
    new_dataset = list()
    emoticons = [':-)', ':)', '(:', '(-:', ':))',
                 '((:', ':-D', ':D', 'X-D', 'XD',
                 'xD', 'xD', '<3', '</3', ':\*',
                 ';-)', ';)', ';-D', ';D', '(;',
                 '(-;', ':-(', ':(', '(:', '(-:',
                 ':,(', ':\'(', ':"(', ':((', ':D',
                 '=D', '=)', '(=', '=(', ')=', '=-O',
                 'O-=', ':o', 'o:', 'O:', 'O:', ':-o',
                 'o-:', ':P', ':p', ':S', ':s', ':@',
                 ':>', ':<', '^ ^', '^.^', '>.>', 'T T',
                 'T-T', '-.-', '*.*', '~.~', ':*', ':-*',
                 'xP', 'XP', 'XP', 'Xp', ':-|', ':->',
                 ':-<', '$ $', '8-)', ':-P', ':-p',
                 '=P', '=p', ':*)', '*-*', 'B-)',
                 'O.o', 'X-(', ')-X']

    for data in dataset:
        value = data.replace(".", " ").lower()
        value = re.sub(r"[^a-zA-Z?.!,¿]+", " ", value)
        users = re.findall("[@]\w+", value)
        for user in users:
            value = value.replace(user, "<user>")
        urls = re.findall(r'(https?://[^\s]+)', value)
        if len(urls) != 0:
            for url in urls:
                value = value.replace(url, "<url >")
        for emoji_ in value:
            if emoji_ in emoji.UNICODE_EMOJI:
                value = value.replace(emoji_, "<emoticon >")
        for emoji_ in emoticons:
            value = value.replace(emoji_, "<emoticon >")
        digits = re.findall('[0-9]+', value)
        for digit_ in digits:
```

```
            value = value.replace(digit_, "<number >")
        value = value.replace('#', "<hashtag >")
        value = re.sub(r"([?.!,¿])", r" ", value)
        value = "".join(l for l in value if l not in string.punctuation)
        value = re.sub(r'[" "]+', " ", value)
        new_dataset.append(value)
    return new_dataset
```

Data Preprocessing Function.

Code for tokenizing tweets to make them more digestible by the BERT model.

```
def dataset_preprocessing(dataset, labels):
    input_ids = []
    attention_masks = []
    bert_tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-uncased'")
    for sentence in dataset:
        bert_inp = bert_tokenizer.__call__(
            sentence, max_length=36,
            padding='max_length',
            pad_to_max_length=True,
            truncation=True,
            return_token_type_ids=False
            )
        input_ids.append(bert_inp['input_ids'])
        attention_masks.append(bert_inp['attention_mask'])
    input_ids = np.asarray(input_ids)
    attention_masks = np.array(attention_masks)
    labels = np.array(labels)
    return input_ids, attention_masks, labels
```

Creating input ids and attention masks

Code for getting attention masks and input ids.

```
def load_preprocess_dataset():
    dataset, labels = load dataset()
    input ids, attention masks, labels = dataset preprocessing(
        dataset cleaning(dataset),
        labels
        )
    return input_ids, attention_masks, labels
```

Function for splitting the data set.

Split the data set into 80-10-10 for train set, test set and validation set respectively.

```
input_ids, attention_masks, labels = load_preprocess_dataset()
dataset = pd.DataFrame(
    list(zip(input_ids, attention_masks)),
    columns=['input_ids', 'attention_masks']
    )

training_set, temporary_set, training_labels, temporary_labels = train_test_split(
```

```
    dataset, labels, random_state=2018,
    test_size=0.2, stratify=labels
    )
validation_set, testing_set, validation_labels, testing_labels = train_test_split(
    temporary_set, temporary_labels,
    random_state=2018, test_size=0.5, stratify=temporary_labels
    )
```

Convert datasets into PyTorch tensors.

```
training sequence = torch.tensor(training set['input ids'].tolist())
training mask = torch.tensor(training set['attention masks'].tolist())
training y = torch.tensor(training labels.tolist())

validation sequence = torch.tensor(validation set['input ids'].tolist())
validation mask = torch.tensor(validation set['attention masks'].tolist())
validation y = torch.tensor(validation labels.tolist())

testing sequence = torch.tensor(testing set['input ids'].tolist())
testing mask = torch.tensor(testing set['attention masks'].tolist())
testing y = torch.tensor(testing labels.tolist())
)
```

Define data loaders with a batch size of 32.

```
batch_size = 32
training_data = TensorDataset(training_sequence, training_mask, training_y)
training_sampler = RandomSampler(training_data)
training_dataloader = DataLoader(training_data, sampler=training_sampler, batch_size=bat
ch_size)
validation_data = TensorDataset(validation_sequence, validation_mask, validation_y)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler, batch_si
ze=batch_size)
del temporary set
gc.collect()
torch.cuda.empty cache()
```

Model definition class with neural networks fine-tuning.

This code sample shows the model class with fine-tuning, and hyper-parameters.

```
import gc
import torch
from torch import nn
from transformers import BertModel
class CROSS LINGUAL MODEL(nn.Module):
    def   init  (self):
        super(CROSS LINGUAL MODEL, self).  init  ()
        self.bert = BertModel.from pretrained('bert-base-multilingual-uncased')
        self.conv = nn.Conv2d(
            in_channels=13,
            out_channels=13,
            kernel_size=(3, 768),
            padding=True)
```

```
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(
            kernel_size=3,
            stride=1
            )
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(442, 3)
        self.flat = nn.Flatten()
        self.softmax = nn.LogSoftmax(dim=1)
    def forward(self, sent_id, mask):
        _, _, all_layers = self.bert(
            sent_id, attention_mask=mask,
            output_hidden_states=True
            )
        x = torch.transpose(torch.cat(
            tuple([t.unsqueeze(0) for t in all_layers]), 0), 0, 1)
        del all_layers
        gc.collect()
        torch.cuda.empty_cache()
        x = self.pool(self.dropout(self.relu(self.conv(self.dropout(x)))))
        x = self.fc(self.dropout(self.flat(self.dropout(x))))
        return self.softmax(x)
model = CROSS_LINGUAL_MODEL()
model = model.to(device)
from transformers import AdamW
optimizer = AdamW(model.parameters(), lr=2e-5)
```

Training function


```
def train_model():
    print("\n\nTraining Our Model...")
    model.train()
    total_losses, all_accuracies = 0, 0
    total_predictions = []
    total_data_length = len(training_dataloader)
    for i, batch in enumerate(training_dataloader):
        steps = i+1
        _percent = "{0:.2f}".format(100 * (steps / float(total_data_length)))
        losses = "{0:.2f}".format(total_losses/(total_data_length*batch_size))
        filled_length = int(100 * steps // total_data_length)
        _bar = '' * filled_length + '>'  *(filled_length < 100) + '.' * (99 - filled_le
ngth)
        _m_1 = f"\rBatch {steps}/{total_data_length} |{_bar}| {_percent}% complete"
        print(f'{_m_1}, loss={losses}, accuracy={all_accuracies}', end='')
        batch = [r.to(device) for r in batch]
        sent_id, mask, labels = batch
        del batch
        gc.collect()
        torch.cuda.empty_cache()
        model.zero_grad()
        _predictions = model(sent_id.to(device).long(), mask)
        losses = cross_entropy(_predictions, labels)
        total_losses += float(losses.item())
        losses.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        total_predictions.append(_predictions.detach().cpu().numpy())
```

```
    gc.collect()
    torch.cuda.empty_cache()
    average_losses = total_losses / (len(training_dataloader)*batch_size)
    total_predictions = np.concatenate(total_predictions, axis=0)
    return average_losses, total_predictions
```

Evaluation function

```
def evaluate_model():
    print("\n\nEvaluating Our Model...")
    model.eval()
    total_losses, all_accuracies = 0, 0
    total_predictions = []
    total_data_length = len(validation_dataloader)
    for i, batch in enumerate(validation_dataloader):
        step = i+1
        _percent = "{0:.2f}".format(100 * (step / float(total_data_length)))
        _losses = "{0:.2f}".format(total_losses/(total_data_length*batch_size))
        filled_length = int(100 * step // total_data_length)
        _bar = '' * filled_length + '>' * (filled_length < 100) + '.' * (99 - filled_le
ngth)
        _m_1 = f'\rBatch {step}/{total_data_length} |{_bar}| {_percent}% complete'
        print(f'{_m_1}, loss={_losses}, accuracy={all_accuracies}', end='')
        batch = [t.to(device) for t in batch]
        sent_id, mask, labels = batch
        del batch
        gc.collect()
        torch.cuda.empty_cache()
        with torch.no_grad():
            _predictions = model(sent_id, mask)
            losses = cross_entropy(_predictions, labels)
            total_losses += float(losses.item())
            total_predictions.append(_predictions.detach().cpu().numpy())
    gc.collect()
    torch.cuda.empty cache()
    average losses = total losses / (len(validation dataloader)*batch size)
    total predictions = np.concatenate(total predictions, axis=0)
    return average losses, total predictions
```

Code for training the model in a GPU computational environment.

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

cross_entropy = nn.NLLLoss()
best_validation_loss = float('inf')

epochs = 3
current_epoch = 1
while current_epoch <= epochs:
    print(f'\nEpoch {current_epoch} / {epochs}:')
    training_loss, _ = train()
    validation_loss, _ = evaluate()
    if validation_loss < best_validation_loss:
        best_validation_loss = validation_loss
    print(f'\n\nTraining Loss: {training_loss:.3f}')
```

```
    print(f'Validation Loss: {validation_loss:.3f}')
    current_epoch = current_epoch + 1
```

Performance matrices.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
                            confusion_matrix,classification_report,roc_curve, auc
gc.collect()
torch.cuda.empty_cache()
with torch.no_grad():
    predictions = model(testing_sequence.to(device), testing_mask.to(device))
    predictions = predictions.detach().cpu().numpy()

print("Model's Performance:")
predictions = np.argmax(predictions, axis=1)

print("Model's Classification Report")
print(classification_report(testing_y, predictions))

print("Model's Accuracy: " + str(accuracy_score(testing_y, predictions)))
```

Experiments with other text classifiers.

The code below shows the experiments done with Naïve Bayes, SVM, Logistic Regression and Ensemble Classifiers. The models were fitted with TF-IDF feature extractor.

```
import pandas as pd
import numpy as np
import seaborn as sns
import emoji as emoji
import re
import string
import matplotlib.pyplot as plt
from sklearn.feature extraction.text import TfidfVectorizer
from sklearn.naive bayes import MultinomialNB
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, con
fusion_matrix,classification_report,roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import VotingClassifier
import torch

if torch.cuda.is_available():
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_count())
    print('GPU is:', torch.cuda.get_device_name(0))
else:
    print('No GPU available, using the C
```

```
from google.colab import drive
drive.mount('/content/drive')
data_path = "/content/drive/My Drive/Thesis/data"

def pre_process_dataset(value):
    emoticons = [':-)', ':)', '(:', '(-:', ':))', '((:', ':-D', ':D', 'X-
D', 'XD', 'xD', 'xD', '<3', '</3', ':\*',
                 ';-)',
                 ';)', ';-D', ';D', '(;', '(-;', ':-(', ':(', '(:', '(-
:', ':,(', ':\'(', ':"(', ':((', ':D', '=D',
                 '=)',
                 '(=', '=(', ')=', '=-O', 'O-=', ':o', 'o:', 'O:', 'O:', ':-o', 'o-
:', ':P', ':p', ':S', ':s', ':@',
                 ':>',
                 ':<', '^_^', '^.^', '>.>', 'T_T', 'T-T', '-.-', '*.*', '~.~', ':*', ':-
*', 'xP', 'XP', 'XP', 'Xp',
                 ':-|',
                 ':->', ':-<', '$_$', '8-)', ':-P', ':-p', '=P', '=p', ':*)', '*-*', 'B-
)', 'O.o', 'X-(', ')-X']
    text = value.replace(".", "").lower()
    text = re.sub(r"[^a-zA-Z?.!,¿]+", " ", text)
    users = re.findall("[@]\w+", text)
    for user in users:
        text = text.replace(user, "<user>")
    urls = re.findall(r'(https?://[^\s]+)', text)
    if len(urls) != 0:
        for url in urls:
            text = text.replace(url, "<url >")
    for emo in text:
        if emo in emoji.EMOJI_DATA:
            text = text.replace(emo, "<emoticon >")
    for emo in emoticons:
        text = text.replace(emo, "<emoticon >")
    numbers = re.findall('[0-9]+', text)
    for number in numbers:
        text = text.replace(number, "<number >")
    text = text.replace('#', "<hashtag >")
    text = re.sub(r"([?.!,¿])", r" ", text)
    text = "".join(l for l in text if l not in string.punctuation)
    text = re.sub(r'[" "]+', " ", text)
    return text

data = pd.read_csv(f"{data_path}/swahili_class.csv")
print(len(data))

data["text"] = data.text.apply(lambda x: pre process dataset(x))
train df, temp df = train test split(data, random state=2018, test size=0.2)

val df, test df = train test split(temp df, random state=2018, test size=0.5)
tfidf vectorizer = TfidfVectorizer()

X tfidf train = tfidf vectorizer.fit transform(train df['text'].values.astype('U'))
X tfidf valid = tfidf vectorizer.transform(val df['text'].values.astype('U'))
X tfidf test = tfidf vectorizer.transform(test df['text'].values.astype('U'))

y train = train df['class']
y_valid = val_df['class']
```

```
y_test = test_df['class']

label_encoder = preprocessing.LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_valid = label_encoder.fit_transform(y_valid)
y_test = label_encoder.fit_transform(y_test)

nb = MultinomialNB()
nb.fit(X_tfidf_train, y_train)
alphas = [0.1, 0.5, 1.0, 1.5]
best_alpha = None
best_acc = 0.0
for alpha in alphas:
    nb.alpha = alpha
    y_pred_valid = nb.predict(X_tfidf_valid)
    acc = accuracy_score(y_valid, y_pred_valid)
    if acc > best_acc:
        best_alpha = alpha
        best_acc = acc

nb.alpha = best_alpha
nb_test_predictions = nb.predict(X_tfidf_test)
nb_valid_predictions = nb.predict(X_tfidf_valid)

cm_nb = confusion_matrix(y_test, nb_test_predictions)
print(cm_nb)

sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Naive Bayes Model')
plt.show()

report = classification_report(y_test, nb_test_predictions)
print(report)

fpr, tpr, thresholds = roc_curve(y_test, nb_test_predictions)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=1, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=1, color='gray', label='Random guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Naive Bayes Classifier')
plt.legend()
plt.show()

svm = SVC(kernel='linear', probability=True)
svm.fit(X tfidf train, y train)
svm test preds = svm.predict(X tfidf test)
svm valid preds = svm.predict(X tfidf valid)

cm = confusion matrix(y test, svm test preds)
print(cm)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Support Vector Machine Classifier')
plt.show()
```

```
report = classification_report(y_test, svm_test_preds)
print(report)


y_pred = svm.predict_proba(X_tfidf_valid)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, svm_test_preds)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=1, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=1, color='gray', label='Random guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for SVM Classifier')
plt.legend()
plt.show()

lgr = LogisticRegression(max_iter=100000)
lgr.fit(X_tfidf_train, y_train)
lgr_test_preds = lgr.predict(X_tfidf_test)
lgr_valid_preds = lgr.predict(X_tfidf_valid)

cm = confusion_matrix(y_test, lgr_test_preds)
print(cm)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Logistic Regration Classifier')
plt.show()

report = classification_report(y_test, lgr_test_preds)
print(report)

y_pred = lgr.predict_proba(X_tfidf_valid)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, lgr_test_preds)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=1, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=1, color='gray', label='Random guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Logistic Regression Classif
ier')
plt.legend()
plt.show()

ensemble model = VotingClassifier(
    estimators=[('lgr', lgr),
                ('svm', svm),
                ('nb', nb)],
                voting='soft')
ensemble model.fit(X tfidf train, y train)
ensemble test predictions = ensemble model.predict(X tfidf test)
ensemble valid predictions = ensemble model.predict(X tfidf valid)

cm = confusion matrix(y test, ensemble test predictions)
print(cm)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
plt.title('Confusion Matrix for Ensemble Model')
plt.show()

report = classification_report(y_test, ensemble_test_predictions)
print(report)

y_pred = ensemble_model.predict_proba(X_tfidf_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, ensemble_test_predictions)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=1, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=1, color='gray', label='Random guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Ensemble Classifier')
plt.legend()
plt.show()
```