HATE SPEECH RECOGNIZATION

A MINI PROJECT REPORT

18CSC304J - COMPILER DESIGN

Submitted by

ALTHAF KADHER(RA2011027010074)
KOTHA NARASIMHA RAO(RA2011027010120)
VADLAPUTI DINESH(RA2011027010121)

Under the guidance of

Dr. S. SHARANYA

Assistant Professor, Department of Computer Science and Engineering

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "**HATE SPEECH RECOGNIZATION**" is the bonafide work of
ALTHAF KADHER (RA2011027010074), KOTHA NARASIMHA RAO (RA2011027010120), VADLAPUTI DINESH (RA2011027010121) who carried out the project work under my supervision.

Dr. S. Sharanya
Assistant Professor
Department of Data Science and Business Systems
SRM Institute of Science and Technology
and Technology

Dr.M.Lakshmi
Head of department
Department of DSBS
SRM Institute of Science

ABSTRACT

Hate speech recognition is the process of identifying and categorizing speech that is intended to be derogatory, offensive, or harmful toward a particular group or individual. This process involves the use of natural language processing techniques and machine learning algorithms to analyze and classify speech based on its content, context, and intent.

The recognition of hate speech is an important task in the field of computational linguistics and is essential for ensuring online safety and promoting civil discourse. Hate speech recognition can be used in various applications, such as content moderation, social media monitoring, and online harassment prevention.

The process of hate speech recognition typically involves several stages, including data collection, preprocessing, feature extraction, and classification. In data collection, large datasets of text, audio, or video data are gathered from various sources, such as social media platforms, news websites, and online forums. Preprocessing involves cleaning and normalizing the data, such as removing stop words and punctuation.

Feature extraction is the process of transforming the raw data into a set of numerical features that can be used for classification. This involves techniques such as word embedding, which converts words into vectors, and topic modeling, which identifies the underlying topics in the data. Finally, classification algorithms such as support vector machines (SVMs), neural networks, or decision trees are used to classify the data into different categories based on their hate speech content.

Overall, hate speech recognition is a challenging and complex task due to the diversity and complexity of language use, as well as the subjectivity of hate speech definitions. Nevertheless, advances in natural language processing and machine learning techniques have shown promising results in detecting and preventing hate speech online.

TABLE OF CONTENTS

CHAPTER 1 – INTRODUCTION

Hate speech is a form of speech that is intended to be derogatory, offensive, or harmful towards a particular group or individual based on their race, ethnicity, religion, gender, sexual orientation, or other personal characteristics. In recent years, the rise of online platforms has made hate speech more pervasive and accessible, leading to concerns about its impact on social cohesion and individual well- being.

Hate speech recognition is the process of identifying and categorizing speech as hate speech or not hate speech. This is a complex and challenging task that requires a deep understanding of language use and the ability to detect the nuances of language and intent. Hate speech recognition is important for promoting online safety, protecting vulnerable groups from harm, and maintaining civil discourse in online communities.

Recent advances in natural language processing and machine learning techniques have made hate speech recognition more feasible and effective. These techniques allow us to analyze and classify large volumes of text data in real time, enabling us to detect and respond to hate speech quickly and efficiently.

Overall, hate speech recognition is an important and challenging task that requires ongoing research and development. By developing effective hate speech recognition tools and techniques, we can help create a safer and more inclusive online environment for everyone.

CHAPTER 2 – LIMITATIONS OF EXISTING METHODS

There are several limitations and challenges to hate speech recognition that researchers and practitioners need to consider. Some of these limitations include:

1. Contextual ambiguity: Hate speech can be highly contextual, and its meaning can be influenced by various factors such as the speaker's tone, intent, and social and cultural context. This makes it challenging to develop automated hate speech recognition systems that can accurately detect hate speech in all contexts.

2. Subjectivity: Hate speech is a subjective term, and different people may have different opinions about what constitutes hate speech. This can make it difficult to develop a universal definition of hate speech that can be used to train automated systems.

3. Multilingualism: Hate speech can occur in different languages, making it challenging to develop hate speech recognition systems that can work across different languages and cultures.

4. Data bias: Hate speech datasets may be biased towards certain types of hate speech, which can affect the accuracy and fairness of automated hate speech detection systems.

Existing methods for hate speech recognition include rule-based systems, machine learning algorithms, and deep learning models. Rule-based systems use predefined rules to identify hate speech based on patterns and keywords in the text. Machine learning algorithms use statistical models to learn patterns and features in the data and classify it into different categories, including hate speech. Deep learning models, such as recurrent neural networks and convolutional neural networks, use neural networks to learn complex patterns and relationships in the data and achieve high accuracy in hate speech detection.

Despite their effectiveness, these methods have their own limitations and challenges, such as the need for large amounts of labeled data, over-reliance on textual features, and susceptibility to adversarial attacks. To address these limitations, researchers are exploring new approaches, such as multimodal methods that combine text, audio, and visual data, and unsupervised methods that can data.

CHAPTER 3 – PROPOSED METHOD WITH ARCHITECTURE

The rough workflow of our compiler as shown below in the diagram:

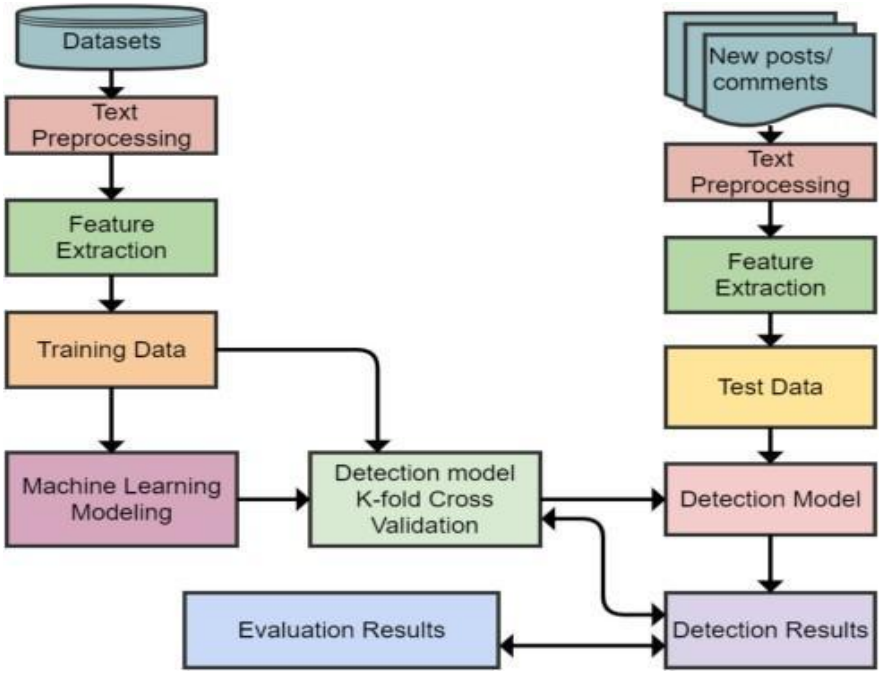

Fig-1: Hate speech detection architecture diagram

The proposed architecture for detecting Odia-English posts and comments as hate, offensive or normal speech is shown in Figure 1. It receives bilingual data as the input and then pre-processes it based on the language nature, which involves removing punctuations, normalization, tokenization and other basic necessary pre-processes. Feature extraction then extracts the feature using TF-IDF, n-gram, and word2vec. The output of this task is an important feature vector (training data) of the dataset for training the model. After feature extraction, the models are trained using SVM, NB, and RF machine learning algorithms. The resulting models are then evaluated by K-fold cross-validation and, based on the validation results, the best detection model is selected. The outcome of these tasks is a detection model for detecting hate and offensive speech. The detection model is evaluated and selected, based on the results of the model evaluation method discussion. The final selected detection model is used to develop a prototype that can take new Odia-English texts as input and classify the input according to whether it contains hate, offensive or normal

Removing (Cleaning) irrelevant Characters, Punctuation symbols , Emoji

The posts and comments on social media text usually contain special characters, punctuations, symbols, and emojis to express different opinions and feelings. Therefore, the cleaning task involves removing all irrelevant special characters, symbols, and emojis. The source code of the cleaning Algorithm 1 is given below:

Algorithm 1: Cleanup Odia-English Mixed Text.

Input: Text in a dataset

Output: Clean text

Begin:

1. Read the text in the dataset

2. While (end of the text in a dataset): If the text contains special char [! @#$%&] then Remove special char, If the text contains symbol [o] then Replace symbol and add space: If a text contains odia Punc-[!?"] then Remove Odia Punc If text contain number 10-9] then Remove number I If a text contains emoji-1 I then Remove emoji: If a text contains extra white space then Trim the text:

3. Return clean text;

End:

Tokenization:

After the cleaning and normalization tasks, the tokenization splits the post and comment text into individual words or tokens by using spaces between words or punctuation marks. This is important because the meaning of text generally depends on the relations of words in that text and this helps the feature extraction methods to obtain the appropriate features from the dataset.

Proposed Feature Extractions:

The proposed feature extractions perform the extraction of important features of the dataset. It goes along with the input of prepossessed and tokenized dataset words and performs extractions, as shown in Figure 3. The extracted features are used for training

the models and to predict the class of posts and comments as hate, offensive, and normal speech. The adopted feature extraction methods, word2vec, TF-IDF, and n-gram, are well-known in text mining approaches [42]. Each method provides the feature vectors used to train the machine learning classifier.

n-Gram Feature Extraction:

This paper proposes a word n-gram feature extraction method that is experimented on a different value of n that ranges from one to three where n is the number of words used in the probability sequences. An n-gram of two words is called a bigram (2-g). The feature extraction is performed by using unigram, bigram, trigram, and combination n-grams. The performance of the n-gram needs a proper choice of the n value. In addition, it provides a different feature model to train and compare the n-gram features with one another.

Dataset Description:

In order to build the dataset for this paper, the authors collected posts and comments from Facebook manually. Firstly, we selected 35 different public Facebook pages, which belonged to categories that contain a range of three to six selected pages based on the selection criteria of public pages. We then collected all posts on the page from April 2019 to April 2020 and recorded the comments under each post. Next, we filtered the Odia-English mixed posts and comments by removing non-textual data. This process resulted in a total number of 837,077 posts and comments. There were, in total, 27,162 posts and comments on unique pages being filtered through the keywords. The keywords helped to filter the posts and comments which were likely to have hateful or offensive speech in the content. Finally, 5000 posts and comments were annotated and labeled as hate speech (HS), offensive speech (OFS), and neither offensive nor hate speech (OK) categories.

Preprocessing Implementation:

The authors utilized a simple random sampling technique to select the posts and comments to be annotated. This technique provided an equal chance for all of the filtered posts and comments to be annotated. Because of the time limitation of the research and the resources of the annotation process for filtered posts and comments, 5000 posts and comments were selected to be annotated. Four annotators were in total and everyone labeled the posts and comments based on the same guidelines. Three of the annotators

were given 500 similar instances and 1000 unique posts and comments. The same instances were used to evaluate the consistency of the annotations among the annotators. The fourth annotator was the researcher who oversaw the whole process of the annotation and annotated 1500 unique posts and comments. The process of building the dataset is tedious, challenging, and time-consuming, therefore, we consider only 500 of the same posts and comments to be annotated by the three annotators and the researcher decided the final class using the majority vote's method.

The annotation process resulted in a distribution of classes shown in Table 3 for each annotator. The labeling result for the common 500 instances of posts and comments by the annotators is presented in Table 4. The resulting three-class distribution in dataset is shown in Table 5. The dataset used to train the machine learning models consisted of 4500 uniquely annotated posts and the final class of 500 posts and comments were decided by the researcher using the majority vote method. A total of 5000 posts and comments were collected in the animation

CHAPTER 4 – MODULES WITH DESCRIPTION

Natural Language Toolkit (NLTK): NLTK is a Python library that provides a wide range of NLP tools and techniques such as tokenization, stemming, and stopwords removal. It is used in this code for the preprocessing of error messages.

Regular Expressions (regex): Regular expressions are a sequence of characters that define a search pattern. In this code, regex is used to remove non-alphabetic characters and digits from the error messages.

Seaborn: Python Seaborn library is a widely popular data visualization library that is commonly used for data science and machine learning tasks.

XGBoost: It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.

Wordcloud: It is basically a visualization technique to represent the frequency of words in a text where the size of the word represents its frequency.

LightGBM: It is a gradient-boosting ensemble method that is used by the Train Using AutoML tool and is based on decision trees. As with other decision tree-based methods, LightGBM can be used for both classification and regression.

Pandas: Pandas is a Python library used for data manipulation and analysis. In this code, it is used to create a data frame to store the token frequency counts.

NumPy: It is an open source Python library that's used in almost every field of science and engineering.

Scikit-learn: It is an open source data analysis library, and the gold standard for Machine Learning (ML) in the Python ecosystem.

Urllib.parse: It is a module in Python's standard library that provides various functions for parsing and manipulating URLs (Uniform Resource Locators). It is part of the urllib package, which also includes other modules for working with URLs and handling network requests.

These modules are used in the code to preprocess and analyze the error messages, extract features, and visualize the results in a user-friendly way.

CHAPTER 5 – CODE

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

We have imported all the necessary python libraries which will be used in this project.

```
[ ] import nltk
    nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Let's Download the stop words to check for every

Now Let's Try to import the nlp module, stemmer.

```
import re
import nltk
from nltk.util import pr
stemmer = nltk.SnowballStemmer("english")
from nltk.corpus import stopwords
import string
stopword = set(stopwords.words("english"))
```

Read the dataset

```
[6] df=pd.read_csv("/twitter_data.csv")
```

0s

Let's check the first five rows of the dataset.

```
print(df.head())
```

```
   Unnamed: 0  count  hate_speech  offensive_language  neither  class  \
0           0      3            0                   0        3      2
1           1      3            0                   3        0      1
2           2      3            0                   3        0      1
3           3      3            0                   2        1      1
4           4      6            0                   6        0      1

                                               tweet
0  !!! RT @mayasolovely: As a woman you shouldn't...
1  !!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2  !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3  !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4  !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
```

In the given dataset there is no label column present in the dataset we need to add the label column because we are checking whether offensive language is detected or not

So add the new column to the dataset with the column name label .

```
df['labels'] = df['class'].map({0:"Hate Speech Detected", 1:"Offensive language detected",3:"No hate and offensive speech"})
print(df.head())
```

```
   Unnamed: 0  count  hate_speech  offensive_language  neither  class  \
0           0      3            0                   0        3      2
1           1      3            0                   3        0      1
2           2      3            0                   3        0      1
3           3      3            0                   2        1      1
4           4      6            0                   6        0      1

                                               tweet  \
0  !!! RT @mayasolovely: As a woman you shouldn't...
1  !!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2  !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3  !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4  !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

                      labels
0                        NaN
1  Offensive language detected
2  Offensive language detected
3  Offensive language detected
4  Offensive language detected
```

In the given dataset for finding whether the hate speech is detected or not we need only tweet columns and Label columns. Because by using the tweet column only going to check whether the hate speech is detected or not.

Remove the all columns except tweet and the labels.

In the given tweet column, it contains the unwanted variables like plus minus etc... remove those unwanted words because it is not useful for the finding the output.

```
def clean(text):
    text = str(text).lower()
    text = re.sub('\[.*?\]','',text)
    text = re.sub('https?://\S+|ww\.\s+','',text)
    text = re.sub('<_*?>+','',text)
    text = re.sub('[%s]'% re.escape(string.punctuation),'',text)
    text = re.sub('\n','',text)
    text = re.sub('\w*\d\w*','',text)
    text = [word for word in text.split(' ') if word not in stopword]
    text = " ".join(text)
    return text
df["tweet"] = df["tweet"].apply(clean)
print(df.head())
```

```
                                               tweet  \
0   rt mayasolovely woman shouldnt complain clean...
1   rt  boy dats coldtyga dwn bad cuffin dat hoe ...
2   rt urkindofbrand dawg rt  ever fuck bitch sta...
3           rt cganderson vivabased look like tranny
4   rt shenikaroberts shit hear might true might ...

                       labels
0                         NaN
1  Offensive language detected
2  Offensive language detected
3  Offensive language detected
4  Offensive language detected
```

Consider x as the tweet and y as the labels.

Now fit the x and y in to the decision tree

model. So

```
x = np.array(df["tweet"]).astype(str)
y = np.array(df["labels"]).astype(str)



cv = CountVectorizer()
x = cv.fit_transform(x)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=100)

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

By seeing the above output we fit the x and y in to the decision tree classifier which will take decisions at each node and if decisions are satisfied according to the given condition then it will generate the output.

Let’s check our output is generating output or not.

```
[12] test_data = "i want be your friend"
     df = cv.transform([test_data]).toarray()
     print(clf.predict(df))

     ['nan']
```

In the above input there is no hate word present that’s why it was showing NAN.
Let’s try for another input.

By the giving the input data it will divide the each and every word in to the tokens and check with the stop words in the stop words library it each token will check if any hate word is

```
test_data = "He will kill you"
df = cv.transform([test_data]).toarray()
print(clf.predict(df))

['Hate Speech Detected']
```

present in the given input then it will give output as the hate speech detected if in the given input all are good words then the input will be given as no hate word will be given by the output.

CHAPTER 6 – CONCLUSION

In conclusion, hate speech recognition is an important area of research that aims to automatically identify and classify hate speech content in various forms of digital media. With the rise of social media and online communication platforms, the need for effective hate speech recognition systems has become more pressing.

Various approaches and techniques have been proposed for hate speech recognition, including natural language processing, machine learning, and deep learning. These approaches involve different steps, such as data collection, preprocessing, feature extraction and selection, model training, evaluation, and deployment.

While significant progress has been made in hate speech recognition, there are still many challenges and limitations that need to be addressed. These include issues related to data bias, model generalization, and ethical considerations.

Overall, the development of effective hate speech recognition systems is important for promoting a safe and inclusive online environment, and it requires collaboration among researchers, industry, and policymakers.

CHAPTER 7 – REFERENCES

[1] Dhanalakshmi Ranganayakulu, Chellappan C., Detecting Malicious URLs in E-mail – An Implementation, AASRI Procedia, Vol. 4, 2013, Pages 125-131, ISSN 2212-6716, https://doi.org/10.1016/j.aasri.2013.10.020.

[2] Yu, Fuqiang, Malicious URL Detection Algorithm based on BM Pattern Matching, International Journal of Security and Its Applications, 9, 33- 44, 10.14257/ijsia.2015.9.9.04.

[3] K. Nirmal, B. Janet and R. Kumar, Phishing - the threat that still exists, 2015 International Conference on Computing and Communications Technologies (ICCCT), Chennai, 2015, pp. 139-143, doi: 10.1109/ICCCT2.2015.7292734.

[4] F. Vanhoenshoven, G. Napoles, R. Falcon, K. Vanhoof and M. K ´ oppen, ¨ Detecting malicious URLs using machine learning techniques, 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, 2016, pp. 1-8, doi: 10.1109/SSCI.2016.7850079.

[5] https://www.kaggle.com/xwolf12/ malicious-and-benign-websites accessed on 27.01.2021

[6] https://openphish.com/ accessed on 27.01.2021

[7] Doyen Sahoo, Chenghao lua, Steven C. H. Hoi, Malicious URL Detection using Machine Learning: A Survey, arXiv:1701.07179v3 [cs.LG], 21 Aug 2019

[8] Rakesh Verma, Avisha Das, What's in a URL: Fast Feature Extraction and Malicious URL Detection, ACM ISBN 978-1-4503-4909-3/17/03

[9] https://github.com/ShantanuMaheshwari/Malicious Website Detection

[10] Frank Vanhoenshoven, Gonzalo Napoles, Rafael Falcon, Koen Vanhoof and Mario Koppen, Detecting Malicious URLs using Machine Learning Techniques, 978-1-5090-4240-1/16 2016, IEE