GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

BANGALORE

PRESIDENCY GROUP OVER 40 YEARS OF ACADEMIC WISDOM

A Project Report

On

AI BASED CONTENT MODERATOR

Batch Details

Sl. No.	Roll Number	Student Name
1	20211CAI0078	B MEENU
2	20211CAI0201	V DEEKSHITHA
3	20211CAI0115	AISHWARYA VILAS PATIL
4	20211CAI0180	GADDAM SAI LIKHITHA

School of Computer Science,

Presidency University, Bengaluru.

Under the guidance of,

Dr. Alamelu Mangai Jothidurai
Associate Professor
School of Computer Science and Engineering
Presidency University

CONTENTS

SL.NO	CONTENTS	PG.NO
1	Introduction	3
2	Literature review	3-4
3	Objectives	4
4	Software usages	4
5	Methodology	4-13
6	Outcomes	13
7	Timeline of the project	13
8	Conclusion	13

1. INTRODUCTION

Natural Language Processing (NLP) enables machines to understand and process human languages. FastText, a tool developed by Facebook's AI Research, excels in multilingual text categorization by representing words through subword-level character n-grams. This feature allows FastText to handle out-of-vocabulary words and complex languages, supporting both supervised and unsupervised learning for tasks like sentiment analysis and language identification.

Natural language processing (NLP) combines computational linguistics, machine learning, and deep learning models to process human language. Computational linguistics is the science of understanding and constructing human language models with computers and software tools. Machine learning is a technology that trains a computer with sample data to improve its efficiency. Deep learning is a specific field of machine learning which teaches computers to learn and think like humans Natural language processing (NLP) is a machine learning technology that gives computers the ability to interpret, manipulate, and comprehend human language. Organizations today have large volumes of voice and text data from various communication channels like emails, text messages, social media newsfeeds, video, audio, and more. They use NLP software to automatically process this data, analyze the intent or sentiment in the message, and respond in real time to human communication. Natural language processing (NLP) is critical to fully and efficiently analyze text and speech data. It can work through the differences in dialects, slang, and grammatical irregularities typical in day-to-day conversations.

Analyze customer feedback or call center recordings

- Run chatbots for automated customer service
- Answer who-what-when-where questions
- Classify and extract text

Generative AI, a subfield of artificial intelligence, creates content using patterns learned from vast datasets. Models like GPT and DALL·E mimic human creativity for applications
in content creation, design, and research. Despite its transformative potential, generative AI raises ethical concerns, including copyright infringement and misinformation.

Three Devanagari-script-based NLP tasks address key challenges in multilingual contexts. Task 1: Language Identification classifies sentences in languages like Nepali, Marathi, Sanskrit, Bhojpuri, and Hindi, critical for translation and speech recognition. Task 2: Hate Speech Detection identifies harmful content in underrepresented languages like Bhojpuri and Nepali. Task 3: Target Identification for Hate Speech focuses on recognizing specific targets of hate speech (individuals, organizations, or communities), aiding in content moderation and conflict resolution. Together, these tasks contribute to digital safety and multilingual content processing.

2. LITERATURE REVIEW

With the advent of the World Wide Web, the amount of data on web increased tremendously. Although, such a huge accumulation of information is valuable and most of this information is texts, it becomes a problem or a challenge for humans to identify the most relevant information or knowledge. So, text classification helps to overcome this challenge. Text classification is the act of dividing a set of input documents into two or more classes where each document can be said to belong to one or multiple classes. Text Classification is a text mining technique which is used to classify the text documents into predefined classes. Classification can be manual or automated. Unlike manual classification, which consumes time and requires high accuracy, Automated Text Classification makes the classification process faster and more efficient since it automatically categorizes documents. Language is used as medium for written as well as spoken communication. With the use of Unicode encoding, text on web may be present in different languages. This will add complexity of natural language processing to text classification. Text Classification is a combination of Text Mining as well as Natural Language Processing. It has many applications such as document indexing, document organization and hierarchical categorization of web pages.

K.E. Abdelfatah [1] delves into the nature and rapid spread of hate speech on social media, emphasizing how digital platforms, through user anonymity and algorithmic amplification, facilitate the dissemination of harmful content, often leading to polarization and violence in society. Abozinadah, E.A [3] broadens the discussion by exploring hate speech beyond radicalization or criminal activity, focusing on its role in influencing social dynamics, cultural identities, and intergroup relations. This study highlights how hate speech perpetuates stereotypes, fuels social tensions, and disproportionately affects marginalized communities, stressing the need to understand hate speech as a multifaceted societal issue. Furthermore, E.A. Abozinadah [4] underscores the critical need for automated hate speech detection, especially given the limitations of manual moderation in managing the overwhelming volume of content on digital platforms. The study suggests that automated systems could enhance accuracy and efficiency in identifying harmful speech, given the potential for hate speech to incite violence or deepen divisions. A. Alakrot [11] offers a perspective on hate speech within the realms of gender, religion, and race, analyzing its role in cyberterrorism and the challenges it poses for international law, while noting that the absence of Twitter datasets or

machine learning limits its applicability to social media analysis. Albadi, N [13] contributes by investigating the geographical and cultural dimensions of hate speech, revealing its prevalence across regions and the unique features of different social media platforms that influence its spread. The study integrates both qualitative and quantitative methods, emphasizing the need for comprehensive policy responses and interventions to address the complex and varied manifestations of hate speech across different contexts.

Surendrabikram Thapa et al. [1] developed the RUHate-MM dataset to improve hate speech detection during the Russia-Ukraine crisis, focusing on identifying specific targets—individuals, organizations, and communities through multimodal content like text and images. Their approach faced challenges in accurately linking text and images, particularly in memes and sarcastic posts, and the subjectivity in manual annotation made target identification difficult. Fernando H. Calderón et al. [2] used a linguistic pattern-based approach to dynamically detect hidden hate speech terms, focusing on identifying targets using contextual word embeddings rather than predefined terms. While effective, the method demanded high computational resources and struggled with ambiguous content. Michael Miller Yoder et al. [3] analyzed how hate speech varies across targeted identity groups and found that models trained on specific groups performed poorly when generalized to other targets, complicating detection efforts. Patricia Chiril et al. [4] developed a multi-target hate speech detection model that integrated affective knowledge to improve accuracy but noted the difficulty in generalizing across datasets without target-specific data. Zewdie Mossie and Jenq-Haur Wang [5] focused on identifying vulnerable communities in Ethiopia using hate speech detection, specifically targeting the Tigre ethnic group, but faced limitations due to a lack of comprehensive hate lexicons for Amharic texts. Across these studies, accurate target identification remains a critical yet challenging aspect of hate speech detection.

3. OBJECTIVES

- Determine the language it belongs to among Nepali, Marathi, Sanskrit, Bhojpuri, and Hindi which addresses the critical need for accurate language identification in multilingual contexts.
- Identify whether it contains hate speech or not.
- Identify the targets of hate speech in each hateful text.
- Developing a prototype app to simulate functionality and test features before full-scale implementation.

4. SOFTWARES USED

- Google Colab
- Fasttext
- LangID
- LangDetect

5. METHODOLOGY

5.1 Algorithms

1. FastText

1. fastText as a library for efficient learning of word representations and sentence classification.
2. It is written in C++ and supports multiprocessing during training.
3. FastText allows you to train supervised and unsupervised representations of words and sentences.
4. FastText supports training continuous bag of words (CBOW) or Skip-gram models using negative sampling, softmax or hierarchical softmax loss functions.

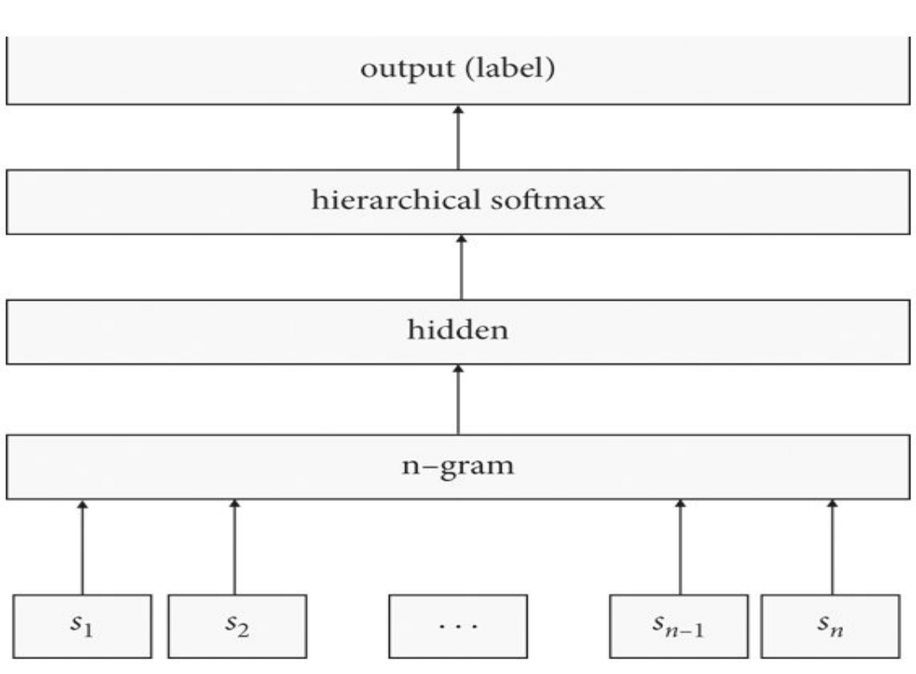

Fig1: Architecture of fasttext

1. **Input Layer:**
   - Represents the input data, in this case, an n-gram.
   - Each input node (s1, s2, ..., sn-1, sn) corresponds to a word or token in the n-gram.
2. **Hidden Layer:**
   - Processes the input n-gram and extracts relevant features.
   - The number of nodes in this layer determines the model's capacity to learn complex patterns.
3. **Hierarchical Softmax Layer:**
   - A specialized layer designed for efficient prediction of categorical variables, especially when the vocabulary size is large.
   - Instead of computing probabilities for all possible output classes (as in traditional softmax), hierarchical softmax organizes the output space into a tree structure.
   - This reduces the computational complexity of the prediction process, making it more efficient.
4. **Output Layer:**
   - Represents the predicted output, which is the label or class associated with the input n-gram.
5. **Overall Function:**
   - The network takes an n-gram as input, processes it through the hidden layer, and then uses the hierarchical softmax layer to predict the most likely label or class.

1.2 Pseudo code

2. pip install fasttext
3. IMPORT pandas as pd, fasttext, numpy as np, requests, matplotlib. Pyplot as plt, seaborn as sns
4. from sklearn. metrics IMPORT roc_auc_score, classification_report
5. from sklearn. preprocessing IMPORT LabelBinarizer,label_binarize
6. from sklearn. model_selection IMPORT train_test_split
7. SET df TO pd.read_csv('E:/sharedtasks capstone/subtask A_edited/taskA_train.csv')
8. Pre-Processing:
   df.info ()
   df.isnull().sum()
   df.dropna(subset=['text'],inplace=True)
   df.isnull().sum()
   df['label'].value_counts()
9. SET url TO "https://dl.fbaipublicfiles.com/fasttext/supervised-models/lid.176.bin"
   SET filename TO "lid.176.bin"
   SET response TO requests.get(url)
   IF response.status_code EQUALS 200:
      with open (filename, 'wb') as f:
         f.write(response.content)
      OUTPUT (f"Downloaded {filename} successfully.")
   ELSE:
      OUTPUT (f"Error downloading the file. Status code: {response.status_code}")

10. DEFINE FUNCTION train_language_model(train_data, output_model_path):
    TRY:
      SET model TO fasttext.train_supervised(INPUT=train_data, lr=0.1, epoch=100,wordNgrams=2)
      model.save_model(output_model_path)
      OUTPUT("Model trained successfully and saved to:", output_model_path)
    except Exception as e:
      OUTPUT("Error during model training:", e)

    IF __name__ EQUALS "__main__":
    SET train_data_path TO 'E:/sharedtasks capstone/subtask A_edited/taskA_train.csv'
    SET output_model_path TO 'language_model.bin'
    train_language_model(train_data_path, output_model_path)
11. DEFINE FUNCTION detect_languages_batch_eval(texts, model, batch_size=1000):
      SET predicted_languages TO []
      FOR i IN range(0, len(texts), batch_size):

```
        SET batch TO texts[i:i+batch_size]
        SET batch TO [str(text) IF not pd.isnull(text) else '' FOR text IN batch]
        SET labels, probabilities TO model.predict(batch, k=1)
        SET language_codes TO [label[0].replace('__label__', '') FOR label IN labels]
        SET label_mapping TO { 'ne': 1,'mr': 2,'sa': 3, 'hi': 4}
        SET mapped_language_codes TO [label_mapping.get(code, -1) FOR code IN language_codes]
        predicted_languages.extend(mapped_language_codes)
        OUTPUT(f"Processed {i+batch_size} out of {len(texts)} texts")  # Progress tracking
    RETURN predicted_languages
IF __name__ EQUALS "__main__":
    SET df1 TO pd.read_csv('E:/sharedtasks capstone/subtask A_edited/taskA_(index,text(eval.csv')
    SET model TO fasttext.load_model('lid.176.bin')
    SET all_sentences TO df1['text'].tolist()
    SET predicted_languages TO detect_languages_batch_eval(all_sentences, model)
    SET df1['predicted_language'] TO pd.Series(predicted_languages)
     df1.to_csv('output_predicted_onevaldata.csv', index=False)
12. DEFINE FUNCTION detect_languages_batch_test(texts, model, batch_size=1000):
        SET predicted_languages TO []
        FOR i IN range(0, len(texts), batch_size):
            SET batch TO texts[i:i+batch_size]
            SET batch TO [str(text) IF not pd.isnull(text) else '' FOR text IN batch]
            SET labels, probabilities TO model.predict(batch, k=1)
            SET language_codes TO [label[0].replace('__label__', '') FOR label IN labels]
            SET label_mapping TO {'ne': 1,'mr': 2, 'sa': 3, 'hi':4}
            SET mapped_language_codes TO [label_mapping.get(code, -1) FOR code IN language_codes]  # Map
unknown codes to -1
            predicted_languages.extend(mapped_language_codes)
            OUTPUT(f"Processed {i+batch_size} out of {len(texts)} texts")  # Progress tracking
        RETURN predicted_languages
    IF __name__ EQUALS "__main__":
        SET df_test TO pd.read_csv('E:/sharedtasks capstone/subtask A_edited/taskA(index,text)test.csv')
        SET model TO fasttext.load_model('lid.176.bin')
        SET all_sentences_test TO df_test['text'].tolist()
        SET predicted_languages_test TO detect_languages_batch_eval(all_sentences_test, model)
        SET df_test['predicted_language'] TO pd.Series(predicted_languages_test)
        df_test.to_csv('output_predicted_test_data.csv', index=False)
13. truth=pd.read_csv("E:/sharedtasks capstone/subtask A_edited/taskA_(index,label)eval.csv")
14. predicted=pd.read_csv("E:/sharedtasks capstone/output_predicted_onevaldata.csv")
15. tl=truth['label'].tolist()
16. pl=predicted['predicted_language'].tolist()
17. label_mapping={1:1,2:2,3:3,4:4}
18. SET tl TO truth['label'].tolist()
19. SET pl TO predicted['predicted_language'].tolist()
20. OUTPUT(classification_report(tl, pl))
21. SET cm TO confusion_matrix(tl, pl)
22. plt.figure(figsize=(8, 6))
23. sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['nepali', 'marathi', 'sanskrit', 'hindi'],
    yticklabels=['nepali', 'marathi', 'sanskrit', 'hindi'])
24. plt.xlabel('Predicted')
25. plt.ylabel('True')
26. plt.title('Confusion Matrix')
27. plt.show()
28. SET y_true_bin TO label_binarize(tl, classes=list(label_mapping.values()))
29. SET y_pred_bin TO label_binarize(pl, classes=list(label_mapping.values()))
30. SET auc_scores TO []
31. FOR i IN range(y_true_bin.shape[1]):
```

```
        SET auc TO roc_auc_score(y_true_bin[:, i], y_pred_bin[:, i])
        auc_scores.append(auc)
32. FOR i, auc IN enumerate(auc_scores):
        OUTPUT(f"AUC FOR DEFINE CLASS {list(label_mapping.keys())[i]}: {auc}")
33. plt.figure(figsize=(8, 6))
34. plt.bar(list(label_mapping.keys()), auc_scores)
35. plt.xlabel('Classes')
36. plt.ylabel('AUC Score')
37. plt.title('AUC Score FOR Each Class')
38. plt.show()
39. SET macro_auc TO roc_auc_score (y_true_bin, y_pred_bin, average='macro')
40. OUTPUT (f"Macro-average AUC: {macro_auc}")
41. SET weighted_auc TO roc_auc_score (y_true_bin, y_pred_bin, average='weighted')
42. OUTPUT (f"Weighted-average AUC: {weighted_auc}")
43. SET overall_auc TO roc_auc_score (y_true_bin, y_pred_bin)
44. OUTPUT (f"Overall AUC: {overall_auc}")
```

2. LangID

- The **LangID-1.1.6** model is a language identification system that uses a **feedforward neural network**.
- It extracts **character n-grams** from each word and embeds them into vectors through lookup tables.
- This approach allows the model to efficiently identify langauges based on the input text.

Here's a more detailed breakdown:

1. **Character N-grams**: The model extracts character n-grams from each word in the input text. N-grams are contiguous sequences of n characters within a word, which help capture language-specific patterns.
2. **Vector Embedding**: These character n-grams are then embedded into vectors using look-up tables. This step converts the n-grams into a numerical format that the neural network can process.
3. **Feedforward Neural Network**: The embedded vectors are fed into a feedforward neural network. This network consists of multiple layers of neurons that process the input and learn to identify the language based on the patterns in the n-grams.
4. **Output Layer**: The final layer of the network outputs a probability distribution over the possible languages. The language with the highest probability is chosen as the identified language.

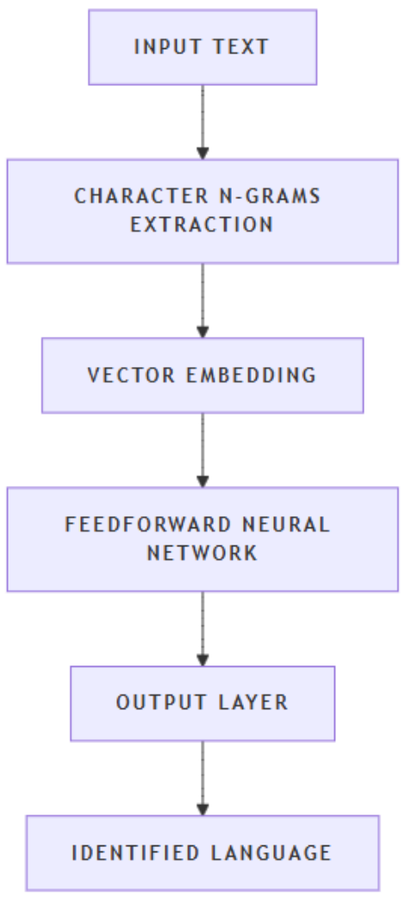

Fig2: Architecture of langID

2.1 Pseudo code

```
1. pip install langid
2. IMPORT numpy as np, pandas as pd, seaborn as sns, matplotlib.pyplot as plt
3. from sklearn.metrics IMPORT classification_report,confusion_matrix, roc_auc_score
4. from sklearn.preprocessing IMPORT label_binarize
5. DEFINE FUNCTION detect_languages_batch_eval_langid(texts, batch_size=1000):
```

```
                SET predicted_languages TO []
                FOR i IN range(0, len(texts), batch_size):
                    SET batch TO texts[i:i + batch_size]
                    SET batch TO [str(text) IF not pd.isnull(text) else '' FOR text IN batch]
                    SET language_codes TO [langid.classify(text)[0] FOR text IN batch]
                    SET label_mapping TO {'ne': 1,'mr': 2,'sa': 3,'hi': 4}
                    SET mapped_language_codes TO [label_mapping.get(code, -1) FOR code IN language_codes]  # Map
unknown codes to -1
                    predicted_languages.extend(mapped_language_codes)
                    OUTPUT(f"Processed {i + batch_size} out of {len(texts)} texts")  # Progress tracking
                RETURN predicted_languages
            IF __name__ EQUALS "__main__":
                SET df1 TO pd.read_csv('/content/taskA_(index,text(eval.csv')
                SET all_sentences TO df1['text'].tolist()
                SET predicted_languages TO detect_languages_batch_eval_langid(all_sentences)
                SET df1['predicted_language'] TO pd.Series(predicted_languages)
                df1.to_csv('output_predicted_onevaldata_langid.csv', index=False)
6. df1=pd.read_csv("/content/taskA_(index,label)eval.csv")
7. df2=pd.read_csv("/content/output_predicted_onevaldata_langid.csv")
8. SET y_true TO df1['label']  # Replace 'label' with the actual column name FOR true labels
9. SET y_pred TO df2['predicted_language']  # Replace 'predicted_language' with the actual column name FOR
   predictions
10. SET label_mapping TO { 'ne': 1,’mr’:2,’sa’:3,’hi’:4}
11. SET y_true_numerical TO y_true.map(label_mapping) #Map string labels to numerical labels
12. SET report TO classification_report(y_true_numerical, y_pred)
13. OUTPUT(report)
14. SET label_mapping TO {'ne': 1,’mr’:2,’sa’:3,’hi’:4}
15. SET y_true_numerical TO y_true.map(label_mapping)
16. SET confusion_mat TO confusion_matrix(y_true_numerical, y_pred)
17. plt.figure(figsize=(8, 6))
18. sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues’,xticklabels=label_mapping.keys(),
    yticklabels=label_mapping.keys())
19. plt.xlabel('Predicted')
20. plt.ylabel('True')
21. plt.title('Confusion Matrix')
22. plt.show()
23. SET y_true_binarized TO label_binarize(y_true_numerical, classes=[1, 2, 3, 4])
24. SET y_pred_binarized TO label_binarize(y_pred, classes=[1, 2, 3, 4])
25. SET auc_scores TO []
26. FOR i IN range(y_true_binarized.shape[1]):
        TRY:
            SET auc TO roc_auc_score(y_true_binarized[:, i], y_pred_binarized[:, i])
            auc_scores.append(auc)
        except ValueError:
            OUTPUT(f"Unable to calculate AUC FOR DEFINE CLASS {i+1} due to insufficient data")
27. IF auc_scores:
        SET average_auc TO np.mean(auc_scores)
        OUTPUT(f"Average AUC: {average_auc:.4f}")
        FOR i, auc IN enumerate(auc_scores):
            OUTPUT(f"AUC FOR DEFINE CLASS {i+1}: {auc:.4f}"
```

3. LangDetect

- LangDetect supports over 50 languages and is known for its simplicity and efficiency.
- It's a great choice for quick and straightforward language detection tasks.
- The **LangDetect** model architecture is relatively straightforward and is based on statistical methods rather than dee p learning. Here's a detailed breakdown:

1. **Character N-grams Extraction**: LangDetect extracts character n-grams from the input text. These n-grams are sequences of characters that help capture language-specific patterns.
2. **Feature Vector Creation**: The extracted n-grams are converted into a feature vector, which is a numerical representation of the text.
3. **Language Models**: LangDetect uses pre-trained language models for each supported language. These models are statistical models trained on large text corpora in different languages.
4. **Classification**: The feature vector is compared against the pre-trained language models to determine the most likely language of the input text.

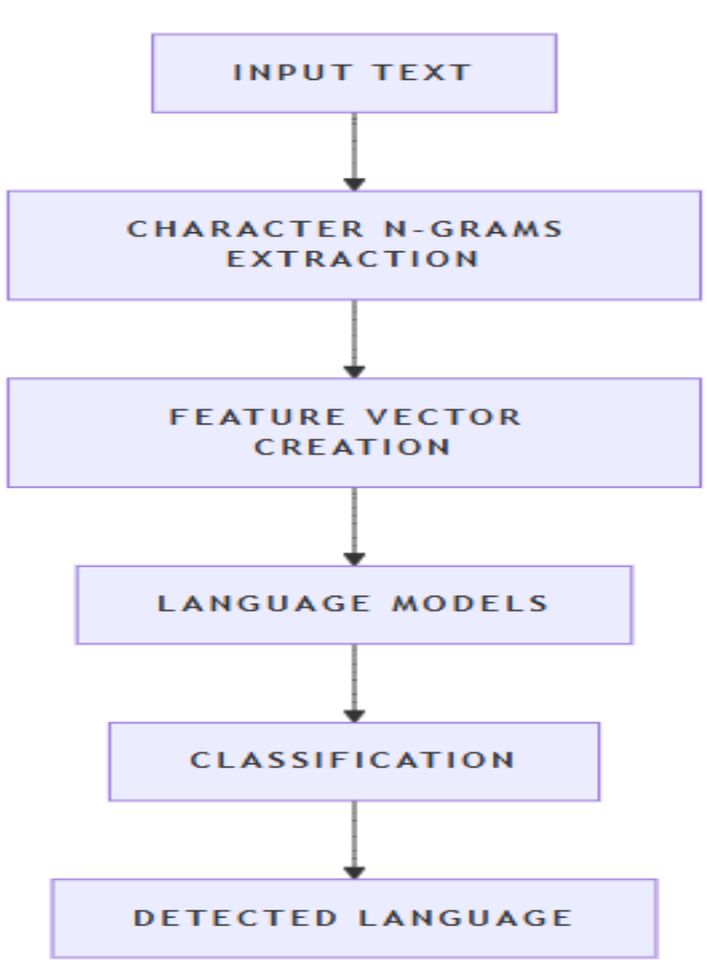

Fig 3: Architecture of langDetect

3.1 Pseudo code

1. pip install langdetect
2. IMPORT langdetect, pandas as pd, langdetect,numpy as np,matplotlib.pyplot as plt, seaborn as sns
3. from sklearn.metrics IMPORT classification_report, confusion_matrix, roc_auc_score
4. from sklearn.preprocessing IMPORT LabelBinarizer
5. DEFINE FUNCTION detect_languages_batch_eval(texts, batch_size=1000):
   SET predicted_languages TO []
   FOR i IN range(0, len(texts), batch_size):
   SET batch TO texts[i:i + batch_size]
   SET batch TO [str(text) IF not pd.isnull(text) and isinstance(text, str) and len(text.strip()) > 0
   and any(c.isalpha() FOR c IN text) else " FOR text IN batch]
   SET language_codes TO [langdetect.detect(text) FOR text IN batch IF text]
   predicted_languages.extend(language_codes)
   OUTPUT(f"Processed {i + batch_size} out of {len(texts)} texts") # Progress tracking
   RETURN predicted_languages
   IF __name__ EQUALS "__main__":
   SET df1 TO pd.read_csv('/content/taskA_(index,text(eval.csv')
   SET all_sentences TO df1['text'].tolist()
   SET predicted_languages TO detect_languages_batch_eval(all_sentences)
   SET df1['predicted_language'] TO pd.Series(predicted_languages)
   df1.to_csv('output_predicted_onevaldata_langdetect.csv', index=False)
6. SET predicted_data TO pd.read_csv('/content/output_predicted_onevaldata_langdetect.csv')
7. SET evaluation_data TO pd.read_csv('/content/taskA_(index,label)eval.csv')
8. SET merged_data TO pd.merge(predicted_data, evaluation_data, on='index')
9. SET true_labels TO merged_data['label'].tolist()
10. SET predicted_labels TO merged_data['predicted_language'].tolist()

```
11. SET report TO classification_report(true_labels, predicted_labels)
12. OUTPUT(report)
13. DEFINE FUNCTION detect_languages_batch_test(texts, batch_size=1000):
        SET predicted_languages TO []
        FOR i IN range(0, len(texts), batch_size):
            SET batch TO texts[i:i + batch_size]
            SET batch TO [str(text) IF not pd.isnull(text) else '' FOR text IN batch]
            SET language_codes TO [langdetect.detect(text) FOR text IN batch]
            predicted_languages.extend(language_codes)
            OUTPUT(f"Processed {i + batch_size} out of {len(texts)} texts")  # Progress tracking
        RETURN predicted_languages
    IF __name__ EQUALS "__main__":
        SET df1 TO pd.read_csv('/content/taskA(index,text)test.csv')
        SET all_sentences TO df1['text'].tolist()
        SET predicted_languages TO detect_languages_batch_test(all_sentences)
        SET df1['predicted_language'] TO pd.Series(predicted_languages)
        df1.to_csv('output_predicted_ontestdata_langdetect.csv', index=False)
14. SET cm TO confusion_matrix(true_labels, predicted_labels)
15. plt.figure(figsize=(10, 8))
16. sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
17. plt.xlabel('Predicted labels')
18. plt.ylabel('True labels')
19. plt.title('Confusion Matrix')
20. plt.show()
21. SET lb TO LabelBinarizer()
22. lb.fit(true_labels)
23. SET y_true TO lb.transform(true_labels)
24. SET y_pred TO lb.transform(predicted_labels)
25. SET auc_scores TO roc_auc_score(y_true, y_pred, average=None)
26. FOR i, label IN enumerate(lb.classes_):
        OUTPUT(f"AUC FOR {label}: {auc_scores[i]}")
27. plt.figure(figsize=(10, 6))
28. plt.bar(range(len(auc_scores)), auc_scores)
29. plt.xlabel('Label')
30. plt.ylabel('AUC Score')
31. plt.title('AUC Score FOR Each Label')
32. plt.xticks(range(len(auc_scores)), lb.classes_)
33. plt.show()
34. macro_auc=roc_auc_score(y_true, y_pred, average='macro')
35. OUTPUT("Macro AUC Score:", macro_auc)
36. weighted_auc=roc_auc_score(y_true, y_pred, average='weighted')
37. OUTPUT("Weighted AUC Score:", weighted_auc)
38. SET overall_auc TO roc_auc_score(y_true, y_pred, average='macro')
39. OUTPUT("Overall AUC Score:", overall_auc)
```

5.2 Performance metrics

The performance metrics used for the 3 models are:

1. Accuracy

- Accuracy measures the proportion of correct predictions made by a model over the total number of predictions made.
- It is one of the most widely used metrics to evaluate the performance of a classification model.

$$Accuracy = \frac{True\ positive + True\ negative}{True\ postive + True\ negative + False\ positive + False\ Negative}$$

2. Precision

- Out of all the positive predicted, what percentage is truly positive.

- The precision value lies between 0 and 1.

$$Precision = \frac{True\ positive}{True\ positive + False\ positive}$$

3. Recall

- Out of the total positive, what percentage are predicted positive. It is the same as TPR (true positive rate).

$$Recall = \frac{True\ positive}{True\ positive + False\ negative}$$

4. F1-score

- It is the harmonic mean of precision and recall.
- It takes both false positive and false negatives into account.
- Therefore, it performs well on an imbalanced dataset.

$$F1 - score = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

5. AUC score

- The AUC-ROC curve, or Area Under the Receiver Operating Characteristic curve, is a graphical representation of the performance of a binary classification model at various classification thresholds.
- AUC stands for the Area Under the Curve, and the AUC curve represents the area under the ROC curve.
- It measures the overall performance of the binary classification model.
- Formula = 1 / (M * N) * Σ(i=1 to M) Σ(j=1 to N) I(yi > yj)

  M is the number of positive instances
  N is the number of negative instances.
  y_i is the predicted probability of the i-th positive instance.
  y_j is the predicted probability of the j-th negative instance.
  I($y_i > y_j$) is an indicator function that returns 1 if yi > yj and 0 otherwise.

1. fastText

- **Classification report**

	PRECISION	***RECALL***	***F1-SCORE***	***SUPPORT***
1(ne)	1.00	0.89	0.94	2688
2(mr)	0.84	0.14	0.24	2381
3(sa)	0.99	0.70	0.82	2356
4(hi)	0.35	0.99	0.52	1663
ACCURACY			0.66	9088
MACRO AVERAGE	0.80	0.68	0.63	9088
WEIGHTED AVERAGE	0.84	0.66	0.65	9088

LABELS	***ne (1)***	***mr (2)***	***sa (3)***	***hi (4)***
AUC SCORE	0.9431	0.56515	0.85090	0.7918
MACRO-AVERAGE AUC	0.7877445384497166			
WEIGHTED-AVERAGE AUC	0.7924921459115644			
OVERALL AUC	0.7877445384497166			

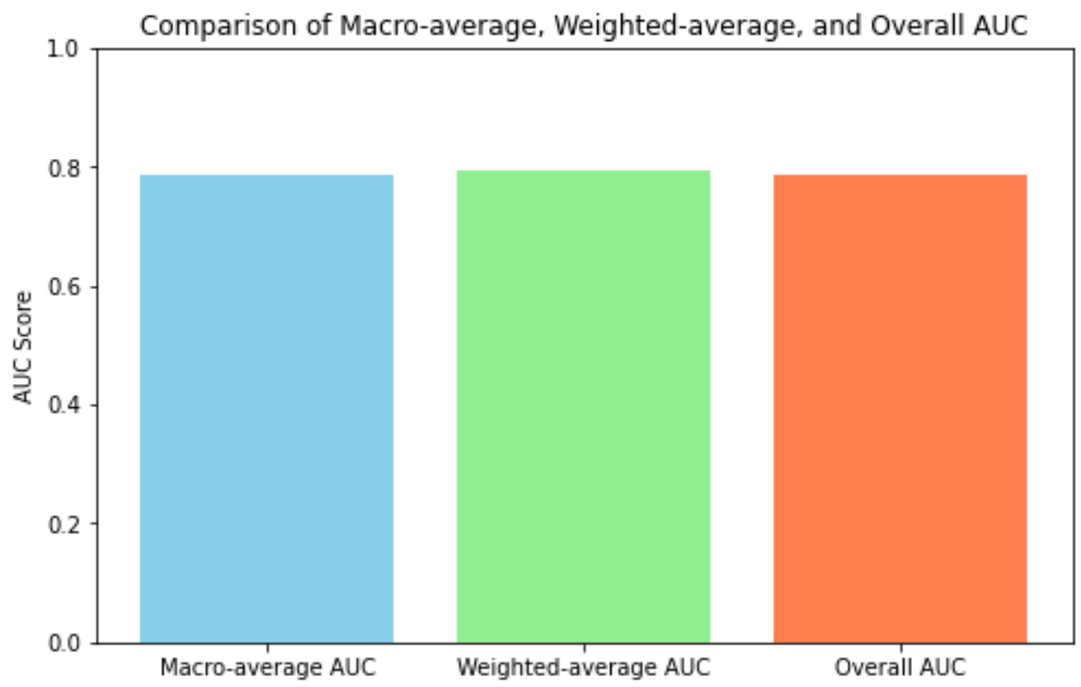

FIG 1: COMPARISON AUC SCORE FOR fastText

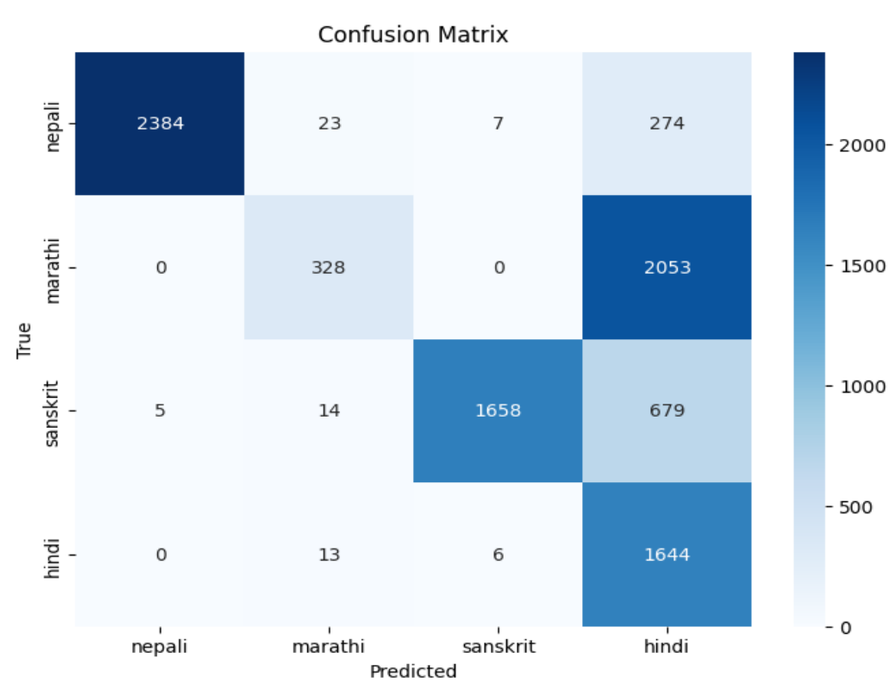

FIG 2: CONFUSION MATRIX FOR fastText

2. langID

- **Classification report**

	PRECISION	*RECALL*	*F1-SCORE*	*SUPPORT*
1(ne)	0.81	0.74	0.77	2688
2(mr)	0.18	0.16	0.17	2381
3(sa)	0.00	0.00	0.00	2356
4(hi)	0.32	0.90	0.48	1663
ACCURACY			0.42	9088
MACRO AVERAGE	0.33	0.45	0.35	9088
WEIGHTED AVERAGE	0.35	0.42	0.36	9088

LABELS	*ne (1)*	*mr (2)*	*sa (3)*	*hi (4)*
AUC SCORE	0.83243	0.4553	0.5	0.7391
MACRO-AVERAGE AUC		0.6317		
WEIGHTED-AVERAGE AUC		0.6303		
OVERALL AUC		0.6317		

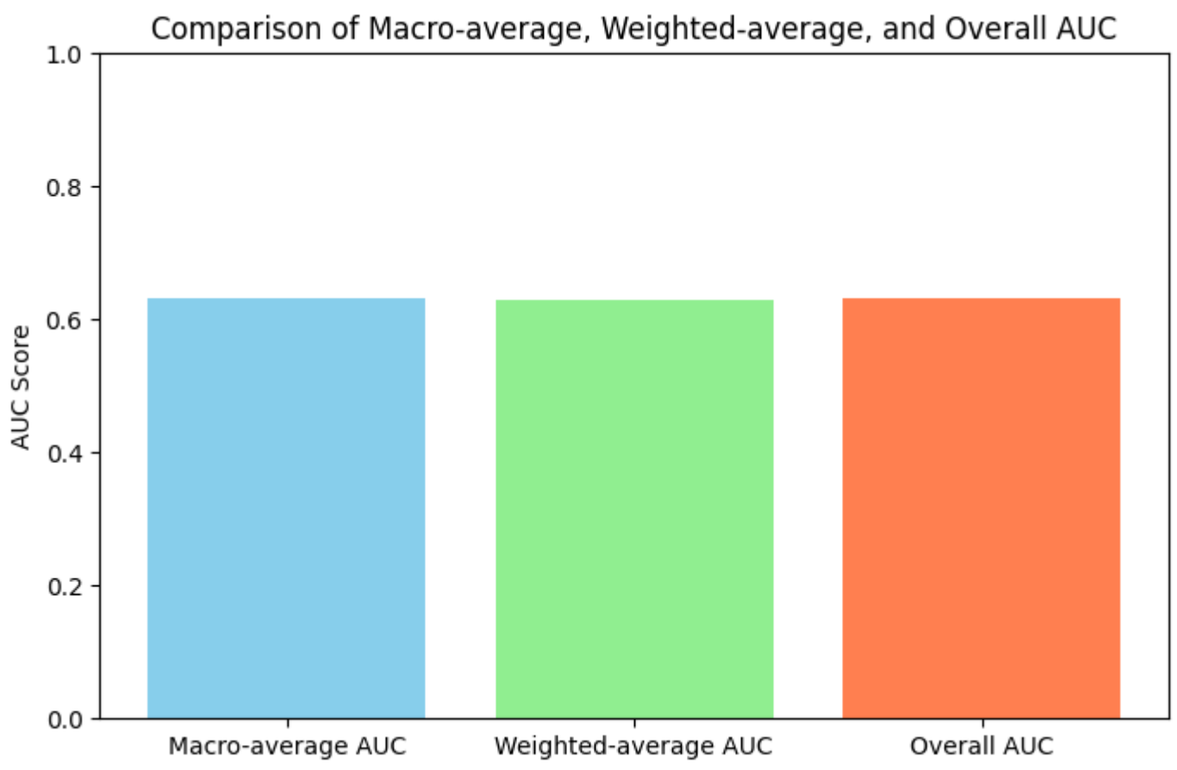

FIG 3: COMPARISON OF AUC SCORE FOR langID

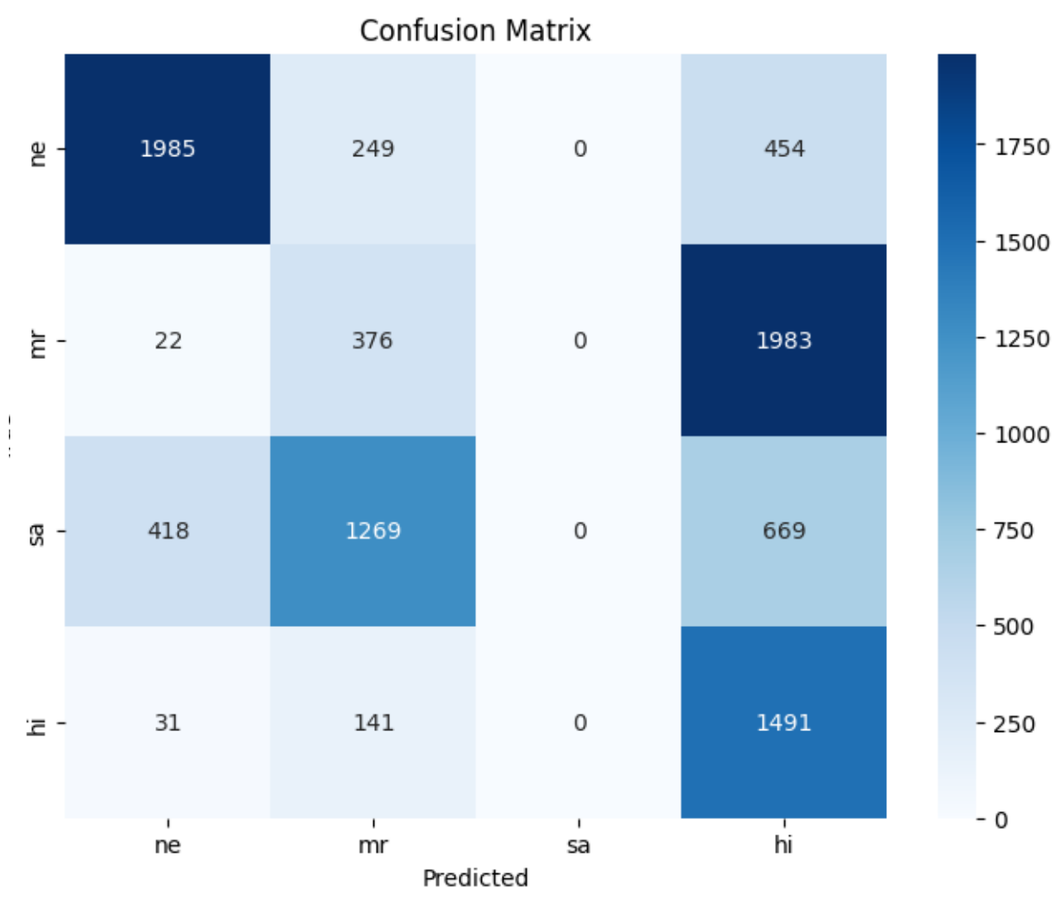

FIG 4: AUC SCORE FOR EACH CLASSS FOR langID

3. LangDetect

- **Classification report**

	PRECISION	*RECALL*	*F1-SCORE*	*SUPPORT*
1(ne)	0.81	0.96	0.88	2682
2(mr)	0.37	0.97	0.54	318
3(sa)	0.00	0.00	0.00	2356
4(hi)	0.54	0.98	0.70	1640
ACCURACY			0.42	9088
MACRO AVERAGE	0.33	0.45	0.35	9088
WEIGHTED AVERAGE	0.35	0.42	0.36	9088

- **AUC score**

LABELS	*ne(1)*	*mr (2)*	*sa (3)*	*hi (4)*
AUC SCORE	0.9117	0.94871	0.5	0.8630
MACRO-AVERAGE AUC		0.8058		
WEIGHTED-AVERAGE AUC		0.7633		
OVERALL AUC		0.8058		

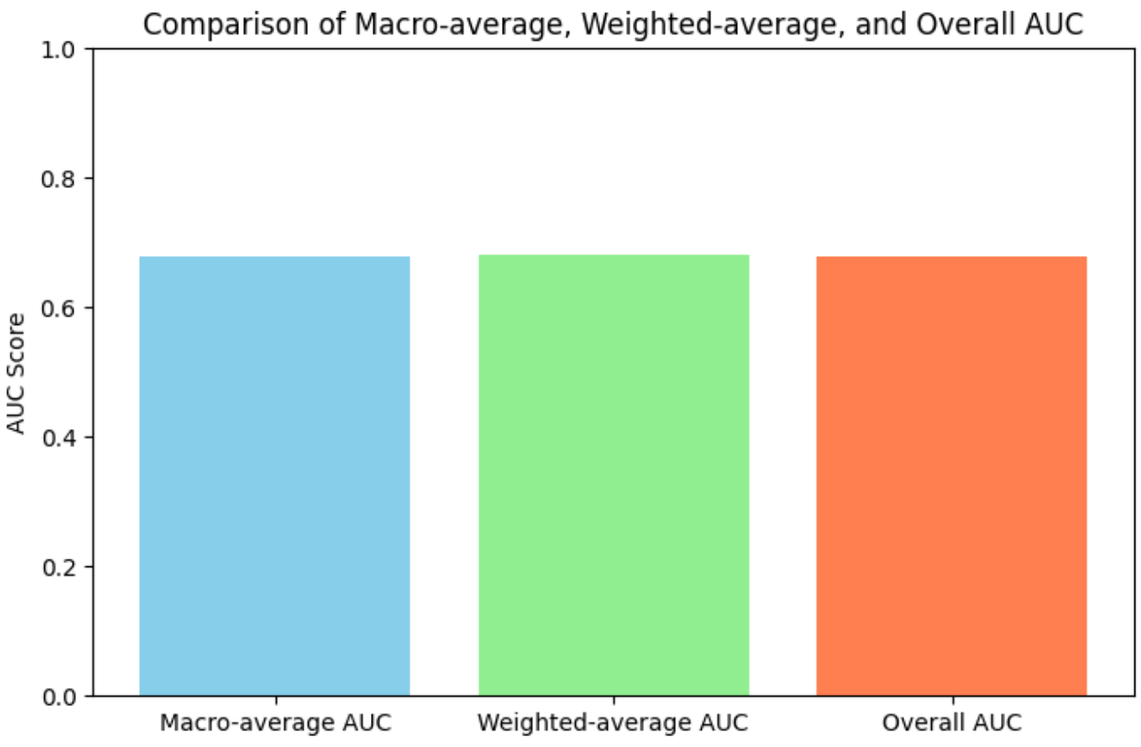

FIG 3: COMPARISON OF AUC SCORE FOR langDetect

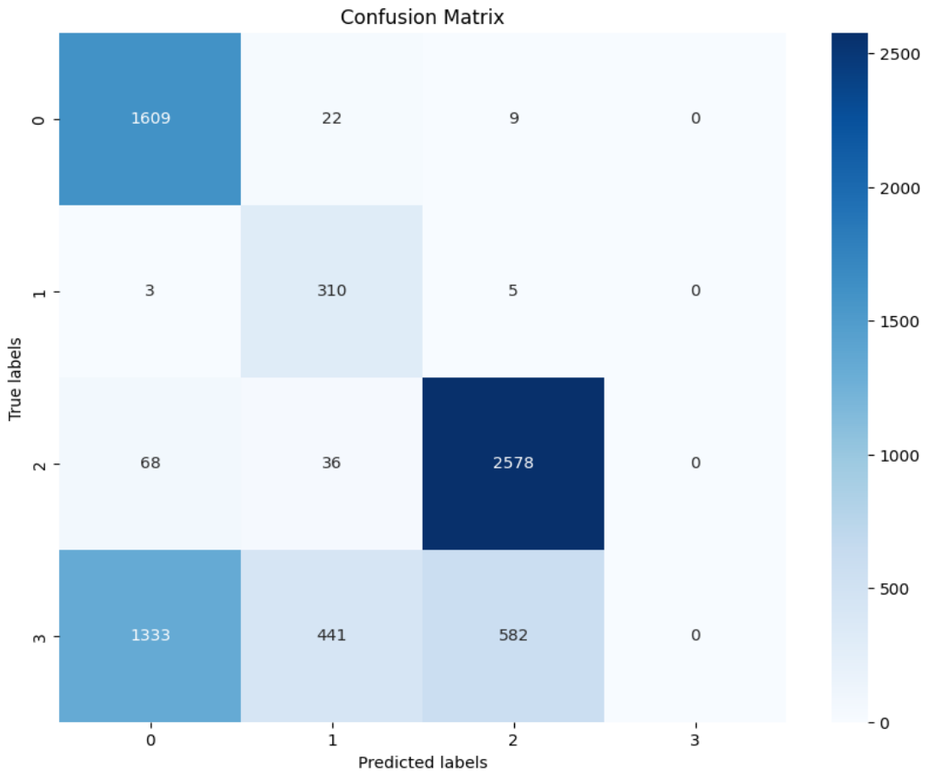

FIG 4: AUC SCORE FOR EACH CLASSS FOR LangDetect

5.3 Overall AUC score for fastText, langID, LangDetect

	fasttext	***langID***	***LangDetect***
Overall AUC Score	0.7877	0.6317	0.8058

5.4 Results and discussions

Even though LangDetect has an overall AUC Score of 80% when compared to langID and fasttext, we choose fasttext due the following reasons:

- LangDetect is a pre-trained model for 55 languages, it cannot be fine-tuned. Due to the imbalance of data in distribution through the languages (Hindi, Marathi, Sanskrit, Nepali) langDetect cannot predict Sanskrit sentences accurately.
- Due to the imbalance in data, we choose fasttext which works well with large datasets, handles data imbalance, handles noise in data.
- We prefer choosing fasttext because the classification report shown for langDetect and langID shows discrepancies which leads to the elimination of langID and langDetect.

6. OUTCOMES

- FastText is computationally efficient in processing, allowing for rapid language detection, especially for large datasets.
- Model robust to noise and errors in the text, making it suitable for real-world applications.
- Handling of Out of Vocabulary Words.
- Effective handling of code-switching, where multiple languages are used within the same text
- Accurate identification of individuals and groups targeted by hate speech.
- Successful classification of hate speech targets across multiple languages.
- Automated alerts generated for hate speech targeting specific groups.
- Scalable identification of hate speech targets across multiple languages and large platforms.

7. TIMELINE OF THE PROJECT

SUBTASKS	TITLE	TIMELINE
Subtask A	Devanagari Language Detection	25th September-11th October
Subtask B	Devanagari Hate Speech Detection	12th October-25th October
Subtask C	Devanagari Target Identification	10th November-18th November

8. CONCLUSION

A crucial component of text mining is text classification, particularly considering the abundance of internet data. The difficulty of identifying texts has increased due to the widespread use of social media, especially in linguistically diverse nations like India. With an accuracy of 66%, the text categorization model FastText beats several of the contemporary deep learning methods. Because of linguistic and cultural variety, dealing with hate speech in languages that utilize the Devanagari script—such as Hindi, Marathi, and others—presents special difficulties. Advanced natural language processing methods such as FastText can improve the detection of hate speech and the people who spread it. By enhancing content moderation, the implementation of such systems enhances online safety; nonetheless, continuous improvements are necessary to accommodate changing language usage and the emergence of hate speech trends.

Our approach determined goals with the help of our methodology, targets—individuals or groups—that hate speech is aimed towards were effectively identified. Comprehending the purpose is essential to understanding the sociocultural dimensions of hate speech, since it facilitates the creation of more focused interventions and the control of its spread. Notwithstanding the advancements, challenges remain in managing caustic or coded language and differentiating between explicit and implicit targeting.

REFERENCES

[1] Haifeng Wang a, Jiwei Li, Hua Wu, Eduard Hovy, Yu Sun. Pre-Trained Language Models and Their Applications. In Baidu Inc., Beijing 100193, China.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov. Enriching Word Vectors with Subword Information. TACL 2017.

[3] Saurav Singla, Vikash Kumar. Multi-Class Sentiment Classification using Machine Learning and Deep Learning Techniques. International Journal of Computer Sciences and Engineering 8(11):14-20

[4] Thapa, S., Jafri, F. A., Rauniyar, K., Nasim, M., & Naseem, U. (2024, May). RUHate-MM: Identification of Hate Speech and Targets using Multimodal Data from Russia-Ukraine Crisis. In Companion Proceedings of the ACM on Web Conference 2024 (pp. 1854-1863).

[5] Calderón, F. H., Balani, N., Taylor, J., Peignon, M., Huang, Y. H., & Chen, Y. S. (2021). Linguistic patterns for code word resilient hate speech identification. Sensors, 21(23), 7859.

[6] Yoder, M. M., Ng, L. H. X., Brown, D. W., & Carley, K. M. (2022). How hate speech varies by target identity: a computational analysis. arXiv preprint arXiv:2210.10839.

[7] Chiril, P., Pamungkeys, E. W., Benamara, F., Moriceau, V., & Patti, V. (2022). Emotionally informed hate speech detection: a multi-target perspective. Cognitive Computation, 1-31.

[8] Mossie, Z., & Wang, J. H. (2020). Vulnerable community identification using hate speech detection on social media. Information Processing & Management, 57(3), 102087.

[9] K.E. Abdelfatah, G. Terejanu, A.A. Alhelbawy. Unsupervised detection of violent content in Arabic social media. Comput. Sci. Inf. Technol. (CS IT) (2017), pp. 1-7

[10] Abozinadah, E.A., Jones Jr, J.H., 2017. A statistical learning approach to detect abusive twitter accounts, in: Proceedings of the International Conference on Compute and Data Analysis, pp. 6–13.

[11] E.A. Abozinadah, A.V. Mbaziira, J. Jones. Detection of abusive accounts with Arabic tweets. Int. J. Knowl. Eng.-IACSIT, 1 (2015), pp. 113-119

[12] A. Alakrot, L. Murray, N.S. Nikolov. Towards correct detection of offensive language in online communication in Arabic Procedia computer science, 142 (2018), pp. 315-320

[13] Albadi, N., Kurdi, M., Mishra, S., 2018. Are they, our brothers? analysis and detection of religious hate speech in the Arabic twittersphere, in: Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ACM. pp. 69–76.