International Institute of Information Technology Bangalore

AI 829
NATURAL LANGUAGE PROCESSING

Mandate 4

Hate and Offensive Speech Detection in Hindi and Marathi Social Media Texts

MT2023158 - Manasi Purkar
MT2023051 - Sanket Patil

GitHub🔗

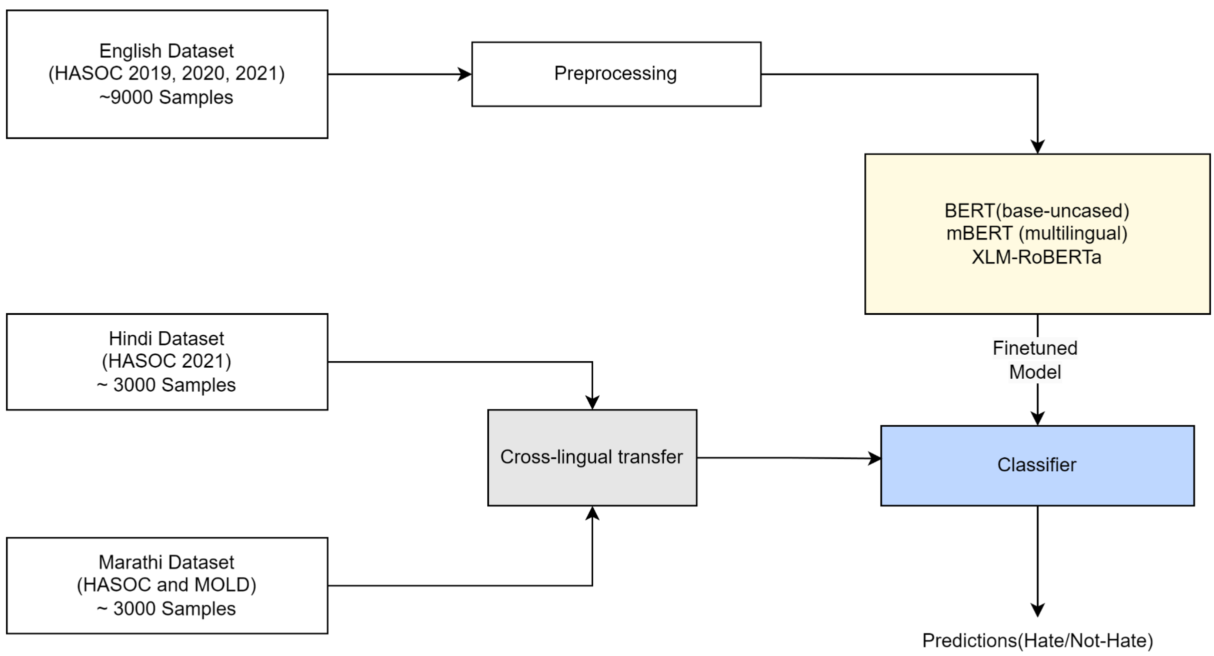
Table of Contents

Problem Statement:

As the use of social media platforms continues to grow, so does the incidence of harmful content, including hate speech, posted online.
This project aims to tackle the issue of hate speech in the Hindi and Marathi languages, two prominent languages spoken in India. There is a lack of attention for low-resource languages like Hindi and Marathi in the domain of hate speech detection. The project leverages deep learning approaches, to classify text as hate or non-hate. The datasets used are sourced from the HASOC and MOLD, focusing on Twitter posts in Hindi and Marathi. The dataset consists of binary labels.

We have Implemented **cross lingual transfer learning** approach to evaluate and tried to improve performance of target languages(Marathi, Hindi) with limited training data. The ultimate objective is to provide effective tools for automatically identifying and mitigating hate speech in online content written in Hindi and Marathi.

Flow Diagram :



Previous mandate submissions overview:

Mandate 2:

- In mandate 2, we focused on **lexical processing** of English, Hindi and Marathi text data. We utilized the HASOC 2021, 2020, 2019 dataset for english, HASOC 2021 dataset for Hindi and a combination of HASOC and MOLD datasets for Marathi.
- Our processing steps included removing punctuation marks and special characters, hashtags and URLs, which often clutter text data.
- We also tackled the challenge of handling emojis, a common element in online communication.
- Along with it we implemented removing stop words (common words with little value for analysis), converting all text to lowercase to maintain consistency, tokenization(breaking text into individual words or tokens), lemmatization and sentence embedding. This thorough lexical processing aims to clean and prepare the data for further analysis effectively.
- **Libraries used**- Pandas, re, NumPy, stopwordsiso, emoji, scikit-learn, WordNetLemmatizer, BertTokenizer, fasttext, NLTK.

Mandate 3:

- In mandate 3, we conducted **syntactic processing** of the data. Which involves breaking down sentences into their grammatical components, such as nouns, verbs, adjectives, and their relationships. Aim was to understand the roles played by each of the words in the sentence, and the relationship among words and to parse the grammatical structure of sentences to understand the proper meaning of the sentence.
- Furthermore, we conducted model selection and **fine-tuning of BERT Based models** on English text for a Hate speech detection task. We compared various models' performances to determine the most effective approach. We also analysed results of fine tuning multilingual bert based models on limited target language data.
- **Models used** -
  - BERT(bert-base-uncased)
  - mBERT(bert-multilingual-base-uncased)
  - XLM-RoBERTa

Results of Mandate 3 -

Hate speech detection evaluation using **F1-score**(average=macro):

Model	Marathi	Hindi	English
XLM-R	0.88	0.80	0.84
mBERT	0.78	0.73	0.82
Bert base	-	-	0.838

XMLR model is giving favorable results across all datasets. For Hindi and Marathi, we initially analyzed the results by directly fine-tuning models on the available data. However, the number of samples for Marathi and Hindi is limited. Subsequently, we applied cross-lingual transfer techniques to the Hindi and Marathi texts.

Mandate 4 Goal:

- Along with models evaluated in previous mandate, in this mandate we evaluated the Hindi and Marathi datasets using the **IndicBERT and Finetuned MURIL** model.

- We applied **cross-lingual transfer learning** on the finetuned models on english text to get results of Marathi and Hindi texts. In this we tried two methods without giving any example of target language and by providing some samples of target language in training.

- Further, we evaluated the results obtained from the **FLANT5** model, which is opensource **Instruction Finetuned LLM** . Without fine-tuning and using prompts, we examined the model's performance.

Cross lingual transfer learning:

Cross-lingual transfer learning involves transferring knowledge from one language to another, typically to improve the performance of natural language processing tasks in a target language for which there is a limited data or resources

One common approach is to train multilingual language models like mBERT (Multilingual BERT), XLM-R (Cross-lingual Language Model) to understand and generate text in multiple languages. These models are pre-trained on large, diverse corpora containing text from many languages.

The cross-lingual transfer enables tasks like zero-shot learning (applying the model to a language it has not seen during fine-tuning) or few-shot learning (adapting the model to a new language with minimal labelled examples).

In all results mentioned below 0 label represents Hate and 1 label represents Non Hate

A. Applying the XLM-RoBERTa and mBERT models to Hindi and Marathi languages, which they **have not encountered during fine-tuning**.

1. XLM-RoBERTa:

Evaluating on **Marathi** text:

```
import torch
from transformers import XLMRobertaModel
from torch import nn
from transformers import AutoTokenizer

class CustomXLMRobertaForClassification(nn.Module):
    def __init__(self, num_labels):
        super(CustomXLMRobertaForClassification, self).__init__()
        self.num_labels = num_labels
```

```
        self.roberta =
XLMRobertaModel.from_pretrained("xlm-roberta-base")
        self.classifier =
CustomRobertaClassificationHead(input_size=self.roberta.config.hidden_s
ize, hidden_size=768, num_labels=num_labels)
    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.roberta(input_ids=input_ids,
attention_mask=attention_mask)
        pooled_output = outputs.pooler_output
        logits = self.classifier(pooled_output)
        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()
            loss = loss_fct(logits.view(-1, self.num_labels),
labels.view(-1))
            return loss,logits
        else:
            return logits
tokenizer = AutoTokenizer.from_pretrained('xlm-roberta-base')
num_labels=2
# Load model state dict
model = CustomXLMRobertaForClassification(num_labels)
model.load_state_dict(torch.load('/kaggle/input/xmlr-custom-model/custo
m_model.pth',map_location=torch.device('cpu')))
```

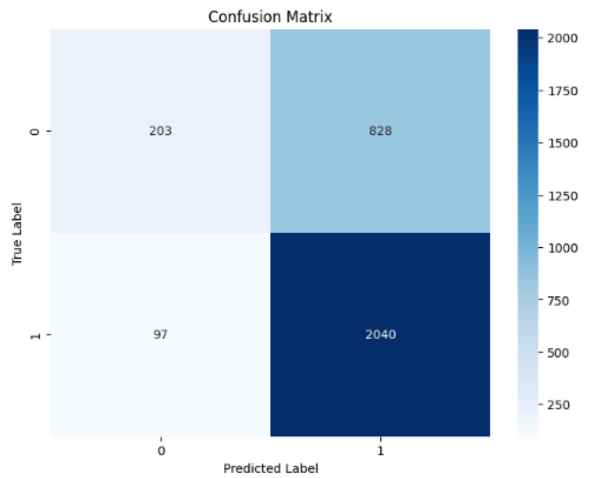
```
from sklearn.metrics import classification_report
# Testing
model.eval()
test_preds = []
test_labels_all = []
with torch.no_grad():
    for mb_x, mb_m, mb_y in test_dataloader:
        mb_x = mb_x.to(device)
        mb_m = mb_m.to(device)
        mb_y = mb_y.to(device)
        outputs = model(mb_x, attention_mask=mb_m, labels=mb_y)
        test_preds += torch.argmax(outputs[1], dim=1).cpu().tolist()
        test_labels_all += mb_y.cpu().tolist()

# Compute and print classification report for testing
test_report = classification_report(test_labels_all, test_preds,
target_names=['hate', 'non-hate']) #0 for hate
print(f'Testing report:\n{test_report}')
```

Tried different hyperparameter tunings(few results added here)

Learning rate=2e-5
Training Batch size=16
Testing Batch size(Marathi)=16
No. of epochs=5
Training max Token length=116
Testing max Token length=80

Testing report:

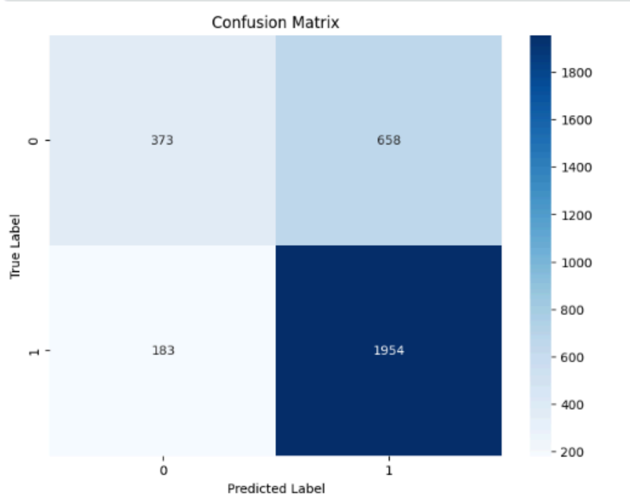
	precision	recall	f1-score	support
hate	0.68	0.20	0.31	1031
non-hate	0.71	0.95	0.82	2137
accuracy			0.71	3168
macro avg	0.69	0.58	0.56	3168
weighted avg	0.70	0.71	0.65	3168



Learning rate=1e-5
Training Batch size=32
Testing Batch size(Marathi)=16
No. of epochs=5
Training max Token length=116
Testing max Token length=100
Making hate and non hate samples equal in training phase **by downsampling**

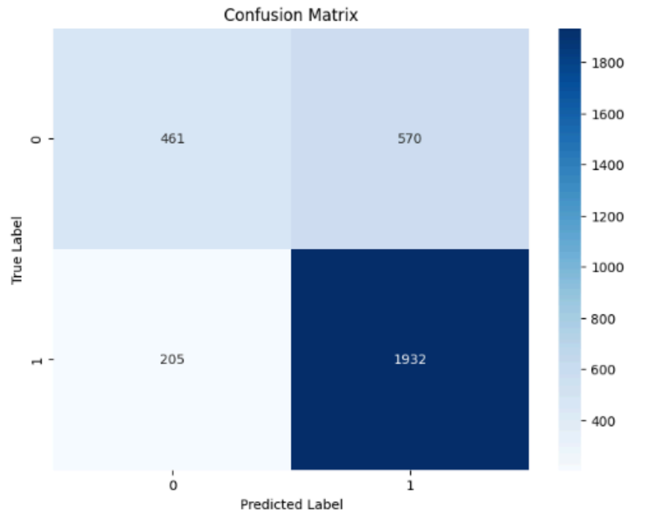
Testing report:

	precision	recall	f1-score	support
hate	0.67	0.36	0.47	1031
non-hate	0.75	0.91	0.82	2137
accuracy			0.73	3168
macro avg	0.71	0.64	0.65	3168
weighted avg	0.72	0.73	0.71	3168



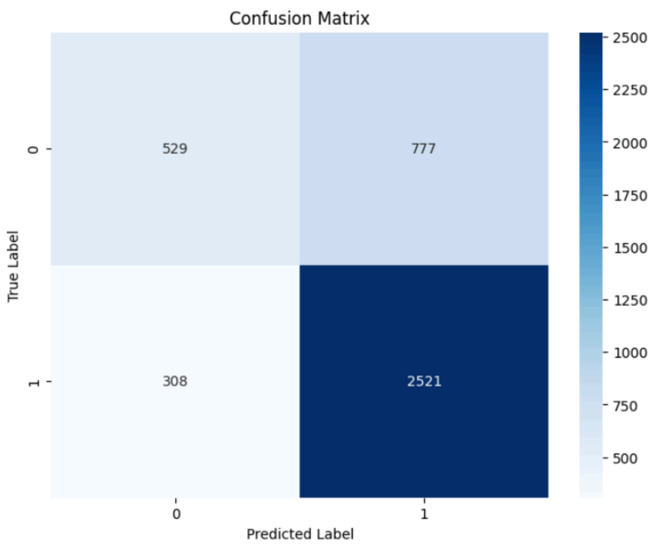
With same hyperparameters but keeping complete data (not making equal size of hate and non hate) in training

```
Testing report:
              precision    recall  f1-score   support

        hate       0.69      0.45      0.54      1031
    non-hate       0.77      0.90      0.83      2137

    accuracy                           0.76      3168
   macro avg       0.73      0.68      0.69      3168
weighted avg       0.75      0.76      0.74      3168
```



Evaluating XLM-RoBERTa model on **Hindi** text:

```
Testing report:
              precision    recall  f1-score   support

        hate       0.63      0.41      0.49      1306
    non-hate       0.76      0.89      0.82      2829

    accuracy                           0.74      4135
   macro avg       0.70      0.65      0.66      4135
weighted avg       0.72      0.74      0.72      4135
```



2. mBERT:

Evaluating on **Hindi** text:

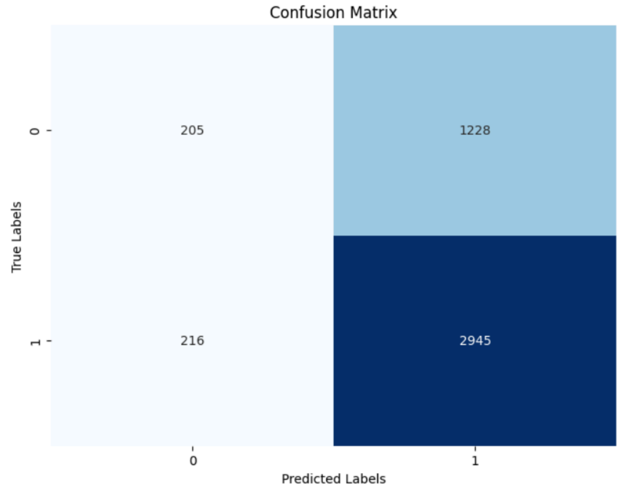
```
# Prediction on hindi dataset
print('Predicting labels for {:,} test
sentences...'.format(len(input_ids)))

# Put model in evaluation mode
model.eval()

# Tracking variables
predictions , true_labels = [], []

# Predict
count =0
for batch in prediction_dataloader:
    # Add batch to GPU
    batch = tuple(t.to(device) for t in batch)

    # Unpack the inputs from our dataloader
    b_input_ids, b_input_mask, b_labels = batch

    # Telling the model not to compute or store gradients, saving
memory and
    # speeding up prediction
    with torch.no_grad():
        # Forward pass, calculate logit predictions.
        count+=len(b_input_ids)
        result = model(b_input_ids,
                       token_type_ids=None,
                       attention_mask=b_input_mask,
                       return_dict=True)

        logits = result.logits

        # Move logits and labels to CPU
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()



        # Store predictions and true labels
```

```
        predictions.append(logits)
        true_labels.append(label_ids)
print(f"count = {count}")
print('     DONE.')
```

F1 Score: 0.51

```
from sklearn.metrics import f1_score

# Combine the predictions from all batches
all_predictions = np.concatenate(predictions, axis=0)
all_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
predicted_labels = np.argmax(all_predictions, axis=1)

# Compute F1 score
f1 = f1_score(all_true_labels, predicted_labels, average='macro'

print("F1 Score:", f1)
```

```
F1 Score: 0.5121261409304068
```




```
from sklearn.metrics import classification_report

target_names = ['class 0', 'class 1']
print(classification_report(all_true_labels, predicted_labels, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.49      0.14      0.22      1433
     class 1       0.71      0.93      0.80      3161

    accuracy                           0.69      4594
   macro avg       0.60      0.54      0.51      4594
weighted avg       0.64      0.69      0.62      4594
```

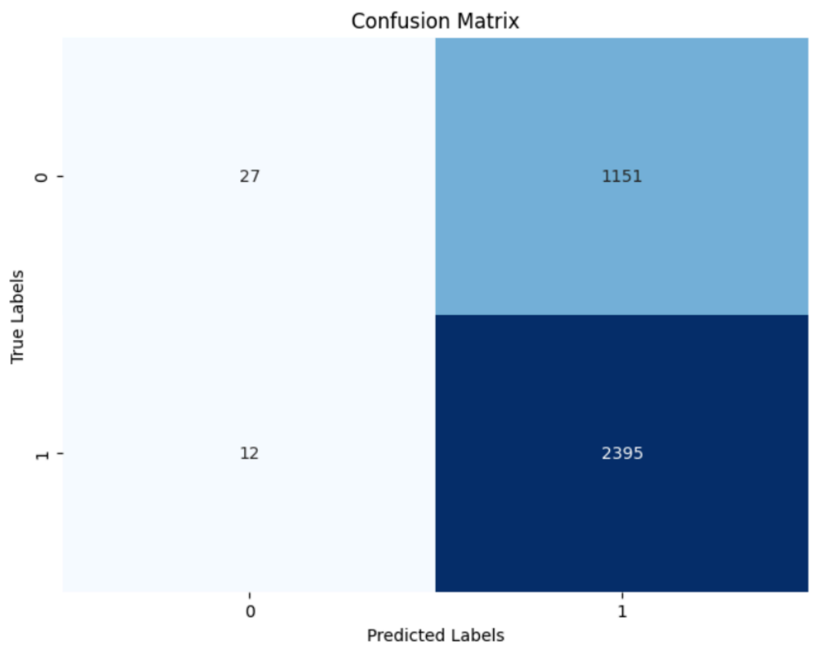
Evaluating mBERT model on **Marathi** text:

```
# Prediction on marathi dataset
print('Predicting labels for {:,} test
sentences...'.format(len(input_ids)))

# Put model in evaluation mode
model.eval()

# Tracking variables
predictions , true_labels = [], []

# Predict
count =0
for batch in prediction_dataloader:
    # Add batch to GPU
    batch = tuple(t.to(device) for t in batch)

    # Unpack the inputs from our dataloader
    b_input_ids, b_input_mask, b_labels = batch

    # speeding up prediction
    with torch.no_grad():
        # Forward pass, calculate logit predictions.
        count+=len(b_input_ids)
        result = model(b_input_ids,
                       token_type_ids=None,
                       attention_mask=b_input_mask,
                       return_dict=True)
        logits = result.logits

        # Move logits and labels to CPU
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        # Store predictions and true labels
        predictions.append(logits)
        true_labels.append(label_ids)

print(f"count = {count}")
print('    DONE.')
```

F1 Score: 0.42

```python
from sklearn.metrics import f1_score

# Combine the predictions from all batches
all_predictions = np.concatenate(predictions, axis=0)
all_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
predicted_labels = np.argmax(all_predictions, axis=1)

# Compute F1 score
f1 = f1_score(all_true_labels, predicted_labels, average='macro')

print("F1 Score:", f1)
```

```
F1 Score: 0.4245038614587205
```



B. Providing a **few Hindi and Marathi text examples** alongside the English dataset during fine tuning to guide the model’s predictions.

XLM-RoBERTa:

1. Evaluating on Marathi text:

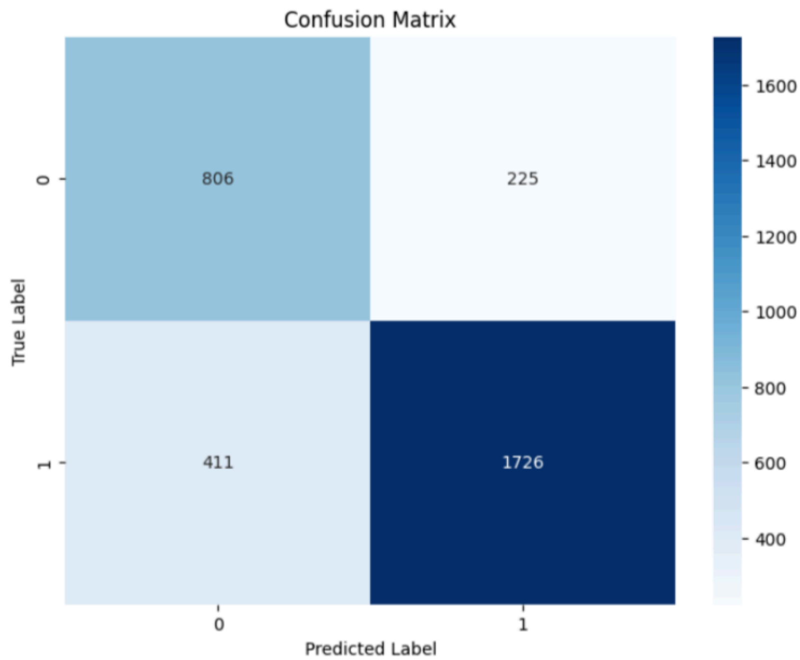
Marathi samples added = 351 (10% of data)

```
Training report after iteration 1:
              precision    recall  f1-score   support

         NOT       0.86      0.83      0.85      3804
         HOF       0.82      0.85      0.84      3425

    accuracy                           0.84      7229
   macro avg       0.84      0.84      0.84      7229
weighted avg       0.84      0.84      0.84      7229

Validation report after iteration 1:
              precision    recall  f1-score   support

         NOT       0.75      0.93      0.83       475
         HOF       0.89      0.66      0.76       429

    accuracy                           0.80       904
   macro avg       0.82      0.79      0.79       904
weighted avg       0.82      0.80      0.80       904

Time: 1.0m 24.351079702377732s
Training report after iteration 2:
              precision    recall  f1-score   support

         NOT       0.87      0.89      0.88      3804
         HOF       0.87      0.86      0.87      3425

    accuracy                           0.87      7229
   macro avg       0.87      0.87      0.87      7229
weighted avg       0.87      0.87      0.87      7229

Validation report after iteration 2:
              precision    recall  f1-score   support

         NOT       0.80      0.89      0.84       475
         HOF       0.86      0.75      0.80       429

    accuracy                           0.83       904
   macro avg       0.83      0.82      0.82       904
weighted avg       0.83      0.83      0.82       904
```

```
Training report after iteration 3:
              precision    recall  f1-score   support

         NOT       0.90      0.91      0.91      3804
         HOF       0.90      0.89      0.89      3425

    accuracy                           0.90      7229
   macro avg       0.90      0.90      0.90      7229
weighted avg       0.90      0.90      0.90      7229

Validation report after iteration 3:
              precision    recall  f1-score   support

         NOT       0.84      0.87      0.85       475
         HOF       0.85      0.81      0.83       429

    accuracy                           0.84       904
   macro avg       0.84      0.84      0.84       904
weighted avg       0.84      0.84      0.84       904

Time: 1.0m 23.737436294555664s
Training report after iteration 4:
              precision    recall  f1-score   support

         NOT       0.92      0.93      0.92      3804
         HOF       0.92      0.91      0.91      3425

    accuracy                           0.92      7229
   macro avg       0.92      0.92      0.92      7229
weighted avg       0.92      0.92      0.92      7229

Validation report after iteration 4:
              precision    recall  f1-score   support

         NOT       0.83      0.88      0.85       475
         HOF       0.86      0.79      0.82       429

    accuracy                           0.84       904
   macro avg       0.84      0.84      0.84       904
weighted avg       0.84      0.84      0.84       904

Time: 1.0m 23.629955768585205s
```

Testing report:

	precision	recall	f1-score	support
NOT	0.66	0.78	0.72	1031
HOF	0.88	0.81	0.84	2137
accuracy			0.80	3168
macro avg	0.77	0.79	0.78	3168
weighted avg	0.81	0.80	0.80	3168

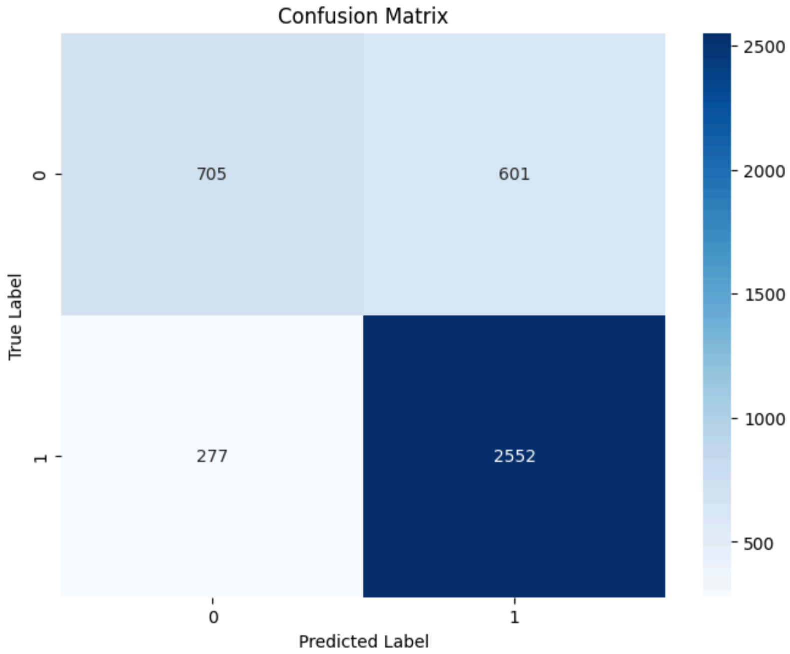


2. Evaluating on Hindi Text:

Hindi samples added= 459 (10% of data)

Testing report:

	precision	recall	f1-score	support
NOT	0.72	0.54	0.62	1306
HOF	0.81	0.90	0.85	2829
accuracy			0.79	4135
macro avg	0.76	0.72	0.73	4135
weighted avg	0.78	0.79	0.78	4135

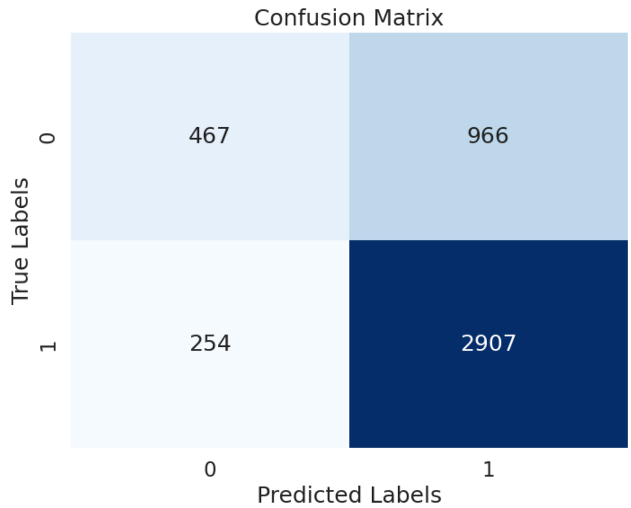


mBERT:

1. Evaluating on Marathi text:

```
from sklearn.metrics import f1_score

# Combine the predictions from all batches
all_predictions = np.concatenate(predictions, axis=0)
all_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
predicted_labels = np.argmax(all_predictions, axis=1)

# Compute F1 score
f1 = f1_score(all_true_labels, predicted_labels, average='macro')

print("F1 Score:", f1)
```

```
F1 Score: 0.6300843046732293
```



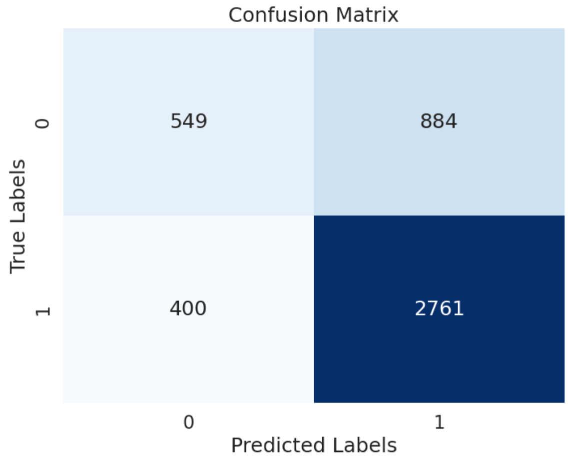
	precision	recall	f1-score	support
class 0	0.65	0.33	0.43	1433
class 1	0.75	0.92	0.83	3161
accuracy			0.73	4594
macro avg	0.70	0.62	0.63	4594
weighted avg	0.72	0.73	0.70	4594

2. Evaluating on Hindi Text:

F1 Score: 0.636

```
from sklearn.metrics import f1_score

# Combine the predictions from all batches
all_predictions = np.concatenate(predictions, axis=0)
all_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
predicted_labels = np.argmax(all_predictions, axis=1)

# Compute F1 score
f1 = f1_score(all_true_labels, predicted_labels, average='macro')

print("F1 Score:", f1)
```

```
F1 Score: 0.6361500557738726
```



	precision	recall	f1-score	support
class 0	0.58	0.38	0.46	1433
class 1	0.76	0.87	0.81	3161
accuracy			0.72	4594
macro avg	0.67	0.63	0.64	4594
weighted avg	0.70	0.72	0.70	4594

C. Using Pretrained **Indic-BERT**:

Evaluating on **hindi** text:

```
from transformers import AutoModel, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained('ai4bharat/indic-bert')
model = AutoModel.from_pretrained('ai4bharat/indic-bert')
```

```
class IndicBertClassfier(torch.nn.Module):
    def __init__(self,bert_model,bert_config,num_class):
        super(IndicBertClassfier,self).__init__()

        self.bert_model=bert_model
        self.dropout=torch.nn.Dropout(0.1)
self.fc1=torch.nn.Linear(bert_config["hidden_size"],bert_config["hidden
_size"])
        self.fc2=torch.nn.Linear(bert_config["hidden_size"],num_class)
    def forward(self, token_ids, attention_mask=None):
        bert_out = self.bert_model(input_ids=token_ids,
attention_mask=attention_mask)[1]  # Sentence vector
        bert_out=self.dropout(bert_out)
        bert_out=self.fc1(bert_out)
        bert_out=self.dropout(bert_out)
        bert_out=self.fc2(bert_out) #[batch_size,num_class]
        return bert_out
```

```
config = {
  "model_type": "albert",
  "attention_probs_dropout_prob": 0,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0,
  "embedding_size": 128,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "num_hidden_groups": 1,
  "net_structure_type": 0,
```

```
  "gap_size": 0,
  "num_memory_blocks": 0,
  "inner_group_num": 1,
  "down_scale_factor": 1,
  "type_vocab_size": 2,
  "vocab_size": 200000
}
```

```
IndicBertClassfier = IndicBertClassfier(model,config,2)
preds = []
for steps,(token_ids,label) in tqdm(enumerate(test_dataload)):
    #print(steps)
    with torch.no_grad():
        token_ids = token_ids.to(device)
        attention_mask = (token_ids != 0).float()
        out = IndicBertClassfier(token_ids,
attention_mask=attention_mask)
        res = list(out.argmax(1))
        for r in res:
            preds.append(r.tolist())
```

F1 Score: 0.49

```
from sklearn.metrics import f1_score

# Compute F1 score
f1 = f1_score(test_labels, preds, average='macro')

print("F1 Score:", f1)
```

```
F1 Score: 0.4947286959534621
```

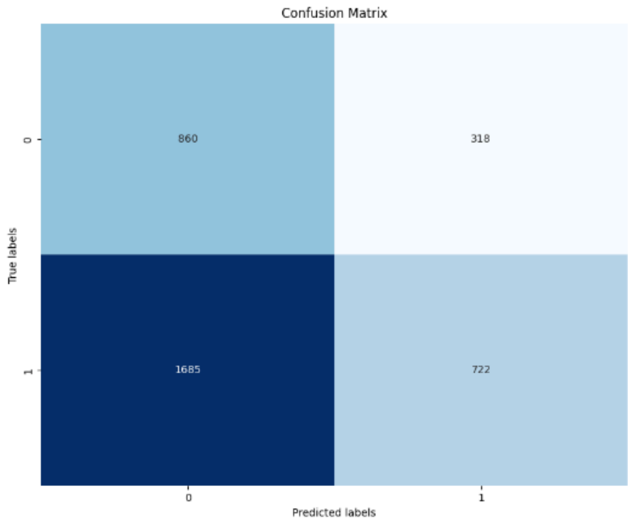
	precision	recall	f1-score	support
0	0.31	0.27	0.29	1433
1	0.69	0.72	0.70	3161
accuracy			0.58	4594
macro avg	0.50	0.50	0.49	4594
weighted avg	0.57	0.58	0.57	4594

Evaluating Indic-BERT on **marathi** text:

F1 Score: 0.44

```
from sklearn.metrics import f1_score

# Compute F1 score
f1 = f1_score(test_labels, preds, average='macro')

print("F1 Score:", f1)
```

```
F1 Score: 0.4404540074670497
```




```
from sklearn.metrics import classification_report

print(classification_report(test_labels, preds))
```

	precision	recall	f1-score	support
0	0.34	0.73	0.46	1178
1	0.69	0.30	0.42	2407
accuracy			0.44	3585
macro avg	0.52	0.52	0.44	3585
weighted avg	0.58	0.44	0.43	3585

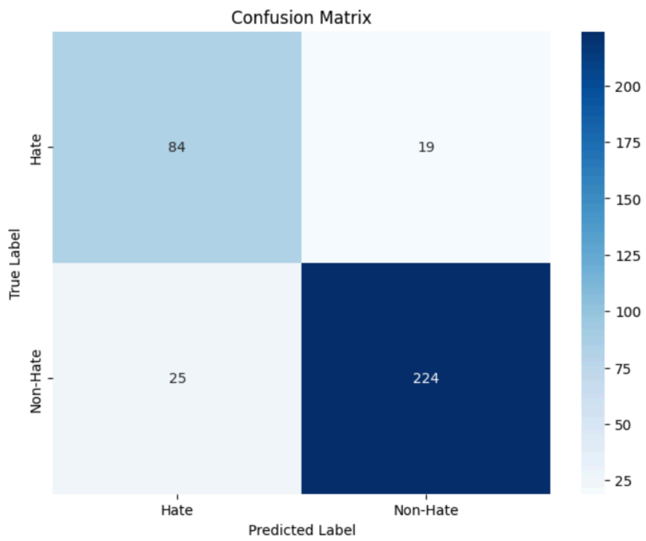
D. Evaluating on finetuned **MuRIL**

Finetuning on **Marathi** Text-

```
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.82      0.79       103
           1       0.92      0.90      0.91       249

    accuracy                           0.88       352
   macro avg       0.85      0.86      0.85       352
weighted avg       0.88      0.88      0.88       352
```

F1 Score- 0.91

```
from sklearn.metrics import f1_score

# After training the model using model_fit function

# 1. Make predictions on the validation set
pred_probs = model.predict(test_x)

# 2. Convert predicted probabilities to class labels
pred_labels = np.argmax(pred_probs, axis=1)

# 3. Calculate F1 score
f1 = f1_score(np.argmax(test_y, axis=1), pred_labels)

print("F1 Score:", f1)

11/11 [==============================] - 3s 280ms/step
F1 Score: 0.910569105691057
```



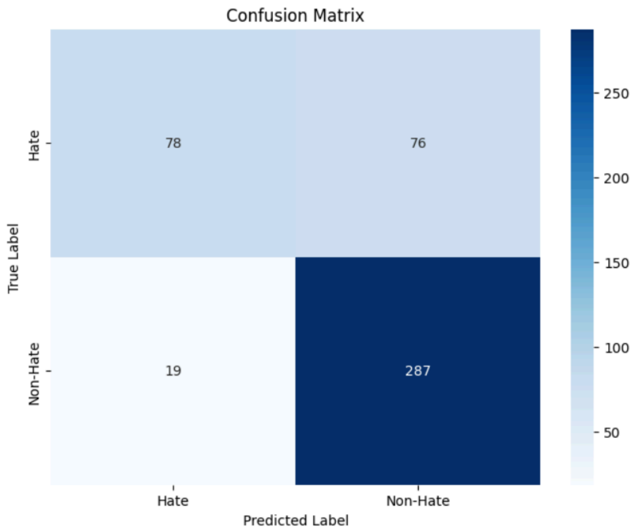
Finetuning on **Hindi** text-

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.51      0.62       154
           1       0.79      0.94      0.86       306

    accuracy                           0.79       460
   macro avg       0.80      0.72      0.74       460
weighted avg       0.80      0.79      0.78       460
```

F1 Score- 0.85

```
from sklearn.metrics import f1_score

# After training the model using model_fit function

# 1. Make predictions on the validation set
pred_probs = model.predict(test_x)

# 2. Convert predicted probabilities to class labels
pred_labels = np.argmax(pred_probs, axis=1)

# 3. Calculate F1 score
f1 = f1_score(np.argmax(test_y, axis=1), pred_labels)

print("F1 Score:", f1)
```

```
15/15 [==============================] - 4s 299ms/step
F1 Score: 0.8579970104633783
```

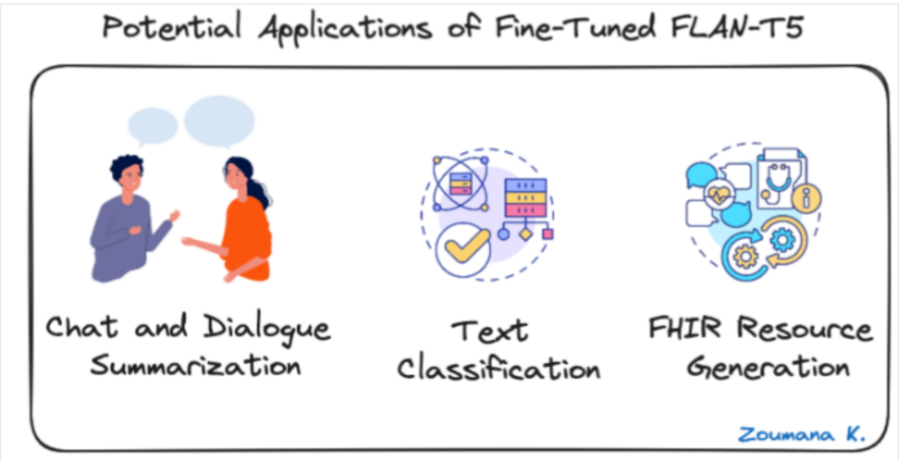


Instruction Fine Tuning-

Instruction fine-tuning trains the model using examples that demonstrate how it should respond to a specific instruction.
The LLM Eval module allows you to evaluate your fine-tuned model using prompt templates.It acts as a structured query or input that guides the model in generating the desired response.

FLAN T5-

FLAN-T5 is an open-source, sequence-to-sequence, large language model that can be also used commercially. The model was published by Google researchers in late 2022, and has been fine-tuned on multiple tasks. FLAN-T5 was released in the paper Scaling Instruction-Finetuned Language Models - it is an enhanced version of T5 that has been finetuned in a mixture of tasks. One can directly use FLAN-T5 weights without finetuning the model.

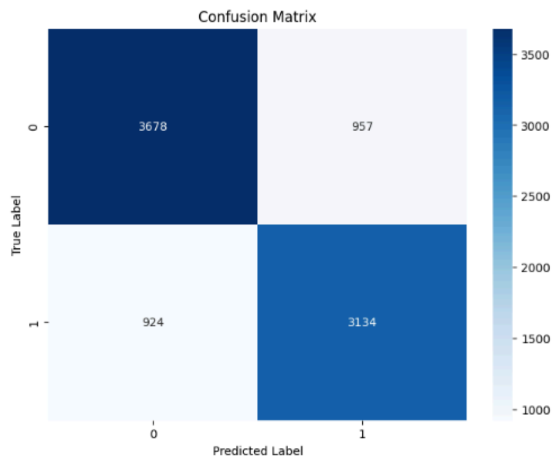


reference-https://www.datacamp.com/tutorial/flan-t5-tutorial

```
# Step 1: Initialize the prompting class instance
prompter2 = prompting(model="flant5")

# Step 2: Prepare your DataFrame with text and labels
data = df[['text', 'label']]

#Step 3: Initialize prompt template
prompt_template = """Classify this text as hate or non-hate.
The definition of hate speech is ‘Hate speech’ is
speech that attacks a person or group on the basis
of attributes such as race, religion, ethnic origin,
national origin, sex, disability, sexual orientation,
or gender identity.Examples:
1. Text: "bitch what"
   Answer: Hate
2. Text: "technically that is still turning back the clock dick head"
   Answer: Hate
3. Text: "fuck off this is you"
   Answer: Hate
   Text:"""
output_indicator = "Answer:"

# Step 3: Call the predict method of the prompting class
predictions2 = prompter2.predict(prompt_template=prompt_template, output_indicator=output_indicat

# Now predictions contains the generated predictions based on the input data
```

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.79	0.80	4635
1	0.77	0.77	0.77	4058
accuracy			0.78	8693
macro avg	0.78	0.78	0.78	8693
weighted avg	0.78	0.78	0.78	8693



```python
prompt_template = """Classify this text as hate or non-hate.
The definition of hate speech is ‘Hate speech’ is
speech that attacks a person or group on the basis
of attributes such as race, religion, ethnic origin,
national origin, sex, disability, sexual orientation,
or gender identity.Examples:
1. Text: "काही लोकांना सांगून उपयोग नाही त्यांच्या गुडघ्यात मेंदू आहे"
   Answer: hate
2. Text: "भिकारचोट पत्रकार दलाल साले"
   Answer: hate
3. Text: "बावळट लोकं सत्ताधारी पक्षात अपेक्षा करणार"
   Answer: hate
4. Text:"लवड्या अगोदरच रडायला लागलास तू"
   Answer:hate
Text:"
"""
output_indicator = "Answer:"


# Step 3: Call the predict method of the prompting class
predictions = prompter.predict(prompt_template=prompt_template, output_indicator=output_indicator, data=data)
```

```python
prompt_template = """हा मजकूर द्वेष किंवा द्वेष नसलेला म्हणून वर्गीकृत करा.
द्वेषयुक्त भाषणाची व्याख्या ‘हेट स्पीच’ अशी आहे
एखाद्या व्यक्तीवर किंवा गटाच्या आधारावर हल्ला करणारे भाषण
वंश, धर्म, वांशिक मूळ,
राष्ट्रीय मूळ, लिंग, अपंगत्व, लैंगिक प्रवृत्ती,
किंवा लिंग ओळख. उदाहरणे:
1. मजकूर: "काही लोकांना सांगून उपयोग नाही त्यांच्या गुडघ्यात मेंदू आहे"
   उत्तर: द्वेष
2. मजकूर: "भिकारचोट पत्रकार दलाल साले"
   उत्तर: द्वेष
3. मजकूर: "बावळट लोकं सत्ताधारी पक्षाची अपेक्षा करणार"
   उत्तर: द्वेष
4. मजकूर: "लवड्या अगोदरच रडायला लागलास तू"
   उत्तर: द्वेष
मजकूर:"""
output_indicator = " उत्तर:"


# Step 3: Call the predict method of the prompting class
predictions = prompter.predict(prompt_template=prompt_template, output_indicator=output_indicator, data=data)
```

```python
from sklearn.metrics import f1_score
# Calculate the F1 score
f1 = f1_score(true_labels, predicted_labels, average='macro')
print("F1 Score:", f1)
```

```
F1 Score: 0.40214067278287463
```

Results:

Cross-lingual hate speech detection evaluation using
F1-score(average=macro):

	Marathi		Hindi	
Model	On unseen language	With few samples alongside english data during finetuning	On unseen language	With few samples alongside english data during finetuning
XLM-R	0.69	0.79	0.70	0.72
mBERT	0.42	0.63	0.51	0.636

From different models and techniques used we observed that **XLM-R** Model gives the best performance on cross lingual transfer approach

Fine Tuning with only marathi, hindi text data evaluation using
F1-score(average=macro):

Model	Marathi	Hindi	English
XLM-R	0.88	0.80	0.84
mBERT	0.78	0.73	0.82
muRIL	0.85	0.79	-
Indicbert(Pretrained)	0.44	0.49	-
Bert base	-	-	0.838

Challenges Faced:

1. Finetuning Various Models
2. Finding open source LLM which gives results by giving prompts
3. Adding custom classification layer in XMLR model for english dataset
4. Less validation and test data for Indian Languages.
5. Identifying Suitable Stopwords Libraries for Hindi and Marathi.
6. Cleaning scrapped tweets in Hindi and Marathi involved addressing unwanted patterns and noise inherent in social media language.

Future Work:

1. Cross Lingual accuracy is still less we can try to improve it further using some FSL algorithms.
2. More LLMs can be tried like GPT3 and can try to increase accuracy by designing prompts efficiently.
3. More dataset can give better results.

Notebooks🔗

A. CrossLingual without providing any samples of target language

1. XLM-R:
    a. Marathi: xlm-r-crosslingual-marathi
    b. Hindi: hindi-cross-lingual-xmlr
2. mBERT:
    a. Marathi: cross-lingual-marathi
    b. Hindi: cross-lingual-hindi

B. After providing a few Hindi and Marathi text examples alongside the English dataset during fine tuning to guide the model's predictions:

1. XLM-R:
    a. Marathi:
       Marathi-CrossLingual-XML-with 2 methods
       Marathi-CrossLingual-XML-CombinedData Method

    b. Hindi:
       Hindi-CrossLingual-XMLR-with 2 methods

2. mBERT:
    a. Marathi- mbert-finetuned-eng-marathi
    b. Hindi- mbert-finetuned-eng-hindi

C. IndicBERT(Evaluation on the pretrained model):
    a. Hindi: indicbert-hindi
    b. Marathi: indicbert-marathi

D. Finetuning MuRIL-
    a. Marathi-MuRIL-Marathi
    b. Hindi- MuRIL-Hindi

E. Using Instruction finetuned LLM FlanT5 - Using FLANT5 LLM

Datasets -

https://drive.google.com/drive/u/2/folders/1IfrRIiTDk6NrsDe9mSJar3poT-MKzj4Y

References:

1. https://arxiv.org/abs/2401.03346
2. https://www.graphcore.ai/posts/flan-t5-sweet-results-with-the-smaller-more-efficient-llm
3. https://medium.com/dailymotion/how-we-used-cross-lingual-transfer-learning-to-categorize-our-content-c8e0f9c1c6c3
4. https://medium.com/neuralspace/indic-transformers-an-analysis-of-transformer-language-models-for-indian-languages-c6b4db0643b
5. https://indicnlp.ai4bharat.org/pages/indic-bert/
6. https://spotintelligence.com/2023/09/22/cross-lingual-transfer-learning/#:~:text=Cross%2Dlingual%20transfer%20learning%20is,be%20limited%20data%20or%20resources.
7. https://huggingface.co/google/muril-large-cased