International Institute of Information Technology Bangalore

AI 829

NATURAL LANGUAGE PROCESSING

Mandate 3

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

MT2023158 - Manasi Purkar
MT2023051 - Sanket Patil

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 shared task, focusing on Twitter posts in Hindi and Marathi. The dataset consists of binary labels.
We are going to Implement cross lingual transfer learning approach to improve performance of target languages 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.

Mandate 3 Goal:

- In Mandate 2, we identified the dataset and performed lexical processing on the data, focusing on data cleaning and feature extraction. Techniques employed included stemming, lemmatization, correcting misspelled words, etc for english, hindi and marathi text. In this mandate, 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 is 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.

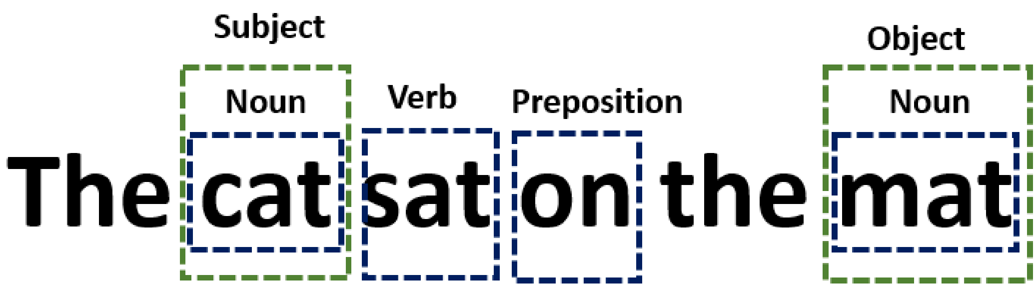
Libraries used:

- transformers
- PyTorch
- BertTokenizer
- BertForSequenceClassification
- numpy
- sklearn
- spacy
- stanza
- tensorflow

Syntactic Processing:

Syntactic processing is the process of analyzing the grammatical structure of a sentence to understand its meaning. This involves identifying the different parts of speech in a sentence, such as nouns, verbs, adjectives, and adverbs, and how they relate to each other in order to give proper meaning to the sentence.

For example, consider the sentence "*The cat sat on the mat*."
Syntactic processing would involve identifying important components in the sentence such as "cat" as a noun, "sat" as a verb, "on" as a preposition, and "mat" as a noun. It would also involve understanding that "cat" is the subject of the sentence and "mat" is the object.



Syntactic processing involves a series of steps, including tokenization, part-of-speech tagging, parsing, and semantic analysis.

Part-of-Speech (POS) tagging:

POS tagging involves assigning a grammatical category (such as noun, verb, adjective, etc.) to each word in a sentence. The goal is to understand the syntactic structure of a sentence and identify the grammatical roles of individual words. POS tagging provides essential information for text analysis.

POS tags: short codes representing specific parts of speech. Common POS tags include: Noun (NN), Verb (VB), Adjective (JJ), Adverb (RB), Pronoun (PRP), Preposition (IN), Conjunction (CC), Determiner (DT), Interjection (UH)

Words may have multiple possible POS tags based on context. For example, “lead” can be a noun (the metal) or a verb (to guide).

Role of POS tagging in Hate Speech Detection Task-

Understanding the syntactic structure of hate speech can provide insights into the context in which hateful content is expressed. Dependency parsing can reveal relationships between words and phrases, helping to identify the subject, object, and verb of sentences. This contextual understanding is crucial for accurately detecting hate speech, as hateful language often relies on specific linguistic cues and contextual factors.

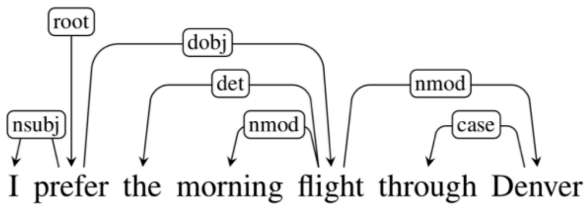
e.g:

yeah especially when there will be muslim political party in the state against one secular party bjp

(yeah, INTJ), (especially, ADV), (when, SCONJ), (there, PRON), (will, AUX), (be, AUX), (muslim, ADJ), (political, ADJ), (party, NOUN), (in, ADP), (the, DET), (state, NOUN), (against, ADP), (one, NUM), (secular, ADJ), (party, NOUN), (bjp, NOUN)

Dependency Parsing:

Dependency parsing is about unravelling the relationships between words in a sentence. Analyzing how words depend on one another constructs a tree-like structure known as a dependency tree or syntactic tree, which graphically represents the syntactic and semantic relationships within the sentence.



- Consists of relations between lexical items, normally binary, asymmetric relations (“arrows”) called dependencies
- The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc)

Libraries used for POS tagging and dependency parsing-

1. **spaCy-**

   The spaCy library’s POS tagger is based on a statistical model trained on the OntoNotes 5 corpus, and it can tag the text with high accuracy. It also can tag other features, like lemma, dependency, ner, etc.

As Indian languages are not available in spacy we used below libraries for Marathi and Hindi.

2. **Udpipe-**

UDPipe is a software tool and service that analyzes (plain) text in about 100 different natural languages up to the dependency syntax level. Users specify the desired function (tokenization, segmentation, morphological analysis, lemmatization, POS tagging, dependency parsing), output format, and input text (file(s)). The resulting analysis can be used to index and search documents by lemmas instead of multiple word forms, extract syntactic dependencies with POS information to get relations between words or lemmas, or get grammatical information for every word in the text. UDPipe software allow to train on any language for which a treebank is available in the CoNLL-U format, such as all the Universal Dependencies corpora.

3. **Stanza NLP-**

The Stanza NLP library takes it a bit further and is able to tag each word with their universal POS (UPOS) tags, universal morphological features (UFeats), and treebank-specific POS (XPOS) tags.

In the Stanza NLP library, each input word is returned with its corresponding lemma form by using the LemmaProcessor.

Syntactic parsing, Dependency parsing, is the task of assigning a syntactic structure to our text data and identifying the dependency parses. We can then identify the “top” or “head” words in our sentences. In Stanza NLP, these tree-like representations follow the Universal Dependencies.

- spacy_stanza uses Stanza models for POS tagging and spacy_udpipe utilises UDPipe models for POS tagging. UDPipe models generally rely on neural networks to predict POS tags based on word context.The POS tagging models in Stanza are typically based on recurrent neural networks (RNNs) or transformers and are trained on large annotated corpora to predict POS tags based on word context.

Implementation:

1. **For English data-**

POS Tagging-

```
import spacy

# Load the English language model
nlp = spacy.load("en_core_web_sm")

# Define a function to perform POS tagging
def pos_tagging(text):
    doc = nlp(text)
    pos_tags = [(token.text, token.pos_) for token in doc]
    return pos_tags

# Apply the function to the 'text' column
df['pos_tags'] = df['text'].apply(pos_tagging)
```

	text	label	pos_tags
0	if you made it through this were not only able to start making money for yourself but sustain living that way all from home fuck these company corporate pig power to the people always	0	[(if, SCONJ), (you, PRON), (made, VERB), (it, PRON), (through, ADP), (this, PRON), (were, AUX), (not, PART), (only, ADV), (able, ADJ), (to, PART), (start, VERB), (making, VERB), (money, NOUN), (for, ADP), (yourself, PRON), (but, CCONJ), (sustain, VERB), (living, NOUN), (that, DET), (way, NOUN), (all, ADV), (from, ADP), (home, NOUN), (fuck, NOUN), (these, DET), (company, NOUN), (corporate, ADJ), (pig, NOUN), (power, NOUN), (to, ADP), (the, DET), (people, NOUN), (always, ADV)]
1	technically that is still turning back the clock dick head	0	[(technically, ADV), (that, PRON), (is, AUX), (still, ADV), (turning, VERB), (back, ADV), (the, DET), (clock, NOUN), (dick, PROPN), (head, NOUN)]
2	and you are the govt stop thinking about world medium liberal gang or any optic whatsoever and act now already if this is what a person at your level is facing then shudder to think the plight of common people in bengal bengalburning	1	[(and, CCONJ), (you, PRON), (are, AUX), (the, DET), (govt, NOUN), (stop, VERB), (thinking, VERB), (about, ADP), (world, NOUN), (medium, ADJ), (liberal, ADJ), (gang, NOUN), (or, CCONJ), (any, DET), (optic, NOUN), (whatsoever, ADV), (and, CCONJ), (act, VERB), (now, ADV), (already, ADV), (if, SCONJ), (this, PRON), (is, AUX), (what, PRON), (a, DET), (person, NOUN), (at, ADP), (your, PRON), (level, NOUN), (is, AUX), (facing, VERB), (then, ADV), (shudder, NOUN), (to, PART), (think, VERB), (the, DET), (plight, NOUN), (of, ADP), (common, ADJ), (people, NOUN), (in, ADP), (bengal, ADJ), (bengalburning, NOUN)]
3	soldier of japan who ha dick head	0	[(soldier, NOUN), (of, ADP), (japan, PROPN), (who, PRON), (ha, INTJ), (dick, PROPN), (head, PROPN)]
4	you would be better off asking who doe not think he is a sleazy shitbag lmao	0	[(you, PRON), (would, AUX), (be, AUX), (better, ADJ), (off, ADP), (asking, VERB), (who, PRON), (doe, AUX), (not, PART), (think, VERB), (he, PRON), (is, AUX), (a, DET), (sleazy, ADJ), (shitbag, NOUN), (lmao, PROPN)]

Dependency Parsing-

a. Using spaCy

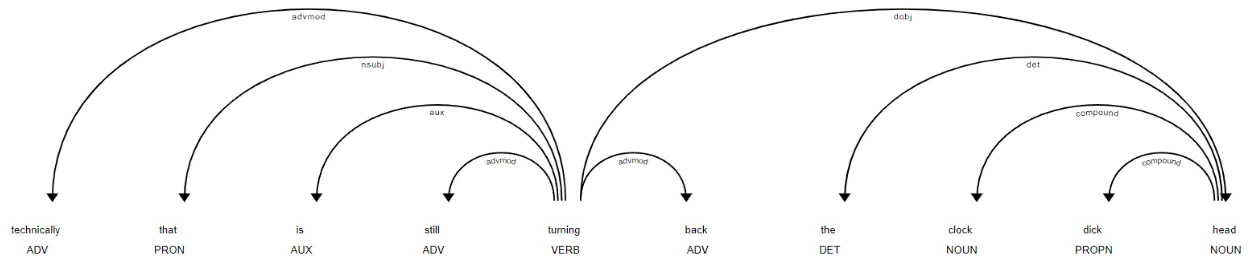
```
# Load the English language model
nlp = spacy.load("en_core_web_sm")

doc = nlp('technically that is still turning back the clock dick head')

dependency_features = ['Id', 'Text', 'Head', 'Dep']
head_format = "\033[1m{!s:>11}\033[0m" * (len(dependency_features) )
row_format = "{!s:>11}" * (len(dependency_features) )

print(head_format.format(*dependency_features))
# Printing dependency features for each token
for token in doc:
    print(row_format.format(token.i, token.text, token.head.i, 
token.dep_))
```

```
         Id       Text       Head        Dep
          0technically          4     advmod
          1       that          4      nsubj
          2         is          4        aux
          3      still          4     advmod
          4    turning          4       ROOT
          5       back          4     advmod
          6        the          9        det
          7      clock          9   compound
          8       dick          9   compound
          9       head          4       dobj
```

Dependency Parsing using spaCy

spaCy provides a built-in dependency visualizer called displaCy that you can use to generate dependency graph for sentences.

```
# SpaCy visualization tool
from spacy import displacy

# Run in a terminal
displacy.serve(doc, style='dep')
```



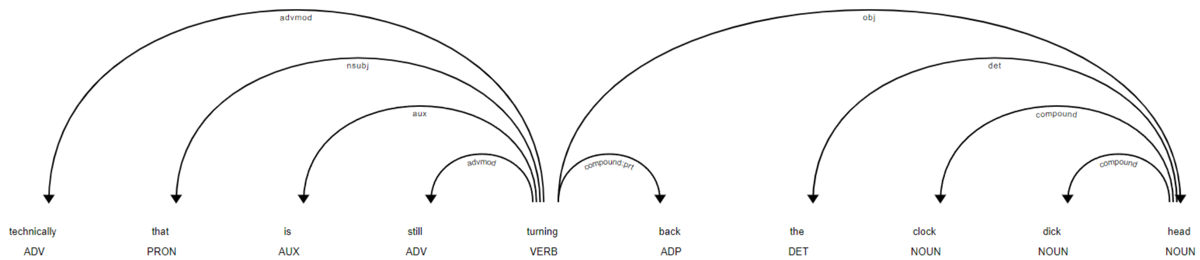
b. Using Stanza -

```
!pip install stanza
```

```
import stanza
import spacy_stanza

stanza.download("en")
nlp = spacy_stanza.load_pipeline("en")

doc = nlp("technically that is still turning back the clock dick head")
for token in doc:
    print(token.text, token.lemma_, token.pos_, token.dep_,
token.ent_type_)
print(doc.ents)
```

```
technically technically ADV advmod
that that PRON nsubj
is be AUX aux
still still ADV advmod
turning turn VERB root
back back ADP compound:prt
the the DET det
clock clock NOUN compound
dick dick NOUN compound
head head NOUN obj
```

```
from spacy import displacy
# Visualize dependency parse
displacy.render(doc, style="dep", jupyter=True)
```



Constituency Parsing-

```
import stanza

nlp = stanza.Pipeline(lang='en',
processors='tokenize,pos,constituency')
doc = nlp('american of all age and race agree trump is a racist resist
resistrump fucktrump')
```

```
for sentence in doc.sentences:
    print(sentence.constituency)
```

```
(ROOT (S (NP (ADJP (JJ american)) (PP (IN of) (NP (DT all) (NML (NN
age) (CC and) (NN race))))) (VP (VBP agree) (SBAR (S (NP (NN trump))
(VP (VBZ is) (NP (NP (DT a) (JJ racist) (NN resist)) (NP (NN
resistrump) (NN fucktrump)))))))))
```

For Marathi Text-

1. **POS Tagging-**

```
!pip install spacy_udpipe
import spacy
import spacy_udpipe
spacy_udpipe.download("mr") # download Marathi model

# Load the Marathi language model
nlp = spacy_udpipe.load("mr")

# Define a function to perform POS tagging
def pos_tagging(text):
    doc = nlp(text)
    pos_tags = [(token.text, token.pos_) for token in doc]
    return pos_tags

# Apply the function to the 'text' column
df['pos_tags'] = df['tweet'].apply(pos_tagging)
```

	tweet	label	pos_tags
0	आजच्या जनता दरबारात जळगाव जिल्ह्यातील चाळीसगावचे रहिवासी माजी सैनिक सोनू महाजन भाजपचे तत्कालीन...	1	[(आजच्या, DET), (जनता, NOUN), (दरबारात, NOUN), (जळगाव, ADJ), (जिल्ह्यातील, AUX), (चाळीसगावचे, ADV), (रहिवासी, NOUN), (माजी, ADV), (सैनिक, NOUN), (सोनू, VERB), (महाजन, ADV), (भाजपचे, NOUN), (तत्कालीन, VERB), (..., PUNCT)]
1	कुणी कविता करत असतं कुणी कविता जगत असतं कुणी कविता वाचत असतं कुणाला कविताच वाचवत असते पल्लवी	1	[(कुणी, ADV), (कविता, PRON), (करत, VERB), (असतं, PRON), (कुणी, NOUN), (कविता, ADJ), (जगत, NOUN), (असतं, PRON), (कुणी, ADV), (कविता, PRON), (वाचत, VERB), (असतं, PRON), (कुणाला, VERB), (कविताच, NUM), (वाचवत, VERB), (असते, VERB), (पल्लवी, VERB)]
2	आम्हाला इतिहासातील औरंगजेबशी घेणे आमच्या कडे आमचा बेकायदेशीर रित्या आलेला हक्काचा औरंगजेब क...	1	[(आम्हाला, NOUN), (इतिहासातील, AUX), (औरंगजेबशी, ADV), (घेणे, VERB), (आमच्या, NOUN), (कडे, ADP), (आमचा, VERB), (बेकायदेशीर, CCONJ), (रित्या, NOUN), (आलेला, VERB), (हक्काचा, ADV), (औरंगजेब, ADV), (क, VERB), (..., PUNCT)]
3	गँभीर प्रकरण महाराष्ट्राची अवस्था बिकट भाषणात मोठे शब्द वापरणे ऐकले कृती करावी उद्योग भी...	1	[(गँभीर, CCONJ), (प्रकरण, NOUN), (महाराष्ट्राची, ADV), (अवस्था, ADJ), (बिकट, NOUN), (भाषणात, NOUN), (मोठे, VERB), (शब्द, NOUN), (वापरणे, VERB), (ऐकले, VERB), (कृती, PRON), (करावी, VERB), (उद्योग, VERB), (भी, ADJ), (..., PUNCT)]
4	कब्झा कन्नड चित्रपट लवकरच मराठी मध्ये डब्ब होऊन प्रदर्शित जर ह्या चित्रपटाला चांगला प्रतिसाद मिळाला आप...	1	[(कब्झा, ADJ), (कन्नड, ADV), (चित्रपट, ADJ), (लवकरच, NUM), (मराठी, NOUN), (मध्ये, ADP), (डब्ब, ADV), (होऊन, VERB), (प्रदर्शित, VERB), (जर, CCONJ), (ह्या, DET), (चित्रपटाला, NOUN), (चांगला, AUX), (प्रतिसाद, NOUN), (मिळाला, VERB), (आप, ADJ), (..., PUNCT)]

2. **Dependency Parsing -**
   a. **using Spacy_udpipe:**

```
text = "एकट्या कंगणाने तुमच्या गोट्या चोळून पार धूर काढला"

nlp = spacy_udpipe.load("mr")

doc = nlp(text)

dependency_features = ['Id', 'Text', 'Lemma', 'Head','POS', 'Dep']
head_format = "\033[1m{!s:>11}\033[0m" * (len(dependency_features) )
row_format = "{!s:>11}" * (len(dependency_features) )
```

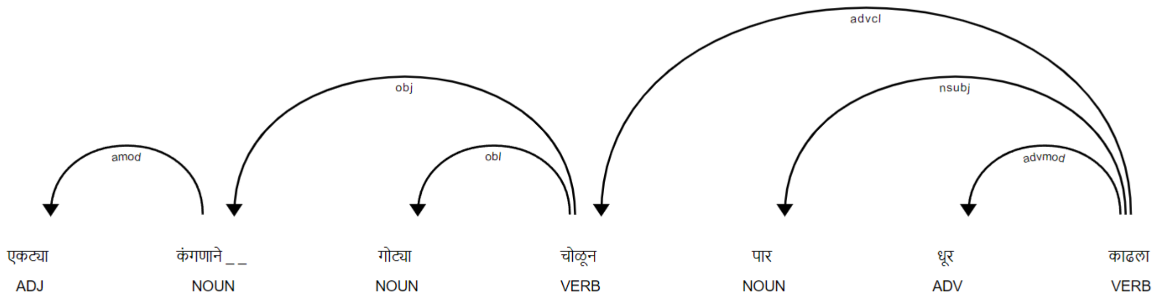
```
print(head_format.format(*dependency_features))
# Printing dependency features for each token
for token in doc:
    print(row_format.format(token.i, token.text, token.lemma_,
token.head.i, token.pos_, token.dep_))
```

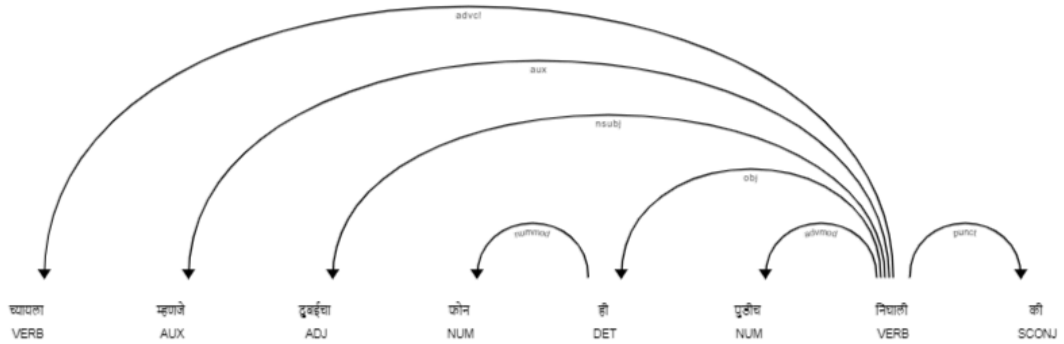
Downloaded pre-trained UDPipe model for 'mr' language

Id	Text	Lemma	Head	POS	Dep
0	एकट्या	एकटा	1	ADJ	amod
1	कंगणाने	कंगण	5	NOUN	obj
2	_	तो	5	NOUN	obl
3	_	चा	2	ADP	case
4	गोट्या	गोटी	5	NOUN	obl
5	चोळून	चोळणे	8	VERB	advcl
6	पार	पार	8	NOUN	nsubj
7	धूर	धूर	8	ADV	advmod
8	काढला	काढणे	8	VERB	ROOT

Id	Text	Lemma	Head	POS	Dep
0	च्यायला	च्याय	6	VERB	advcl
1	म्हणजे	म्हणजे	6	AUX	aux
2	दुबईच्या	दुबईचणे	6	ADJ	nsubj
3	फोन	फोन	4	NUM	nummod
4	ही	हा	6	DET	obj
5	पुढीच	पुढीच	6	NUM	advmod
6	निघाली	निघाली	6	VERB	ROOT
7	की	की	6	SCONJ	punct

```
from spacy import displacy
displacy.serve(doc, style='dep')
```



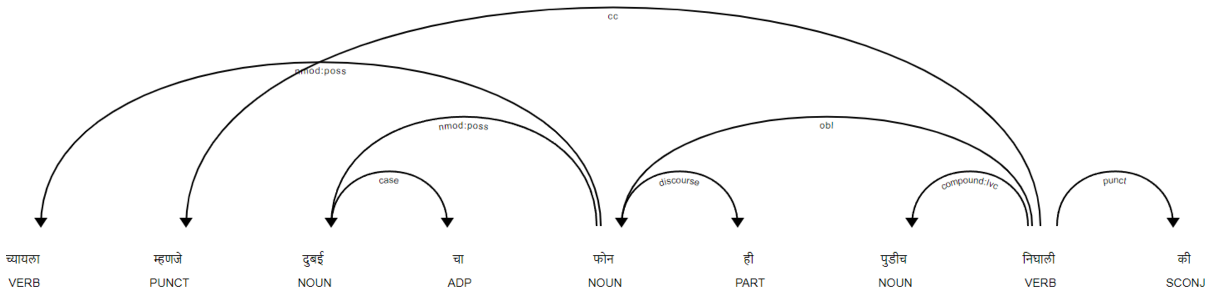


b. Using Spacy_Stanza:

```
!pip install stanza
!pip install spacy_stanza

import stanza
import spacy_stanza

stanza.download("mr")
nlp = spacy_stanza.load_pipeline("mr")

doc=nlp("च्यायला म्हणजे दुबईचा फोन ही पुडीच निघाली की")
# doc = nlp("एकट्या कंगणाने तुमच्या गोट्या चोळून पार धूर काढला")
dependency_features = ['Id', 'Text', 'Lemma', 'Head','POS', 'Dep']
head_format = "\033[1m{!s:>22}\033[0m" * (len(dependency_features) )
row_format = "{!s:>22}" * (len(dependency_features) )

print(head_format.format(*dependency_features))
# Printing dependency features for each token
for token in doc:
    print(row_format.format(token.i, token.text, token.lemma_,
token.head.i, token.pos_, token.dep_))
```

Id	Text	Lemma	Head	POS	Dep
0	च्यायला	चाणे	4	VERB	nmod:poss
1	म्हणजे	म्हणजे	7	PUNCT	cc
2	दुबई	दुबई	4	NOUN	nmod:poss
3	चा	चा	2	ADP	case
4	फोन	फोन	7	NOUN	obl
5	ही	ही	4	PART	discourse
6	पुडीच	पुडीच	7	NOUN	compound:lvc
7	निघाली	निघाली	7	VERB	root
8	की	की	7	SCONJ	punct

```python
# SpaCy visualization tool
from spacy import displacy

# Run in a terminal
displacy.serve(doc, style='dep')
```



For Hindi Text -

1. **POS Tagging**

```python
# Load the Hindi language model
nlp = spacy_udpipe.load("hi")


# Define a function to perform POS tagging
def pos_tagging(text):
    doc = nlp(text)
    pos_tags = [(token.text, token.pos_) for token in doc]
    return pos_tags


# Apply the function to the 'text' column
df['pos_tags'] = df['text'].apply(pos_tagging)
```

1	सब लोग इतने पैसे डोनेट आम आदमी सिलेंडर कन्सेंट्रेटर ख़ुद ख़रीदना पड़ पैसे कहाँ बीयर्ड ऑयल	1	[(सब, DET), (लोग, NOUN), (इतने, DET), (पैसे, NOUN), (डोनेट, X), (आम, ADJ), (आदमी, NOUN), (सिलेंडर, NOUN), (कन्सेंट्रेटर, NOUN), (ख़ुद, PRON), (ख़रीदना, VERB), (पड़, AUX), (पैसे, NOUN), (कहाँ, PRON), (बीयर्ड, PROPN), (ऑयल, PROPN)]
2	शेरए सिवान शहाबुद्दीन साहब रिश्ता क्या لا إله إلا الله محمد رسول الله	1	[(शेरए, PROPN), (सिवान, PROPN), (शहाबुद्दीन, PROPN), (साहब, NOUN), (रिश्ता, NOUN), (क्या, PRON), (لا, PROPN), (إله, PROPN), (إلا, PROPN), (الله, PROPN), (محمد, PROPN), (رسول, PROPN), (الله, PROPN)]
3	आसमानी किताब नाजायज औलाद	0	[(आसमानी, ADJ), (किताब, PROPN), (नाजायज, PROPN), (औलाद, PROPN)]
4	दोगला पंती सपा दम सफर माया इज्ज़त बचा पाई आज सपा मिटाने बात रही	1	[(दोगला, PROPN), (पंती, PROPN), (सपा, PROPN), (दम, ADV), (सफर, NOUN), (माया, VERB), (इज्ज़त, NOUN), (बचा, VERB), (पाई, AUX), (आज, NOUN), (सपा, PROPN), (मिटाने, VERB), (बात, NOUN), (रही, VERB)]

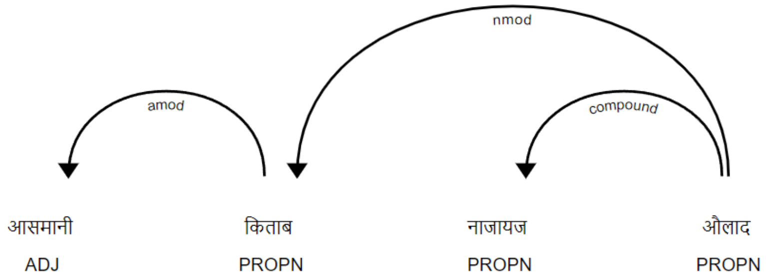
2. Dependency Parsing

a. Using spaCy:

```
import spacy
import spacy_udpipe
spacy_udpipe.download("hi") # download Hindi model

# Load the hindi language model
nlp = spacy_udpipe.load("hi")

def pos_tagging(text):
    doc = nlp(text)
    pos_tags = [(token.text, token.pos_) for token in doc]
    return pos_tags

# Apply the function to the 'text' column
df['pos_tags'] = df['text'].apply(pos_tagging)
text = "आसमानी किताब नाजायज औलाद"
nlp = spacy_udpipe.load("hi")
doc = nlp(text)
dependency_features = ['Id', 'Text', 'Lemma', 'Head','POS', 'Dep']
head_format = "\033[1m{!s:>11}\033[0m" * (len(dependency_features) )
row_format = "{!s:>11}" * (len(dependency_features) )

print(head_format.format(*dependency_features))
# Printing dependency features for each token
for token in doc:
    print(row_format.format(token.i, token.text, token.lemma_,
token.head.i, token.pos_, token.dep_))
```

```
from spacy import displacy
displacy.serve(doc, style='dep')
```

Id	Text	Lemma	Head	POS	Dep
0	आसमानी	आसमाना	1	ADJ	amod
1	किताब	किताब	3	PROPN	nmod
2	नाजायज	नाजायज	3	PROPN	compound
3	औलाद	औलाद	3	PROPN	ROOT

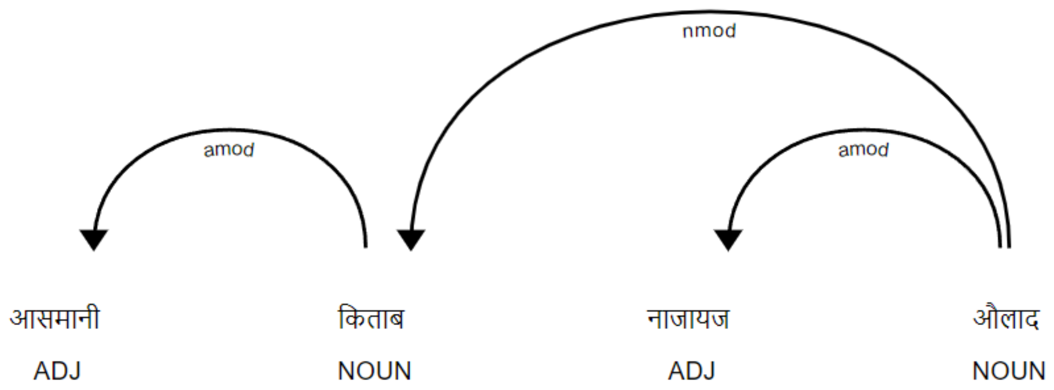


b. Using spacy_stanza:

```
import stanza
import spacy_stanza
stanza.download("hi")
nlp = spacy_stanza.load_pipeline("hi")

doc=nlp("आसमानी किताब नाजायज औलाद")

dependency_features = ['Id', 'Text', 'Lemma', 'Head','POS', 'Dep']
head_format = "\033[1m{!s:>22}\033[0m" * (len(dependency_features) )
row_format = "{!s:>22}" * (len(dependency_features) )

print(head_format.format(*dependency_features))
# Printing dependency features for each token
for token in doc:
    print(row_format.format(token.i, token.text, token.lemma_,
token.head.i, token.pos_, token.dep_))
```

```
from spacy import displacy
displacy.serve(doc, style='dep')
```

Id	Text	Lemma	Head	POS	Dep
0	आसमानी	आसमानी	1	ADJ	amod
1	किताब	किताब	3	NOUN	nmod
2	नाजायज	नाजायज	3	ADJ	amod
3	औलाद	औलाद	3	NOUN	root



Observations-
spacy-udpipe is slightly less accurate than stanza but much faster. In above example of Hindi you can see results of Stanza are more accurate than udpipe.

FineTuning BERT Based Models on Hate Speech Detection Task:

Fine-tuning is taking a pre-trained model and training at least one internal model parameter (i.e. weights). The goal is to optimize the model's performance on a new, related task without starting the training process from scratch.

The key upside of this approach is that models can achieve better performance while requiring (far) fewer manually labelled examples compared to models that solely rely on supervised training.

Fine-tuning not only improves the performance of a base model, but a smaller (fine-tuned) model can often outperform larger (more expensive) models on the set of tasks on which it was trained. This was demonstrated by OpenAI with their first generation "InstructGPT" models, where the 1.3B parameter InstructGPT model completions were preferred over the 175B parameter GPT-3 base model despite being 100x smaller.

Models Used:

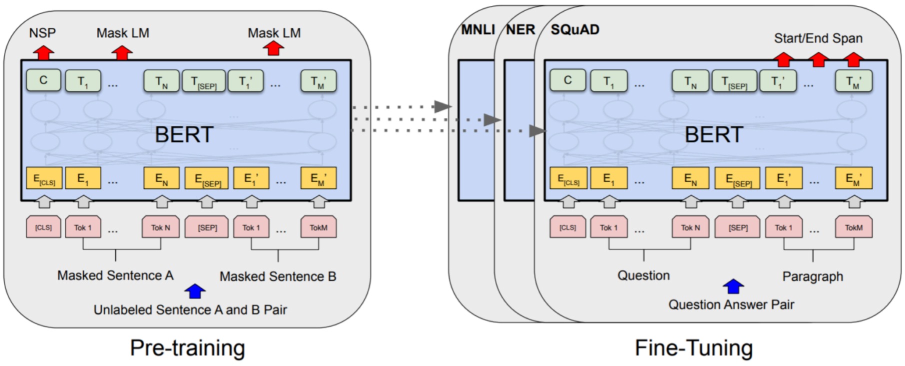
1. BERT(bert-base-uncased)
2. mBERT(bert-multilingual-base-uncased)
3. XLM-RoBERTa

1. BERT

Bidirectional Encoder Representations from Transformers (BERT) is designed to pre-train deep bidirectional representations from an unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

1. BERT is basically a trained Transformer Encoder stack, with twelve in the Base version, and twenty-four in the Large version, compared to 6 encoder layers in the original Transformer.

2. BERT encoders have larger feedforward networks (768 and 1024 nodes in Base and Large respectively) and more attention heads (12 and 16 respectively). BERT was trained on Wikipedia and Book Corpus, a dataset containing +10,000 books of different genres.



A model pre-trained on text from only a single language is called **monolingual**, while those trained on text from multiple languages are called **multilingual**.

Different languages have different amounts of training data available to create large, BERT-like models. These are referred to as high, medium, and low-resource languages. High-resource languages like English have lots of freely available text online that can be used as training data.

2. BERT multilingual base model:

102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters

- Instead of pretraining BERT on just english text, it is pretrained on 100's of different languages.
- It allows to perform **cross-lingual transfer learning**

3. XLM-RoBERTa:

- "Cross-Lingual Language Model - Roberta" from Facebook,
  While the original BERT was pre-trained on English Wikipedia and BooksCorpus (a collection of self-published books) XLM-R was pre-trained on Wikipedia and Common Crawl data from 100 different languages.
- There is a single, shared vocabulary (with 250k tokens) to cover all 100 languages.
- There is no special marker added to the input text to indicate what language it is.
- It wasn't trained with "parallel data" (the same sentence in multiple languages).

Cross-Lingual Transfer:

XLM-R is able to take it's task-specific knowledge that it learned in English and apply it to Hindi, Marathi, even though we never showed it any Hindi, Marathi examples. Transfer learning applied from one language to another.

Implementation :-

We finetuned different models on our hate speech datasets for comparative analysis
Labels-
0 -Hate speech
1- Non Hate Speech

1. Fine Tuning BERT "bert-base-uncased" on English Hate Speech Dataset

Loaded the Preprocessed English text dataset

```
import pandas as pd

# Load the dataset into a pandas dataframe.
df =  pd.read_csv("/kaggle/input/eng-data/english_data (2).csv")

df.rename(columns = {'text':'sentence', 'label':'label'}, inplace = True)

print('Number of training sentences: {:,}\n'.format(df0.shape[0]))

df0.sample(10)
```

```
Number of training sentences: 8,693
```

BERT Tokernizer: To feed the text to BERT, it must be split into tokens, and then these tokens must be mapped to their index in the tokenizer vocabulary.
The tokenization must be performed by the tokenizer included with BERT

```
from transformers import BertTokenizer

# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
do_lower_case=True)
```

```
Loading BERT tokenizer...
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 4.04kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 4.93MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 14.4MB/s]
config.json: 100%  570/570 [00:00<00:00, 51.2kB/s]
```

```
import torch
# Tokenize all of the sentences and map the tokens to thier word IDs.
input_ids = []
attention_masks = []
# For every sentence...
for sent in sentences:
    # `encode_plus` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent,
                        add_special_tokens = True,
                        max_length = 70,
                        pad_to_max_length = True,
                        return_attention_mask = True,
                        return_tensors = 'pt',
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_dict['input_ids'])

    # And its attention
    attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Print sentence 0, now as a list of IDs.
print('Original: ', sentences[0])
print('Token IDs:', input_ids[0])
```

Training & Validation Split:

```
from torch.utils.data import TensorDataset, random_split

# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, labels)
```

```
# Create a 90-10 train-validation split.

train_size = int(0.90 * len(dataset))
val_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size,
val_size])

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

BertForSequenceClassification:

This is the normal BERT model with an added single linear layer on top for classification that we will use as a sentence classifier. As we feed input data, the entire pre-trained BERT model and the additional untrained classification layer is trained on our specific task.

```
from transformers import BertForSequenceClassification, AdamW,
BertConfig

# Load BertForSequenceClassification
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased
vocab.
    num_labels = 2, # The number of output labels--2 for binary
classification.
    output_attentions = False,
    output_hidden_states = False,
)

# Tell pytorch to run this model on the GPU.
model.cuda()
```

Model's Parameters:

```
The BERT model has 201 different named parameters.

==== Embedding Layer ====

bert.embeddings.word_embeddings.weight                  (30522, 768)
bert.embeddings.position_embeddings.weight                (512, 768)
bert.embeddings.token_type_embeddings.weight                (2, 768)
bert.embeddings.LayerNorm.weight                              (768,)
bert.embeddings.LayerNorm.bias                                (768,)

==== First Transformer ====

bert.encoder.layer.0.attention.self.query.weight          (768, 768)
bert.encoder.layer.0.attention.self.query.bias                (768,)
bert.encoder.layer.0.attention.self.key.weight            (768, 768)
bert.encoder.layer.0.attention.self.key.bias                  (768,)
bert.encoder.layer.0.attention.self.value.weight          (768, 768)
bert.encoder.layer.0.attention.self.value.bias                (768,)
bert.encoder.layer.0.attention.output.dense.weight        (768, 768)
bert.encoder.layer.0.attention.output.dense.bias              (768,)
bert.encoder.layer.0.attention.output.LayerNorm.weight        (768,)
bert.encoder.layer.0.attention.output.LayerNorm.bias          (768,)
bert.encoder.layer.0.intermediate.dense.weight           (3072, 768)
bert.encoder.layer.0.intermediate.dense.bias                 (3072,)
bert.encoder.layer.0.output.dense.weight                 (768, 3072)
bert.encoder.layer.0.output.dense.bias                        (768,)
bert.encoder.layer.0.output.LayerNorm.weight                  (768,)
bert.encoder.layer.0.output.LayerNorm.bias                    (768,)

==== Output Layer ====

bert.pooler.dense.weight                                  (768, 768)
bert.pooler.dense.bias                                        (768,)
classifier.weight                                           (2, 768)
classifier.bias                                                 (2,)
```

```
optimizer = AdamW(model.parameters(),
                  lr = 2e-5, # args.learning_rate - default is 5e-5
                  eps = 1e-8 # args.adam_epsilon  - default is 1e-8
                )
```

Training:

```
import random
import numpy as np
seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()
```

```
for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================

    # Perform one full pass over the training set.

    # Measure how long the training epoch takes.
    t0 = time.time()

    # Reset the total loss for this epoch.
    total_train_loss = 0

    # Put the model into training mode. Don't be mislead--the call to
    model.train()

    # For each batch of training data...
    for step, batch in enumerate(train_dataloader):

        # Progress update every 40 batches.
        if step % 40 == 0 and not step == 0:
            elapsed = format_time(time.time() - t0)

            #  progress.
            print('  Batch {:>5,}  of  {:>5,}.    Elapsed:
{:}.'.format(step, len(train_dataloader), elapsed))

        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        model.zero_grad()

        # forward pass .
        result = model(b_input_ids,
                       token_type_ids=None,
                       attention_mask=b_input_mask,
                       labels=b_labels,
                       return_dict=True)

        loss = result.loss
```

```
        logits = result.logits

        total_train_loss += loss.item()

        # =backward pass to calculate the gradients.
        loss.backward()

        # Clip the norm of the gradients to 1.0.
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        # Update parameters and take a step using the computed
gradient.
        optimizer.step()

        # Update the learning rate.
        scheduler.step()

    # Calculate the average loss over all of the batches.
    avg_train_loss = total_train_loss / len(train_dataloader)

    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)

    # ========================================
    #               Validation
    # ========================================
    print("")
    print("Running Validation...")

    t0 = time.time()

    # Put the model in evaluation mode
    model.eval()

    total_eval_accuracy = 0
    total_eval_loss = 0
    nb_eval_steps = 0

    # Evaluate data for one epoch
    for batch in validation_dataloader:

        b_input_ids = batch[0].to(device)
```

```
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        with torch.no_grad():
            # Forward pass
            result = model(b_input_ids,
                           token_type_ids=None,
                           attention_mask=b_input_mask,
                           labels=b_labels,
                           return_dict=True)

        # Get the loss and "logits" output by the model.
        loss = result.loss
        logits = result.logits

        # Accumulate the validation loss.
        total_eval_loss += loss.item()

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

        # Calculate the accuracy for this batch of test sentences, and
        # accumulate it over all batches.
        total_eval_accuracy += flat_accuracy(logits, label_ids)

    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
    print("  Accuracy: {0:.2f}".format(avg_val_accuracy))

    # Calculate the average loss over all of the batches.
    avg_val_loss = total_eval_loss / len(validation_dataloader)

    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)

    print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    print("  Validation took: {:}".format(validation_time))
```

```
    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Valid. Accur.': avg_val_accuracy,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
    )
print("")
print("Training complete!")
print("Total training took {:}
(h:mm:ss)".format(format_time(time.time()-total_t0)))
```

Summary of the training process:

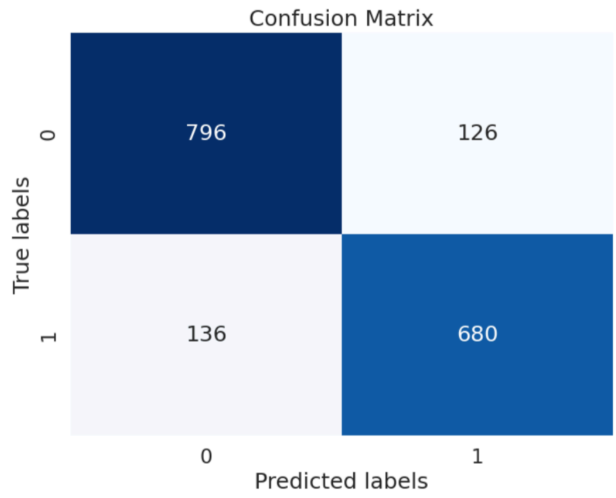
epoch	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
1	0.436902	0.359569	0.848958	0:01:16	0:00:03
2	0.286462	0.336906	0.859848	0:01:16	0:00:03
3	0.213986	0.368237	0.849432	0:01:16	0:00:03

Evaluating on Test Dataset :

F1 Score: 0.838

```
from sklearn.metrics import f1_score

# Calculate the F1 score
f1 = f1_score(flat_true_labels, flat_predictions)

print('F1 Score: %.3f' % f1)
```

```
F1 Score: 0.838
```



Classification report:

```
from sklearn.metrics import classification_report

target_names = ['class 0', 'class 1']
print(classification_report(flat_true_labels, flat_predictions, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.85      0.86      0.86       922
     class 1       0.84      0.83      0.84       816

    accuracy                           0.85      1738
   macro avg       0.85      0.85      0.85      1738
weighted avg       0.85      0.85      0.85      1738
```

2. Finetuning mBERT on English, Marathi, Hindi Hate Speech Datasets :-

a) Training on English Text:

Bert-base-multilingual-uncased Tokenizer:

```
from transformers import BertTokenizer

# Load the BERT tokenizer.
tokenizer =
BertTokenizer.from_pretrained('bert-base-multilingual-uncased',
do_lower_case=True)
```

BERT multilingual base model (uncased):

```
from transformers import BertForSequenceClassification, AdamW,
BertConfig

# Load BertForSequenceClassification
model = BertForSequenceClassification.from_pretrained(
    "bert-base-multilingual-uncased", # Use the 12-layer BERT model,
with an uncased vocab.
    num_labels = 2, # The number of output labels--2 for binary
classification.
    output_attentions = False,
    output_hidden_states = False,
)

model.cuda()
```

```
optimizer = AdamW(model.parameters(),
                  lr = 2e-5, # args.learning_rate - default is 5e-5
                  eps = 1e-8 # args.adam_epsilon  - default is 1e-8
                )
```

Training the mBERT Model:

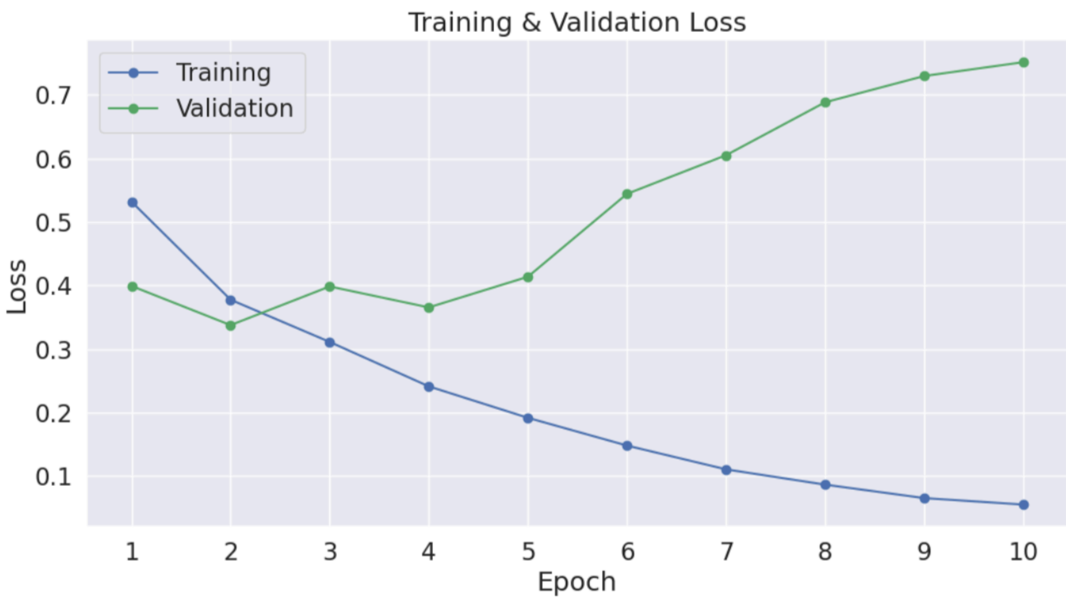
Hyperparameters:

Epochs = 10

Batch size = 32

Maximum length of the sentence after tokenization = 64

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.532019	0.399266	0.841218	0:01:09	0:00:03
2	0.377641	0.337671	0.859684	0:01:14	0:00:03
3	0.311189	0.398517	0.848938	0:01:14	0:00:03
4	0.241249	0.365539	0.838995	0:01:14	0:00:03
5	0.191745	0.413768	0.837574	0:01:14	0:00:03
6	0.148048	0.544261	0.842144	0:01:14	0:00:03
7	0.110587	0.605139	0.838130	0:01:14	0:00:03
8	0.086511	0.688519	0.826210	0:01:14	0:00:03
9	0.065340	0.729864	0.834980	0:01:14	0:00:03
10	0.055171	0.751605	0.835289	0:01:14	0:00:03

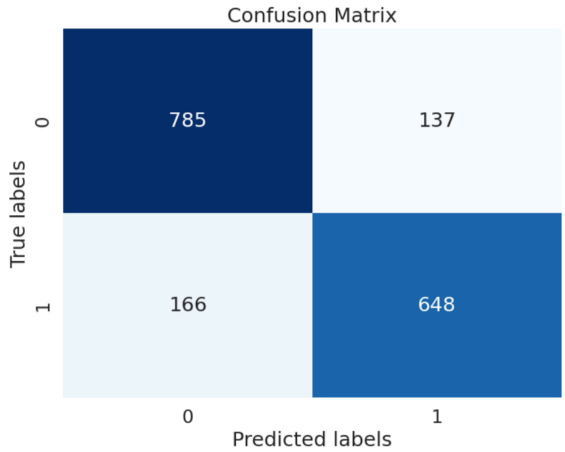


Evaluating on Test Dataset

F1 Score: 0.811

```
from sklearn.metrics import f1_score

# Calculate the F1 score
f1 = f1_score(flat_true_labels, flat_predictions)

print('F1 Score: %.3f' % f1)
```

```
F1 Score: 0.811
```



Hyperparameter Tuning -

After changing the number of **epochs** from 10 to 2:

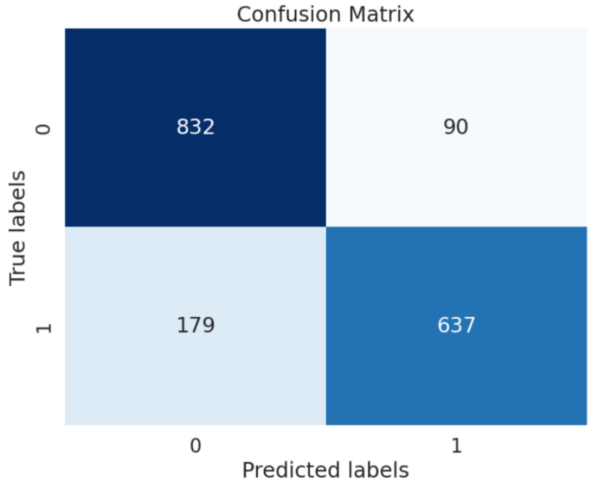
	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.473194	0.370573	0.841856	0:01:15	0:00:03
2	0.330847	0.344052	0.849432	0:01:19	0:00:03

Evaluating on Test Dataset:

F1 Score: 0.826

```
from sklearn.metrics import f1_score

# Calculate the F1 score
f1 = f1_score(flat_true_labels, flat_predictions)

print('F1 Score: %.3f' % f1)
```

```
F1 Score: 0.826
```



Classification Report:

```
from sklearn.metrics import classification_report

target_names = ['class 0', 'class 1']
print(classification_report(flat_true_labels, flat_predictions, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.82      0.90      0.86       922
     class 1       0.88      0.78      0.83       816

    accuracy                           0.85      1738
   macro avg       0.85      0.84      0.84      1738
weighted avg       0.85      0.85      0.84      1738
```

b) Training on Marathi Text:

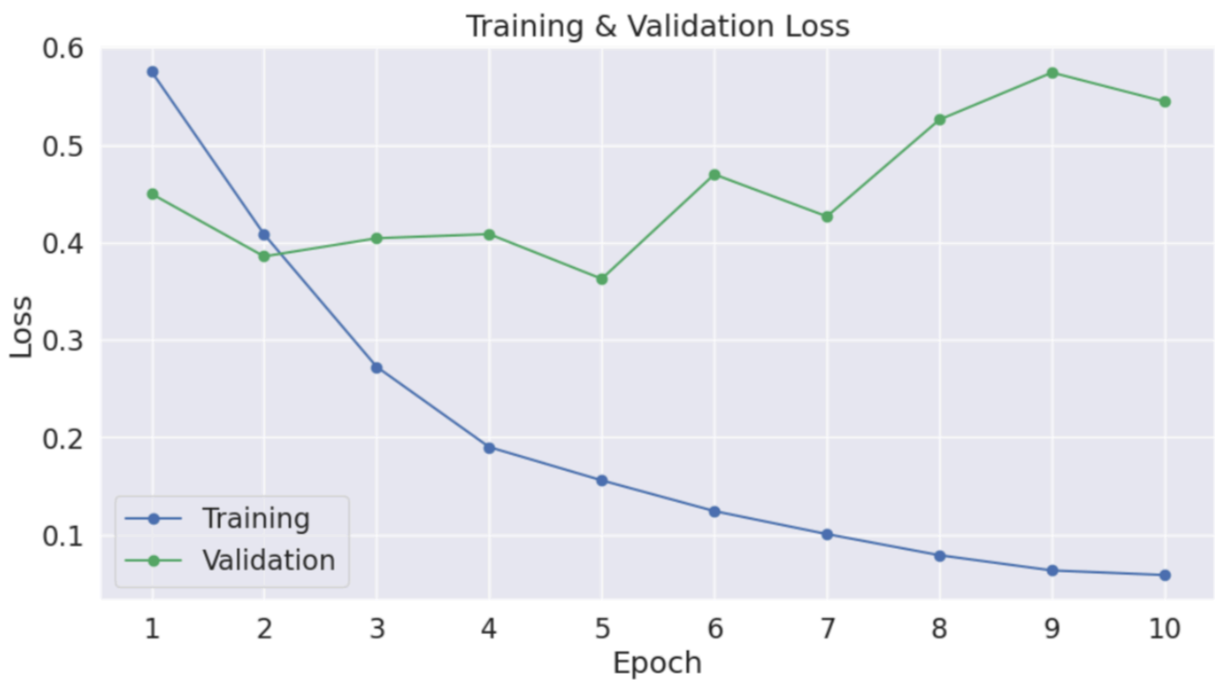
Number of sentences in dataset: 3,585
Training: 2,533
Validation: 282
Testing: 704

Hyperparameters **:**

Epochs = 10
Batch size = 32
Maximum length of the sentence after tokenization = 64

Training Process Summary:

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.575416	0.449900	0.821314	0:00:27	0:00:01
2	0.408376	0.385684	0.865652	0:00:28	0:00:01
3	0.272431	0.404342	0.864850	0:00:29	0:00:01
4	0.190051	0.408580	0.864049	0:00:28	0:00:01
5	0.155898	0.362662	0.869124	0:00:28	0:00:01
6	0.124386	0.469860	0.883814	0:00:28	0:00:01
7	0.100786	0.426710	0.880342	0:00:28	0:00:01
8	0.079106	0.525988	0.857906	0:00:28	0:00:01
9	0.063477	0.574327	0.857906	0:00:28	0:00:01
10	0.058774	0.544548	0.883814	0:00:28	0:00:01

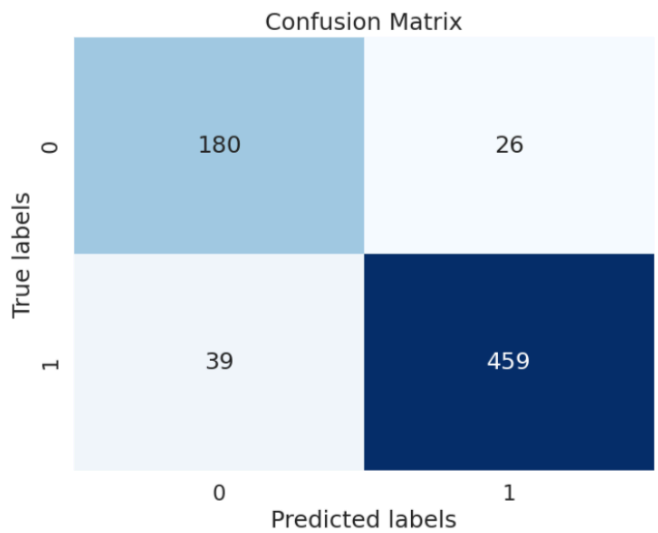


Evaluating on Test Dataset:

F1 Score: 0.934

```
from sklearn.metrics import f1_score

# Calculate the F1 score
f1 = f1_score(flat_true_labels, flat_predictions)

print('F1 Score: %.3f' % f1)
```

```
F1 Score: 0.934
```



c) Training on Hindi Text:

Number of sentences in dataset: 4,594
Training: 3,307
Validation: 368
Testing: 919

Hyperparameters :

Epochs = 5
Batch size = 32
Maximum length of the sentence after tokenization = 64

Training Process Summary:

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.587784	0.466270	0.776042	0:00:36	0:00:01
2	0.458799	0.432973	0.812500	0:00:37	0:00:01
3	0.351996	0.456544	0.807292	0:00:38	0:00:01
4	0.276683	0.507982	0.817708	0:00:38	0:00:01
5	0.226102	0.499675	0.809896	0:00:38	0:00:01

Evaluating on Test Dataset:
F1 Score: 0.837

```
from sklearn.metrics import f1_score

# Calculate the F1 score
f1 = f1_score(flat_true_labels, flat_predictions)

print('F1 Score: %.3f' % f1)
```

```
F1 Score: 0.837
```

Classification report:

```
from sklearn.metrics import classification_report

target_names = ['class A', 'class B']
print(classification_report(flat_true_labels, flat_predictions, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class A       0.67      0.57      0.62       297
     class B       0.81      0.87      0.84       622

    accuracy                           0.77       919
   macro avg       0.74      0.72      0.73       919
weighted avg       0.76      0.77      0.77       919
```

3. Finetuning XML-R on English, Marathi, Hindi Hate Speech Datasets :

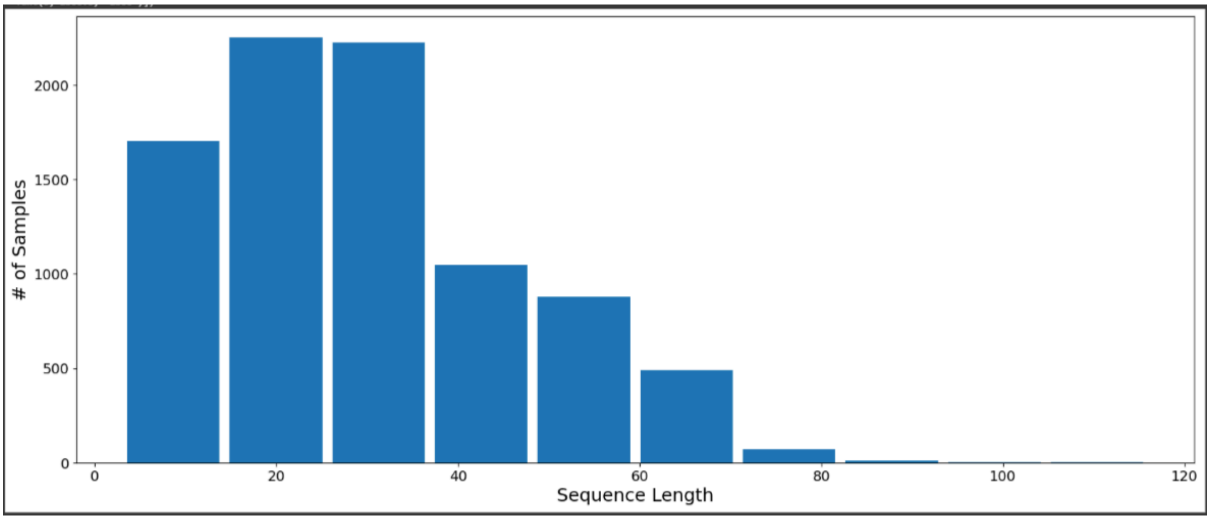
For English Text -
Steps-

1. **Import Tokenizer from XMLR model**

```
# load tokens
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained('xlm-roberta-base')
```

2. **Finding Max length of tokenized sentence**

```
# install matplotlib
!pip install matplotlib
# tokenize the text feature
tokenized_feature_raw = tokenizer.batch_encode_plus(
                            # Sentences to encode
                            df.text.values.tolist(),
                            # Add '[CLS]' and '[SEP]'
                            add_special_tokens = True
                   )
# collect tokenized sentence length
token_sentence_length = [len(x) for x in
tokenized_feature_raw['input_ids']]
print('max: ', max(token_sentence_length))
print('min: ', min(token_sentence_length))
# plot the distribution
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 8))
plt.hist(token_sentence_length, rwidth = 0.9)
plt.xlabel('Sequence Length', fontsize = 18)
plt.ylabel('# of Samples', fontsize = 18)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
```



3. Tokenize Features

```
# tokenize features
MAX_LEN = 116
tokenized_feature = tokenizer.batch_encode_plus(
                            # Sentences to encode
                            features,
                            # Add '[CLS]' and '[SEP]'
                            add_special_tokens = True,
                            # Add empty tokens if len(text)<MAX_LEN
                            padding = 'max_length',
                            # Truncate all sentences to max length
                            truncation=True,
                            # Set the maximum length
                            max_length = MAX_LEN,
                            # Return attention mask
                            return_attention_mask = True,
                            # Return pytorch tensors
                            return_tensors = 'pt'
                   )
```

4. Create Dataloader

```
# define batch_size
batch_size = 16
# Create the DataLoader for our training set
train_data = TensorDataset(train_inputs, train_masks,
torch.tensor(train_labels))
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler,
batch_size=batch_size)
# Create the DataLoader for our validation set
```

```
validation_data = TensorDataset(validation_inputs, validation_masks,
torch.tensor(validation_labels))
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data,
sampler=validation_sampler, batch_size=batch_size)
# Create the DataLoader for our test set
test_data = TensorDataset(test_inputs, test_masks,
torch.tensor(test_labels))
test_sampler = SequentialSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler,
batch_size=batch_size)
```

5. **Custom classification head created**

With custom classification head accuracy improved little bit.

```
from torch import nn
class CustomRobertaClassificationHead(nn.Module):
    def __init__(self, input_size, hidden_size, num_labels):
        super(CustomRobertaClassificationHead, self).__init__()
        print(input_size)
        self.dense = nn.Linear(input_size, hidden_size)
        self.dropout = nn.Dropout(0.1)
        self.additional_linear = nn.Linear(768, 768)  # Add one more
linear layer
        self.out_proj = nn.Linear(hidden_size, num_labels)

    def forward(self, x):
        x = self.dropout(x)
        batch_size = x.size(0)  # Get the batch size dynamically
        x = x.view(-1, x.size(-1))  # Reshape without hardcoding
input_size
        x = nn.functional.relu(self.dense(x))
        x = self.dropout(x)
        x = x.view(batch_size, -1, 768)  # Reshape to original shape
for additional_linear
        x = nn.functional.relu(self.additional_linear(x))  # Pass
through additional linear layer
        x = self.dropout(x)
        x = x.mean(dim=1)  # Pooling operation to aggregate information
across tokens
        x = self.out_proj(x)
        return x
```

Model-

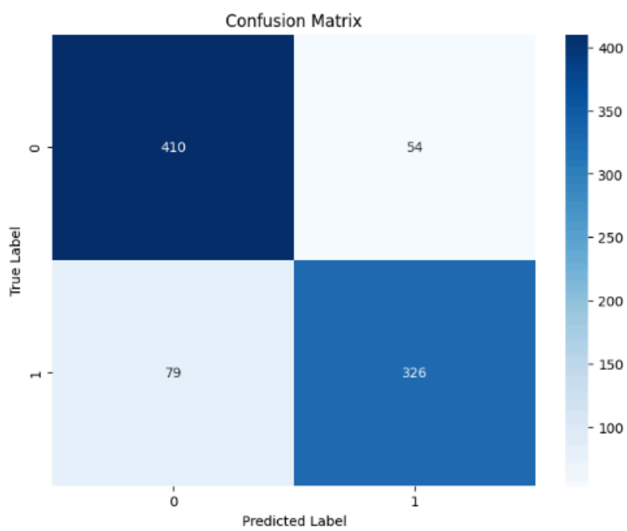
```
XLMRobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(250002, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (classifier): CustomRobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (additional_linear): Linear(in_features=768, out_features=768, bias=True)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

6. Training and validation

```
Time: 3.0m 1.5450448989868164s
Training report after iteration 3:
              precision    recall  f1-score   support

         NOT       0.88      0.91      0.89      3708
         HOF       0.89      0.86      0.88      3240

    accuracy                           0.89      6948
   macro avg       0.89      0.88      0.89      6948
weighted avg       0.89      0.89      0.89      6948

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

         NOT       0.82      0.92      0.87       463
         HOF       0.89      0.77      0.83       406

    accuracy                           0.85       869
   macro avg       0.86      0.85      0.85       869
weighted avg       0.86      0.85      0.85       869

Time: 3.0m 1.5740792751312256s
Training report after iteration 4:
              precision    recall  f1-score   support

         NOT       0.90      0.93      0.92      3708
         HOF       0.92      0.88      0.90      3240

    accuracy                           0.91      6948
   macro avg       0.91      0.91      0.91      6948
weighted avg       0.91      0.91      0.91      6948

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

         NOT       0.86      0.87      0.87       463
         HOF       0.85      0.84      0.85       406

    accuracy                           0.86       869
   macro avg       0.86      0.86      0.86       869
weighted avg       0.86      0.86      0.86       869

Time: 3.0m 1.62030029296875s
```

7. Testing

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

         NOT       0.84      0.88      0.86       464
         HOF       0.86      0.80      0.83       405

    accuracy                           0.85       869
   macro avg       0.85      0.84      0.85       869
weighted avg       0.85      0.85      0.85       869
```



View detailed code in notebook shared.

For Marathi-

For marathi and hindi same above model used but custom classifier not created.

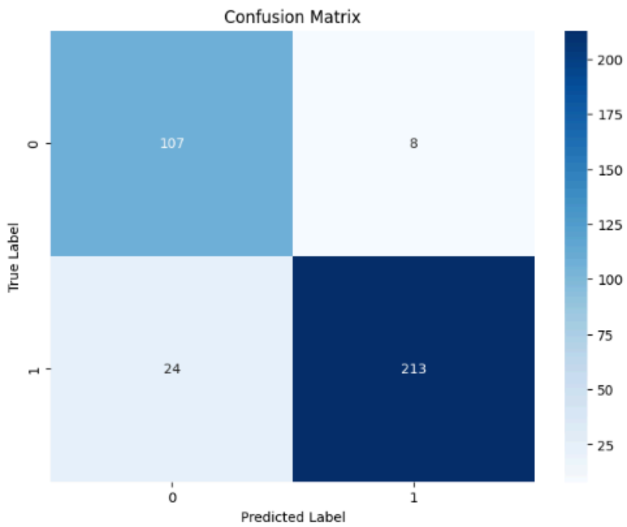
```
XLMRobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(250002, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (classifier): RobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

Training and Validation Results-

```
Time: 0.0m 53.438422203063965s
Training report after iteration 5:
              precision    recall  f1-score   support

         NOT       0.92      0.90      0.91       921
         HOF       0.95      0.96      0.96      1894

    accuracy                           0.94      2815
   macro avg       0.93      0.93      0.93      2815
weighted avg       0.94      0.94      0.94      2815

Validation report after iteration 5:
              precision    recall  f1-score   support

         NOT       0.88      0.85      0.87       115
         HOF       0.93      0.95      0.94       237

    accuracy                           0.91       352
   macro avg       0.91      0.90      0.90       352
weighted avg       0.91      0.91      0.91       352

Time: 0.0m 53.46840691566467s
Training report after iteration 6:
              precision    recall  f1-score   support

         NOT       0.92      0.92      0.92       921
         HOF       0.96      0.96      0.96      1894

    accuracy                           0.95      2815
   macro avg       0.94      0.94      0.94      2815
weighted avg       0.95      0.95      0.95      2815

Validation report after iteration 6:
              precision    recall  f1-score   support

         NOT       0.86      0.85      0.86       115
         HOF       0.93      0.93      0.93       237

    accuracy                           0.91       352
   macro avg       0.89      0.89      0.89       352
weighted avg       0.91      0.91      0.91       352

Time: 0.0m 53.506038188934326s
```

Testing Results-

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

         NOT       0.84      0.87      0.85       464
         HOF       0.84      0.81      0.83       405

    accuracy                           0.84       869
   macro avg       0.84      0.84      0.84       869
weighted avg       0.84      0.84      0.84       869
```



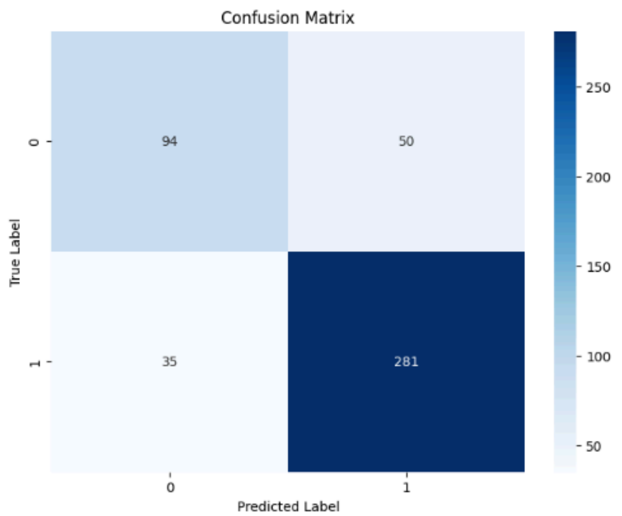
For Hindi Text-

Training and Validation Results-

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

         NOT       0.72      0.61      0.66      1146
         HOF       0.84      0.89      0.86      2529

    accuracy                           0.80      3675
   macro avg       0.78      0.75      0.76      3675
weighted avg       0.80      0.80      0.80      3675

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

         NOT       0.72      0.62      0.66       143
         HOF       0.84      0.89      0.86       316

    accuracy                           0.80       459
   macro avg       0.78      0.75      0.76       459
weighted avg       0.80      0.80      0.80       459

Time: 1.0m 49.26625728607178s
Training report after iteration 4:
              precision    recall  f1-score   support

         NOT       0.76      0.68      0.72      1146
         HOF       0.86      0.90      0.88      2529

    accuracy                           0.83      3675
   macro avg       0.81      0.79      0.80      3675
weighted avg       0.83      0.83      0.83      3675

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

         NOT       0.71      0.63      0.67       143
         HOF       0.84      0.88      0.86       316

    accuracy                           0.80       459
   macro avg       0.77      0.76      0.76       459
weighted avg       0.80      0.80      0.80       459

Time: 1.0m 49.31631803512573s
```

Testing Results-

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

         NOT       0.73      0.65      0.69       144
         HOF       0.85      0.89      0.87       316

    accuracy                           0.82       460
   macro avg       0.79      0.77      0.78       460
weighted avg       0.81      0.82      0.81       460
```



Analysis-

F1 Scores of different models on test data-

Model	Dataset		
	English	Hindi	Marathi
bert-base	0.838	-	-
mBERT	0.826	0.837	0.934
XML-R	0.86	0.82	0.91

XMLR model is giving good results on all datasets.
Here for Hindi and Marathi we analysed results of directly finetuning models on available data but as number of samples of marathi and hindi are less we are going to implement cross lingual transfer learning in next mandate and going to analyse its results.

Challenges Faced-

1. Finding Libraries for Indian languages dependency parsing. After going through some resources we get to know about spacy udpipe and spacy stanza.
2. Finetuning different models. After going through documentations and some articles we get to know process of fine tuning BERT based models.
3. Adding custom classification layer in XMLR model for english dataset
4. Less validation and test data for Indian Languages. We are going to use transfer learning in next mandate.

Future Work-

In the work we have referred for Hindi and Marathi Hate speech detection they are comparing results of different models on Hindi and Marathi datasets[6]. In this mandate we compared results directly finetuning on available datasets and in further mandate we are going to analyse the effect of **cross lingual transfer learning**. We tried to collect different datasets apart from datasets mentioned in reference.

Notebook Links-

1. POS Tagging and Dependency Parsing
   https://colab.research.google.com/drive/1Y4ETKYswWK3e3suhShquThnTy159yg6Z?usp=sharing

2. Finetuning On English Dataset
   a. XMLR: https://www.kaggle.com/code/manasipurkar/english-xmlr
   b. mBERT:https://www.kaggle.com/code/sanketp029/bert-finetuning
   c. BERT: https://www.kaggle.com/code/sanketp029/bert-base-finetuning

3. Finetuning On Marathi Dataset
   a. XMLR : https://www.kaggle.com/code/manasipurkar/marathi-xmlr
   b. MBERT: https://www.kaggle.com/code/sanketp029/bert-marathi

4. Finetuning On Hindi Dataset
   a. XMLR : https://www.kaggle.com/code/manasipurkar/hindi-xmlr
   b. MBERT: https://www.kaggle.com/code/sanketp029/bert-finetuning-hindi

Dataset:

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

References:

1. https://huggingface.co/google-bert/bert-base-multilingual-cased
2. https://towardsdatascience.com/fine-tuning-large-language-models-llms-23473d763b91
3. https://medium.com/@manjindersingh_10145/sentiment-analysis-with-bert-using-huggingface-88e99deeec9a
4. https://wisdomml.in/syntactic-processing-what-it-is-and-how-it-works/
5. https://huggingface.co/docs/transformers/en/model_doc/xlm-roberta
6. https://arxiv.org/pdf/2110.12200.pdf
7. https://spacy.io/universe/project/spacy-stanza
8. https://spacy.io/universe/project/spacy-udpipe
9. https://spotintelligence.com/2023/10/22/dependency-parsing/