Project Report

On

Social Media Comment Analyzer

Submitted
In partial fulfilment

For the award of the Degree of

PG-Diploma in Artificial Intelligence

(C-DAC, ACTS (Pune))

Guided by:

Dr. Krishnanjan Bhattacharjee (AAI Group, C-DAC)

Submitted by:

Manali Singh (220340128022)

Shraddha Bhide (220340128004)

Vishal Joshi (220340128014)

Shubham Narad (220340128031)

Gaurav Pol (220340128011)

Centre for Development of Advanced Computing (C-DAC), ACTS (Pune- 411008)

Acknowledgement

This is to acknowledge our indebtedness to our Project Guide, Dr **Krishnanjan B.,** C-DAC ACTS, Pune for her constant guidance and helpful suggestion for preparing this project **Social Media Comment Analyzer.** We express our deep gratitude towards him for his inspiration, personal involvement, and constructive criticism that he provided us along with technical guidance during the course of this project.

We take this opportunity to thank the Head of the department, **Mr. Gaur Sunder** and the Program Head, **Mrs. Risha P R** for providing us with such a great infrastructure and environment for our overall development.

We express sincere thanks to **Ms. Namrata Ailawar**, Process Owner, for their kind cooperation and support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude toward **Dr. Priyanka Ranade** (Course Coordinator, PG-DAI) for their valuable guidance and constant support throughout this work and helps to pursue additional studies.

Also our warm thanks to **C-DAC ACTS, Pune** which provides us with this opportunity to carry out this prestigious Project and to enhance our learning in various technical fields.

Manali Singh (220340128022)

Shraddha Bhide (220340128004)

Vishal Joshi (220340128014)

Shubham Narad (220340128031)

Gaurav Pol (220340128011)

Abstract

Many microblogging sites have millions of people sharing their thoughts daily. We propose and investigate the sentiment from a popular real-time microblogging service, YouTube, where real-time reactions are posted by the user and we find their opinions on almost “everything”. Social networking sites like YouTube, Twitter, Facebook, Instagram, Orkut etc. are great sources of communication for internet users. So this becomes an important source for understanding the opinions, views or emotions of people.

General idea about our model - There is a growing fear that toxic comments are a threat to the community of people in online conversations. The organizations and groups that are involved in maintaining the social media sites and online platforms are attempting to detect and reduce toxic contents, to stop losing members of their user groups, and also prevent the loss of future users. This project aimed to establish a toxicity classification scheme in online comments, based on the vocabulary and other characteristics in sentences. In this project, standard dataset of https://huggingface.co/datasets/ucberkeley-dlab/measuring-hate-speech and a multi-label distilbert model has been used

Table of Contents

Chapter 1

Introduction

Social networking sites and online communities have been very popular in recent times and essential for the exchange of information and social experiences. These sites empower their users with an excellent forum for sharing their feelings and views. Cyberbullying and abuse have now become severe issues for many users who cause psychological problems, such as depression and even suicide (Alexander and Krans 2016).

Online abuses may fall under many toxic definitions, such as identity-based hatred, threat, absurd, and insult (Young 2016). Besides, a single statement may contain various types of toxicity at the same time. If the actions, thoughts, or feelings of one person are influenced in a social network by another person, a process called "social impact". Social impact is the main question for social networking analysis, and it gives rise to numerous real-world applications. Recognizing the impact protocol of a social network may promote the achievement of the complicated structures that govern the social network processes. (Latkin and Knowlton 2015)

Recently, social media has become an essential medium of expression and a way for user preferences to be observed and predictions in various contexts chosen. Many social media sites provide easy dissemination of multimedia content. They can also be used as information for early notice, publicity, viral ads, emergency management, and, most generally, to help and/or educate other consumers. Natural Language Processing has been around a few decades now and can be used to gain a greater understanding of the text, such as the structure of the text and meaning at several levels that include document, paragraph, and sentence. This allows the individual to define classes inside language, how the terms refer to one another perhaps one of the most challenging activities — toxicity analysis (Mohammad n.d.). The detection of harmful remarks is an important area of study in natural language processing (NLP). The main objective is to evaluate the toxicity and behaviors expressed in words and their contexts.

Chapter 2
Literature Review

With the ongoing pandemic, there are more people online than ever. In fact, 84% of young Canadians actively rely on social media such as Facebook to communicate with others. However, the internet also gives negative individuals "a safe space to explore extreme ideologies and intensify their hate without consequence ... until it explodes in the real world", said Shannon Martinez from the Free Radicals Project. The goal of this project is to create a safer environment online. By detecting toxic comments on social media, they can be easily reported and removed. In the long term, this would allow people to better connect with each other in this increasingly digital world. This problem can be tackled best with a neural network because it is hard for hard-coded algorithms to understand and detect human speech, and costly for companies to hire humans to identify these comments. Therefore, NLP with neural networks has been found to be the most effective way to do so in the industry, with an estimated market size of US$ 26.4 billion by 2024 . Background Before deep learning (NLP), companies resorted to ineffective methods of identifying hate speech, such as simple keyword searches (bag of word). This method has “high recall but leads to high rates of false positives”, mistakenly removing normal conversation. Recently, research has already been conducted in the deep learning field to identify hate speech. A paper published in 2018 utilizes various word embeddings to train a CNN_GRU model, achieving 90% accuracy on 3 different classes. In addition, many social media companies have invested in methods to eliminate online hate speech. In July 2020, Facebook Canada announced that it is "teaming up with Ontario Tech University's Centre on 3 Hate, Bias and Extremism to create what it calls the Global Network Against Hate", for which Facebook will invest $500,000 to spot online extremism and countering methods.

Chapter 3

Methodology and Techniques

3.1 Approach & Methodology/Techniques:

In the machine learning field, classification methods have been developed, which use different strategies to classify unlabeled data. Classifiers could require training data. It is important to mention that training a classifier effectively will make future predictions easier.

The approach In this section, we will present the dataset used, and our methodologies to recognize emotions of English tweets using:

1. **LSTM: LSTM stands for Long-Short Term Memory. LSTM is a type of recurrent neuralnetwork but is better than traditional recurrent neural networks in terms of memory**
2. **BERT: BERT stands for Bidirectional Encoder Representations from Transformers and itis a state-of-the-art machine learning model used for NLP tasks**
3. **DistilBERT: DistilBERT stands for Distillation BERT. It is a small, fast, cheap and light Transformer model based on the BERT architecture. Knowledge distillation is performed during the pre-training phase to reduce the size of a BERT model by 40%.**

3.2 Dataset and Model Preparation

Dataset

This project uses dataset of measuring-hate-speech project which began in early 2017 at UC Berkeley's D-Lab, to create a multi-label model. Which can distinguish the different toxicity categories in a message. Though this contains many comments, there is a class disparity in the dataset. This is a public release of the dataset described in Kennedy et al. (2020), consisting of 39,565 comments annotated by 7,912 annotators, for 135,556 combined rows.

The primary outcome variable is the "hate speech score" but the 10 constituent labels (sentiment, (dis)respect, insult, humiliation, inferior status, violence, dehumanization, genocide, attack/defense, hate speech benchmark) can also be treated as outcomes. Includes 8 target identity groups (race/ethnicity, religion, national origin/citizenship, gender, sexual orientation, age, disability, political ideology) and 42 identity subgroups.

This dataset card is a work in progress and will be improved over time. This project tries to deal with toxic behavior which discourages online participation by helping to simplify the identification of toxic comment.

This contributes to growing research on online dispute identification.

Key dataset columns

- **hate_speech_score - continuous hate speech measure, where higher = more hateful and lower = less hateful**
- **text - lightly processed text of a social media post**
- **comment_id - unique ID for each comment**
- **annotator_id - unique ID for each annotator**
- **sentiment - ordinal label that is combined into the continuous score**
- **respect - ordinal label that is combined into the continuous score**
- **insult - ordinal label that is combined into the continuous score**
- **humiliate - ordinal label that is combined into the continuous score**
- **status - ordinal label that is combined into the continuous score**
- **dehumanize - ordinal label that is combined into the continuous score**
- **violence - ordinal label that is combined into the continuous score**
- **genocide - ordinal label that is combined into the continuous score**
- **attack_defend - ordinal label that is combined into the continuous score**
- **hatespeech - ordinal label that is combined into the continuous score**
- **annotator_severity - annotator's estimated survey interpretation bias**

3.2.1. Data Visualization

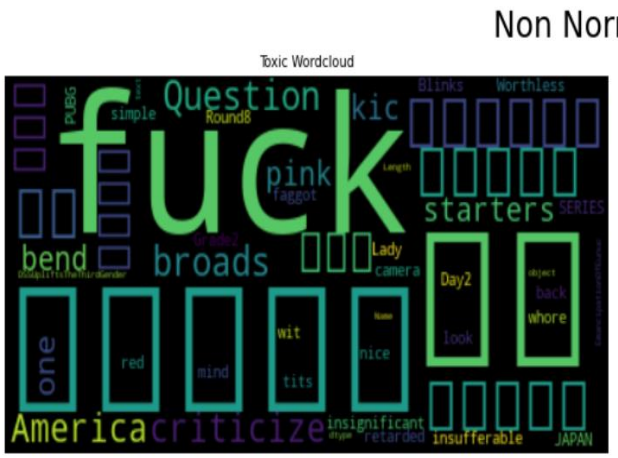

Fig 3.2.1.1 Word cloud for toxic words

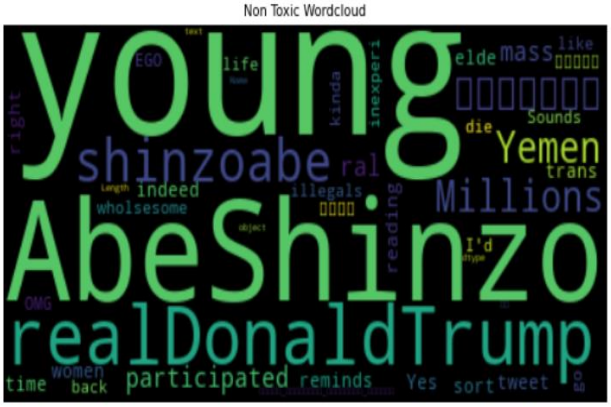

Fig 3.2.1.2Word cloud for non-toxic words

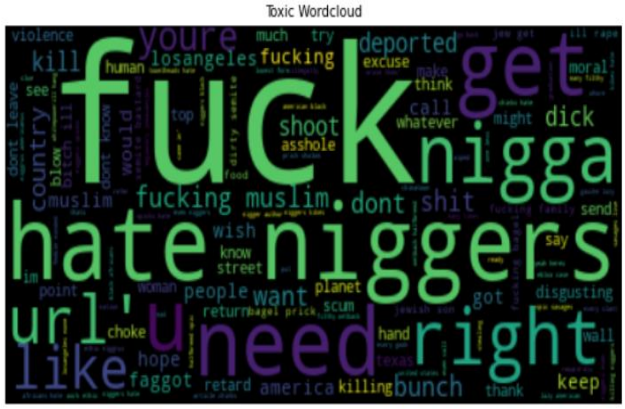

Fig 3.2.1.3 Word cloud for toxic words

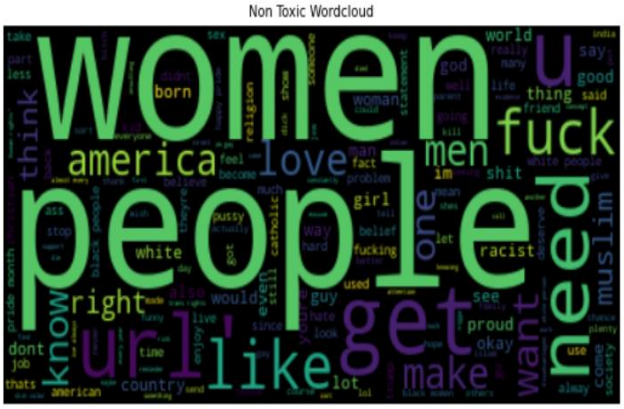

Fig 3.2.1.4 Word cloud for non-toxic words

The above images show the word clouds for the toxic and non-toxic word where the size of each wordindicates its frequency/importance.

3.2.2 Data Preprocessing

Data Preprocessing Pre-processing is the set of transformations applied to the data before doing the actual analytics. Data collected from various sources exist in raw format which needs to beprocessed before it is analyzed.

It helps in transforming the raw data into a clean data set that excludes noise and thus gives better results. Preprocessing the text involves various steps which are depicted in the Figure and explained below: Conversion of text into uniform format: Tweets will contain letters both in upper and lower case, all letters are converted into lowercase to maintain uniformity which will speed up model training

Removal of punctuation and HTML tags: Punctuation marks and other special characters do not impact the sentiment analysis hence they are removed.

Tokenization: Tokenization is the process of splitting text into smaller units called tokens. In the tweets, sentences or paragraphs are broken down into individual words.

Stop-word removal: Most common words appearing in the sentences like “a”, “this” etc. are called stop-words.

The presence or absence of these words does not play a role in analyzing the emotion of the sentence; hence these stop words are removed and their removal helps in reducing the computation time and also helps in processing large amounts of data. The steps included in sentiment analysis with emoticons are as follows: Convert emoticon to Unicode. Check the Unicode sequence in the dataset and match it with the respective textual sequence. Replace the Emoticon Unicode sequence with the textual sequence.

3.3.1 Model Description

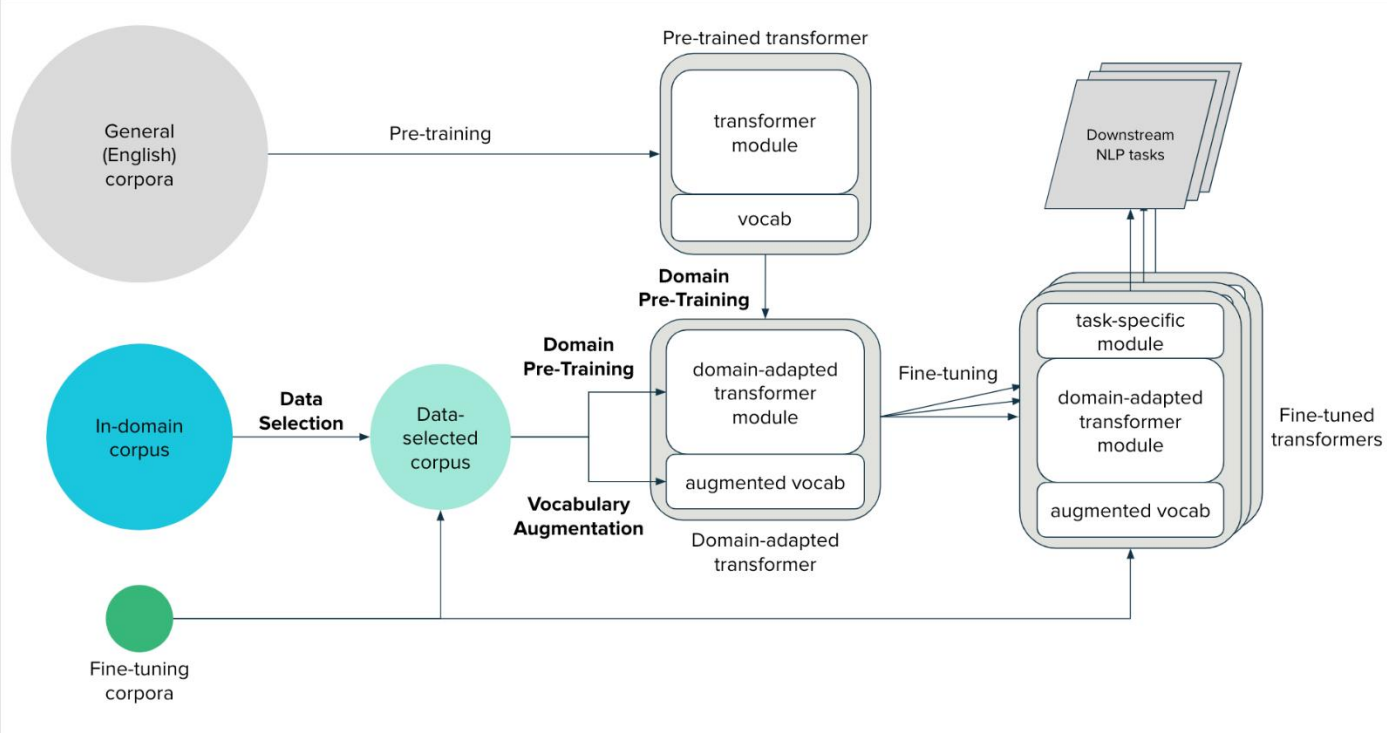

Fig. 3.3.1 - Transformers

3.3.1.1 LSTM

LSTM stands for Long-Short Term Memory. LSTM is a type of recurrent neural network but is better than traditional recurrent neural networks in terms of memory. Having a good hold over memorizing certain patterns LSTMs perform fairly better. As with every other NN, LSTM can have multiple hidden layers and as it passes through every layer, the relevant information is kept and all the irrelevant information gets discarded in every single cell.

LSTM has 3 main gates:

- **FORGET Gate**
- **INPUT Gate**
- **OUTPUT Gate**

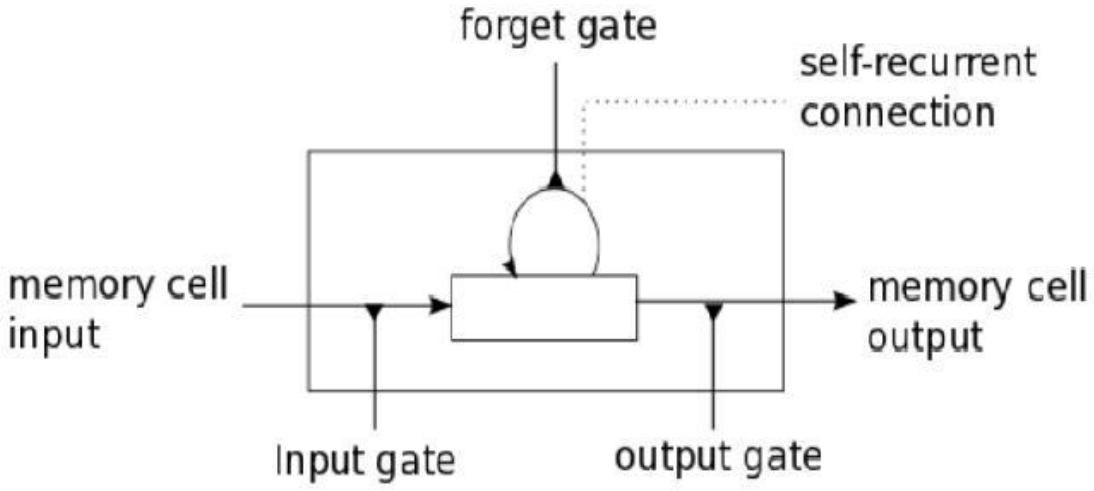

Fig 3.3.1.1.1- Gate of LSTM.

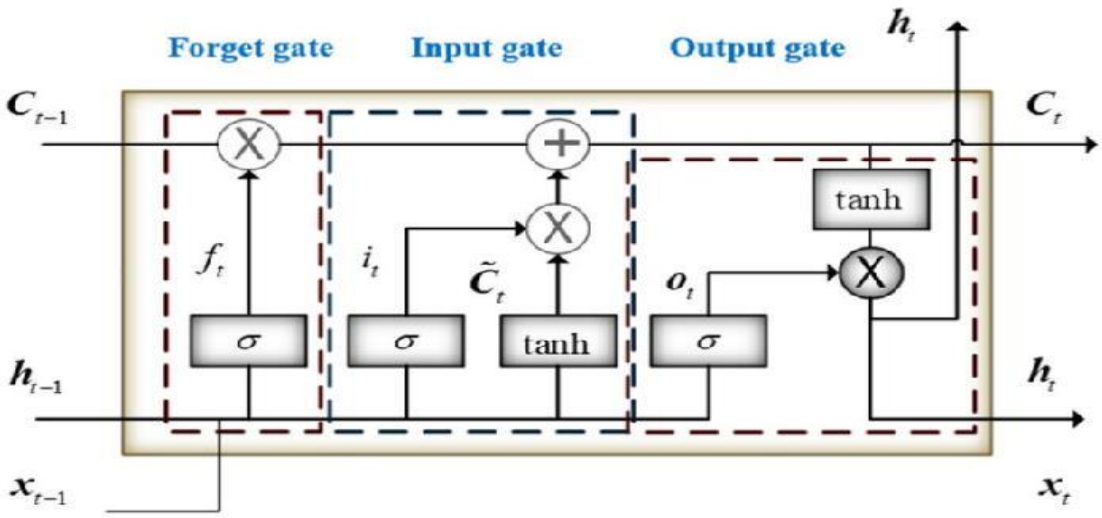

Fig 3.3.1.1.2 -LSTM architecture.

3.3.1.1.1 FORGET Gate:

This gate is responsible for deciding which information is kept for calculating the cell state and which is not relevant and can be discarded. The ht-1 is the information from the previous hidden state (previous cell) and the xt is the current cell information. These are the 2 inputs given to the Forget gate. They are passed through a sigmoid function and the ones tending towards 0 are discarded, and others are passed further to calculate the cell state.

3.3.1.1.2 INPUT Gate:

Input Gate updates the cell state and decides which information is important and which is not. As forget gate helps to discard the information, the input gate helps to find out important information and store certain data in the relevant memory. ht-1 and xt are the inputs that are both passed through sigmoid and tanh functions respectively. tanh function regulates the network and reduces bias.

3.3.1.1.3 OUTPUT Gate:

The Output gate's last gate decides what the next hidden state should be. ht-1 and xt are passed to a sigmoid function. Then the newly modified cell state is passed through the tanh function.

3.3.1.2 BERT

BERT stands for **Bidirectional Encoder Representations from Transformers** and it is a state-of-the-art machine learning model used for NLP tasks. Jacob Devlin and his colleagues developed BERT at Google in 2018. Devlin and his colleagues trained the BERT on English Wikipedia (2,500M words) and Books Corpus (800M words) and achieved the best accuracy for some of the NLP tasks in 2018.

BERT can be used for a variety of NLP tasks such as:

- **Text Classification or Sentence Classification**
- **Semantic Similarity between pairs of Sentences**
- **Question Answering Task with paragraph**
- **Text summarization**
- **Machine translation**
- **Text Generator**
- **Normal Question answering task, etc.**

It is designed to pre-train deep bidirectional representations from the unlabeled text by jointly conditioning on both left and right contexts. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks. First, it’s easy to get that BERT stands for Bidirectional Encoder Representations from Transformers. Each word here has a

meaning to it and we will encounter that one by one in this article. For now, the key takeaway from this line is that BERT is based on the Transformer architecture. Second, BERT is pre-trained on a large corpus of unlabeled text including the entire Wikipedia and Book Corpus. This pre-training step is half the magic behind BERT's success. This is because as we train a model on a large text corpus, our model starts to pick up a deeper and more intimate understanding of how the language works. This knowledge is the swiss army knife that is useful for almost any NLP task. Third, BERT is a "deeply bidirectional" model. Bidirectional means that BERT learns information from both the left and the right sides of a token's context during the training phase.

BERT is an Encoder stack of transformer architecture. A transformer architecture is an encoder-decoder network that uses self-attention on the encoder side and attention on the decoder side. **BERT-BASE** has 12 layers in the Encoder stack while **BERT-LARGE** has 24 layers in the Encoder stack. These are more than the Transformer architecture described in the original paper. BERT architectures also have larger feedforward networks and more attention heads than the Transformer architecture suggested in the original paper.

3.3.1.2.1 BERT's Architecture:

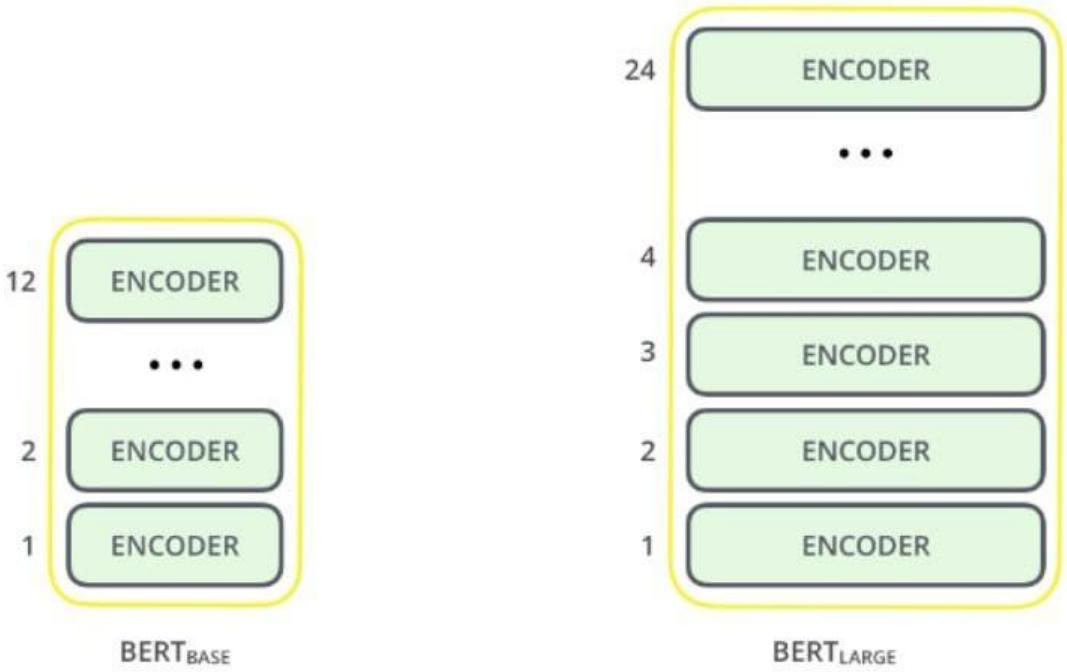

Fig 3.3.1.2.1 BERT ARCHITECTURE.

All of these Transformer layers are Encoder-only blocks.

The BERT architecture builds on top of the Transformer. We currently have two variants available :

- **BERT Base: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters.**
- **BERT Large: 24 layers (transformer blocks), 16 attention heads and 340 million parameters.**
- **DistilBERT: DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% fewer parameters than Bert-base-uncased and runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.**

3.3.1.2.2 Text Preprocessing

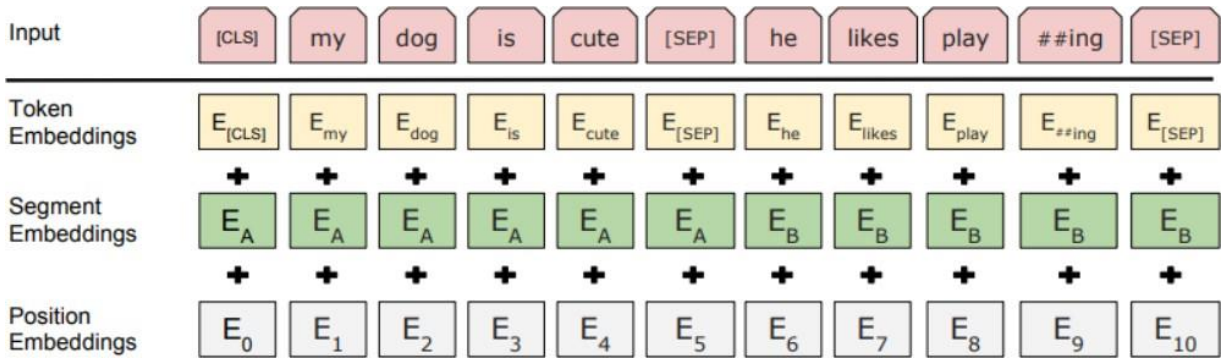

Fig 3.3.1.2.2 BERT Embedding.

In-Text preprocessing BERT uses three types of embeddings :

- **Position Embeddings: BERT learns and uses positional embeddings to express the position of words in a sentence. These are added to overcome the limitation of the Transformer which, unlike an RNN, is not able to capture "sequence" or "order" information**
- **Segment Embeddings: BERT can also take sentence pairs as inputs for tasks (Question-Answering). That's why it learns a unique embedding for the first and the second sentences to helpthe model distinguish between them.**
- **Token Embeddings: These are the embeddings learned for the specific token from the Word Piecetoken vocabulary**

These combinations of preprocessing steps make BERT so versatile. This implies that without making any major change in the model's architecture, we can easily train it on multiple kinds of NLP tasks.

3.2.1.2.3 Pre-training Tasks

BERT's state-of-the-art performance is based on two things. First, novel pre-training tasks called **Masked Language Model (MLM)** and **Next Sentence Prediction (NSP)**. Second, a lot of data and compute power to train BERT-MLM makes it possible to perform bidirectional learning from the text, i.e. it allows the model to learn the context of each word from the words appearing both before and after it.

3.3.1.3. DistilBERT

Transfer Learning methods are primarily responsible for the breakthrough in Natural Learning Processing(NLP) these days. It can give state-of-the-art solutions by using pre-trained models to save us from the high computation required to train large models. This post gives a brief overview of DistilBERT, one outstanding performance shown by TL on natural language tasks, using some pre-trained model with knowledge distillation.

Due to the large size of BERT, it is difficult for it to put it into production. Suppose we want to use these models on mobile phones, so we require a less weight yet efficient model, that's when Distil-BERT comes into the picture. Distil-BERT has 97% of BERT's performance while being trained on half of the parameters of BERT. BERT-base has 110 parameters and BERT-large has 340 parameters, which are hard to deal with. For this problem's solution, distillation technique is used to reduce the size of these large models.

3.3.1.3.1 Knowledge Distillation

It is considered as a knowledge-transfer model from student to teacher. In this technique, a larger

model/ensemble of models is trained, and a smaller model is created to mimic that large one. Distillation refers to copy dark knowledge, for example, a desk chair can be mistaken for an armchair but it should not be mistaken with mushroom. In another way, it's concept is similar to label smoothing, it prevents the model to be too sure about its prediction.

3.3.1.3.2 DistilBERT Architecture

Student Architecture/DistilBERT Architecture: General Architecture is the same as BERT except for removing token-type embeddings and the pooler while reducing the number of layers by a factor of 2, which largely impacts computation efficiency.

Student initialization: It is important to find the right time to initialize the sub-network for its convergence during the model training. Hence, initialize the student from the teacher by taking one layer out of two.

Distillation: The model has been distilled on very large batches using dynamic masking and with the next sentence prediction(NSP). Here, masking and NSP referred to the process where a word to be predicted is converted to ["MASK"] in the Masked Language model, and the entire sequence is trained to predict that particular word.

Data and compute power: The model trained on the concatenated dataset of English Wikipedia and Toronto Book Corpus[Zhu et al., 2015] on 8 16GB V100 GPUs for approximately 90 hours.

3.3.1.3.3 Experiment Results

General Language Understanding: DistilBERT retains 97% performance of the BERT with 40% fewer parameters. This performance is checked on the General Language Understanding Evaluation(GLUE) benchmark, which contains 9 datasets to evaluate natural language understanding systems.

	BERT	RoBERTa	DistilBERT	XLNet
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base:** 66	**Base**: ~110 **Large**: ~340
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large:** 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.	**Large:** 512 TPU Chips x 2.5 days; 5 times more than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT	2-15% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.	**Base**: 16 GB BERT data **Large**: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation	Bidirectional Transformer with Permutation based modeling

Fig. 3.3.1.3.1 - BERT, RoBERTa, DistilBERT, XLNet — which one to use?

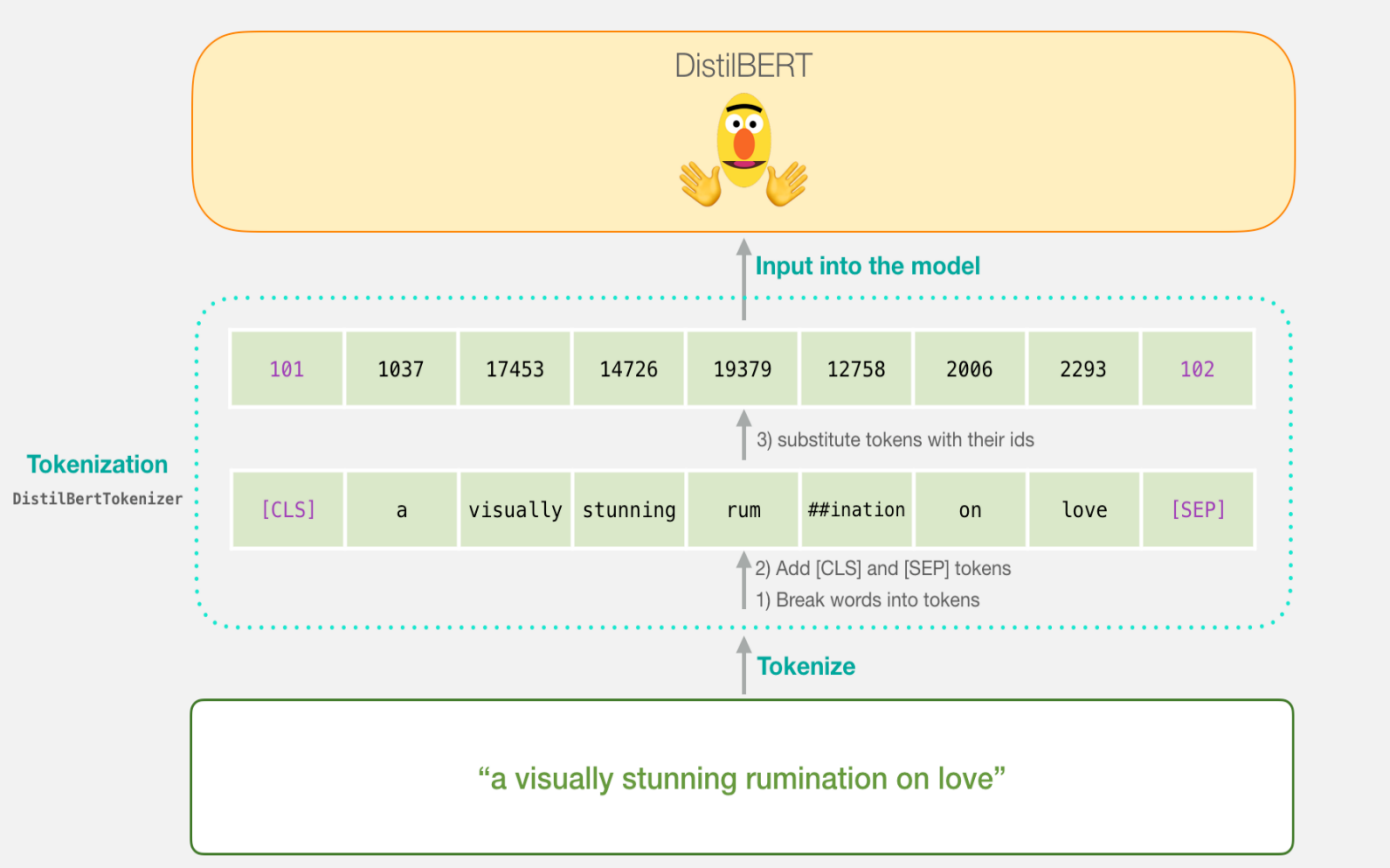

Fig. 2.3.5.2 – A Visual Guide to DistilBert

Chapter 4

Implementation

4.1 Bidirectional LSTM Model Implementation

```
In [57]: model.summary()
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, None, 32)          6400032

 bidirectional_3 (Bidirectio  (None, 64)               16640
 nal)

 dense_12 (Dense)            (None, 128)               8320

 dense_13 (Dense)            (None, 256)               33024

 dense_14 (Dense)            (None, 128)               32896

 dense_15 (Dense)            (None, 10)                1290

=================================================================
Total params: 6,492,202
Trainable params: 6,492,202
Non-trainable params: 0
_________________________________________________________________
```

Fig 4.1.1 Bidirectional LSTM Model.

4.2 DistilBERT Model Implementation

DistilBERT Model is as follows:

```
DistilBERTClass(
  (l1): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
```

```
      (out_lin): Linear(in_features=768, out_features=768, bias=True)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
      (lin1): Linear(in_features=768, out_features=3072, bias=True)
      (lin2): Linear(in_features=3072, out_features=768, bias=True)
      (activation): GELUActivation()
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (1): TransformerBlock(
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
      (q_lin): Linear(in_features=768, out_features=768, bias=True)
      (k_lin): Linear(in_features=768, out_features=768, bias=True)
      (v_lin): Linear(in_features=768, out_features=768, bias=True)
      (out_lin): Linear(in_features=768, out_features=768, bias=True)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
      (lin1): Linear(in_features=768, out_features=3072, bias=True)
      (lin2): Linear(in_features=3072, out_features=768, bias=True)
      (activation): GELUActivation()
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (2): TransformerBlock(
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
      (q_lin): Linear(in_features=768, out_features=768, bias=True)
      (k_lin): Linear(in_features=768, out_features=768, bias=True)
      (v_lin): Linear(in_features=768, out_features=768, bias=True)
      (out_lin): Linear(in_features=768, out_features=768, bias=True)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
      (lin1): Linear(in_features=768, out_features=3072, bias=True)
      (lin2): Linear(in_features=3072, out_features=768, bias=True)
      (activation): GELUActivation()
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (3): TransformerBlock(
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
      (q_lin): Linear(in_features=768, out_features=768, bias=True)
      (k_lin): Linear(in_features=768, out_features=768, bias=True)
      (v_lin): Linear(in_features=768, out_features=768, bias=True)
      (out_lin): Linear(in_features=768, out_features=768, bias=True)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
      (4): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
      (5): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=10, bias=True)
)
```

In [ ]:

```python
# Creating the customized model, by adding a drop out and a dense layer on top of distil bert to get the final output for th

class DistilBERTClass(torch.nn.Module):
    def __init__(self):
        super(DistilBERTClass, self).__init__()
        self.l1 = DistilBertModel.from_pretrained("distilbert-base-uncased")
        self.pre_classifier = torch.nn.Linear(768, 768)
        self.dropout = torch.nn.Dropout(0.1)
        self.classifier = torch.nn.Linear(768, 10)

    def forward(self, input_ids, attention_mask, token_type_ids):
        output_1 = self.l1(input_ids=input_ids, attention_mask=attention_mask)
        hidden_state = output_1[0]
        pooler = hidden_state[:, 0]
        pooler = self.pre_classifier(pooler)
        pooler = torch.nn.Tanh()(pooler)
        pooler = self.dropout(pooler)
        output = self.classifier(pooler)
        return output

model = DistilBERTClass()
model.to(device)
```

Fig .4.2.1 DistilBERT Model.

Chapter 5

RESULTS

5.1 Bidirectional LSTM

4. Evaluate Model

```
In [29]: from tensorflow.keras.metrics import Precision, Recall, CategoricalAccuracy

In [30]: pre = Precision()
         re = Recall()
         acc = CategoricalAccuracy()

In [31]: for batch in test.as_numpy_iterator():
             # Unpack the batch
             X_true, y_true = batch
             # Make a prediction
             yhat = model.predict(X_true)

             # Flatten the predictions
             y_true = y_true.flatten()
             yhat = yhat.flatten()

             pre.update_state(y_true, yhat)
             re.update_state(y_true, yhat)
             acc.update_state(y_true, yhat)

In [32]: print(f'Precision: {pre.result().numpy()}, Recall:{re.result().numpy()}, Accuracy:{acc.result().numpy()}')
         Precision: 0.7311154007911682, Recall:0.6935917139053345, Accuracy:0.3199999928474426
```

Fig 5.1.1 Bidirection LSTM Precision,Recall,Accuracy

```
In [21]: from matplotlib import pyplot as plt

In [22]: plt.figure(figsize=(8,5))
         pd.DataFrame(history.history).plot()
         plt.show()
         <Figure size 800x500 with 0 Axes>
```

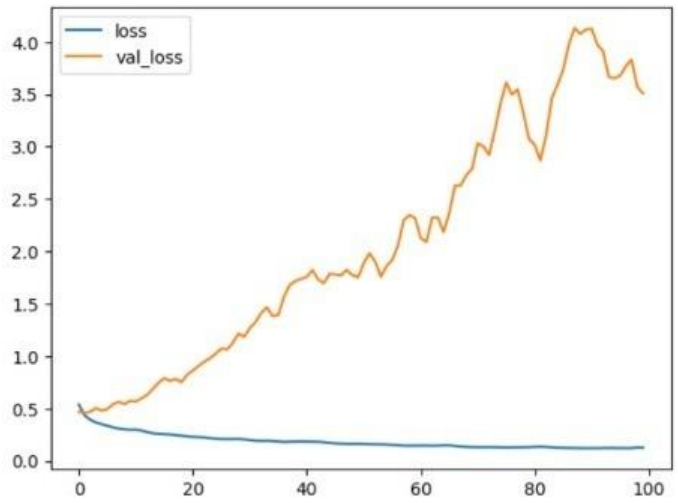

Fig 4.1.2 BIdirection LSTM Loss Vs Val_Loss Plot

5.2 DistilBERT (Distillation Bi-Directional Encoder Representations from Transformers)

In [21]:

```
val_hamming_loss = metrics.hamming_loss(targets, final_outputs)
val_hamming_score = hamming_score(np.array(targets), np.array(final_outputs))

print(f"Hamming Score = {val_hamming_score}")
print(f"Hamming Loss = {val_hamming_loss}")
```

```
Hamming Score = 0.6947781255543787
Hamming Loss = 0.1525321825089447
```

Fig 4.2.1 DistilBERT Model classification report.

Chapter 6

Conclusion

Upon evaluating all the models, we can conclude the following details:

Accuracy:

As far as the accuracy of the model is concerned DistilBERT better than the Bidirectional LSTM model which in turn performsbetter than RNN and other ML models. DistilBERT has shown the accuracy up to 69.47% while Bidirectional LSTM which has an accuracy of 31.99%.

Speed:

DistilBERT model is faster in comparison to Bidirectional LSTM

Chapter 7

Reference

- https://huggingface.co/datasets/go_emotions
- https://www.kaggle.com/datasets/ishantjuyal/emotions-in-text?select=Emotion_final.csv
- https://drive.google.com/file/d/17mtskWyPaztNqPyZ7vDH3Rrn5K0feHRb/view?
- https://data.world/crowdflower/sentiment-analysis-in-text
- https://huggingface.co/datasets/emotion
- https://www.analyticsvidhya.com/blog/2021/06/lstm-for-text-classification/
- https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural- networks-python-keras/
- https://www.kaggle.com/code/ngyptr/multi-class-classification-with-lstm/notebook/
- https://analyticsindiamag.com/guide-to-sentiment-analysis-using-bert/
- https://www.tensorflow.org/text/tutorials/classify_text_with_bert
- https://huggingface.co/docs/transformers/model_doc/bert#berttokenizer
- https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Chapter 8

Future Work

- **The project model can be made on the BERT (Bidirectional Encoder Representations from Transformers) which will take more time but will give better accuracy**
- **The deployment can be done on streaming data**
- **Also instead of one YouTube video, more than one video/channels can be taken as input and;**
- **The video with most toxic or dangerous comments can be flagged.**