

A project on

MULTI-DOMAIN RECOGNITION & TEXT CLASSIFICATION

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology in Information Technology

by

PRIYAL BHARDWAJ (18BIT0272)



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SITE

May, 2022

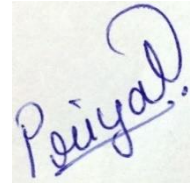
DECLARATION

I, Priyal Bhardwaj hereby declare that the thesis entitled “Multi-Domain Recognition & Text Classification” submitted by me, for the award of the degree of *Bachelor of Technology in Information Technology* to VIT is a record of bonafide work carried out by me under the supervision of Prof. Senthil Kumar N C.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 25-04-2022



Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled “MULTI-DOMAIN RECOGNITION & TEXT CLASSIFICATION” submitted by PRIYAL BHARDWAJ (18BIT0272) SITE VIT, for the award of the degree of Bachelor of Technology in Information Technology is a record of bonafide work carried out by her under my supervision.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The Project report fulfils the requirements and regulations of VIT and in my opinion meets the necessary standards for submission.



25.04.2022

Signature of the Guide

Signature of the HoD

Internal Examiner

External Examiner

ABSTRACT

Along with the advancements made in deep learning, the performance of text classification models have been improved significantly. Nevertheless, the successful training of a good classification model requires a sufficient amount of labelled data, while it is always expensive and time consuming to annotate data. With the rapid growth of digital data, similar classification tasks can typically occur in multiple domains, while the availability of labelled data can largely vary across domains. Some domains may have abundant labelled data, while in some other domains there may only exist a limited amount (or none) of labelled data. Meanwhile text classification tasks are highly domain-dependent — a text classifier trained in one domain may not perform well in another domain.

For the course of this project, I will be investigating the performance of various models ranging from LSTM and KERAs to heavy encoder algorithms like BERT for multi-domain text classification, which separate the domain-specific and domain common knowledge by using a set of shared word-level parameters to encode word hidden states with domain knowledge, and using a set of shared sentence-level parameters to extract the common information, using a set of domain-specific parameters for storing private knowledge.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. N C Senthilkumar, Department of Software and Systems Engineering, School of Information Technology and Engineering, Vellore Institute of Technology, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine Learning.

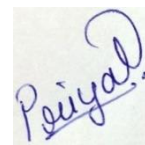
I would like to express my gratitude to Dr. G. Viswanathan, Mr. G.V. Selvam, Prof. Rambabu Kodali, Dr. S. Narayanan, Dr. Sumathy S., School of Information Technology and Engineering, Vellore Institute of Technology, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Usha Devi G, The Head of Department, School of Information Technology and Engineering, Vellore Institute of Technology, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: 27-03-2022



Priyal Bhardwaj

CONTENTS

LIST OF TABLES.....	viii
----------------------------	-------------

LIST OF FIGURES.....	ix
-----------------------------	-----------

LIST OF ACRONYMS.....	x
------------------------------	----------

CHAPTER 1

INTRODUCTION.....	1
--------------------------	----------

1.1 OBJECTIVE.....	1
--------------------	---

1.2 MOTIVATION.....	3
---------------------	---

1.3 BACKGROUND.....	4
---------------------	---

CHAPTER 2

LITERATURE SURVEY.....	6
-------------------------------	----------

CHAPTER 3

REQUIREMENTS.....	12
--------------------------	-----------

3.1 SOFTWARE.....	12
-------------------	----

3.2 HARDWARE.....	13
-------------------	----

CHAPTER 4

DETAILED DESIGN.....	14
-----------------------------	-----------

4.1 SYSTEM ARCHITECTURE.....	14
------------------------------	----

4.2 UML DIAGRAM.....	15
----------------------	----

4.3 MODULE DESCRIPTION.....	16
-----------------------------	----

4.4 DATA FLOW DIAGRAM.....	22
 CHAPTER 5	
IMPLEMENTATION.....	23
 CHAPTER 6	
CONCLUSION.....	51
 CHAPTER 7	
FUTURE WORK.....	52
 REFERENCES.....	53

LIST OF TABLES

5.1	F1 Score – Severe Toxic.....	45
5.2	F1 Score – Obscene.....	45
5.3	F1 Score – Threat.....	46
5.4	F1 Score – Insult.....	47
5.5	F1 Score – Identity Hate.....	48
5.6	F1 Score – Master Dataframe.....	48

LIST OF FIGURES

1.1.1	Conditional Probability.....	2
1.3.1	Multi-Class vs Multi-Domain.....	4
4.1.1	ML-Architecture.....	14
4.1.2	Overall Architecture.....	14
4.2.1	UML Diagram.....	15
4.3.1	Data Sampling.....	16
4.3.2	LSTM with Single Output Layer.....	18
4.3.3	LSTM with Multiple Output Layers.....	19
4.3.4	BERT Architecture.....	20
4.4.1	Data Flow Diagram.....	21
5.1	IDF Score Formula.....	31
5.2	TF-IDF Score Formula.....	31

LIST OF ACRONYMS

DM	Data Mining
ML	Machine Learning
NLP	Natural Language Processing
MDTC	Multi-Domain Text Classification
MCC	Multi-Class Classifier
MDC	Multi-Domain Classifier
BR	Binary Relevance
CC	Classifier Chains
KNN	K-Nearest Neighbor
SVM	Support Vector Machine
HAN	Hierarchical Attention Network
TC	Text Classifier
LSTM	Long-Short Term Memory
Bi-LSTM	Bidirectional Long-Short Term Memory
MAN	Multinomial Adversarial Network
CAN	Conditional Adversarial Network
BERT	Bidirectional Encoder Representations from Transformers
ICA	Independent Component Analysis
TTS	Text-to-Speech
UML	Unified Modelling Language
DFD	Data Flow Diagram
EDA	Exploratory Data Analysis
NA	Not Available

Chapter 1

INTRODUCTION

1.1. OBJECTIVE

The digital information available on the internet is growing day by day. Human intervention is not at all sufficient to analyse all this data. Many techniques are required to extract all this data, analyse it and draw meaningful patterns. Text classification is a machine-learning technique, in which the text is classified into different categories by different classification algorithms. Every classifier has its own way of gathering the features and using them in order to classify the text. Text classification has wide range of applications like spam filtering, genre categorization, language identification, routing the emails, sentiment analysis and many such applications. All these applications use wide range of text classification techniques.

Now-a-days sentiment classification is widely used by the businesses to know their brand value and to gain business intelligence. Feedback given by the customers about businesses is of great importance to them. Using the text from the feedback(reviews), the data can be mined for various interesting facts which might be helpful in increasing the market for a particular business. Mining the text might be of high importance when compared to the star rating system, because sentiment of the customer can't be seen in the latter. A particular user might give a business 3 star rating, but the review written by that user might not match with the star rating given. In this manner, sentiment classification has gained huge importance in the business world.

In this Project, supervised learning is the learning approach that has been adapted by the classifiers. In supervised learning, the training dataset consists of the data instances along with the labels provided for them. Initially the classifiers learn the data in a supervised fashion and then the model is evaluated with the data instances(test set) that doesn't consist of labels. The classifier, during evaluation, outputs the labels for testing instances and then the accuracies are computed.

Many features like stop-words, stemmers, n-grams, parts of speech tagging have

been used on this dataset for the classification task. Models like Bayesian, Random Forest and Stochastic Gradient Descent have been tried on this dataset for classifying the reviews under different labels.

Given one or several sentences, Multi-Domain Text Classification (MDTC) is a complex natural language processing task which requires to predict multiple domains related to one instance. First of all, To describe the MDTC task we would like to define some notations, assuming that we have a domain list with a domains in total, $Y = \{Y_1, Y_2, Y_3, \dots, Y_a\}$, MDTC is asked to assign a subset y containing n domains in the domain space Y to instance x . Unlike traditional Single-Domain Classification that each sample has only one domain, each instance in MDTC task may have several related domains. The MDTC task can be modelled as finding a domain list subset y^* that maximizes the conditional probability $p(y|x)$:

$$p(y|x) = \prod_{i=1}^n p(y_i | y_1, y_2, y_3, \dots, y_{i-1}, x)$$

Fig.1.1.1 Conditional Probability

where y_i is the next predict Domain to a sequence x , and y_1, y_2, \dots, y_{i-1} are predicted domains.

1.2. MOTIVATION

The motivation of this paper is to effectively solve limited-information text classification problem.

Both Multi-Domain Text Classification and Sentiment Analysis for short texts records are challenging because of the limited contextual information and semantic diversity which can lead to interference building language model. To build a robust and effective model, an intuition concept is to combine limited text content with prior technical knowledge by using more informative word embedding methods and implementing more powerful feature extraction method to represent information-limited data. In this paper, we compared result from each model and it shows that convolutional neural network is the most suitable feature extraction model for key words contributing data set, further we discussed the contribution between key words and contextual information in a practical text classification problem.

Word Clouds are the most effective way of visualizing the text, as the words in the text are shown in a way as if their frequency is proportional to the font size. Usually this is done in a static way, meaning the summarization is done for static text. The authors of [4], have done research on the importance of word clouds for text analytics. A prototypical system called the word cloud explorer, into which the natural language processing techniques are integrated, have been developed in [4], and this completely relies on word clouds as the visualization method. Features like search, click based filter, part-of-speech filter, stop word editor and co-occurrence cloud have also been implemented in this system. They evaluated this system in such a way that is useful in qualitative study.

1.3. BACKGROUND

Multi-domain Text Classification (MDTC) is an important task in the field of natural language processing (NLP), which can be applied in many real-world scenarios, such as information retrieval (Gopal and Yang (2010)), tag recommendation (Katakis et al. (2008)), and so on. The target of the MDTC task is to assign multiple labels to each sample in the data set. The difference between Multi-class Classification and Multi-domain Classification is the number of domains that can be assigned to one instance. Consider an example of three classes $C = [\text{Sun}, \text{Moon}, \text{Cloud}]$. In Multi-class case, each sample belongs to only one of C classes, while in Multi-domain case, more than one class in C can associate with a sentence, the figure below shows a scenario to compare MCC and MDC. Normally, for a Multi-domain Classification problem, the data set is more complicated to solve. Currently, there are two groups of methods to solve this type of problem: Problem Transformation and Algorithm Adaptation. (Zhang and Zhou (2014)).

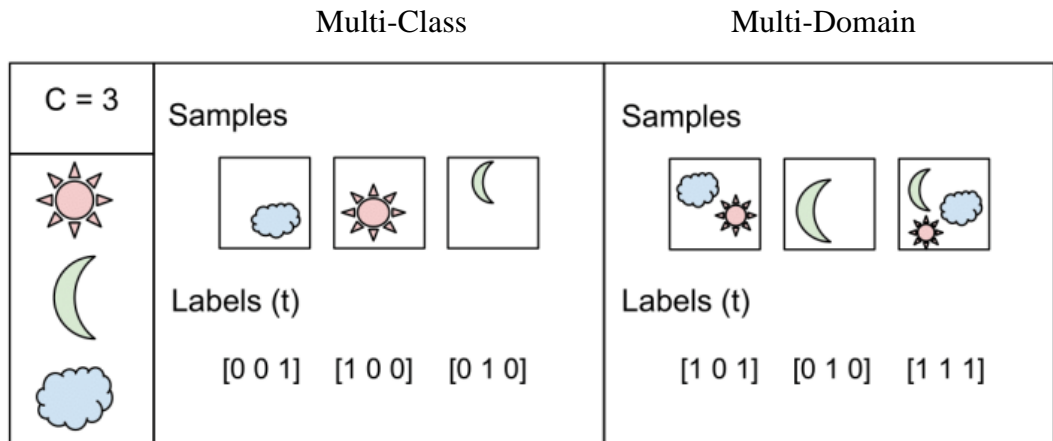


Fig.1.3.1 Multi-Class vs Multi-Domain

The first attempt of Problem Transformation techniques is transforming the Multi-domain Classification problem into one or more Single-domain Classification problems. One way to accomplish this task is called Binary relevance (BR) (Boutell et al. (2004)) which will create an independent binary classifier for each domain of the problem, this is one of the earliest attempts to solve the MDTC task by transforming concept. The main problem of the BR method is that it neglects the correlations between domains, thus ignoring some characteristics of the

problem. One way to solve this is through the Classifier chains (CC) method (Read et al. (2011)), The CC method uses the output of a binary classifier as an input attribute to the next classifier, hence adding relationships among classes in a problem. However, the main flaw of this method is the difficulty of getting the best order of the classifier, and experiments show that it is computationally expensive processing large data sets.

Another solution to MDTC task is algorithm adaptation method. The concept is trying to work directly with a Multi-domain problem by using data mining algorithms. One optimization is a Multi-domain lazy learning approach named ML-KNN (Zhang and Zhou (2007)), in this work, the authors perform an adaptation of the KNN algorithm to allow the use of Multi-domain data. In the first step, all the k nearest neighbors of each instance are identified, after identification, statistical information obtained from the neighbors' domain sets is used to determine domains. However, just as other similar methods such as ML-DT (Clare and King (2001)), Rank-SVM (Elisseeff and Weston (2002)), they are only able to capture domain correlations within the first or second order, and computationally expensive as well. In recent years, neural networks have achieved great success in many fields including NLP. Some neural network models have also been applied in the MLTC task and achieved important progress. For instance, (Zhang and Zhou (2006)) used fully connected neural network with pairwise ranking loss function to tackle the problem. (Kim (2014)) is the first to implement convolutional neural network into Sentence-Level Classification tasks, (Kurata et al. (2016)) introduced CNN to Multi-domain Classification. (Chen et al. (2017)) use CNN and RNN to capture the semantic information of texts. (Yang et al. (2016)) purposed Hierarchical Attention Networks (HAN) to solve Document-level Classification task, this hierarchical structure has two levels of Attention mechanisms used at the word and sentence-level, it purposed an idea of capturing important information in different levels, which is useful especially in scenario that each instance has many sentences.

Chapter 2

LITERATURE SURVEY

[1]. In this paper, the authors investigate SP-LSTM for multi-domain text classification, which separates the domain-specific and domain-common knowledge by using a set of shared word-level parameters to encode word hidden states with domain knowledge, and using a set of shared sentence-level parameters to extract the common information, using a set of domain-specific parameters for storing private knowledge. Results on 16 domains show that their method significantly outperforms traditional shared-private architectures for transfer learning.

[2]. This research paper proposes a new approach for dealing with multi class sentiment classification problems. Generally, a classification model works better when the number of labels in the dataset are less in number, usually two or three. This paper proposes a classification model that deals with multiple labels. This model works in multiple phases and the sentiment of the data instance is fine-tuned in every phase. For instance, phase 1 classifies the data instance under three labels and the other phases further fine tune the sentiment and predicts the final sentiment for the data instance. In this way, as the classification model works in multiple phases, the accuracy doesn't get degraded even when there are multiple labels.

[3]. In this paper, the authors take a step further in their research towards developing a new strategy for high quality text-to-speech (TTS) synthesis among different domains. In this context, it is necessary to select the most appropriate domain for synthesizing the text input to the TTS system, task that can be solved including a text classifier (TC) in the classic TTS architecture. Since speech speaking style and prosody depend on the sequentially and text structure of the message, the TC should consider not only thematic but also stylistic aspects of text. To this end, they introduce a new text modelling scheme based on an associative relational network, which represents texts as a weighted word-based graph. The conducted experiments validate the proposal in terms of both objective (text classification efficiency) and subjective (perceived synthetic speech quality)

evaluation criteria.

[4]. In this paper, the authors have described two approaches for addressing domain adaptation. They have shown that generalizable features can be useful for doing classification in mixed domain datasets, and also shown how establishing relationship mappings between words of different domains can help better understand domain adaptation. Neither technique provided a final solution to this challenging problem, but both provided important glimpses into how to make further headway in the construction of text categorization models that work in multiple domains.

[5]. In this paper, the authors propose a novel *multi-domain active learning* framework to jointly select data instances from all domains with duplicate information considered. In their solution, a shared subspace is first learned to represent common latent features of different domains. By considering the common and the domain-specific features together, the model loss reduction induced by each data instance can be decomposed into a common part and a domain-specific part. In this way, the duplicate information across domains can be encoded into the common part of model loss reduction and taken into account when querying. They compared their method with the state-of-the-art active learning approaches on several text classification tasks: sentiment classification, newsgroup classification and email spam filtering. The experiment results showed that their method reduces the human labelling efforts by 33.2%, 42.9% and 68.7% on the three tasks, respectively.

[6]. In this paper, the authors propose a novel completely-shared multidomain neural sentiment classification model to learn domain-aware word embeddings and make use of domain-aware attention mechanism. Their model first utilizes BiLSTM for domain classification and extracts domain-specific features for words, which are then combined with general word embeddings to form domain-aware word embeddings. Domain-aware word embeddings are fed into another BiLSTM to extract sentence features. The domain-aware attention mechanism is used for selecting significant features, by using the domain-aware sentence representation as the query vector. Evaluation results on public datasets with 16

different domains demonstrated the efficacy of their proposed model. Further experiments showed the generalization ability and the transferability of their model.

[7]. In this paper , the authors propose a novel dual adversarial co-learning approach for multi-domain text classification (MDTC). The approach learns shared-private networks for feature extraction and deploys dual adversarial regularizations to align features across different domains and between labelled and unlabelled data simultaneously under a discrepancy-based co-learning framework, aiming to improve the classifiers' generalization capacity with the learned features. They conducted experiments on multi-domain sentiment classification datasets. The results showed the proposed approach achieves the state-of-the-art MDTC performance.

[8]. In this work, the authors propose a multinomial adversarial network1 (MAN) to tackle the real-world problem of multi-domain text classification (MDTC) in which labelled data may exist for multiple domains, but in insufficient amounts to train effective classifiers for one or more of the domains. They provided theoretical justifications for the MAN framework, proving that different instances of MANs are essentially minimizers of various f-divergence metrics (Ali and Silvey, 1966) among multiple probability distributions. MANs are thus a theoretically sound generalization of traditional adversarial networks that discriminate over two distributions. More specifically, for the MDTC task, MAN learns features that are invariant across multiple domains by resorting to its ability to reduce the divergence among the feature distributions of each domain. They presented experimental results showing that MANs significantly outperform the prior art on the MDTC task. They also showed that MANs achieve state-of-the-art performance for domains with no labelled data.

[9]. In this paper, the authors propose conditional adversarial networks (CANs), a framework that explores the relationship between the shared features and the label predictions to impose more discriminability to the shared features, for multi-domain text classification (MDTC). The proposed CAN introduces a conditional domain discriminator to model the domain variance in both shared feature

representations and class-aware information simultaneously and adopts entropy conditioning to guarantee the transferability of the shared features. They provided theoretical analysis for the CAN framework, showing that CAN’s objective is equivalent to minimizing the total divergence among multiple joint distributions of shared features and label predictions. Therefore, CAN is a theoretically sound adversarial network that discriminates over multiple distributions. Evaluation results on two MDTC benchmarks showed that CAN outperforms prior methods. Further experiments demonstrated that CAN has a good ability to generalize learned knowledge to unseen domains.

[10]. In this paper, the authors explored multiple Active Learning strategies using BERT. Their goal was to understand if BERT-based models can prove effective in an Active Learning setting for multi-class text classification. We observed that EGL performed reasonably well across datasets in multi-class text classification. Moreover, Random, EGL and DAL captured diverse and representative samples with relatively lower-class bias. However, unlike Random, EGL and DAL had longer execution times. In future work, they plan to perform a deeper analysis of the observations and also explore ensemble approaches by combining the advantages of each strategy to explore potential performance improvements.

[11]. In this paper, the author proposes a novel approach for multi-class classification problem. In their research, they proposed a solution for text classification to predict the sentiment of movie reviews on a scale of 5 levels using different machine learning algorithms. This paper also demonstrated a multi-tier prediction model for multi-class classification system. The classification of data is improved based on feature selection methods. These experiments and results have outperformed and gave a better classifier for supervised text classification.

[12]. In this paper, the authors seek 1) to advance the understanding of commonly used text classification techniques, and 2) through that understanding, improve the tools that are available for text classification. They begin by clarifying the assumptions made in the derivation of Naive Bayes, noting basic properties and proposing ways for its extension and improvement. Next, they investigated the

quality of Naive Bayes parameter estimates and their impact on classification. Their analysis led to a theorem which gives an explanation for the improvements that can be found in multiclass classification with Naive Bayes using Error-Correcting Output Codes. They used experimental evidence on two commonly-used data sets to exhibit an application of the theorem. Finally, they showed fundamental flaws in a commonly-used feature selection algorithm and develop a statistics-based framework for text feature selection. Greater understanding of Naive Bayes and the properties of text allowed them to make better use of it in text classification.

[13]. In this paper, the authors present a hierarchical text classifier based on Independent Component Analysis (ICA), which is capable of *(i)* organizing the contents of the corpus in a hierarchical manner and *(ii)* classifying the texts to be synthesized according to the learned structure. The document organization and classification performance of our ICA-based hierarchical classifier are evaluated in several encouraging experiments conducted on a journalistic-style text corpus for speech synthesis in Catalan. The experiments demonstrated its good performance for text corpus hierarchization. Moreover, the classifier achieved encouraging results for different training and testing configurations, even when few documents are available. This ability is essential in the MD-TTS context due to the reduced size of the corpus. In addition, as could be expected, the final hierarchical structure is highly dependent on the contents of the corpus.

[14]. In this paper, the authors propose the S-MDMT model for stance detection on tweets. They novelly formulated the task as a multidomain multi-task learning problem. And they employed the shared-private structure in stance detection for the first time to fully exploit the shared stance features across different targets. Experimental results on the SemEval 2016 dataset demonstrated the effectiveness of the proposed S-MDMT model, which outperforms existing target-aware models that using external resources and additional sentiment annotations. They provided a detailed analysis of the shared features across different targets.

[15]. In this paper, the researchers at Amazon proposed X-BERT, the first deep learning approach with finetuned BERT models that achieves state-of-the-art

performance in the XMC problem. The novel semantic label indexing stage endows heterogeneous label partitions that bootstrap various BERT models, resulting in a powerful ensemble model for XMC problem. Quantitatively, on the Wiki-500K dataset, the precision@1 is increased from 60:91% to 67:87% when comparing X-BERT to the strong XMC method Parabel. This amounts to a 11:43% relative improvement over Parabel, which is indeed significant compared to the recent state-of-the-art approach SLICE which has 5:53% relative improvement over Parabel.

Chapter 3

REQUIREMENTS

3.1. SOFTWARE REQUIREMENTS

- Python: Python is a high-level, object-oriented, interactive, and interpreted programming language. It is easy to write, learn and read. It is a free and open-source language. It has a large set of libraries. Python is used in almost all the domains in the computer industry.
- Tensorflow: TensorFlow is an open-source software library for numerical computation. It is widely used in Machine Learning.
- Keras: Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test neural network with minimal line of code. It contains implementation of commonly used neural network components such as layers, activation function, optimizers, etc. to make working with images and text easier.
- Flask: Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.
- Anaconda:
- Visual Studio Code:

3.2. HARDWARE REQUIREMENTS

- CPU: Preferably a quad core intel i3 processor (or equivalent AMD Ryzen processor) or above is required to run the project smoothly.
- GPU: Dedicated NVIDIA GPU is required to run TensorFlow and Keras library.

If the CPU or the GPU of the system is of lower version, then online Jupyter notebook environment such as Kaggle or Google Colab can be used to run the project. They provide optimum CPU and GPU for free and Jupyter notebooks to implement the project. They don't need any setup and runs completely on cloud.

I will be proceeding with implementation on Kaggle.

Chapter 4

DETAILED DESIGN

4.1. SYSTEM ARCHITECTURE

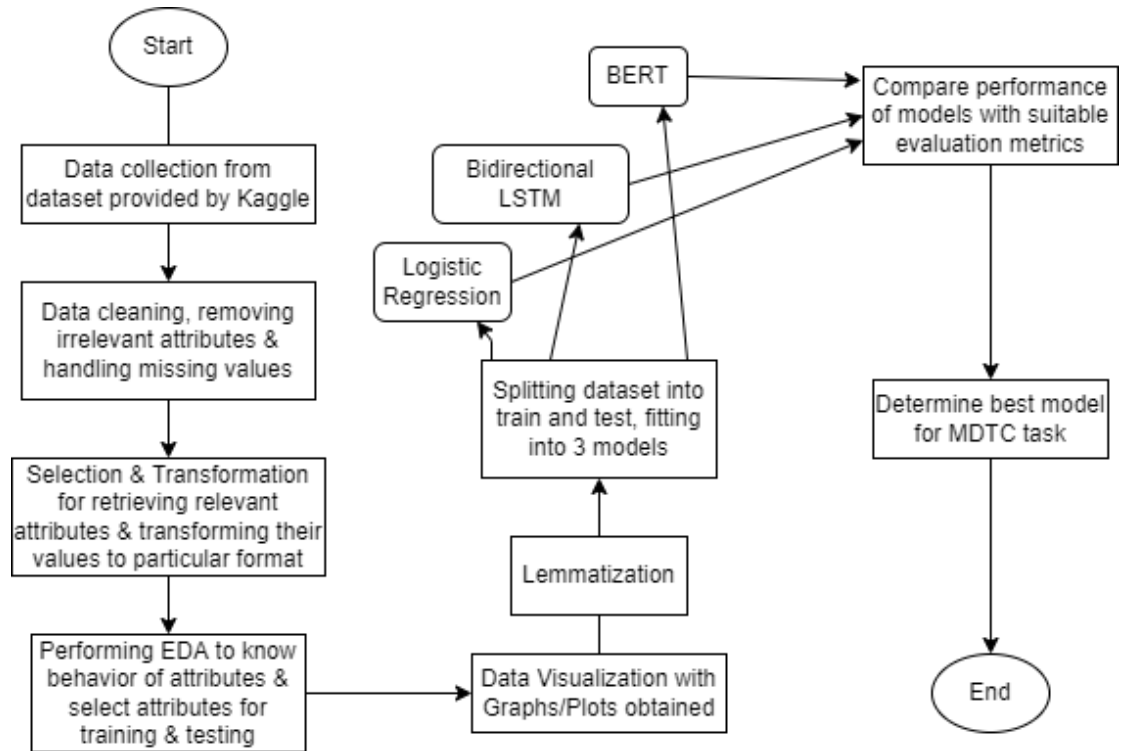


Fig.4.1.1 ML-Architecture

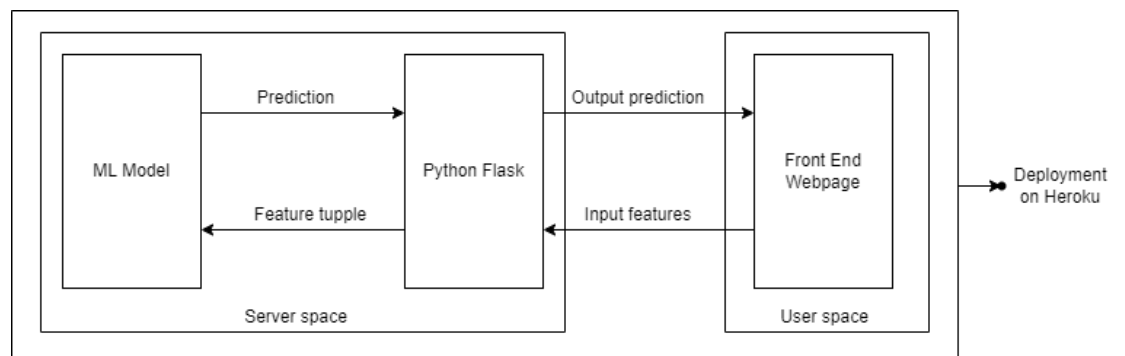


Fig.4.1.2 Overall Architecture

4.2.UML DIAGRAM

Use-Case Diagram:

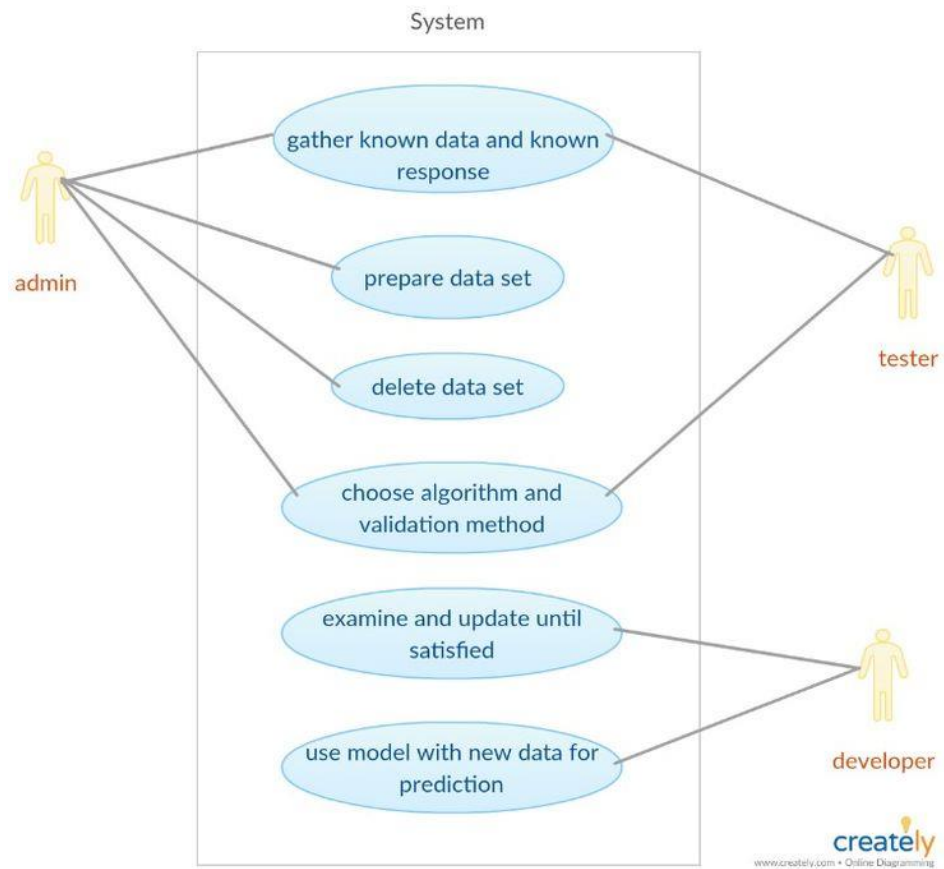


Fig.4.2.1 UML Diagram

4.3.MODULE DESCRIPTION

- Data Pre-processing

This can be considered as one of the most important steps to be performed before selecting the features from the dataset. The main agenda of this particular phase is to remove the unwanted data and retain only the data that is required for feature selection. This phase can be sub-divided into many other steps like data validation, data cleansing, data sampling and so on. The actions performed in each of these steps is discussed below in detail.

- Data Cleansing

Data Cleansing or Data Scrubbing is the process of checking the inappropriate attributes from the dataset and removing them.

- Data Sampling

As discussed earlier the dataset consists of both train-set and test-set. The train-set consists of a valid Sentiment label for each and every instance. The test-set doesn't consist of a Sentiment label and that is what, is to be determined for the test-set. As, the test-set doesn't consist of Sentiment labels, accuracy of the model can't be determined. Therefore, the initial train-set is split into two parts with the split factor being 80:20, 80 for train-set and 20 for test-set.

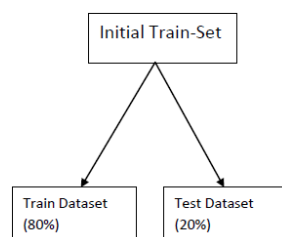


Fig.4.3.1 Data Sampling

Splitting should be done in such a way, that the Sentiment label distribution in the actual train-set should be properly maintained.

- Feature Selection

Feature selection is the process of selecting the features that are most essential and that are more relevant to the machine learning problem,

which is being considered. To further simplify, in the context of Text classification, only the features that help in determining the sentiment of a sentence are to be considered. In Sentiment classification, words play a major role in determining the sentiment of a sentence. Hence, words are said to be the features in the context of Text classification.

There are many advantages in performing the feature selection before building the classifier and modelling the data.

- *Accuracy might improve:* When the relevant features are only selected for modelling, there is every chance for the accuracy to get improved.
- *Training time is reduced:* After removing all the unwanted features, it is very obvious that the training time is reduced.
- *Over-fitting gets reduced:* When the redundant data is less, there is very less probability for the model to make decisions based on noise.

In this project, for the dataset that is being considered, there are 14k unique words, that are being considered as the features for the machine learning model. Furthermore, features like stop-word removal, stemming, n-grams and so on, will also be considered for further fine tuning before modelling the data.

The set of such features used in this project will be discussed in detail in the following sections.

- **Stopwords**

In the case of sentiment classification, after considering words as the features for training the classifier, there are still some words which don't play any role in deciding the sentiment of a particular sentence. In the dataset used for this project, some of the words like {in, the, anywhere, are, around, as, at, be, became, because, become, been, being, between, both, but, by, can, could, detail, each, either, else, elsewhere, etc, even,.....} and many more such words doesn't determine the sentiment of a particular review [10]. So, removing all such words will be advantageous in many ways like:

- Storage space of the feature vector decreases.
- Performance of the classifier increases, which means it's running time decreases.
- There is every chance for the accuracy to improve.

- Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations. EDA is the process of investigating the dataset to discover patterns, and anomalies (outliers), and form hypotheses based on our understanding of the dataset.

EDA involves generating summary statistics for numerical data in the dataset and creating various graphical representations to understand the data better.

The dataset contains comments from Wikipedia's talk page edits. There are six output labels for each comment: toxic, severe_toxic, obscene, threat, insult and identity_hate. A comment can belong to all of these categories or a subset of these categories, which makes it a multi-label classification problem.

The dataset can be downloaded from this [Kaggle link](#). We will only use the "train.csv" file that contains 160,000 records.

- Logistic Regression

By default, logistic regression cannot be used for classification tasks that have more than two class labels, so-called multi-class classification. A logistic regression model that is adapted to learn and predict a multinomial probability distribution is referred to as Multinomial Logistic Regression.

- MDTC model with Single Output Layer

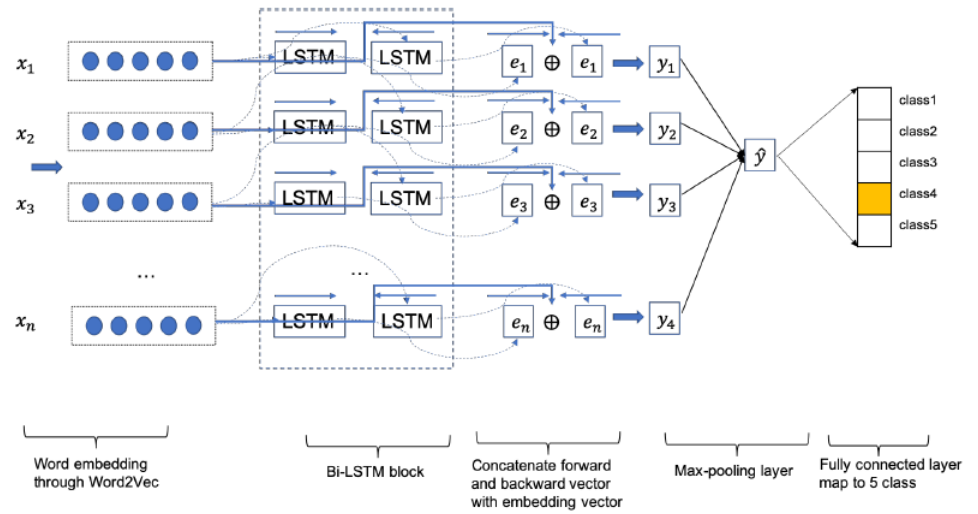


Fig.4.3.2 LSTM with Single Output Layer

- MDTC model with Multiple Output Layers

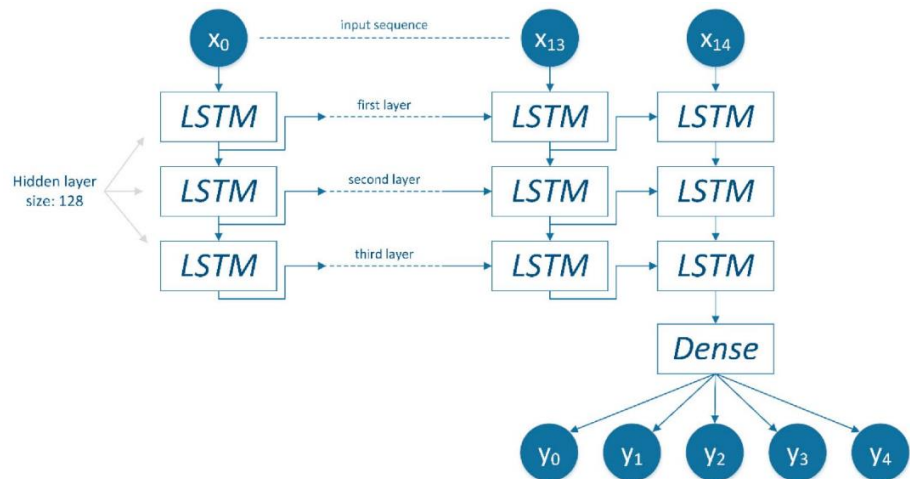


Fig.4.3.3 LSTM with Multiple Output Layers

There are two ways to create multi-label classification models: Using single dense output layer and using multiple dense output layers.

In the first approach, we can use a single dense layer with six outputs with a sigmoid activation functions and binary cross entropy loss functions. Each neuron in the output dense layer will represent one of the six output labels. The sigmoid activation function will return a value between 0 and 1 for each neuron. If any neuron's output value is greater than 0.5, it is assumed that the comment belongs to the class represented by that particular neuron.

In the second approach we will create one dense output layer for each label. We will have a total of 6 dense layers in the output. Each layer will have its own sigmoid function.

- BERT

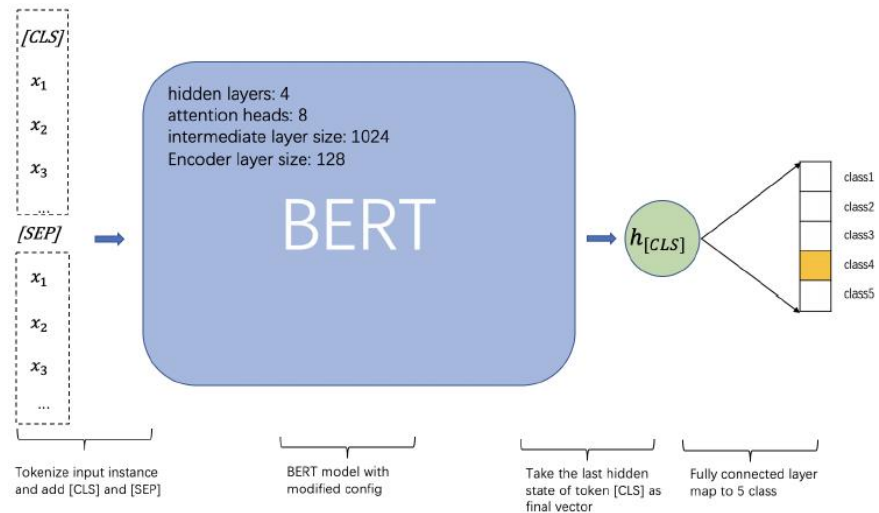


Fig.4.3.4 BERT Architecture

In Oct 2018, Google released a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. BERT builds upon recent work in pre-training contextual representations — including Semi-supervised Sequence Learning, Generative Pre-Training, ELMo, and ULMFit. However, unlike these previous models, BERT is the first deeply bidirectional, unsupervised language representation, pre-trained using only a plain text corpus.

Intuitively, a deep bidirectional model is strictly more powerful than either a left-to-right model or the concatenation of a left-to-right and right-to left model. Unfortunately, standard conditional language models can only be trained left-to-right or right-to-left, since bidirectional conditioning would allow each word to indirectly “see itself” in a multi-layered context.

To solve this problem, BERT uses “MASKING” technique to mask out some of the words in the input and then condition each word bidirectionally to predict the masked words.

BERT also learns to model relationships between sentences by pre-training on a very simple task that can be generated from any text corpus: Given two sentences A and B, is B the actual next sentence that comes after A in the corpus, or just a random sentence?

- Flask App
- To create an end-to-end product a Flask app that outputs probability that a comment falls under various categories of toxicity will be the final module for the course of this project.

4.4. DATA FLOW DIAGRAM

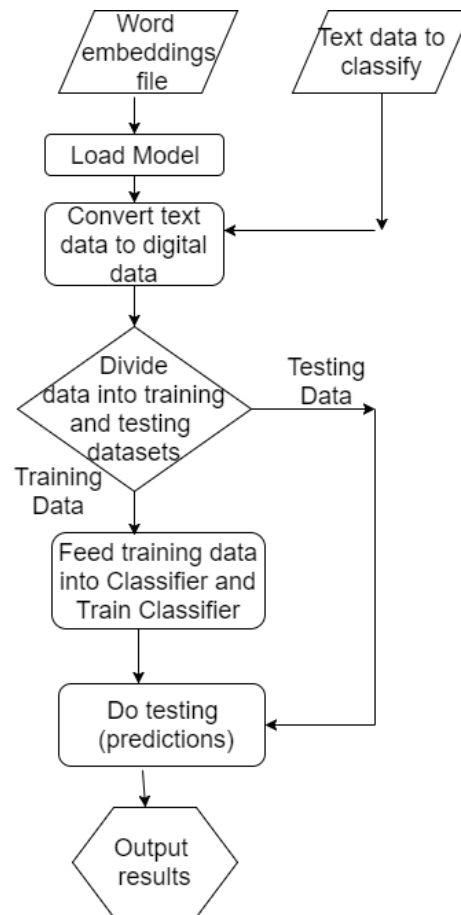


Fig.4.4.1 Data Flow Diagram

Chapter 5

IMPLEMENTATION

[Kaggle Notebook Link](#)

Import the required libraries

```
[2]: import numpy as np
import pandas as pd

from datetime import datetime
from collections import Counter
import re, spacy, string
import en_core_web_sm
nlp = en_core_web_sm.load()

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from pprint import pprint
import time

# hide warnings
import warnings
warnings.filterwarnings('ignore')
# set options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Import and load the dataset

```
In [88]: path = '../input/jigsaw-toxic-comment-classification-challenge/'
df = pd.read_csv(path+'train.csv.zip')
df_test = pd.read_csv(path+'test.csv.zip')
df_submission = pd.read_csv(path+'sample_submission.csv.zip')

df.head()
```

Out[88]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Check for missing values

The concept of missing values is important to understand **in order to successfully manage data**. If the missing values are not handled properly by the researcher, then he/she may end up drawing an inaccurate inference about the data.

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- `isnull()`
- `notnull()`
- `dropna()`
- `fillna()`
- `replace()`
- `interpolate()`

In order to check null values in Pandas DataFrame, we use `isnull()` function this function return dataframe of Boolean values which are True for NaN values.

```
[4]: df.shape
```

```
[4]: (159571, 8)
```

+ Code

+ Markdown

Checking missing values

+ Code

+ Markdown

```
[5]: df.isnull().sum()
```

```
[5]: id          0
comment_text    0
toxic           0
severe_toxic    0
obscene         0
threat          0
insult          0
identity_hate    0
dtype: int64
```

Data Analysis

The `drop()` method **removes the specified row or column**. By specifying the column axis (`axis='columns'`), the `drop()` method removes the specified column. By specifying the row axis (`axis='index'`), the `drop()` method removes the specified row.

```
[6]: comments = df.drop(['id', 'comment_text'], axis = 1)
      comments.columns
```

```
[6]: Index(['toxic', 'severe_toxic', 'obscene', 'threat', 'insult',
        'identity_hate'],
        dtype='object')
```

```
[7]: #Distribution of the target variable data in terms of proportions.

for i in list(comments.columns):
    print("Percent of {0}s: ".format(i), round(100*comments[i].mean(),2), "%")
```

```
Percent of toxics: 9.58 %
Percent of severe_toxics: 1.0 %
Percent of obscenes: 5.29 %
Percent of threats: 0.3 %
Percent of insults: 4.94 %
Percent of identity_hates: 0.88 %
```

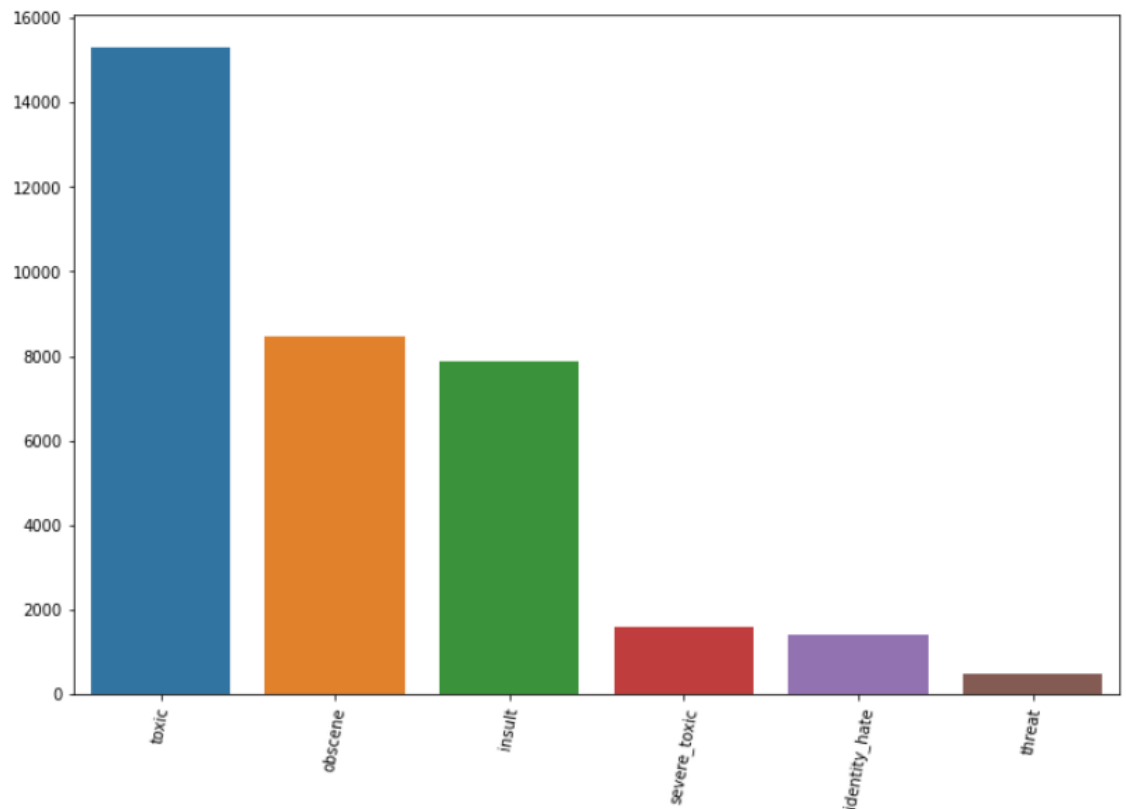
Data Visualization

Data visualization is **the graphical representation of information and data**. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

With interactive visualization, you can take the concept a step further by using technology to drill down into charts and graphs for more detail, interactively changing what data you see and how it's processed.

[9]:

```
plt.figure(figsize=(12,8))
sns.barplot(com_list,comments.sum().sort_values(ascending=False))
plt.xticks(rotation=80)
plt.show()
```



Data Preprocessing

Data preprocessing is **the process of transforming raw data into an understandable format**. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

If the text is in the same case, it is easy for a machine to interpret the words because the lower case and upper case are treated differently by the machine. for example, words like Ball and ball are treated differently by machine. So, we need to make the text in the same case and the most preferred case is a lower case to avoid such problems.

One of the other text processing techniques is removing punctuations. there are total 32 main punctuations that need to be taken care of. we can directly use the string module with a regular expression to replace any punctuation in text with an empty string.

Sometimes it happens that words and digits combine are written in the text which creates a problem for machines to understand. hence, We need to remove the words and digits which are combined like **game57** or **game5ts7**. This type of word is difficult to process so better to remove them or replace them with an empty string. we use regular expressions for this.

```
[10]: # Function to clean the review text and remove all the unnecessary elements.

def clean_review_text(text):
    text = text.lower() # covert the text to lowercase
    text = re.sub('<.*?>', '', text).strip() # remove html chars
    text = re.sub('\[|\(|\)|\]', '', text).strip() # remove text in square brackets and parenthesis
    text = text.translate(str.maketrans('', '', string.punctuation)) # remove punctuation marks
    text = re.sub("(\\W)", " ", text).strip() # remove non-ascii chars
    text = re.sub('\S*\d\S*\s*', '', text).strip() # remove words containing numbers
    return text.strip()

[11]: df.comment_text = df.comment_text.astype(str)
df.comment_text = df.comment_text.apply(clean_review_text)
df.comment_text.head()

[11]: 0    explanation why the edits made under my userna...
1    daww he matches this background colour im seem...
2    hey man im really not trying to edit war its j...
3    more i cant make any real suggestions on impro...
4    you sir are my hero any chance you remember wh...
Name: comment_text, dtype: object
```

Stopwords are the most commonly occurring words in a text which do not provide any valuable information. stopwords like they, there, this, where, etc are some of the stopwords. NLTK library is a common library that is used to remove stopwords and include approximately 180 stopwords which it removes. If we want to add any new word to a set of words then it is easy using the add method.

Lemmatization

Stemming is a process to reduce the word to its root stem for example run, running, runs, runed derived from the same word as run. basically stemming do is remove the prefix or suffix from word like ing, s, es, etc. NLTK library is used to stem the words. The stemming technique is not used for production purposes because it is not so efficient technique and most of the time it stems the unwanted words. So, to solve the problem another technique came into the market as Lemmatization. there are various types of stemming algorithms like porter stemmer, snowball stemmer. Porter stemmer is widely used present in the NLTK library.

Lemmatization is similar to stemming, used to stem the words into root word but differs in working. Actually, Lemmatization is a systematic way to reduce the words into their lemma by matching them with a language dictionary.

```
[12]: # Snowball stemmer
import nltk
from nltk.stem.snowball import SnowballStemmer

snow_stemmer = SnowballStemmer(language='english')

stopwords = nlp.Defaults.stop_words
def apply_stemmer(text):
    words = text.split()
    sent = [snow_stemmer.stem(word) for word in words if not word in set(stopwords)]
    return ' '.join(sent)

+ Code + Markdown

[13]: df.comment_text = df.comment_text.apply(apply_stemmer)
df.comment_text.head()

[13]: 0    explain edit usernam hardcor metallica fan reve...
1    daww match background colour im seem stuck tha...
2    hey man im tri edit war guy constant remov rel...
3    cant real suggest improv wonder section statis...
4             sir hero chanc rememb page that
Name: comment_text, dtype: object
```

Word Cloud

A word cloud (also known as a tag cloud) is a **visual representation of words**. Cloud creators are used to highlight popular words and phrases based on frequency and relevance.

```
[14]: #Using a word cloud find the top 50 words by frequency among all the review texts
!pip install wordcloud
from wordcloud import WordCloud

wordcloud = WordCloud(stopwords=stopwords,max_words=50).generate(str(df.comment_text))

print(wordcloud)
plt.figure(figsize=(10,6))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()

Requirement already satisfied: wordcloud in /opt/conda/lib/python3.7/site-packages (1.8.1)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.7/site-packages (from wordcloud) (3.5.1)
Requirement already satisfied: numpy>=1.6.1 in /opt/conda/lib/python3.7/site-packages (from wordcloud) (1.19.5)
Requirement already satisfied: pillow in /opt/conda/lib/python3.7/site-packages (from wordcloud) (8.2.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib->wordcloud) (4.28.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib->wordcloud) (1.3.2)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib->wordcloud) (3.0.6)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.7/site-packages (from matplotlib->wordcloud) (2.8.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib->wordcloud) (21.3)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from python-dateutil->matplotlib->wordcloud) (1.16.0)
```


Term frequency-inverse document frequency is a text vectorizer that transforms the text into a usable vector. It combines 2 concepts, Term Frequency (TF) and Document Frequency (DF).

The term frequency is the number of occurrences of a specific term in a document. Term frequency indicates how important a specific term in a document. Term frequency represents every text from the data as a matrix whose rows are the number of documents and columns are the number of distinct terms throughout all documents.

Document frequency is the number of documents containing a specific term. Document frequency indicates how common the term is.

Inverse document frequency (IDF) is the weight of a term, it aims to reduce the weight of a term if the term's occurrences are scattered throughout all the documents. IDF can be calculated as follow:

$$idf_i = \log\left(\frac{n}{df_i}\right)$$

Fig.5.1 IDF Score Formula

Where idf_i is the IDF score for term i , df_i is the number of documents containing term i , and n is the total number of documents. The higher the DF of a term, the lower the IDF for the term. When the number of DF is equal to n which means that the term appears in all documents, the IDF will be zero, since $\log(1)$ is zero, when in doubt just put this term in the stopwords list because it doesn't provide much information.

The TF-IDF score as the name suggests is just a multiplication of the term frequency matrix with its IDF, it can be calculated as follow:

$$w_{i,j} = tf_{i,j} \times idf_i$$

Fig.5.2 TF-IDF Score Formula

Where w_{ij} is TF-IDF score for term i in document j , tf_{ij} is term frequency for term i in document j , and idf_i is IDF score for term i .

Transformed dataset

[18]:

```
## transforming the train and test datasets
X_train_transformed = word_vectorizer.transform(X_train)
X_test_transformed = word_vectorizer.transform(X_test)

# # Print the shape of each dataset.
print('X_train_transformed', X_train_transformed.shape)
print('y_train', y_train.shape)
print('X_test_transformed', X_test_transformed.shape)
print('y_test', y_test.shape)
```

```
X_train_transformed (111699, 4257625)
y_train (111699, 6)
X_test_transformed (47872, 4257625)
y_test (47872, 6)
```

Logistic Regression

Logistic regression is a **process of modeling the probability of a discrete outcome given an input variable**. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on.

In natural language processing, logistic regression is the **base- line supervised machine learning algorithm for classification**, and also has a very close relationship with neural networks.

```
[20]: from sklearn.multiclass import OneVsRestClassifier
      from sklearn.problem_transform import BinaryRelevance
```

```
[21]: # Logistic Regression
      time1 = time.time()
      # logistic regression
      log_reg = LogisticRegression(C = 10, penalty='l2', solver = 'liblinear', random_state=seed)

      # fit model
      classifier_ovr_log = OneVsRestClassifier(log_reg)
      classifier_ovr_log.fit(X_train_transformed, y_train)

      time_taken = time.time() - time1
      print('Time Taken: {:.2f} seconds'.format(time_taken))

      y_train_pred_proba = classifier_ovr_log.predict_proba(X_train_transformed)
      y_test_pred_proba = classifier_ovr_log.predict_proba(X_test_transformed)

      roc_auc_score_train = roc_auc_score(y_train, y_train_pred_proba, average='weighted')
      roc_auc_score_test = roc_auc_score(y_test, y_test_pred_proba, average='weighted')

      print("ROC AUC Score Train:", roc_auc_score_train)
      print("ROC AUC Score Test:", roc_auc_score_test)

      Time Taken: 112.67 seconds
      ROC AUC Score Train: 0.9998294941155842
      ROC AUC Score Test: 0.975950081677377
```

LSTM:

LSTM (Long Short-Term Memory) network is a type of RNN (Recurrent Neural Network) that is widely used for learning sequential data prediction problems. As every other neural network LSTM also has some layers which help it to learn and recognize the pattern for better performance. The basic operation of LSTM can be considered to hold the required information and discard the information which is not required or useful for further prediction.

When performing normal text modelling, most of the preprocessing task and modelling task focuses on creating data sequentially. Examples of such tasks can be POS tagging, stopwords elimination, sequencing of the text. These are the methods that try to make data understood by a model with less effort according to the known pattern. It can give the results.

Here applying LSTM networks can have its own special feature. Earlier in the article, we have discussed that LSTM has a feature through which it can memorize the sequence of the data. It has one more feature that it works on the elimination of unused information and as we know the text data always consists a

lot of unused information which can be eliminated by the LSTM so that the calculation timing and cost can be reduced,

So basically, the feature of elimination of unused information and memorizing the sequence of the information makes the LSTM a powerful tool for performing text classification or other text-based tasks.

```
[1]: import os
import zipfile
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from keras.layers import Dense, Input, Dropout, Activation
from keras.layers import Bidirectional, LSTM, Embedding, GlobalMaxPool1D
from keras import initializers, regularizers, constraints, optimizers, layers
from keras import backend as K
from keras import callbacks
from sklearn.model_selection import train_test_split
```

Tokenization

```
[11]: data_train = train['comment_text']
data_labels = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
y_train = train[data_labels].values
data_test = test['comment_text']
```

```
[12]: embed_size = 50 # how big is each word vector
max_features = 20000 # how many unique words to use (i.e num rows in embedding vector)
maxlen = 200 # max number of words in a comment to use

tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(data_train))

list_tokenized_train = tokenizer.texts_to_sequences(data_train)
list_tokenized_test = tokenizer.texts_to_sequences(data_test)

X_train = pad_sequences(list_tokenized_train, maxlen=maxlen)
X_test = pad_sequences(list_tokenized_test, maxlen=maxlen)
```

```
[13]: glove_embedding = '../input/glove6b50dtxt/glove.6B.50d.txt'

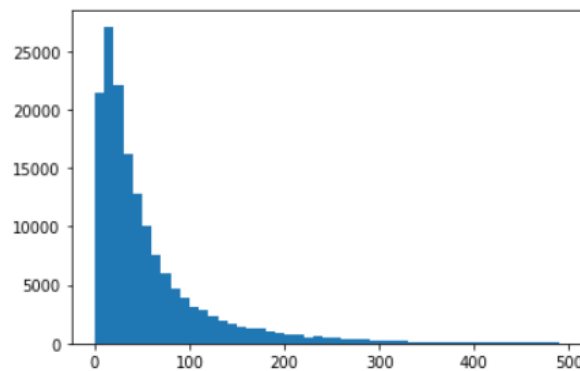
def get_coefs(word, *arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.strip().split()) for o in open(glove_embedding))
```

```
[14]: all_embs = np.stack(embeddings_index.values())
emb_mean, emb_std = all_embs.mean(), all_embs.std()
emb_mean, emb_std
```

```
[15]: word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector
```

```
[16]: total_num_words = [len(one_comment) for one_comment in list_tokenized_train]
```

```
[17]: plt.hist(total_num_words, bins = np.arange(0, 500, 10))
plt.show()
```



```
[18]: model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=max_features, output_dim=embed_size,
                              input_length=maxlen, weights=[embedding_matrix]),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(60, return_sequences=True,
                                                         dropout=0.1, recurrent_dropout=0.1)),
    tf.keras.layers.GlobalMaxPool1D(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(6, activation='sigmoid')]) # only 2 value in each labels

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# using binary crossentropy because each labels or the features only have 2 value, 0 or 1

model.summary()
```

Model: "sequential"

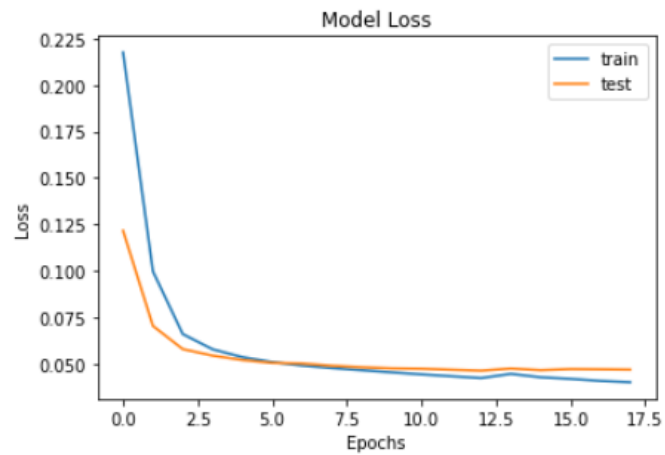
Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 200, 50)	1000000
bidirectional (Bidirectional)	(None, 200, 120)	53280
global_max_pooling1d (Global)	(None, 120)	0
dropout (Dropout)	(None, 120)	0
dense (Dense)	(None, 50)	6050
dropout_1 (Dropout)	(None, 50)	0
dense_1 (Dense)	(None, 6)	306
Total params: 1,059,636		
Trainable params: 1,059,636		
Non-trainable params: 0		

```
batch_size = 2048
epochs = 100

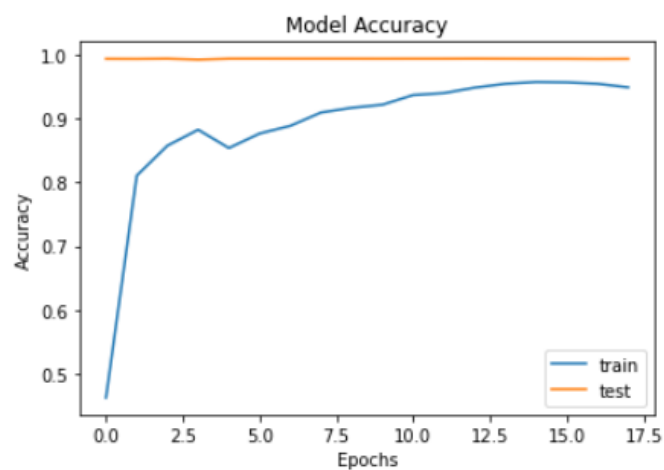
history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                    validation_split=0.2, # validation set is 20% of dataset
                    callbacks=[early_stopping_cb], verbose=1)
```

```
Epoch 1/100
63/63 [=====] - 109s 2s/step - loss: 0.3483 - accuracy: 0.2428 - val_loss: 0.1215 - val_accuracy: 0.9938
Epoch 2/100
63/63 [=====] - 103s 2s/step - loss: 0.1122 - accuracy: 0.7868 - val_loss: 0.0703 - val_accuracy: 0.9937
Epoch 3/100
63/63 [=====] - 103s 2s/step - loss: 0.0692 - accuracy: 0.8528 - val_loss: 0.0578 - val_accuracy: 0.9940
Epoch 4/100
63/63 [=====] - 103s 2s/step - loss: 0.0577 - accuracy: 0.8816 - val_loss: 0.0543 - val_accuracy: 0.9926
Epoch 5/100
63/63 [=====] - 103s 2s/step - loss: 0.0538 - accuracy: 0.8412 - val_loss: 0.0519 - val_accuracy: 0.9940
Epoch 6/100
63/63 [=====] - 103s 2s/step - loss: 0.0515 - accuracy: 0.8774 - val_loss: 0.0504 - val_accuracy: 0.9940
Epoch 7/100
63/63 [=====] - 103s 2s/step - loss: 0.0492 - accuracy: 0.8905 - val_loss: 0.0500 - val_accuracy: 0.9940
Epoch 8/100
63/63 [=====] - 102s 2s/step - loss: 0.0474 - accuracy: 0.9033 - val_loss: 0.0488 - val_accuracy: 0.9940
Epoch 9/100
63/63 [=====] - 102s 2s/step - loss: 0.0468 - accuracy: 0.9172 - val_loss: 0.0480 - val_accuracy: 0.9940
Epoch 10/100
63/63 [=====] - 104s 2s/step - loss: 0.0452 - accuracy: 0.9207 - val_loss: 0.0474 - val_accuracy: 0.9940
Epoch 11/100
63/63 [=====] - 104s 2s/step - loss: 0.0446 - accuracy: 0.9364 - val_loss: 0.0472 - val_accuracy: 0.9940
Epoch 12/100
63/63 [=====] - 103s 2s/step - loss: 0.0431 - accuracy: 0.9357 - val_loss: 0.0467 - val_accuracy: 0.9940
Epoch 13/100
63/63 [=====] - 104s 2s/step - loss: 0.0420 - accuracy: 0.9507 - val_loss: 0.0462 - val_accuracy: 0.9941
Epoch 14/100
63/63 [=====] - 104s 2s/step - loss: 0.0425 - accuracy: 0.9529 - val_loss: 0.0473 - val_accuracy: 0.9940
Epoch 15/100
63/63 [=====] - 102s 2s/step - loss: 0.0429 - accuracy: 0.9605 - val_loss: 0.0465 - val_accuracy: 0.9938
Epoch 16/100
63/63 [=====] - 102s 2s/step - loss: 0.0422 - accuracy: 0.9568 - val_loss: 0.0470 - val_accuracy: 0.9938
Epoch 17/100
63/63 [=====] - 103s 2s/step - loss: 0.0405 - accuracy: 0.9567 - val_loss: 0.0469 - val_accuracy: 0.9935
Epoch 18/100
63/63 [=====] - 104s 2s/step - loss: 0.0396 - accuracy: 0.9524 - val_loss: 0.0467 - val_accuracy: 0.9937
```

```
[21]: plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```



```
[22]: plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```



BERT:

In [7]:

```
import seaborn as sns

import tensorflow as tf
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from kaggle_datasets import KaggleDatasets
import transformers
from tqdm.notebook import tqdm
import tokenizers
from tokenizers import BertWordPieceTokenizer
```

```
2022-01-28 05:41:10.547882: W tensorflow/stream_executor/platform/default/dso_loader.cc:60]
Could not load dynamic library 'libcudart.so.11.0'; dLError: libcudart.so.11.0: cannot open
shared object file: No such file or directory; LD_LIBRARY_PATH: /opt/conda/lib
2022-01-28 05:41:10.547988: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above
cudart dLError if you do not have a GPU set up on your machine.
```

```
def fast_encode(texts, tokenizer, chunk_size=256, maxlen=512):
    print('encoding with', tokenizer)

    # for transformers 3.5
    if isinstance(tokenizer, transformers.DistilBertTokenizer) or \
        isinstance(tokenizer, transformers.DistilBertTokenizerFast):
        # tokenizer.enable_truncation(max_length=maxlen)
        # tokenizer.enable_padding(max_length=maxlen)
        all_ids = []

        for i in tqdm(range(0, len(texts), chunk_size)):
            text_chunk = texts[i:i+chunk_size].tolist()
            # encs = tokenizer.encode_batch(text_chunk)
            encs = tokenizer(text_chunk, padding='max_length', truncation=True, max_length=maxlen)

            # all_ids.extend([enc.ids for enc in encs])
            all_ids.extend(encs['input_ids'])
        elif isinstance(tokenizer, tokenizers.implementations.bert_wordpiece.BertWordPieceTokenizer):
            tokenizer.enable_truncation(max_length=maxlen)
            tokenizer.enable_padding(max_length=maxlen)
            all_ids = []
```



```

        for i in tqdm(range(0, len(texts), chunk_size)):
            text_chunk = texts[i:i+chunk_size].tolist()
            encs = tokenizer.encode_batch(text_chunk)
            all_ids.extend([enc.ids for enc in encs])

    return np.array(all_ids)

```

```

In [9]: # First load the real tokenizer
tokenizer = transformers.DistilBertTokenizerFast.from_pretrained('distilbert-base-multilingual-cased')

save_path = '/kaggle/working/distilbert_base_cased/'
if not os.path.exists(save_path):
    os.makedirs(save_path)
tokenizer.save_pretrained(save_path)
fast_tokenizer = tokenizer

# "faster as the tokenizers from transformers because they are implemented in Rust."
# fast_tokenizer = BertWordPieceTokenizer('distilbert_base_cased/vocab.txt', lowercase=False)

```

TPU Config

```

In [10]: # Detect hardware, return appropriate distribution strategy

try:
    # TPU detection. No parameters necessary if TPU_NAME environment variable is
    # set: this is always the case on Kaggle.
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    # Default distribution strategy in Tensorflow. Works on CPU and single GPU.
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)

```

Running on TPU grpc://10.0.0.2:8470

```
In [11]:
# Configuration
AUTO = tf.data.experimental.AUTOTUNE
SHUFFLE = 2048
EPOCHS1 = 20
EPOCHS2 = 4
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
MAX_LEN = 192
VERBOSE = 2
```

```
In [12]:
train1 = pd.read_csv("/kaggle/input/jigsaw-multilingual-toxic-comment-classification/jigsaw-toxic-comment-train.csv")
train2 = pd.read_csv("/kaggle/input/jigsaw-multilingual-toxic-comment-classification/jigsaw-unintended-bias-train.csv")
valid = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-comment-classification/validation.csv')
test = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-comment-classification/test.csv')
```

```
In [13]:
train2.toxic = train2.toxic.round().astype(int)
train = pd.concat([train1[['comment_text', 'toxic']],
                  train2[['comment_text', 'toxic']].query('toxic==1'),
                  train2[['comment_text', 'toxic']].query('toxic==0').sample(n=100000)
                  ])
#rate=10
```

Out[15]:

	comment_text	toxic	num_words_comment_text
0	Explanation\nWhy the edits made under my usern...	0	43
1	D'aww! He matches this background colour I'm s...	0	17
2	Hey man, I'm really not trying to edit war. It...	0	42
3	"\nMore\nI can't make any real suggestions on ...	0	113
4	You, sir, are my hero. Any chance you remember...	0	13

```
In [16]:
del train['toxic']; gc.collect()
```

Out[16]:
23

```
In [17]:
train['comment_text'] = train['comment_text'].apply(lambda x: clean_text(x))
train['comment_text'] = train['comment_text'].apply(lambda x: text_process(x))
x_train = fast_encode(train['comment_text'].astype(str), fast_tokenizer, maxlen=MAX_LEN)
```

encoding with PreTrainedTokenizerFast(name_or_path='distilbert-base-multilingual-cased', vocab_size=119547, model_max_len=512, is_fast=True, padding_side='right', truncation_side='right', special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]', 'mask_token': '[MASK]'})

Run Model

In [22]:

```
# not train on order to save memory
n_steps = len(y_train) // (BATCH_SIZE*8)

train_history = model.fit(
    train_dataset,
    steps_per_epoch=n_steps,
    validation_data=valid_dataset,
    epochs=EPOCHS1,
    callbacks=callbacks_list1,
    verbose=VERBOSE
)

del train_dataset; gc.collect()
```

Epoch 1/20

425/425 - 80s - loss: 0.1794 - auc: 0.9181 - accuracy: 0.9335 - val_loss: 0.4421 - val_auc: 0.7393 - val_accuracy: 0.8515

Epoch 2/20

425/425 - 29s - loss: 0.1155 - auc: 0.9669 - accuracy: 0.9560 - val_loss: 0.4142 - val_auc: 0.7794 - val_accuracy: 0.8531

Epoch 3/20

425/425 - 29s - loss: 0.1071 - auc: 0.9734 - accuracy: 0.9581 - val_loss: 0.4650 - val_auc: 0.7968 - val_accuracy: 0.8474

425/425 - 29s - loss: 0.1071 - auc: 0.9734 - accuracy: 0.9581 - val_loss: 0.4650 - val_auc: 0.7968 - val_accuracy: 0.8474

Epoch 4/20

425/425 - 29s - loss: 0.1296 - auc: 0.9652 - accuracy: 0.9438 - val_loss: 0.5881 - val_auc: 0.7819 - val_accuracy: 0.8469

Epoch 5/20

425/425 - 29s - loss: 0.0365 - auc: 0.9972 - accuracy: 0.9858 - val_loss: 3.0710 - val_auc: 0.7422 - val_accuracy: 0.2985

Epoch 00005: ReduceLROnPlateau reducing learning rate to 6.9999998231651255e-06.

Epoch 6/20

425/425 - 29s - loss: 3.0698e-05 - auc: 0.0000e+00 - accuracy: 1.0000 - val_loss: 3.6471 - val_auc: 0.7229 - val_accuracy: 0.2648

Epoch 7/20

425/425 - 29s - loss: 0.0486 - auc: 0.9967 - accuracy: 0.9842 - val_loss: 1.2826 - val_auc: 0.5525 - val_accuracy: 0.8464

Epoch 00007: ReduceLROnPlateau reducing learning rate to 4.899999748886329e-06.

Epoch 8/20

425/425 - 29s - loss: 0.0041 - auc: 0.7671 - accuracy: 0.9993 - val_loss: 0.7587 - val_auc: 0.6975 - val_accuracy: 0.8465

Restoring model weights from the end of the best epoch.

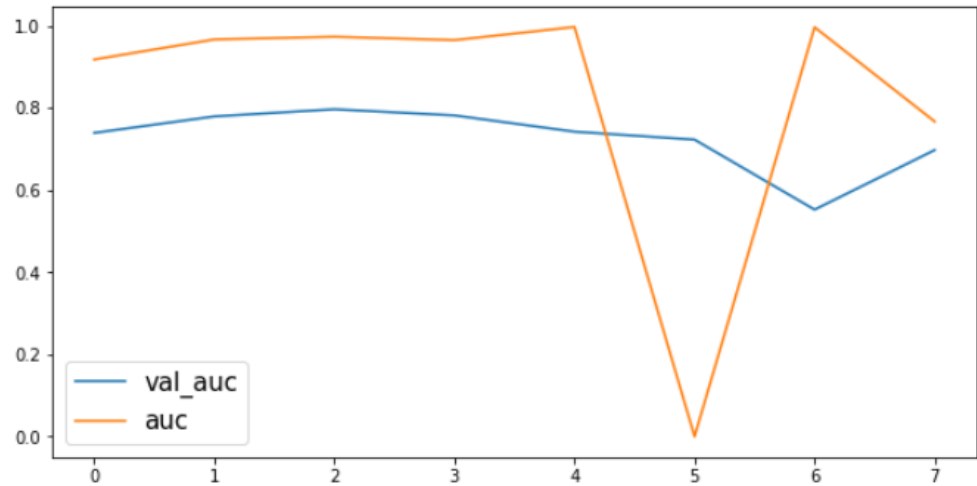
Epoch 00008: early stopping

In [24]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
plt.plot(train_history_df['val_auc'], label='val_auc')
plt.plot(train_history_df['auc'], label='auc')
plt.legend(fontsize=15)
```

Out[24]:

<matplotlib.legend.Legend at 0x7f76d0f3c990>

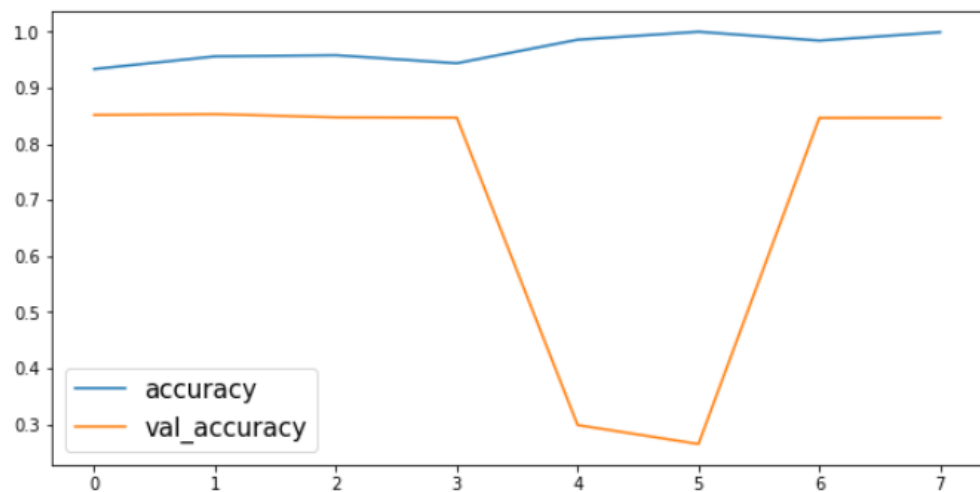


In [25]:

```
plt.figure(figsize=(10, 5))
plt.plot(train_history_df['accuracy'], label='accuracy')
plt.plot(train_history_df['val_accuracy'], label='val_accuracy')
plt.legend(fontsize=15)
```

Out[25]:

<matplotlib.legend.Legend at 0x7f76b27c4a10>



Final adjustments for flask app:

Not all categories have 5000 rows. So we should count them first and make them balanced.

```
data_sev[data_sev['severe_toxic'] == 1].count()
```

Python

```
id          1595
comment_text 1595
severe_toxic 1595
dtype: int64
```

```
data_sev_1 = data_sev[data_sev['severe_toxic'] == 1].iloc[0:1595,:]
data_sev_0 = data_sev[data_sev['severe_toxic'] == 0].iloc[0:1595,:]
data_sev_done = pd.concat([data_sev_1, data_sev_0], axis=0)
data_sev_done.shape
```

Python

```
(3190, 3)
```

We only had 1,595 comments that are severely toxic. We combine it together with another 1,595 comments that are not toxic to form a new dataset that is balanced. We repeat this for all other categories.

```
data_obs[data_obs['obscene'] == 1].count()
```

The number of threat comments of 478 is too miniscule when added with another 478 clean comments for a proper analysis. We decided that the clean comments will comprise 80% at the most of the dataset, as shown below.

```
data_thr_1 = data_thr[data_thr['threat'] == 1].iloc[0:478,:]

# We include 1912 comments that have no threat so that the data with threat (478) will represent 28% of the dataset.
data_thr_0 = data_thr[data_thr['threat'] == 0].iloc[0:1912,:]
data_thr_done = pd.concat([data_thr_1, data_thr_0], axis=0)
data_thr_done.shape
```

Python

```
(2390, 3)
```

```
data_ins[data_ins['insult'] == 1].count()
```

Python

```
id          7877
comment_text 7877
insult       7877
dtype: int64
```

```
data_ins_1 = data_ins[data_ins['insult'] == 1].iloc[0:5000,:]
data_ins_0 = data_ins[data_ins['insult'] == 0].iloc[0:5000,:]
data_ins_done = pd.concat([data_ins_1, data_ins_0], axis=0)
data_ins_done.shape
```

Python

```
(10000, 3)
```

Reminder: Number of comments that fall into the following categories:

- Toxic (14,000+)
- Severe Toxic (1595)
- Obscene (8449)
- Threat (478)
- Insult (7877)
- Identity Hate (1405)

df_****_done refers to the dataframes of each class that has been balanced (at least 20/80 proportion)

Toxic data_tox_done 5000 5000 10000 Severe Toxic data_sev_done 1595 1595 3190 Obscene (8449) data_obs_done 5000 5000 10000 Threat (478) data_thr_done 478 1912 2390 Insult (7877) data_ins_done 5000 5000 10000 Identity Hate (1405) data_ide_done 1405 5620 7025

Import relevant packages for modelling

```
# Import packages for pre-processing
from sklearn import preprocessing
from sklearn.feature_selection import SelectFromModel

# Import tools to split data and evaluate model performance
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import f1_score, precision_score, recall_score, precision_recall_curve, fbeta_score, confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve

# Import ML algos
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
```

Create simple function that takes in a dataset and allows user to choose dataset, toxicity label, vectorizer and number of ngrams

```
'''
df_done: data_tox_done, data_sev_done, ...
label: toxic, severe_toxic, ...
vectorizer values: CountVectorizer, TfidfVectorizer
gram_range values: (1,1) for unigram, (2,2) for bigram
'''
def cv_tf_train_test(df_done,label,vectorizer,ngram):

    ''' Train/Test split'''
    # Split the data into X and y data sets
    X = df_done.comment_text
    y = df_done[label]

    # Split our data into training and test data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    ''' Count Vectorizer/TF-IDF '''

    # Create a Vectorizer object and remove stopwords from the table
    cv1 = vectorizer(ngram_range=(ngram), stop_words='english')

    X_train_cv1 = cv1.fit_transform(X_train) # Learn the vocabulary dictionary and return term-document matrix
    X_test_cv1 = cv1.transform(X_test)      # Learn a vocabulary dictionary of all tokens in the raw documents.

    # Output a DataFrame of the CountVectorizer with unique words as the labels
    # test = pd.DataFrame(X_train_cv1.toarray(), columns=cv1.get_feature_names())

    ''' Initialize all model objects and fit the models on the training data '''
    lr = LogisticRegression()
    lr.fit(X_train_cv1, y_train)
```

```
print('lr done')

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_cv1, y_train)

bnb = BernoulliNB()
bnb.fit(X_train_cv1, y_train)
print('bnb done')

mnb = MultinomialNB()
mnb.fit(X_train_cv1, y_train)
print('mnb done')

svm_model = LinearSVC()
svm_model.fit(X_train_cv1, y_train)

randomforest = RandomForestClassifier(n_estimators=100, random_state=42)
randomforest.fit(X_train_cv1, y_train)
print('rdf done')

# Create a list of F1 score of all models
f1_score_data = {'F1 Score':[f1_score(lr.predict(X_test_cv1), y_test), f1_score(knn.predict(X_test_cv1), y_test),
                             f1_score(bnb.predict(X_test_cv1), y_test), f1_score(mnb.predict(X_test_cv1), y_test),
                             f1_score(svm_model.predict(X_test_cv1), y_test), f1_score(randomforest.predict(X_test_cv1), y_test)]}

# Create DataFrame with the model names as column labels
df_f1 = pd.DataFrame(f1_score_data, index=['Log Regression','KNN', 'BernoulliNB', 'MultinomialNB', 'SVM', 'Random Forest'])

return df_f1
```

Let's create a TF-IDF vectorizer object for each category and calculate the F1 scores across all models

```
'''
def cv_tf_train_test(df_done,label,vectorizer,ngram)
vectorizer values: CountVectorizer, TfidfVectorizer
ngram_range values: (1,1) for unigram, (2,2) for bigram
'''

import time

t0 = time.time()

df_tox_cv = cv_tf_train_test(data_tox_done, 'toxic', TfidfVectorizer, (1,1))
df_tox_cv.rename(columns={'F1 Score': 'F1 Score(toxic)'}, inplace=True)

t1 = time.time()

total = 'Time taken: {} seconds'.format(t1-t0)
print(total)

df_tox_cv
```

F1 Score(severe_toxic)	
Log Regression	0.927879
KNN	0.857416
BernoulliNB	0.803707
MultinomialNB	0.936170
SVM	0.926004
Random Forest	0.934874


```

t0 = time.time()

df_obs_cv = cv_tf_train_test(data_obs_done, 'obscene', TfidfVectorizer, (1,1))
df_obs_cv.rename(columns={'F1 Score': 'F1 Score(obscene)'}, inplace=True)

t1 = time.time()

total = 'Time taken: {} seconds'.format(t1-t0)
print(total)

df_obs_cv

```

Table 5.1 F1 Score – Severe Toxic

```

lr done
bnb done
mnb done
rdf done
Time taken: 18.873051404953003 seconds
lr done
bnb done
mnb done
rdf done
Time taken: 18.873051404953003 seconds

```

F1 Score(obscene)	
Log Regression	0.908655
KNN	0.519056
BernoulliNB	0.787830
MultinomialNB	0.901463
SVM	0.921378
Random Forest	0.909091


```

t0 = time.time()

df_thr_cv = cv_tf_train_test(data_thr_done, 'threat', TfidfVectorizer, (1,1))
df_thr_cv.rename(columns={'F1 Score': 'F1 Score(threat)'}, inplace=True)

t1 = time.time()

total = 'Time taken: {} seconds'.format(t1-t0)
print(total)

df_thr_cv

```

Table 5.2 F1 Score – Obscene

```

lr done
bnb done
mnb done
rdf done
Time taken: 2.1821258068084717 seconds
lr done
bnb done
mnb done
rdf done
Time taken: 2.1821258068084717 seconds

```

	F1 Score(threat)
Log Regression	0.628821
KNN	0.720000
BernoulliNB	0.311828
MultinomialNB	0.504762
SVM	0.786765
Random Forest	0.795539

```

t0 = time.time()

df_ins_cv = cv_tf_train_test(data_ins_done, 'insult', TfidfVectorizer, (1,1))
df_ins_cv.rename(columns={'F1 Score': 'F1 Score(insult)'}, inplace=True)

t1 = time.time()

total = 'Time taken: {} seconds'.format(t1-t0)
print(total)

df_ins_cv

```

Table 5.3 F1 Score – Threat


```

lr done
bnb done
mnb done
rdf done
Time taken: 19.122676849365234 seconds
lr done
bnb done
mnb done
rdf done
Time taken: 19.122676849365234 seconds

```

	F1 Score(insult)
Log Regression	0.896599
KNN	0.257992
BernoulliNB	0.783762
MultinomialNB	0.897411
SVM	0.902619
Random Forest	0.883993

```

t0 = time.time()

df_idc_cv = cv_tf_train_test(data_idc_done, 'identity_hate', TfidfVectorizer, (1,1))
df_idc_cv.rename(columns={'F1 Score': 'F1 Score(identity_hate)'}, inplace=True)

t1 = time.time()

total = 'Time taken: {} seconds'.format(t1-t0)
print(total)

df_idc_cv

```

Table 5.4 F1 Score – Insult

```

lr done
bnb done
mnb done
rdf done
Time taken: 10.205200672149658 seconds
lr done
bnb done
mnb done
rdf done
Time taken: 10.205200672149658 seconds

```

	F1 Score(identity_hate)
Log Regression	0.699029
KNN	0.230159
BernoulliNB	0.549206
MultinomialNB	0.485857
SVM	0.797516
Random Forest	0.768448

```

# Let's combine the dataframes into a master dataframe to compare F1 scores across all categories.
f1_all = pd.concat([df_tox_cv, df_sev_cv, df_obs_cv, df_ins_cv, df_thr_cv, df_ide_cv], axis=1)
f1_all

```

Table 5.5 F1 Score – Identity Hate

	F1 Score(toxic)	F1 Score(severe_toxic)	F1 Score(obscene)	F1 Score(insult)	F1 Score(threat)	F1 Score(identity_hate)
Log Regression	0.861234	0.927879	0.908655	0.896599	0.628821	0.699029
KNN	0.185120	0.857416	0.519056	0.257992	0.720000	0.230159
BernoulliNB	0.776521	0.803707	0.787830	0.783762	0.311828	0.549206
MultinomialNB	0.874958	0.936170	0.901463	0.897411	0.504762	0.485857
SVM	0.876133	0.926004	0.921378	0.902619	0.786765	0.797516
Random Forest	0.838055	0.934874	0.909091	0.883993	0.795539	0.768448

Transpose the combined F1 dataframe to make it suitable for presentation on a graph

```

f1_all_trp = f1_all.transpose()
f1_all_trp

```

	Log Regression	KNN	BernoulliNB	MultinomialNB	SVM	Random Forest
F1 Score(toxic)	0.861234	0.185120	0.776521	0.874958	0.876133	0.838055
F1 Score(severe_toxic)	0.927879	0.857416	0.803707	0.936170	0.926004	0.934874
F1 Score(obscene)	0.908655	0.519056	0.787830	0.901463	0.921378	0.909091
F1 Score(insult)	0.896599	0.257992	0.783762	0.897411	0.902619	0.883993
F1 Score(threat)	0.628821	0.720000	0.311828	0.504762	0.786765	0.795539
F1 Score(identity_hate)	0.699029	0.230159	0.549206	0.485857	0.797516	0.768448

Table 5.6 F1 Score – Master Dataframe

```

sns.lineplot(data=f1_all_trp, size=[10,10], markers=True)
plt.xticks(rotation='90', fontsize=14)
plt.yticks(fontsize=14)
plt.legend(loc='best')
plt.title('F1 Score of ML models (TF-IDF)', fontsize=20)

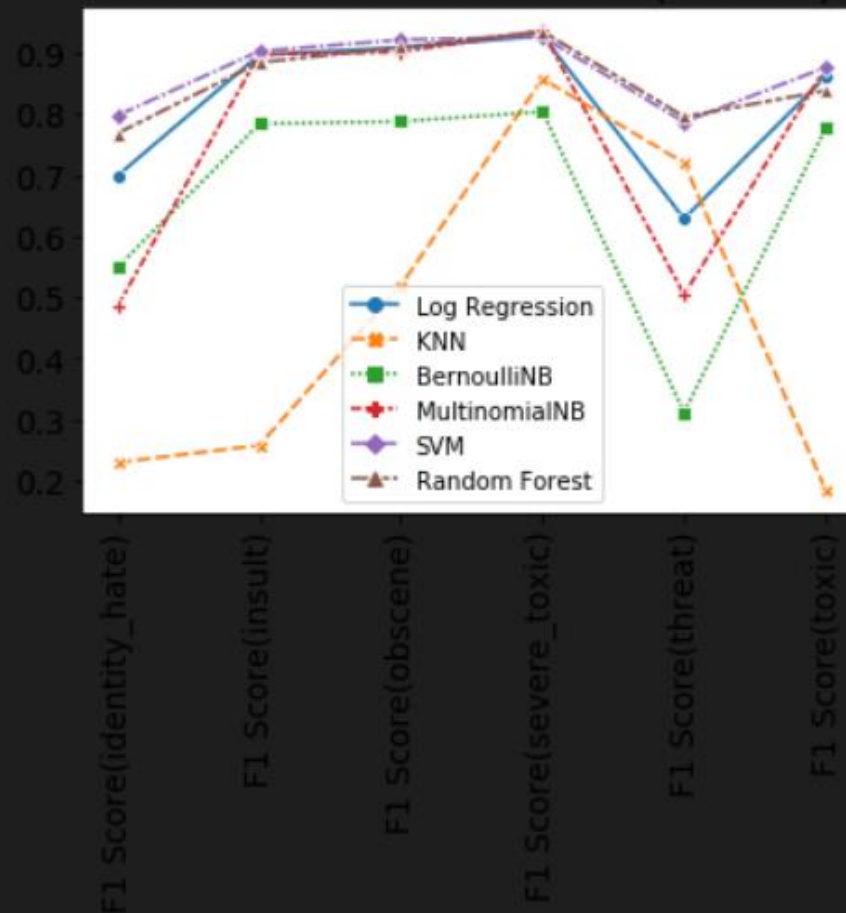
# Repeat this for CountVectorizer as well

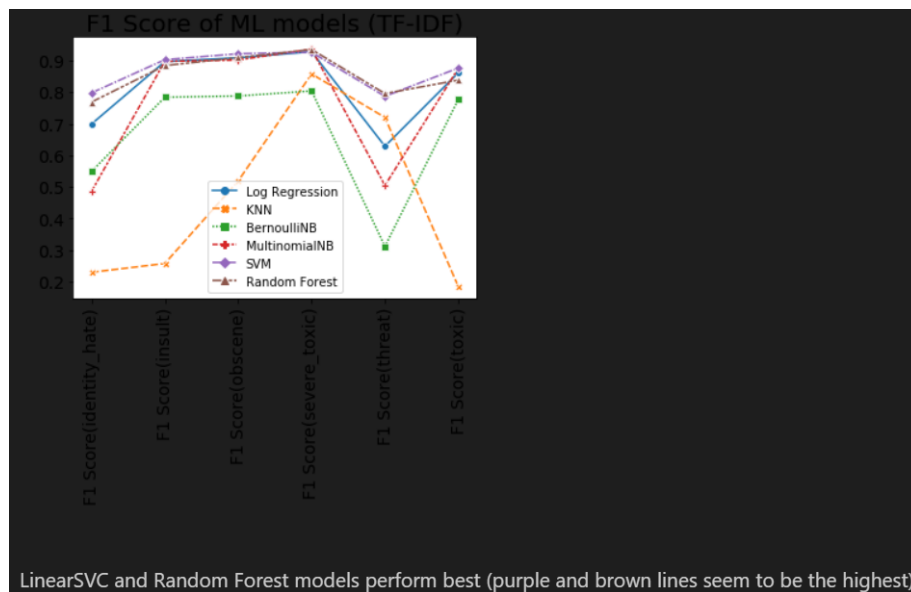
```

Text(0.5, 1.0, 'F1 Score of ML models (TF-IDF)')

Text(0.5, 1.0, 'F1 Score of ML models (TF-IDF)')

F1 Score of ML models (TF-IDF)





Flask WebApp:

Multi-Label Text Classification

127.0.0.1:5000

Is your comment toxic?

Enter your text

Predict

Multi-Label Text Classification

127.0.0.1:5000/predict

Is your comment toxic?

i hate this movie

Predict

Prob (Toxic): 0.96
 Prob (Severe Toxic): 0.78
 Prob (Obscene): 0.39
 Prob (Insult): 0.89
 Prob (Threat): 0.35
 Prob (Identity Hate): 0.56

Chapter 6

CONCLUSION

NLP-ML models like state-of-the-art transformer models like BERT outperformed traditional models like LSTM. Logistic Regression model gave the highest accuracy which can be attributed to the size of the dataset. Logistic Regression gave an AUC score of 99.98%, LSTM gave an AUC score of 95.24% and BERT gave an AUC score of 99.67%.

For the flask app, logistic regression model is used instead of BERT due the higher accuracy as well as the dip in AUC score in BERT model as seen in the graphical representation which suggests that a more balanced and larger dataset is required for BERT to perform well.

Chapter 7

FUTURE WORK

Since the dataset is related to hate speech and toxic comments found on social media, the flask app can be deployed on cloud platforms like github or Heroku and centralized to be used as detection system to identify situations of online bullying and harassment. This can be used to mitigate such situations on various social media platforms.

References

- [1] Haiming Wu, Yue Zhang, Xi Jin, Yun Xue, and Ziwen Wang, “Shared-private LSTM for Multi-domain Text Classification”, School of Physics and Telecommunication Engineering, 2019.
- [2] Siva Charan Reddy Gangireddy, “Supervised Learning for Multi-Domain Text Classification”, SJSU ScholarWorks, 2016.
- [3] Francesc Alías, Xavier Sevillano, Joan Claudi Socoró, “Text Classification based on Associative Relational Networks for Multi-Domain Text-to-Speech Synthesis”, Dept. of Communications and Signal Theory, 2016.
- [4] Evan Cox, Marcelo Worsley, “In Pursuit of an Efficient Multi-Domain Text Classification Algorithm”, Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012.
- [5] Lianghao Li, Xiaoming Jin, Sinno Jialin Pan, Jian-Tao Sun, “Multi-Domain Active Learning for Text Classification”, Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012.
- [6] Yitao Cai and Xiaojun Wan, “Multi-Domain Sentiment Classification Based on Domain-Aware Embedding and Attention”, Institute of Computer Science and Technology, Peking University, 2019.
- [7] YuanWu and Yuhong Guo, “Dual Adversarial Co-Learning for Multi-Domain Text Classification”, School of Computer Science, Carleton University, 2019.
- [8] Xilun Chen and Claire Cardie, “Multinomial Adversarial Networks for Multi-Domain Text Classification”, Cornell University ScholarWorks, 2018.
- [9] YuanWu, Diana Inkpen and Ahmed El-Roby, “Conditional Adversarial Networks for Multi-Domain Text Classification”, Carleton University ScholarWorks, 2021.
- [10] Sumanth Prabhu, Moosa Mohamed and Hemant Misra, “Multi-class Text Classification using BERT-based Active Learning”, Swiggy Applied Research, 2021.

- [11] Abhiteja Gajjala, “Multi-Faceted Text Classification using Supervised Machine Learning Models”, SJSU ScholarWorks, 2016.
- [12] Jason D. M. Rennie, “Improving Multi-Class Text Classification with Naïve Bayes”, B.S. Computer Science Carnegie Mellon University, 1999.
- [13] Xavier Sevillano, Francesc Alías*, Joan Claudi Socoró, “ICA-Based Hierarchical Text Classification for Multi-Domain Text-to-Speech Synthesis”, Department of Communications and Signal Theory, 2014.
- [14] Limin Wang and Dexin Wang, “Solving Stance Detection on Tweets as Multi-Domain and Multi-Task Text Classification”, National Social Science Fund of China Project by the Mechanism of Action of Network Groups in the Evolution of Emergencies, 2021.
- [15] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, Inderjit Dhillon, “X-BERT: eXtreme Multi-label Text Classification using Bidirectional Encoder Representations from Transformers”, NeurIPS Science Meets Engineering of Deep Learning Workshop, 2019.