RIDDHI MEHTA J030
HUSAIN GHADIALI J56

NLP PROJECT

TOXIC COMMENTS CLASSIFIER

PROBLEM FORMULATION

1. What are toxic comments?

Toxic comments are those pieces of texts that users leave online in the form of comments or feedback that are having a malicious or derogatory intention.

They are texts which are written with the intention to cause someone to feel disturbed and can be rude and cruel.

2. Why are they harmful?

Such comments are directed to someone or something and are offensive in nature which can cause a variety of problems or negative effects.

They can make a person feel conscious, mentally disturbed, uncomfortable, and make them indulge in self-doubt.

People leave toxic comments about people, including abusive language, body shaming, comments on looks, and comments on one's behavior among many other abusive content.

NOT TOXIC MILDLY TOXIC TOXIC

Grilled cheese ideas are limitless.

Thanks for the chicken health info! My hens LOVE the mealworms, and would like for me to win.

Yes, the socialist Hitler. Far right-wing he was. Uh huh.

They got their heads so far in the sand they will never recognize a wolf dressed like a sheep.

You are a disgusting, subhuman, painfully stupid waste of cells. You are a racist pig, a slime ball.

Liberals are devil lovers.

3. Why do we need a toxic comments classifier?

A toxic comments classifier is crucial, especially in times like these, with the growing dependence of the current generation on social media and technology.

According to a study conducted, nine out of ten teens aged 13-17 use social media platforms, and most (71%) use more than one.

With a large youth population using social media it becomes easy for anyone to communicate openly and publicly with anyone all over the world.

This gives the notorious people with a questionable intention to openly censure and insult people in public, while also maintaining anonymity in some cases, while leaving the recipient of such comments to feel depressed and affected.

CURRENT SCENARIO

4. Benefits of toxic comments classifier

The benefits of toxic comments classifier are that it can identify texts which are toxic and can classify them as that as soon as the person types in the comment. So there is no need for the victim to go through the pains of manually reporting the abusive comment. And in fact, the victim will not even become a victim in the first place!

This makes it easier for the platform which uses this classifier to report or delete these toxic statements from their platform, preventing the negative consequences of these mindless and toxic comments.

5. Scope of such a Classifier

The number of applications and websites we visit are ever increasing and with it comes the issue of toxic comments, especially on platforms that where the user is allowed to input texts, comment or give their feedback in any form.

These classifiers can be used on all such platforms online and by applications where such problems are prevalent.

It's becoming increasingly important to eliminate such toxic comments with increasing advancement in technology and also with increasing concerns about our mental health while depending on it.

LOGISTICS

1. IDE - google colab and jupyter notebook

Google colab is among the most preferred integrated development environment.

Colab is ideal for everything from improving your Python coding skills to working with deep learning libraries, like PyTorch, Keras, TensorFlow, and OpenCV.

You can create notebooks in Colab, upload notebooks, store notebooks, share notebooks, mount your Google Drive and use whatever you've got stored in there, import most of your favorite directories, upload your personal Jupyter Notebooks, upload notebooks directly from GitHub, upload Kaggle files, download your notebooks, and do just about everything else that you might want to be able to do.

It provides a free cloud service and also a free and faster GPU!

As far as jupyter notebook is concerned, it has full support of GUI, which at the moment is not available on google colab. So **jupyter notebook** has been used at the last stage to develop GUI using "tkinter" supporting library.

2. Language - python 3

The simple syntax and transparent semantics of this language makes it an excellent choice for projects that include Natural Language Processing tasks.

More importantly, it provides developers with an extensive collection of NLP tools and libraries that enable developers to handle a great number of NLP-related tasks such as document classification, topic modeling, part-of-speech (POS) tagging, word vectors, and sentiment analysis.

3. Data - Toxic Comment Classification Challenge

The train dataset is taken from the site 'kaggle.com'.

It includes data with the following columns: **id, comment_text, toxic, severe_toxic, obscene, threat, insult, identity_hate.**

4. Important Libraries used

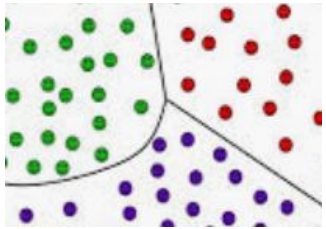
- **Pandas**: Built on the Numpy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables.
- **Numpy**: stands for 'Numerical Python'. It adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **NLTK**: One of the leading platforms for working with human language data and Python; widely used for NLP.
- **Sklearn**: It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- **Tensorflow**: Library for fast numerical computing, that can be used to create Deep Learning models.
- **Keras**: A deep learning library which allows for easy and fast prototyping.

5. Tkinter GUI

Used to display the final output of the input text, categorized as per the nature of the comment. There is a system of letting the user ENTER a comment, and when the user clicks on the BUTTON, the algorithm runs and the nature of the comment is DISPLAYED. Along with the nature, there is a probability graph displayed on the side which shows the different toxicity intensity of every class.

MULTI-CLASS vs MULTI-LABEL

Simply put, multi class categorizes the output into one of the possible outputs. Which means that the output can be only one of the outputs and not more.
Eg. An input picture can be classified **ONLY** as either a dog, a cat, or a cow.

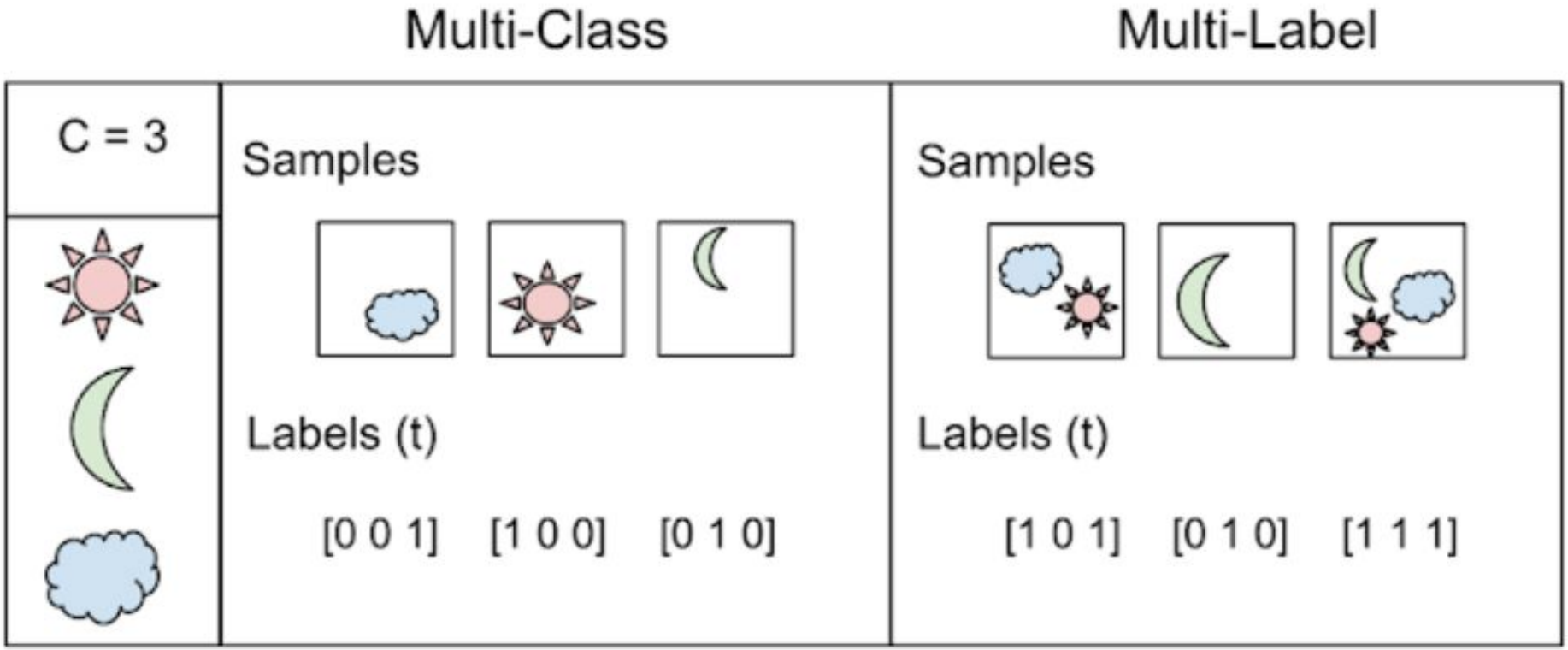


In multi label, the input can be categorized as more than one labels, which are independant to one other.
Eg. A book can be fictional, it can also be of the mystery, it can also be award-winning.

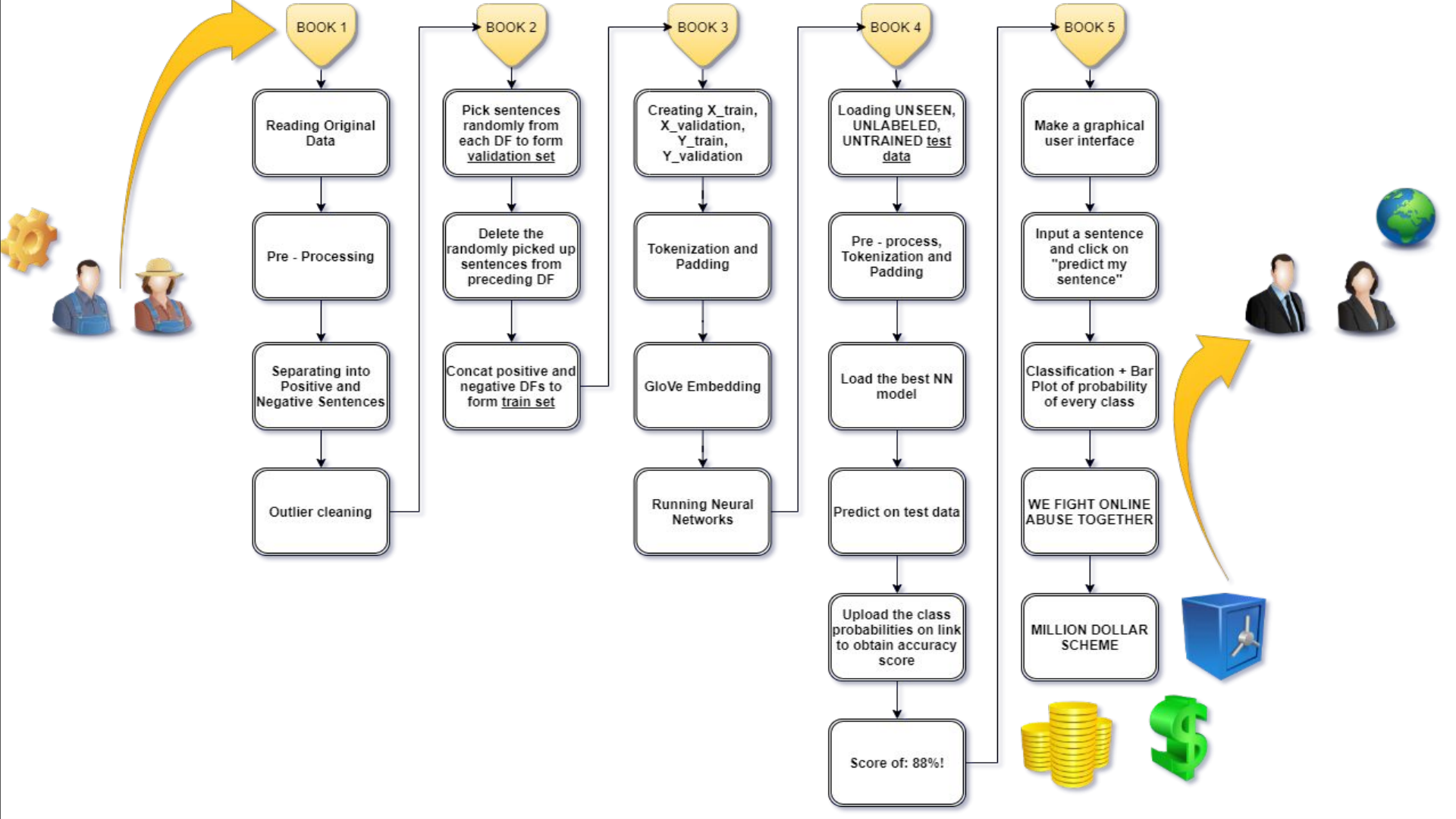
['Drama', 'Thriller']

Our main objective of classifying the toxic comments revolves around the concept of multi labelling. Toxic comments can have more than one label depending on the type of comment, and therefore we are required to go ahead with multi labelling.

EXAMPLE using the same items

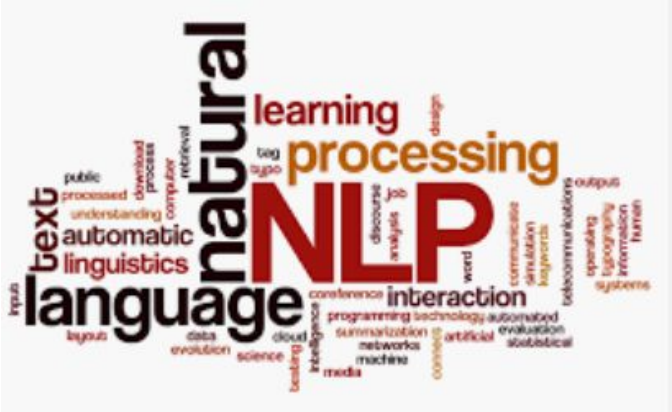


PROJECT FLOWCHART


BOOK 1
Reading Original Data
Pre - Processing
Separating into Positive and Negative Sentences
Outlier cleaning
BOOK 2
Pick sentences randomly from each DF to form validation set
Delete the randomly picked up sentences from preceding DF
Concat positive and negative DFs to form train set
BOOK 3
Creating X_train, X_validation, Y_train, Y_validation
Tokenization and Padding
GloVe Embedding
Running Neural Networks
BOOK 4
Loading UNSEEN, UNLABELED, UNTRAINED test data
Pre - process, Tokenization and Padding
Load the best NN model
Predict on test data
Upload the class probabilities on link to obtain accuracy score
Score of: 88%!
BOOK 5
Make a graphical user interface
Input a sentence and click on "predict my sentence"
Classification + Bar Plot of probability of every class
WE FIGHT ONLINE ABUSE TOGETHER
MILLION DOLLAR SCHEME

CODE NOTEBOOKS

GOOGLE COLAB – BOOK 1



1. Reading original data

The original data consists of 159571 rows and 7 columns.

The first step is to read this data to Google colab.

2. EDA

It is an approach to analyze data sets to get the main characteristics of the data set and summarize it. Visual methods are often used to display the results. A statistical method may or may not be used.

It is a process of performing initial investigations on the data set in order to discover patterns, spot outliers, test hypothesis and check assumptions with the help of summarizing statistics and graphical representations.

3. Pre-processing

Text data available for analysis is not always in the form that can be used to perform tests on.

Most often, the text data requires to be preprocessed before it can be used. This includes converting the data into a more acceptable form.

This conversion allows the user to perform tests and returns better and more accurate results.

Here's a few techniques used to preprocess our text data before using it for analysis:

Noise removal:

- Noise removal is the process of removing digits, characters and pieces of texts that interfere with the text analysis.
- Eg. '1.Start', '...beginning...', '#selfie', are some examples of where digits or special characters can interfere with the text to be analyzed.
- Getting rid of these saves us the unnecessary complication of understanding their significance, which does not contribute to the analysis of the text and adds no meaning to the text.

Lowercasing:

- One of the simplest and most effective form of preprocessing
- Helps specially where the data is not very large
- Helps with consistency of expected output
- Effective in cases where the same word may sometimes be used with upper case and sometimes with lower case. The words may mean the same but are taken as different words by the algorithm.
- Must be avoided for data where the upper case words have significance and must remain upper cased. Eg. US, meaning United States, unlike ‘us’.

Conversion to full forms:

- Not all words in the data set were spelt correctly with many in their short forms.
- Conversion to full form included converting these words from their short form to their full forms.
- Eg. Conversion of ‘scuse’ to ‘excuse’.

Stemming:

- Stemming is the process of reducing words to their root form.
- Eg. Connection, connected, connects. All can be reduced to their root form, being ‘connect’.
- Stemming is done using a crude heuristic method of simply chopping off the ends of the words, with the aim to transform the word into its root form correctly.
- Sometimes, it can be incorrect and is not always effective. Eg. The words, troubled, troubled, troubles, might be converted to troubl instead of trouble because the ends were simply chopped off. It cuts off ‘ed’ since most words have ‘ed’ added to their root form.
- There are different algorithms for stemming, the most common being Porter’s Algorithm.
- It helps with standardizing vocabulary.
- Search applications in particular benefit from stemming immensely.
- While searching for something, we want the results of it with all the forms of the input words used. Eg. “Best coaching classes” should also return results for “Best coaching class”, etc.

Lemmatization:

- Lemmatization is similar to stemming in the sense of transforming a word into its root form.
- The difference is that lemmatization does so in a more effective way.
- Stemming aims to find the root word by simply chopping words off. Lemmatization actually finds the root word for the word in the data.
- Eg. The word ‘better’ would become ‘good’ after being lemmatized.
- It may or may not be effective for the data that the user has. It takes more time to compute and sometimes has little or no advantage over stemming, which is faster and easier to compute.
- One can try lemmatization on their data set and see whether it increases accuracy to be sure of using it.

Stopwords Removal:

- Stop words are a set of words which are commonly used in a language but do not particularly contribute to the analysis of the text.
- It does not play a significant part in the analysis as they contain no information about the purpose of the text.
- The idea behind removing stop words is that, by removing words that do not contribute to the meaning or purpose of the sentence, we can focus on the more important words in the sentence.
- Eg. ‘a’, ‘the’, ‘is’, etc. are considered stop words.
- Stop words lists come in pre-established lists from libraries, or you could also create a custom list for your use.

Normalization:

- Text normalization is the process of transforming a text into a canonical/standard form.
- Eg. The word ‘gooood’, ‘gud’ can be transformed into its canonical form, ‘good’.
- Another purpose of it is to transform near identical words into the same standard word. Eg. The words ‘stopwords’, ‘stop words’ and ‘stop-words’ to just ‘stopwords’.
- This method is especially applicable and important for texts on social media, platforms, comments, etc. where misspellings, abbreviations, and out-of-vocabulary words (oov) are common.

```
[ ]  def preprocessing(text):

       text = text.lower()
       text = re.sub(r"what's", "what is ", text)
       text = re.sub(r"\'s", " ", text)
       text = re.sub(r"\'ve", " have ", text)
       text = re.sub(r"can't", "cannot ", text)
       text = re.sub(r"n't", " not ", text)
       text = re.sub(r"i'm", "i am ", text)
       text = re.sub(r"\'re", " are ", text)
       text = re.sub(r"\'d", " would ", text)
       text = re.sub(r"\'ll", " will ", text)
       text = re.sub(r"\'scuse", " excuse ", text)
       text = re.sub('\W', ' ', text) # substitutes non word character with a space
       text = re.sub('\s+', ' ', text)
       text = text.strip(' ')
       text = re.sub(r"[0-9]"," ",text) # substitutes numbers with a blank space

       my_tokens = list() # Initialising a list of tokens that will be formed
       stop_words= set(stopwords.words('english')) # Extracting english stopwords from nltk corpus

       tokens = word_tokenize(text, preserve_line=False)

       for token in tokens:

         if (token not in stop_words):
           my_tokens.append(token)

       final = ' '.join([str(elem) for elem in my_tokens]) # makes it an entire string

       return final
```

4. Separate sentences into 2 new dataframes

The dataset consists of positive and negative comments. To analyze the negative comments and categorize them, our first step would be to separate them into 2 data frames.

In the original dataset there are **143346** positive sentences and **16225** negative sentences.

5. Outlier Cleaning

Outliers in this case are comments that are either having too sentences or too little to be classified as a comment.

We have eliminated outliers based on the length. Therefore null texts too have been eliminated.

Then, to classify the sentences as good or bad, we have added two more columns, **pos and neg,** indicating '1' for positive in positive sentences and '1' for negative in negative sentences.

(Length of sentence, Dropping empty texts, Dropping outlier texts, Adding 2 more columns - pos & neg, Save those 2 into csv readable format) - **In Summary**

1. Calculating length of each sentence

```
[ ]  def length_of_sentence(text):
       return len(word_tokenize(text))
```

```
[ ]  positive_sentences["length"] = positive_sentences["comment_text"].apply(length_of_sentence)
     negative_sentences["length"] = negative_sentences["comment_text"].apply(length_of_sentence)
```

```
positive_sentences["length"].describe()
```

```
count    143288.000000
mean         35.405009
std          52.178395
min           1.000000
25%           9.000000
50%          19.000000
75%          39.000000
max        1250.000000
Name: length, dtype: float64
```

```
positive_sentences["length"].describe()
```

```
count    74136.000000
mean        20.198689
std          8.460624
min          9.000000
25%         13.000000
50%         19.000000
75%         26.000000
max         39.000000
Name: length, dtype: float64
```

```
negative_sentences["length"].describe()
```

```
count    16225.000000
mean        28.746872
std         68.379872
min          1.000000
25%          6.000000
50%         12.000000
75%         25.000000
max       1250.000000
Name: length, dtype: float64
```

```
negative_sentences["length"].describe()
```

```
count    8688.000000
mean       12.788329
std         5.484364
min         6.000000
25%         8.000000
50%        12.000000
75%        17.000000
max        25.000000
Name: length, dtype: float64
```

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate	positive	negative
0	explanation edits made username hardcore metal...	0	0	0	0	0	0	1	0
1	aww matches background colour seemingly stuck ...	0	0	0	0	0	0	1	0
2	hey man really trying edit war guy constantly ...	0	0	0	0	0	0	1	0
3	sorry word nonsense offensive anyway intending...	0	0	0	0	0	0	1	0
4	oh girl started arguments stuck nose belong be...	0	0	0	0	0	0	1	0

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate	positive	negative
0	bye look come think comming back tosser	1	0	0	0	0	0	0	1
1	sorry sorry screwed around someones talk page ...	1	0	0	0	0	0	0	1
2	get fucked get fuckeeed got drink cant put get...	1	0	1	0	0	0	0	1
3	stupid peace shit stop deleting stuff asshole ...	1	1	1	0	1	0	0	1
4	tony sidaway obviously fistfuckee loves arm ass	1	0	1	0	1	0	0	1

GOOGLE COLAB – BOOK 2

1. Read positive and negative sentences dataframe

After previously storing the positive and negative sentences in different dataframes, we read the data frames individually to carry on with the analysis.

2. Validation dataset

Now to create our validation dataset, we take a certain amount of sentences from both, the negative and positive sentences dataframe. We concat these sentences into one data frame which acts as our validation data.

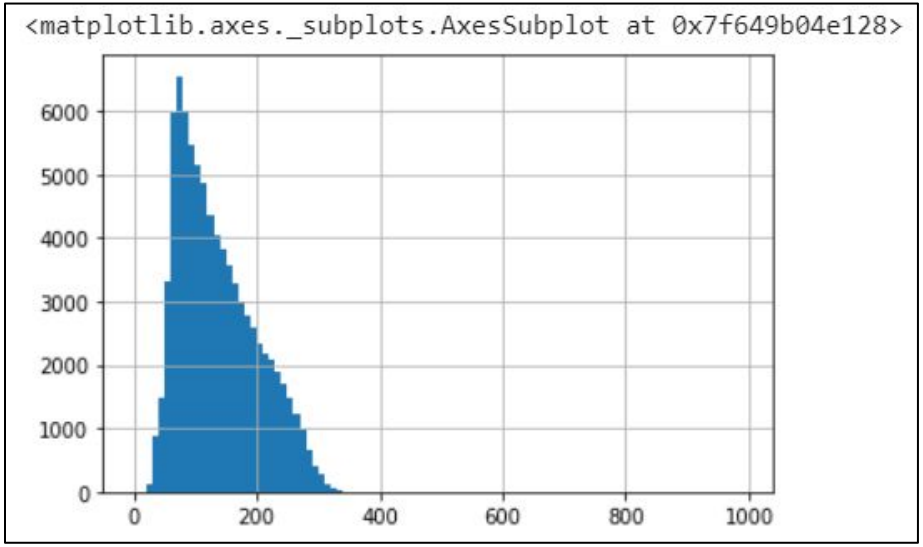
3. Train dataset

The rows selected for making the test data from both the data frames (positive and negative) are now dropped from both the data frames individually. This leaves us with the 2 data frames without the sentences used in the validation data. We concat these 2 data frames now to give us our training dataset.

4. Train and Validation dataset - read to save!

Now we save the two newly formed test and train datasets as different readable csv format files which are to be used for the future analysis.

GOOGLE COLAB – BOOK 3




```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

```
((82754,), (70,), (82754, 2), (70, 2))
```

1. Reading train and test

After saving the train and validation data in the previous notebook, we now read it for further analysis.

2. Histogram of number of characters.

The histogram gives us an intuition of the number of characters in our training data.

3. Creating X_train, X_validation, Y_train, Y_validation

It is essential to divide our data in order to observe how the model predicts on unseen data.

4. Tokenization and Padding

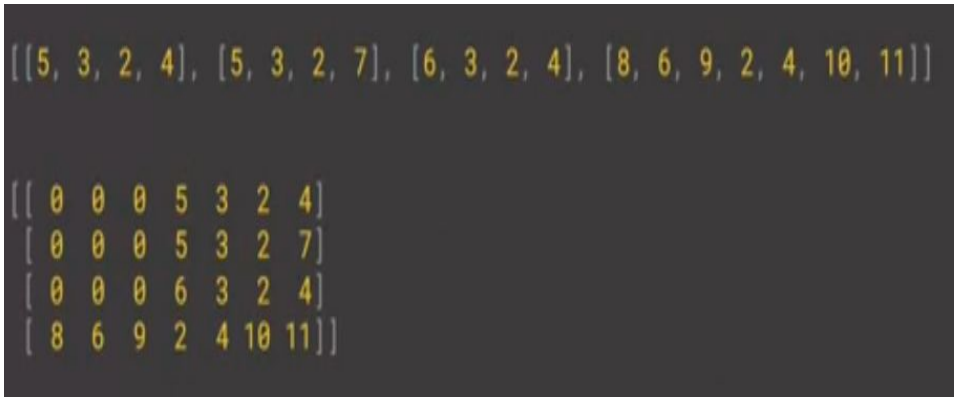
Given a character sequence, tokenization is the task of chopping it up into pieces, called *tokens*, perhaps at the same time throwing away certain characters, such as punctuation. Here is an example of tokenization:

Input: Friends, Romans, Countrymen, lend me your ears;

Output:

Friends	Romans	Countrymen	lend	me	your	ears

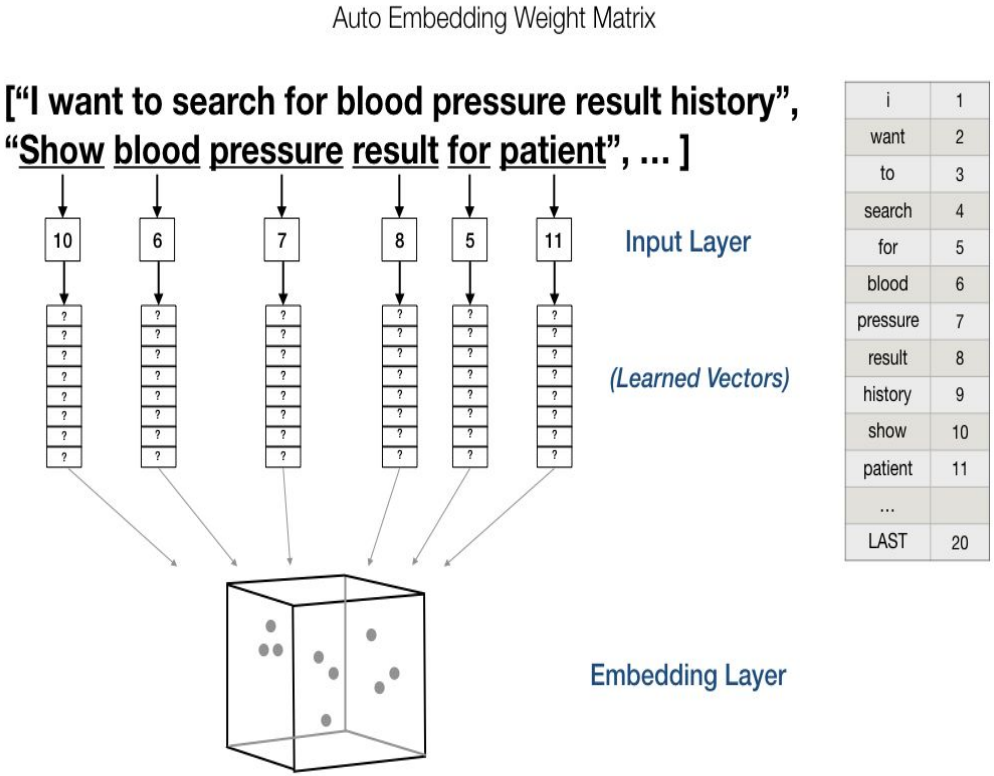
What padding essentially does is, it creates a matrix of as many rows as the number of sentences and as many columns as the number of words in the longest sentence.



5. GloVe Embedding

GloVe stands for Global Vectors and it is an unsupervised learning algorithm for obtaining vector representations for words.

This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity.



6. Running Neural Networks

A neural network is a series of algorithms that aims to identify the underlying relationship in the dataset, mimicking the way in which a human brain functions, acting as a system of neurons.

Summary of all the models:

Using GloVe Embeddings

There are 2 models which uses GloVe Embeddings:

1. Neural Network - ***Type 0***

- It uses 6 one-hot-encoded columns which forms the Y_train and Y_test.
- These columns are ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]
- The best training and validation accuracy is given by:

loss: 0.0737 - accuracy: 0.9935 - val_loss: 0.3127 - val_accuracy: 0.9900

2. Neural Network - ***Type 2***

- It uses 2 one-hot-encoded columns which forms the Y_train and Y_test.
- These columns are ["positive", "negative"]
- The best training and validation accuracy is given by:

loss: 0.1802 - accuracy: 0.9447 - val_loss: 0.5740 - val_accuracy: 0.7800

Without GloVe Embeddings

There are 2 models which uses keras.layers.Embedding (without GloVe Embedding)

1. Neural Network - ***Type 1***

- It uses 6 one-hot-encoded columns which forms the Y_train and Y_test.
- These columns are ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]
- The best training and validation accuracy is given by:

loss: 0.0730 - accuracy: 0.9948 - val_loss: 0.3138 - val_accuracy: 0.9900

2. Neural Network - ***Type 3***

- It uses 2 one-hot-encoded columns which forms the Y_train and Y_test.
- These columns are ["positive", "negative"]
- The best training and validation accuracy is given by:

loss: 0.1434 - accuracy: 0.9552 - val_loss: 0.5210 - val_accuracy: 0.8300

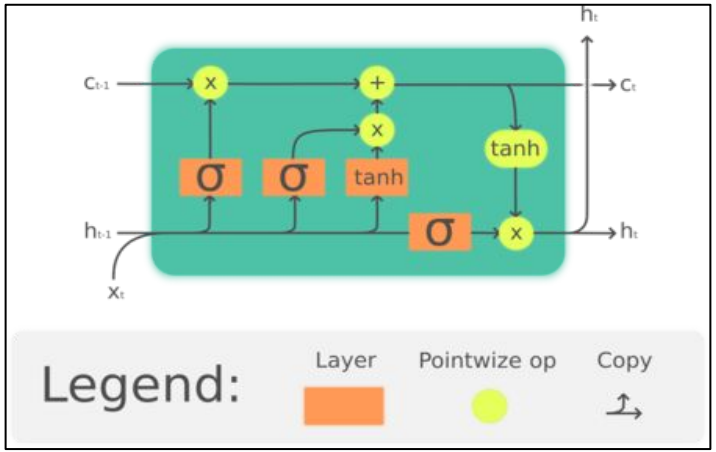
COMPONENTS OF A NEURAL NETWORK

1. **Embedding Layer**

- A word embedding is a class of approaches for representing words and documents using a dense vector representation.
- Keras offers an Embedding layer that can be used for neural networks on text data.
- It requires that the input data be integer encoded, so that each word is represented by a unique integer.
- For integer encoding, we have already used Tokenizer API of Keras.
- The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.
- The Embedding layer is defined as the first hidden layer of a network.
- It must specify 3 arguments:
    1. Input_dim - Size of vocabulary in text data
    2. Output_dim - size of vector space in which words will be embedded
    3. Input_length - Length of input sentences

2. LSTM Layer

- Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems.
- Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points, but also entire sequences of data.



3. Activation Functions

- Activation functions are mathematical equations that determine the output of a neural network.
- The common activation functions used are **sigmoid**, **tanh**, **softmax**, **relu**.
- Sigmoid:
  - Output values bound between 0 and 1.
  - Smooth gradient, preventing “jumps” in output values.
  - Highly used in multi-label classification as it outputs probability wrt to each class.
  - So for example, if 2 classes out of 4 have a probability >=0.5 then these two classes together will be the predicted class.

$$f(x) = \frac{1}{1+e^{-(x)}}$$

- Tanh:
  - Zero centered—making it easier to model inputs that have strongly negative, neutral, and strongly positive values.
  - In other ways, very alike the Sigmoid function but the range of output is different.

$$f(x) = tanh(x)$$

3. Activation Functions (contd.)

- Softmax:
  - Able to handle multiple classes.
  - Gives the probability of the input value being in a specific class, and the highest probability is the predicted class.
  - It does not consider the inter-relation between the classes, and hence it is not useful in multi-label classification.

$$f_i(\vec{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}}$$

- ReLU:
  - Computationally efficient
  - Non-linear, allows for backpropagation.
  - The activation functions used for our project are Sigmoid and ReLU.

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

4. Loss Functions

- Our objective in a neural network is to minimize loss by optimizing its parameters/weights.
- The loss is calculated using loss function by matching the target(actual) value and predicted value by a neural network.
- A few types of loss functions are:
    - Mean Squared Error
    - Root Mean Squared Error
    - Mean Absolute Error
    - Huber Loss
    - **Cross Entropy (Categorical and Binary) -** these two losses are essential to understand and differentiate in order to use them accurately in our problem statement.

- In simple words, ***binary cross entropy is used with sigmoid activation function*** as here we need to classify each class individually. Hence, whether its zero / one wrt to each class, hence we require “binary” approach.
- ***Categorical cross entropy is used with softmax activation function*** as softmax outputs probabilities of every class wrt to one another, hence sum of probabilities is one. And since, there are many classes, we require “categorical” approach.

5. Metrics

- The metrics used to assess the neural network are **accuracy, precision, recall, F1_score**.
- **Accuracy** is the metric that determines how effective the model is to classify the outputs correctly.
- **Precision** talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive.
- **Recall** actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive).
- **F1 Score** is a function of precision and recall. It might actually be a better measure to use if we need to seek a balance between Precision and Recall as well as when there is an uneven class distribution.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{F1} = 2 \times \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

OUR BEST NEURAL NETWORK

Neural Network - Type 1

Layers:

1. Embedding (**WITHOUT glove**)
2. Convolutional 1D layer
3. Max Pooling
4. LSTM
5. Global Max Pooling
6. Dropout
7. Dense
8. Drop-out
9. Dense (final output layer)

Input Data:

1. X_train: normal
2. X_test: normal
3. Y_train: **6 columns** (toxic, severe_toxic, obscene, threat, insult, identity_hate)
4. Y_test: same as Y_train

Architecture

```
[ ]  vocab_size = len(tokenizer.word_index) + 1

     inp = Input(shape = (maxlen,))

     x = Embedding(vocab_size, 250)(inp)

     x = Conv1D(filters = 100, kernel_size = 4, padding = 'same', activation = 'relu')(x)

     x = MaxPooling1D(pool_size=4)(x)

     x = LSTM(50, return_sequences = True, dropout = 0.1, recurrent_dropout = 0.1)(x)

     x = GlobalMaxPool1D()(x)

     x = Dense(50, activation = "relu")(x)

     x = Dropout(0.2)(x)

     x = Dense(6, activation = "sigmoid")(x)

     model = Model(inputs = inp, outputs = x)

     model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

Model Summary

```
[ ]  model.summary()
```

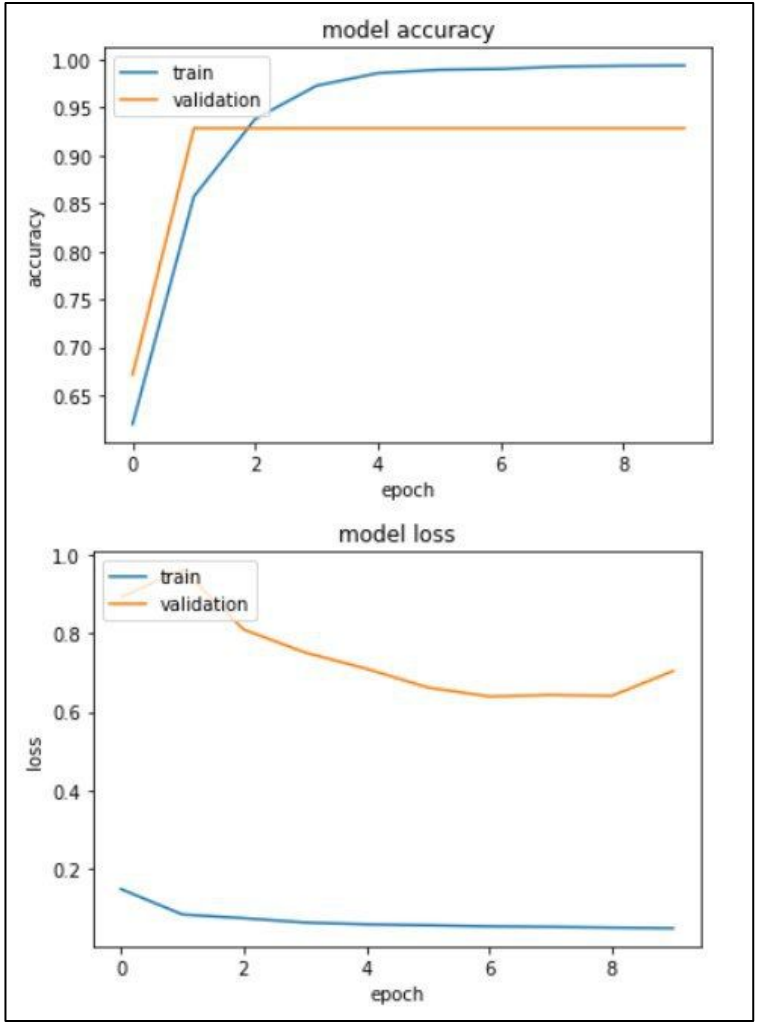
```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 400)]             0
_________________________________________________________________
embedding_2 (Embedding)      (None, 400, 250)          198250
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 400, 100)          100100
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 100, 100)          0
_________________________________________________________________
lstm_2 (LSTM)                (None, 100, 50)           30200
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 50)                0
_________________________________________________________________
dense_4 (Dense)              (None, 50)                2550
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 306
=================================================================
Total params: 331,406
Trainable params: 331,406
Non-trainable params: 0
```

Fitting and Saving the BEST Model Weights

```
[ ] batch_size = 256
    epochs = 10

    file_path = "/content/drive/My Drive/Colab Notebooks/NLP - Sem 6/Project/Models/nn1_new.hdf5"
    checkpoint = ModelCheckpoint(file_path, monitor ='val_loss', verbose = 1, save_best_only = True, mode = 'min')

    early = EarlyStopping(monitor = "val_loss", mode = "min", patience = 20)

    callbacks_list = [checkpoint, early]

    history = model.fit(X_train, Y_train, batch_size = batch_size, epochs = epochs, validation_data = (X_test, Y_test), callbacks = callbacks_list, verbose = 1)
```

```
Epoch 1/10
324/324 [==============================] - ETA: 0s - loss: 0.1492 - accuracy: 0.6198
Epoch 00001: val_loss improved from inf to 0.89243, saving model to /content/drive/My Drive/Colab Notebooks/NLP - Sem 6/Project/Models/nn1_new.hdf5
324/324 [==============================] - 129s 398ms/step - loss: 0.1492 - accuracy: 0.6198 - val_loss: 0.8924 - val_accuracy: 0.6714
Epoch 2/10
324/324 [==============================] - ETA: 0s - loss: 0.0847 - accuracy: 0.8573
Epoch 00002: val_loss did not improve from 0.89243
324/324 [==============================] - 129s 397ms/step - loss: 0.0847 - accuracy: 0.8573 - val_loss: 0.9617 - val_accuracy: 0.9286
Epoch 3/10
324/324 [==============================] - ETA: 0s - loss: 0.0752 - accuracy: 0.9382
Epoch 00003: val_loss improved from 0.89243 to 0.80936, saving model to /content/drive/My Drive/Colab Notebooks/NLP - Sem 6/Project/Models/nn1_new.hdf5
324/324 [==============================] - 129s 398ms/step - loss: 0.0752 - accuracy: 0.9382 - val_loss: 0.8094 - val_accuracy: 0.9286
Epoch 4/10
324/324 [==============================] - ETA: 0s - loss: 0.0641 - accuracy: 0.9729
Epoch 00004: val_loss improved from 0.80936 to 0.75123, saving model to /content/drive/My Drive/Colab Notebooks/NLP - Sem 6/Project/Models/nn1_new.hdf5
324/324 [==============================] - 128s 395ms/step - loss: 0.0641 - accuracy: 0.9729 - val_loss: 0.7512 - val_accuracy: 0.9286
Epoch 5/10
324/324 [==============================] - ETA: 0s - loss: 0.0596 - accuracy: 0.9860
Epoch 00005: val_loss improved from 0.75123 to 0.70980, saving model to /content/drive/My Drive/Colab Notebooks/NLP - Sem 6/Project/Models/nn1_new.hdf5
324/324 [==============================] - 128s 396ms/step - loss: 0.0596 - accuracy: 0.9860 - val_loss: 0.7098 - val_accuracy: 0.9286
Epoch 6/10
324/324 [==============================] - ETA: 0s - loss: 0.0572 - accuracy: 0.9895
Epoch 00006: val_loss improved from 0.70980 to 0.66209, saving model to /content/drive/My Drive/Colab Notebooks/NLP - Sem 6/Project/Models/nn1_new.hdf5
324/324 [==============================] - 128s 396ms/step - loss: 0.0572 - accuracy: 0.9895 - val_loss: 0.6621 - val_accuracy: 0.9286
Epoch 7/10
324/324 [==============================] - ETA: 0s - loss: 0.0544 - accuracy: 0.9904
Epoch 00007: val_loss improved from 0.66209 to 0.63924, saving model to /content/drive/My Drive/Colab Notebooks/NLP - Sem 6/Project/Models/nn1_new.hdf5
324/324 [==============================] - 129s 397ms/step - loss: 0.0544 - accuracy: 0.9904 - val_loss: 0.6392 - val_accuracy: 0.9286
Epoch 8/10
324/324 [==============================] - ETA: 0s - loss: 0.0531 - accuracy: 0.9929
Epoch 00008: val_loss did not improve from 0.63924
324/324 [==============================] - 128s 396ms/step - loss: 0.0531 - accuracy: 0.9929 - val_loss: 0.6430 - val_accuracy: 0.9286
Epoch 9/10
324/324 [==============================] - ETA: 0s - loss: 0.0512 - accuracy: 0.9936
Epoch 00009: val_loss did not improve from 0.63924
324/324 [==============================] - 128s 397ms/step - loss: 0.0512 - accuracy: 0.9936 - val_loss: 0.6408 - val_accuracy: 0.9286
Epoch 10/10
324/324 [==============================] - ETA: 0s - loss: 0.0496 - accuracy: 0.9940
Epoch 00010: val_loss did not improve from 0.63924
```

```
[ ]  # summarize history for accuracy
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'validation'], loc='upper left')
     plt.show()
     # summarize history for loss
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'validation'], loc='upper left')
     plt.show()
```



Prediction on X_test

```
[ ]  model = load_model(r"/content/drive/My Drive/Colab Notebooks/NLP - Sem 6/Project/Models/nn1_new.hdf5")
     y_pred = model.predict(X_test, verbose = 1)

     outer_list = []

     for y in y_pred:

       inner_list = []

       for i in y:

         if i >= 0.5:
           inner_list.append(1)

         else:
           inner_list.append(0)

       outer_list.append(inner_list)

     df = pd.DataFrame.from_records(outer_list)
     df.columns = ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]
     df
```

Prediction on Validation Data

	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
...	...	...	...	...	...	...
65	1	0	1	0	1	0
66	1	0	1	0	1	0
67	1	0	1	0	1	0
68	1	0	1	0	0	0
69	1	0	1	0	1	0

70 rows × 6 columns

Classification Report of Validation Data

```
[ ]  print("F1 score = " + str(f1_score(Y_test, df, average = 'samples')))
     print(classification_report(Y_test, df))
```

```
F1 score = 0.5424489795918367
                precision    recall  f1-score   support

             0       0.96      0.89      0.92        55
             1       0.00      0.00      0.00        11
             2       0.90      0.77      0.83        48
             3       0.00      0.00      0.00        24
             4       0.95      0.82      0.88        44
             5       0.00      0.00      0.00        26

     micro avg       0.94      0.59      0.72       208
     macro avg       0.47      0.41      0.44       208
  weighted avg       0.66      0.59      0.62       208
   samples avg       0.68      0.48      0.54       208
```

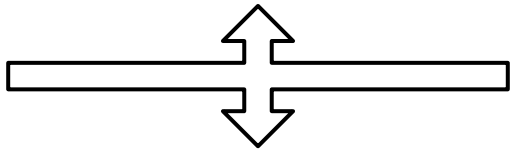
GOOGLE COLAB – BOOK 4

1. **Loading kaggle's unseen, untrained and unlabelled test data.**

```
k_test.head()
```

	id	comment_text
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
3	00017563c3f7919a	:If you have a look back at the source, the in...
4	00017695ad8997eb	I don't anonymously edit articles at all.

2. Preprocess, Tokenization, and Padding on comment_text column
3. Load the best neural network model
4. Predict on test data
5. Obtain probabilities in the form of list of lists; then convert it to dataframe



```
pred
```

```
array([[9.98230636e-01, 3.78732234e-01, 9.95744884e-01, 3.83478068e-02,
        9.53207374e-01, 1.62006870e-01],
       [8.34740877e-01, 2.08991636e-02, 1.68355972e-01, 1.38262389e-02,
        2.54785955e-01, 5.29731847e-02],
       [8.77261698e-01, 2.34035738e-02, 3.05176526e-01, 1.62008721e-02,
        3.22717875e-01, 4.08973806e-02],
       ...,
       [3.47051546e-02, 6.06870803e-04, 7.28017464e-03, 1.09573384e-03,
        1.23752151e-02, 2.59249192e-03],
       [1.15491375e-02, 3.89082212e-04, 3.01267439e-03, 5.43617352e-04,
        4.73518018e-03, 1.11420732e-03],
       [7.42655277e-01, 1.16674146e-02, 4.91627991e-01, 8.05689022e-03,
        3.80228251e-01, 1.68845709e-02]], dtype=float32)
```

```
df.head()
```

	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0.998231	0.378732	0.995745	0.038348	0.953207	0.162007
1	0.834741	0.020899	0.168356	0.013826	0.254786	0.052973
2	0.877262	0.023404	0.305177	0.016201	0.322718	0.040897
3	0.007811	0.000280	0.001792	0.000366	0.002636	0.000558
4	0.755440	0.007590	0.099156	0.007321	0.156043	0.028790

```
sample_submission.head()
```

	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	0.5	0.5	0.5	0.5	0.5	0.5
1	0000247867823ef7	0.5	0.5	0.5	0.5	0.5	0.5
2	00013b17ad220c46	0.5	0.5	0.5	0.5	0.5	0.5
3	00017563c3f7919a	0.5	0.5	0.5	0.5	0.5	0.5
4	00017695ad8997eb	0.5	0.5	0.5	0.5	0.5	0.5

6. Concat it with original
7. Drop column “comment_text”, then save into CSV
8. Upload on kaggle to check the validity of the model

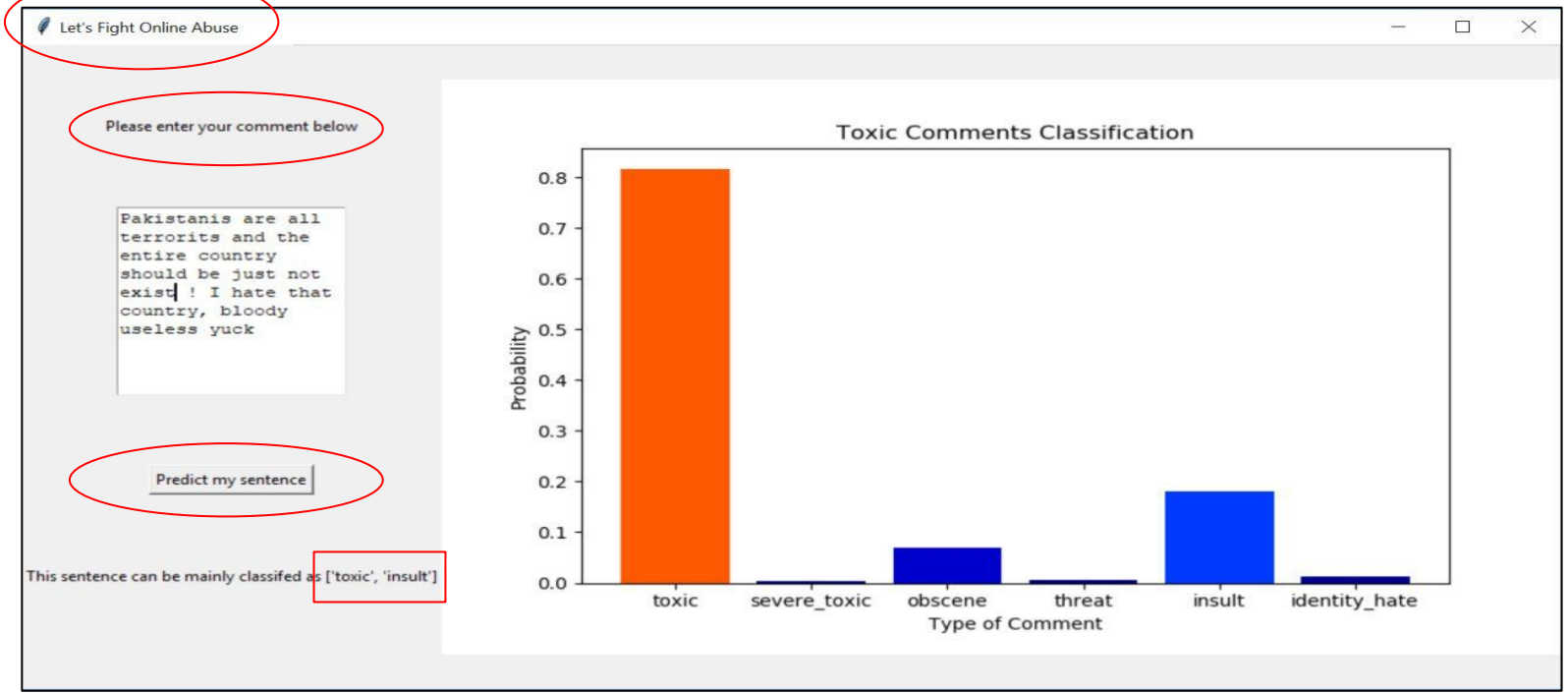
9. Check score

- We achieved a score of **<u>0.88993</u>** after uploading the model on Kaggle.

JUPYTER NOTEBOOK – BOOK 5

Library - Tkinter

- Tkinter is the standard GUI library for Python.
- Python when combined with Tkinter provides a fast and easy way to create GUI applications.
- Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.



THANK - YOU

RIDDHI MEHTA J030
HUSAIN GHADIALI J56