A

Minor Project Report

on

Emotion Classification based on Hinglish text

Submitted for partial fulfillment for the degree of

Bachelor of Technology

(Information Technology)

in

Department of Information Technology

by

Siddhant Bhanot
189402111
&
Sagar Rajput
189303154

Under the Guidance of
Mr. Ankit Mundra

(June-2021)

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

MANIPAL UNIVERSITY JAIPUR

CERTIFICATE

17th June, 2021

This is to certify that the project titled **Emotion Classification based on Hinglish text** is a record of the Bonafede work done by **Siddhant Bhanot** (189402111) & **Sagar Rajput** (189303154) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B. Tech) in **(Discipline)** of Manipal University Jaipur, during the academic year 2020-21.

Mr. Ankit Mundra

Project Guide, Dept of Information Technology

Manipal University Jaipur

Dr. Pankaj Vyas

HOD, Dept of Information Technology

Manipal University Jaipur

ABSTRACT

It is important to classify social media text to remove and report the text which is not suitable from the social media platform and thus control hate speech. The purpose of this project is to classify social media Hinglish text based on aggression into 3 classes namely NAG(Non-Aggressive), CAG (Covertly Aggressive), and OAG (Overly Aggressive). We investigate different ML models for this purpose and the final predictions are made using both conventional ML models and DNN (Deep Neural Networks).

1. INTRODUCTION

1.1. Motivation

- Overview

With the increase in hate speech, it is important to classify social media text to remove and report the text which is not suitable from the social media platform and thus control hate speech.

  - Usage of code-switched languages like Hinglish has increased on the social media platform.
  - However, classifying such sentences is a challenge due to grammar and spelling variations.
  - This project aims to build a working ML model with a good accuracy to classify such sentences.

- Applications

This project is applicable in keeping a check on Hinglish social media posts that violate the social media guidelines owing to hate speech and violence.

Thus, the process of filtering out such posts can be automated and improved in terms of efficiency and speed.

1.2. Project Objective

Emotion Classification in HINGLISH social media text using supervised Machine Learning Algorithms

This project aims at classifying social media Hinglish text based on aggression into 3 classes namely NAG (Non-Aggressive), CAG (Covertly Aggressive) and OAG (Overly Aggressive).

2. BACKGROUND OVERVIEW

2.1. Conceptual Overview

- ### Text Augmentation

It is the process of artificially extending a set of existing data by performing on them some operations which depend on the data type. The goal is to make new labeled data from the prevailing ones.

There are different techniques for text-based data augmentation. One process includes 4 techniques *(Jason Wei et al, 2019)* namely – Synonym replacement (SR), Random Insertion (RI), Random Swap (RS), Random Deletion(RD).

- Synonym Replacement (SR) – Randomly choose n words from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.
-
- Random Insertion (RI) – Find a random synonym of a random word in the sentence that is not a stop word. Insert that synonym into a random position within the sentence
-
- Random Swap (RS) – Randomly choose two words within the sentence and swap their positions
-
- Random Deletion (RD) – Randomly remove each word within the sentence with probability p.

Another popular study generated new data by translating sentences into French and back into English *(Yu et al., 2018)* which is known as back translation.

- ### WORD EMBEDDING

Word embeddings is a method for representing words and documents in numerical format. Word Embedding is a numeric vector input that represents a word in a lower-dimensional space such that words that have similar meaning are closely represented in the dimensional space.

Each word is mapped to one vector having tens or hundreds of dimensions in contrast with thousands or millions of dimensions required for sparse word representations such as in one-hot encoding.

Machine Learning algorithms cannot take strings or plain text as an input in NLP problems, rather they require numbers. There are different methods to map this plain string into numerical representation, one of which is the most effective is Word Embeddings.

Word embeddings are classified into two types: -

2) <u>Frequency Based Embeddings</u> – vectorize the words depending on the frequency of occurrence of the words in the text/ document. E.g. TF-IDF, BOW

2) <u>Prediction Based embeddings</u> – Vectors learned through neural networks, providing probabilities to the words. E.g. Word2Vec

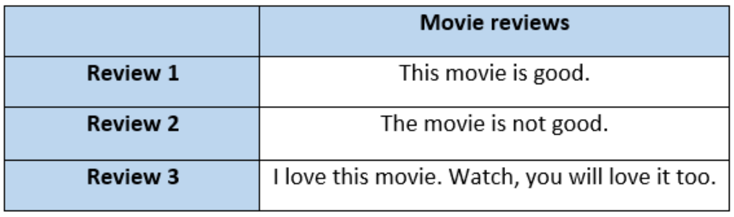
• BOW

Bag of words is the most trivial representation of text into vectors. Each column of a vector represents a word. The values in each cell of a row show the number of occurrences of a word in a sentence.

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

Example:



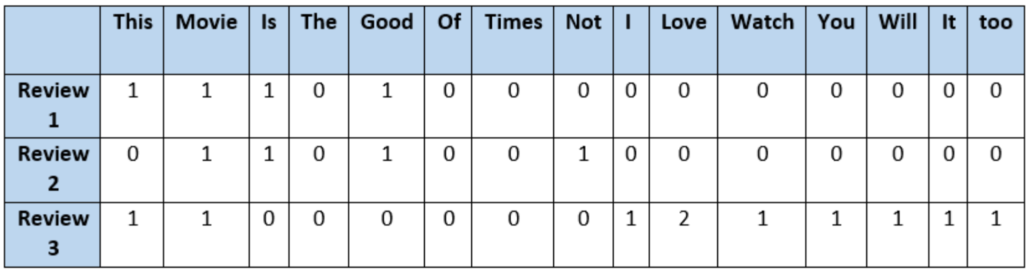
	Movie reviews
Review 1	This movie is good.
Review 2	The movie is not good.
Review 3	I love this movie. Watch, you will love it too.

Figure 1. BoW example

Vocabulary of unique words:

[This, movie, is, the, good, of, times, not, I, love, watch, you, will, it, too]

In our vocabulary, we have 15 unique words. Therefore, each movie review is represented by a vector of 15 dimensions (each word representing a dimension). The values corresponding to each word shows the number of occurrences of a word in a review.



	This	**Movie**	**Is**	**The**	**Good**	**Of**	**Times**	**Not**	**I**	**Love**	**Watch**	**You**	**Will**	**It**	**too**
Review 1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
Review 2	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0
Review 3	1	1	0	0	0	0	0	0	1	2	1	1	1	1	1

Figure 2. BoW distribution

n-grams:

n-grams are a neighboring sequence of n-words. n can be any positive integer.

Example — “Manipal University Jaipur” is a three-gram, “Manipal University” is a two-gram.

Drawbacks of BOW

- The length of the vector can be high, and most of the values are zero. Computationally, it is not efficient if the vocabulary is very high.
- Uni-gram based BOW is unable to capture the context of the text. Bi-grams and Tri-grams can do the job, but they are computationally expensive.

● TF-IDF

The TF-IDF method is used to weigh a word in any text and assign value to it based on how many times it appears in the text.
TF-IDF is a product of term frequency and inverse document frequency.

Term frequency:

- It is the number of times a word appears in a document.
- This can be estimated by counting the number of times a word appears in a document. The frequency may be adjusted based on the length of the document.
-

Inverse document frequency:

- This indicates how frequent or uncommon a term is over the full text collection.
- The closer it is to zero, the more prevalent the term.
- This metric may be computed by taking the total number of documents, dividing them by the number of documents that include the word in consideration, and then computing the logarithm.

Figure 2: Calculation of TF-IDF

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$$tf_{i,j} = \text{number of occurrences of } i \text{ in } j$$
$$df_i = \text{number of documents containing } i$$
$$N = \text{total number of documents}$$

i = word in consideration.
j = refers to the document in consideration.

● Word2Vec

Word2vec is a two-layer neural network that vectorizes words to analyze text. It takes a text corpus as input and produces a set of vectors in the form of feature vectors representing words in the corpus. While Word2vec is not a deep neural network, it converts text to a numerical format that deep neural networks can use.

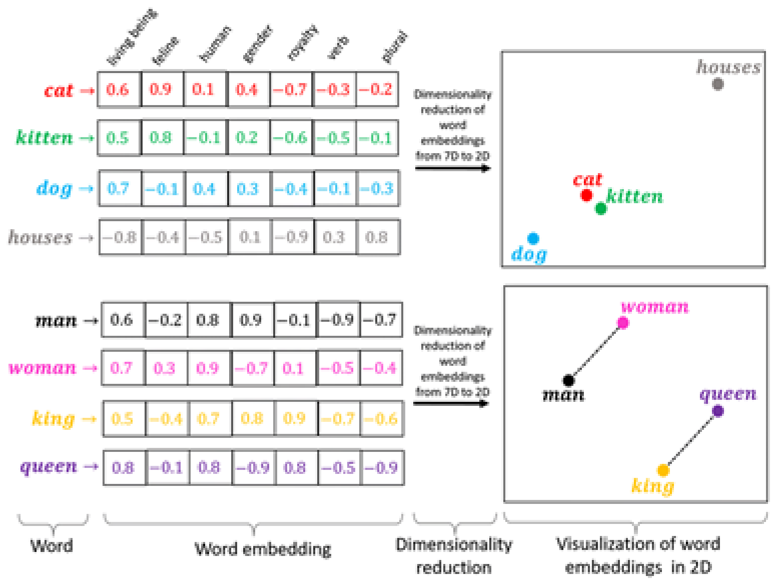
Word2Vec represents words as vectors of 32 or more dimensions.
Similar words will have similar word vectors.

Word2Vec has two competing algorithms namely CBOW and skip-gram which are used to create word vectors.

The dispersed representations of context of a sentence are integrated in the CBOW model to predict the target word in the middle. The Skip-gram model predicts the context and the sentence using the dispersed representation of the input word.

For example:

The quick brown fox ______ over the lazy dog.
Here jumps is the word that fits in the blank.
In CBOW the rest of the sentence is utilized to predict the word jumps.



	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8
man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D
Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

Figure 3: Word representations and their similarity

In the case of skip-gram the word jumps is used to get the context of the sentence.

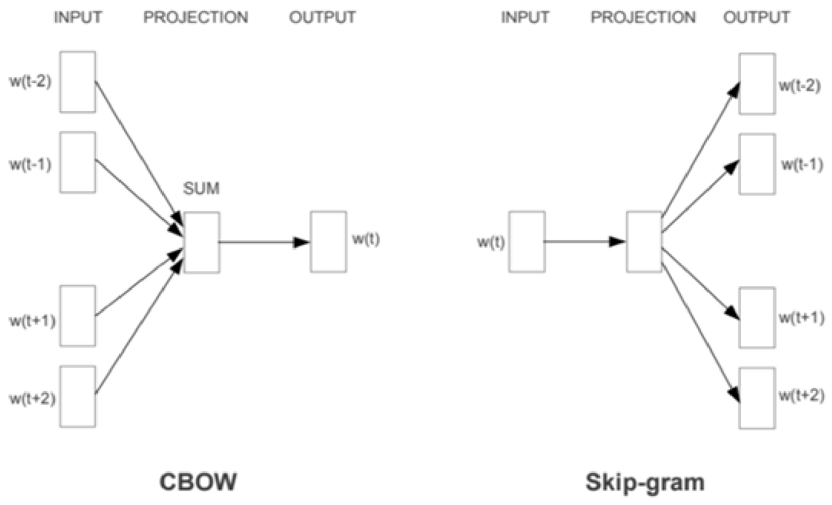


Figure 4: CBOW and Skip-gram architectures

w represents the word with its index in the sentence

• SVM

Support Vector Machines (SVM) is a supervised machine learning algorithm used for classification and regression problems.

In SVM algorithm, we plot each data item as a point in an n-dimensional space where n is the number of features, and so the value of each feature corresponds to the value of a particular coordinate for a given data point.

Then the objective of the SVM algorithm is to find a hyperplane in this n-dimensional space such that the hyperplane distinctly differentiates the data points with respect to their classes with maximum margins.

The data points closest to the hyperplane which affects the position and orientation of the hyperplane are known as support vectors

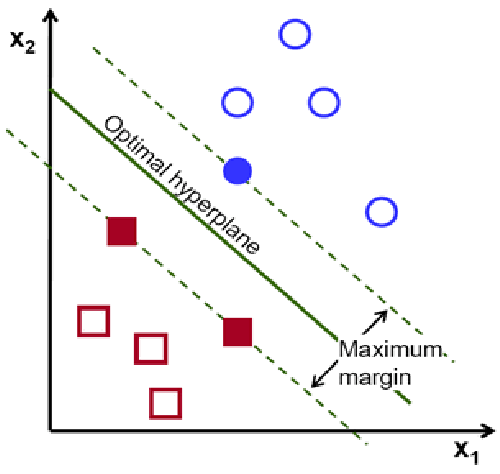


Figure 5: SVM Depiction

● XGBOOST

XGBoost stands for e**X**treme **G**radient **B**oosting.
XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.
The XGBoost library implements the gradient boosting decision tree algorithm.

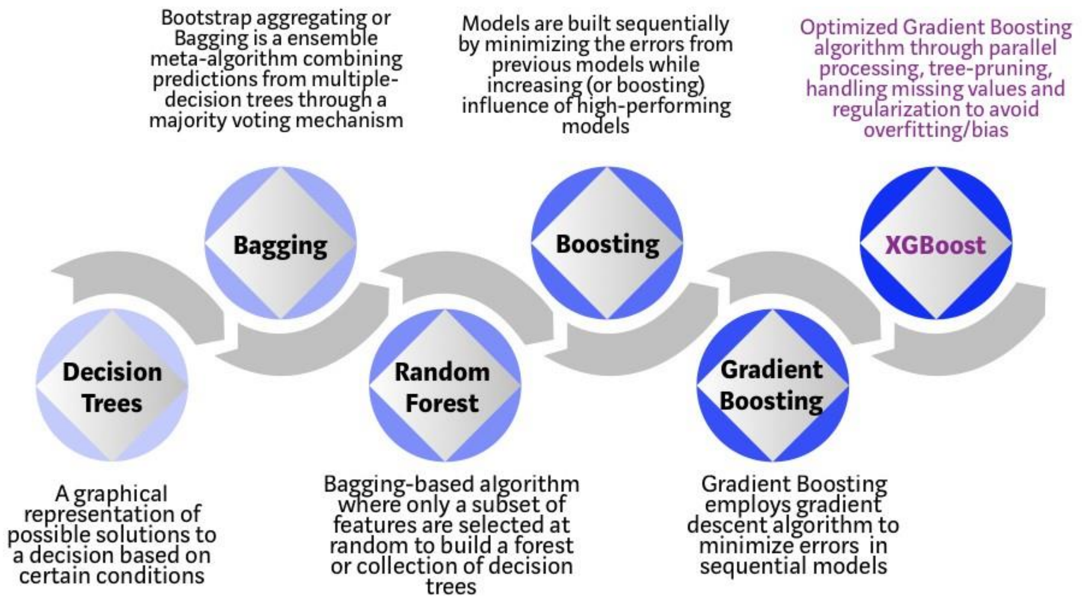

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

Figure 6. XGBoost Evolution.

Advantages:

1. A wide range of applications: Can be used to solve regression, classification, ranking, and user-defined prediction problems.
2. Portability: Runs smoothly on Windows, Linux, and OS X.
3. Languages: Supports all major programming languages including C++, Python, R, Java, Scala, and Julia.
4. Cloud Integration: Supports AWS, Azure, and Yarn clusters and works well with Spark, and other ecosystems.

● LOGISTIC REGRESSION

Logistic Regression is a Machine Learning algorithm which is used for classification problems. It is a predictive analysis algorithm based on the concept of probability.

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumor Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

Sigmoid Function:

Sigmoid function is used to map predicted values of the probabilities, it maps any real value into another value between 0 and 1.

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

Formula of the sigmoid function

Hypothesis Representation:

The hypothesis of logistic regression tends it to limit the cost function between 0 and 1.

$$0 \leq h_\theta(x) \leq 1$$

Logistic regression hypothesis expectation

Cost Function:
The cost function represents optimization objectives i.e., we create a cost function and minimize it so that we can develop an accurate model with minimum error.

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

The Cost function of Logistic regression

- ## CNN

In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most applied to analyze visual imagery.
CNN works on the technique called Convolution. Convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other. Convolutions help in reducing the dimensions of the input data into a form that is easier to process, without losing features that are critical for getting a good prediction.

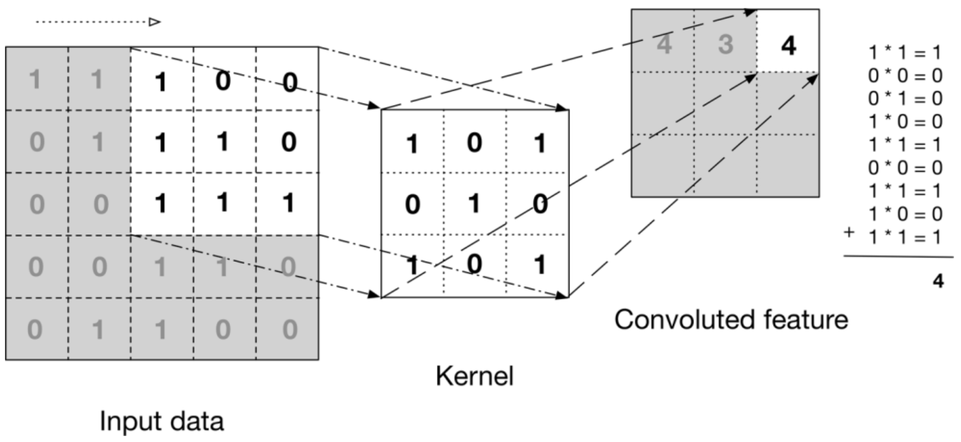


Figure 7. CNN-1

The above image shows what a convolution is. We take a filter/kernel (3×3 matrix) and apply it to the input data to get the convolved feature. This convolved feature is passed on to the next layer.

CNNs are composed of multiple layers of nodes which represent artificial neurons. Artificial neurons as the name suggests are a rough imitation of their biological counterparts. In a single neuron a mathematical function along with trained weights is applied onto the data and the result is passed through an activation function to get the output, which in turn is the input for the next layer of nodes/neuron.

A visual depiction of this process is shown below: -

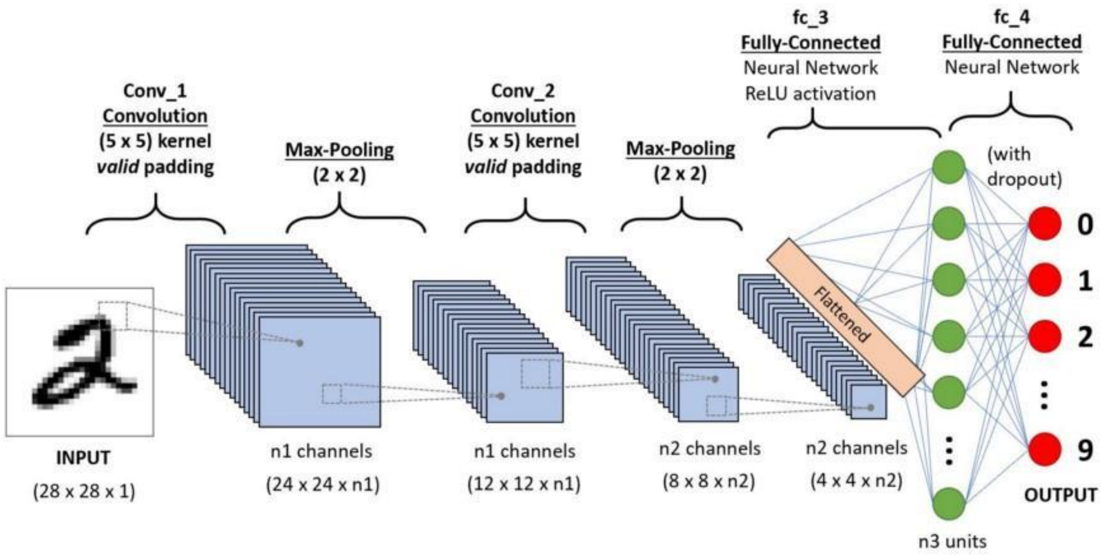


Figure 8: CNN Depiction

With the help of a loss function and backtracking these variable weights of each node are trained to minimize the loss.

- ## BiLSTM

Bidirectional LSTMs are a kind of LSTM that may be used to increase model performance in sequence classification issues. On the input sequence, bidirectional LSTMs train two LSTMs instead of one. The first is based on the original input sequence, while the second is based on a reversed replica of the original input sequence. This can give the network more context and help it understand the problem faster and more thoroughly.

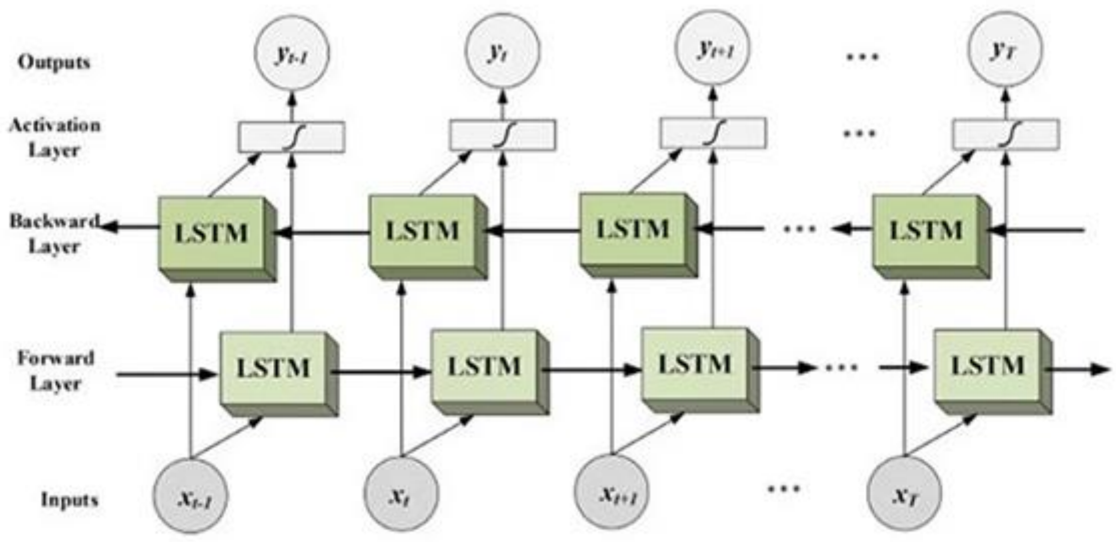


Figure 9 : biLSTM Depiction

2.2 Technologies

- Google trans API
- Google Colab & Jupyter Notebook
- MS Excel
- Language - Python 3.x
- Libraries used
    - Pandas
    - Numpy
    - Indic-trans
    - Scikit-learn
    - Fasttext
    - Gensim
    - TensorFlow & Keras

3. METHODOLOGY & IMPLEMENTATION

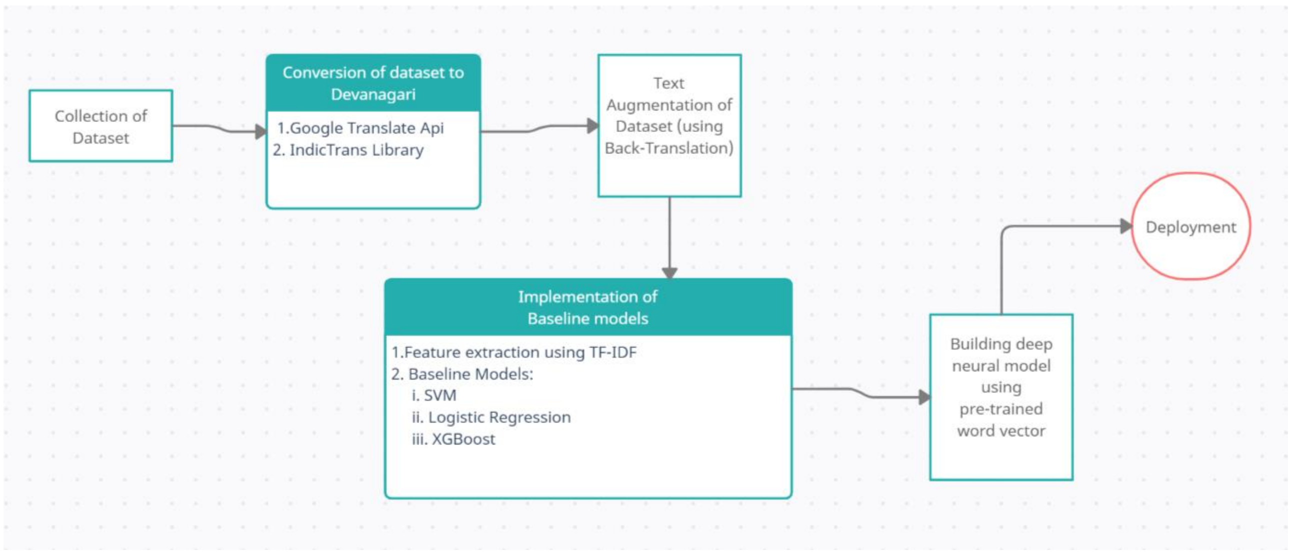

Collection of Dataset
Conversion of dataset to Devanagari
1.Google Translate Api
2. IndicTrans Library
Text Augmentation of Dataset (using Back-Translation)
Implementation of Baseline models
1.Feature extraction using TF-IDF
2. Baseline Models:
i. SVM
ii. Logistic Regression
iii. XGBoost
Building deep neural model using pre-trained word vector
Deployment

Figure 10. Flow of Project

3.1 Collection of Data

Creating an authentic dataset was a limitation. Because of this we needed an already authenticated and published dataset for the same.

TRAC 2 dataset was chosen which was published by the European Language Resources Association in Language Resources and Evaluation Conference (2020).

<u>About the dataset</u>

The dataset consisted of 5,000 aggression-annotated data from social media each in Bangla (in both Roman and Bangla script), Hindi (Roman) and English for training and validation.

For our project, only Hindi corpus (written in roman script) is needed.

The Hindi comments were labelled into 3 categories

1. Covertly Aggressive (CAG): This label was given to comments which had a hint of hate speech or had some moderately bad words.

2. Overtly Aggressive (OAG): This label was given to comments which had conspicuous hate and had explicit abusive words.

3. Not Aggressive (NAG): The rest of the comments were labelled as non-aggressive

3.2 Conversion of data to Devanagari

The original TRAC 2 Hindi dataset had majority of the words of Hindi written in Roman script and the others were English words (which made the language of the dataset as Hinglish).

To apply a pre-trained word vector to this dataset, we needed to convert the dataset either in Hindi or in English (with the translation of the mismatched words).
As most of the words were Hindi (written in Roman script), it was better to convert the dataset into Devanagari script, wherein transliterating the Hindi words and translating the English words, so that pre-trained Hindi word vectors can be applied.

We used Google trans API to translate the dataset. however, google trans failed to translate the Hindi words written in Roman script. To transliterate these words indic-trans library was used.

3.3 Imbalanced dataset

The publicly available dataset was divided into training and testing datasets. it was seen that the dataset was imbalanced with respect to the class labels.

The distribution of the two datasets combined was as follows: -

(Train + Test)
OAG - 1037
CAG - 1121
NAG - 2823

As the imbalances will result in biased classification, hence it was needed to rectify it.
To balance the dataset, text augmentation approach was applied which not only helped in balancing the dataset but also increased the size.
There are different techniques for text augmentation which includes: -

- Synonym Replacement
- Random insertion
- Random Swap
- Random Deletion
- Back Translation

Out of these, back translation provided the best results for data augmentation, which did not change the meaning and context of the data wherein we translated the sentences into Hindi

Two options were available for back translation, either through iNLTK library (using get_similar_sentences function) and Google trans API. Out of these google trans provided with better results.

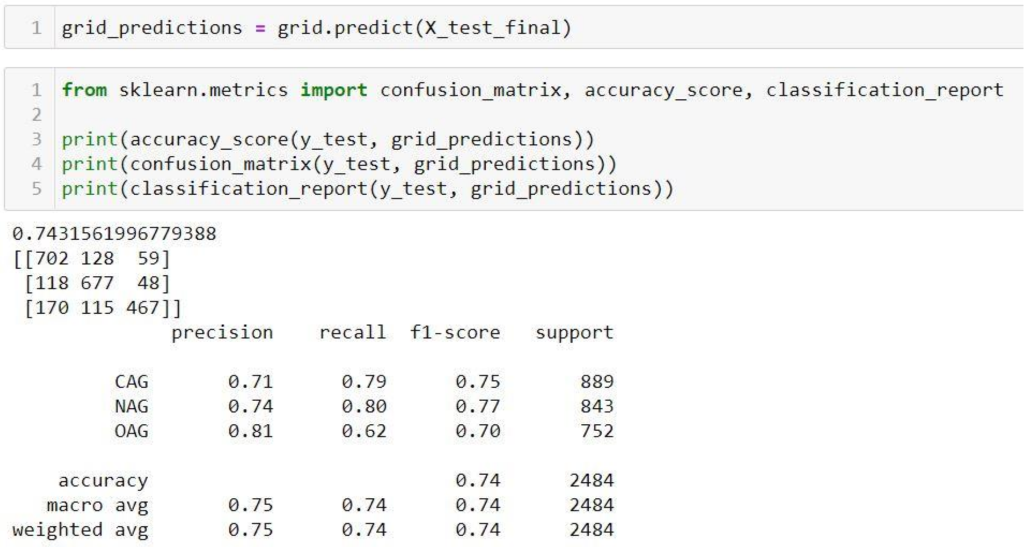
Final Distribution of dataset after text augmentation:

(Train + Test)
OAG - 3915
CAG - 3738
NAG – 2823

3.4 Implementation of Models

- **SVM**

  Accuracy - 74.31 %



```
grid_predictions = grid.predict(X_test_final)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

print(accuracy_score(y_test, grid_predictions))
print(confusion_matrix(y_test, grid_predictions))
print(classification_report(y_test, grid_predictions))
```

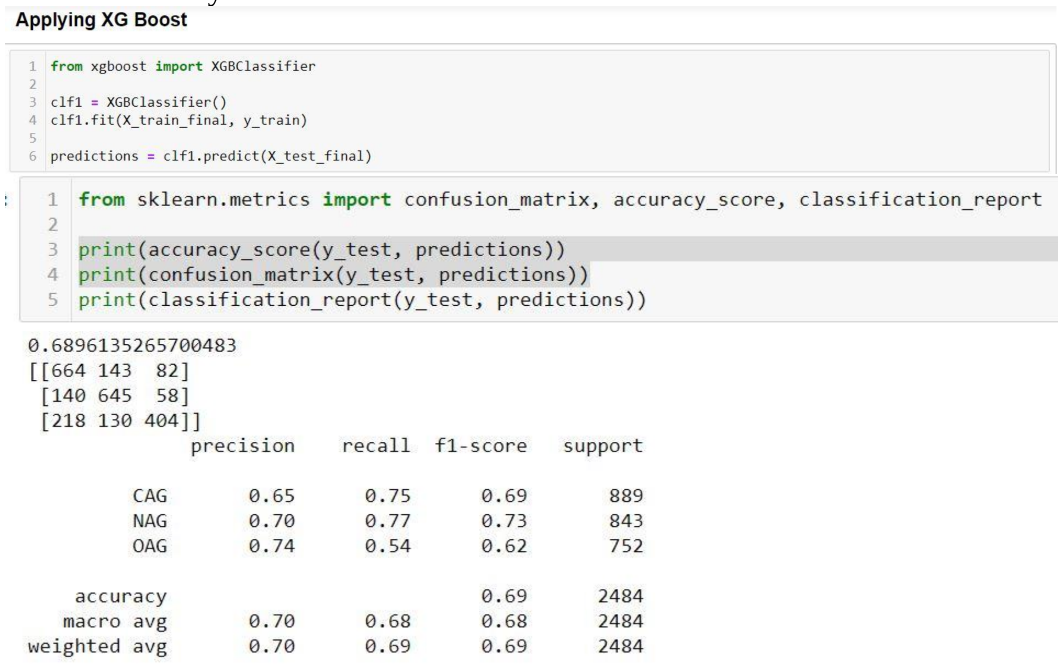
```
0.7431561996779388
[[702 128  59]
 [118 677  48]
 [170 115 467]]
              precision    recall  f1-score   support

         CAG       0.71      0.79      0.75       889
         NAG       0.74      0.80      0.77       843
         OAG       0.81      0.62      0.70       752

    accuracy                           0.74      2484
   macro avg       0.75      0.74      0.74      2484
weighted avg       0.75      0.74      0.74      2484
```

- **XGBoost**

  Accuracy - 68.96 %



Applying XG Boost

```
from xgboost import XGBClassifier

clf1 = XGBClassifier()
clf1.fit(X_train_final, y_train)

predictions = clf1.predict(X_test_final)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

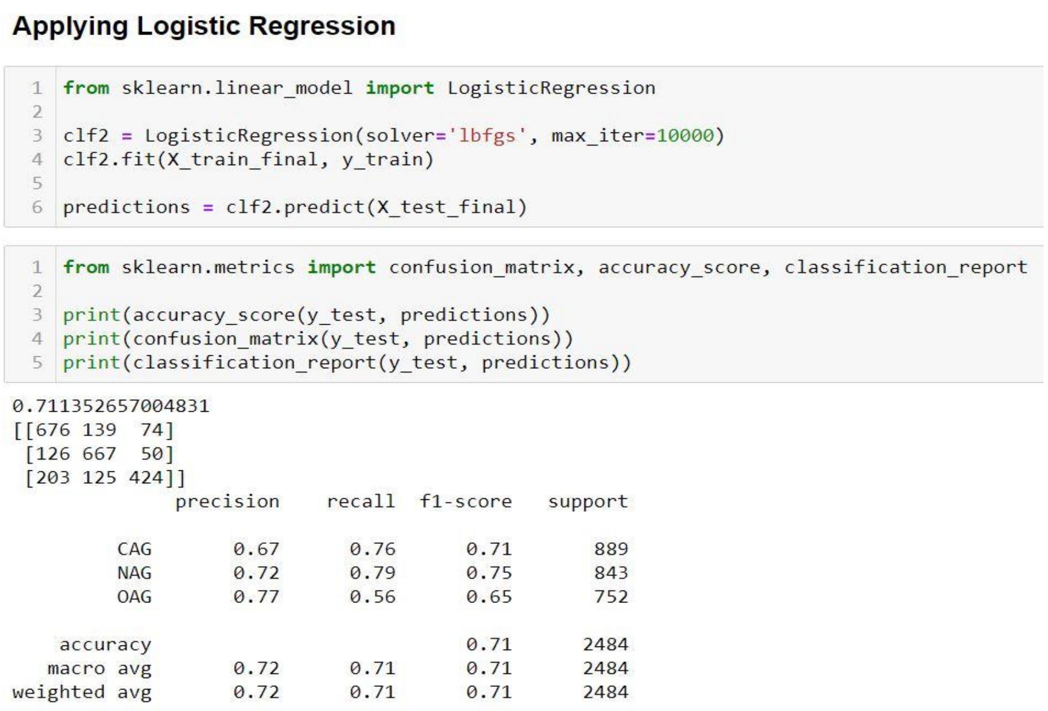
```
0.6896135265700483
[[664 143  82]
 [140 645  58]
 [218 130 404]]
              precision    recall  f1-score   support

         CAG       0.65      0.75      0.69       889
         NAG       0.70      0.77      0.73       843
         OAG       0.74      0.54      0.62       752

    accuracy                           0.69      2484
   macro avg       0.70      0.68      0.68      2484
weighted avg       0.70      0.69      0.69      2484
```

- **Logistic Regression**

  Accuracy - 71.13 %



Applying Logistic Regression

```
from sklearn.linear_model import LogisticRegression

clf2 = LogisticRegression(solver='lbfgs', max_iter=10000)
clf2.fit(X_train_final, y_train)

predictions = clf2.predict(X_test_final)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

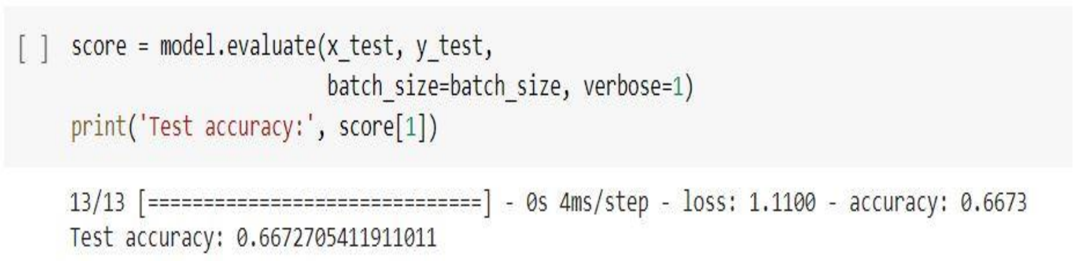
```
0.711352657004831
[[676 139  74]
 [126 667  50]
 [203 125 424]]
              precision    recall  f1-score   support

         CAG       0.67      0.76      0.71       889
         NAG       0.72      0.79      0.75       843
         OAG       0.77      0.56      0.65       752

    accuracy                           0.71      2484
   macro avg       0.72      0.71      0.71      2484
weighted avg       0.72      0.71      0.71      2484
```

- **BOW**

  Accuracy - 66.73 %

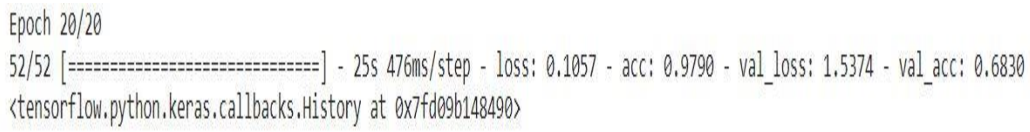


```
[ ]  score = model.evaluate(x_test, y_test,
                            batch_size=batch_size, verbose=1)
     print('Test accuracy:', score[1])
```

```
13/13 [==============================] - 0s 4ms/step - loss: 1.1100 - accuracy: 0.6673
Test accuracy: 0.6672705411911011
```

- **CNN**

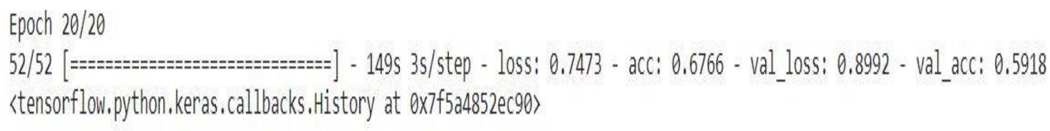
  Accuracy - 68.30 %



```
Epoch 20/20
52/52 [==============================] - 25s 476ms/step - loss: 0.1057 - acc: 0.9790 - val_loss: 1.5374 - val_acc: 0.6830
<tensorflow.python.keras.callbacks.History at 0x7fd09b148490>
```

- **BiLSTM**

  Accuracy - 59.18 %



```
Epoch 20/20
52/52 [==============================] - 149s 3s/step - loss: 0.7473 - acc: 0.6766 - val_loss: 0.8992 - val_acc: 0.5918
<tensorflow.python.keras.callbacks.History at 0x7f5a4852ec90>
```

4. RESULTS

Methods	**Accuracy %**
1. SVM	74.31 %
2. XGBoost	68.96 %
3. Logistic Regression	71.13%
4. BOW	66.73%
5. BiLSTM	59.18%
6. CNN	68.30%

5. CONCLUSION and Future Work

We introduced a machine learning model to classify social media Hinglish text based on aggression. The data in the Hinglish language is translated and transliterated into Hindi text on which ML algorithms are applied. It is difficult to apply ML models on Hindi text data because of varied spellings and grammar due to its different dialects.

It is observed that the accuracy achieved by conventional ML algorithms is more than DNN (Deep Neural Networks). Possible future enhancements in the datasets, word embeddings, and model architecture ought to further increase the accuracy.

6. REFERENCES

1. TRAC 2 dataset, {Bhattacharya, Shiladitya and Singh, Siddharth and Kumar, Ritesh and Bansal, Akanksha and Bhagat, Akash and Dawer, Yogesh and Lahiri, Bornini and Ojha, Atul Kr.}, https://www.aclweb.org/anthology/2020.trac2-1.25

2. Jason Wei and Kai Zou, 2019. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks, arXiv:1901.11196

3. Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. 2018. Qanet: Combining local convolution with global self-attention for reading comprehension. arXiv:1804.09541

4. Md Abul Bashar, Richi Nayak, Nicolas Suzor and Bridget Weir, 2020. Misogynistic Tweet Detection: Modelling CNN with Small Datasets. arXiv:2008.12452

5. Ai4Bharat pre trained Hindi Word Embeddings: https://github.com/AI4Bharat/indicnlp_corpus#word-embeddings