Data Glacier

# Advance NLP: Hate Speech detection using Transformers (Deep Learning)

**Group name: VerbalVigilantes**

**Team member names:** Zohra Bouchamaoui

**Email:** Zohra.bouchamaoui@outlook.com

**Country :** United Kingdom

**Company:** Data Glacier

**Batch code:** LISUM21

**Specialisation:** NLP

## PROBLEM DESCRIPTION

Hate speech is defined as any form of verbal, written, or behavioural communication that uses derogatory or discriminatory language to insult or attack an individual or a group based on attributes such as religion, ethnicity, nationality, race, colour, ancestry, gender, or other identity factors. In this project, our objective is to design a machine learning model, utilising Python, that can accurately detect instances of hate speech.

Hate speech detection typically falls under the umbrella of sentiment classification. To train a model capable of discerning hate speech in a given text, we will utilise a dataset commonly used for sentiment classification. Specifically, for this task, we will train our hate speech detection model using Twitter data, with the aim of identifying tweets that contain hate speech.

## DATA CLEANSING AND TRANSFORMATION

1. **Data Loading:** The train and test datasets are loaded using the pd.read_csv() function from the pandas library. The file paths for the datasets are provided as arguments to the function.
2. **Data Exploration:** We conducted some initial exploratory analysis to understand the data. The following steps are performed:
    - We displayed the first few rows of the train and test datasets using the head() function to get a brief look at the data.
    - We calculated the basic statistics of the train dataset using the describe() function to get information like count, mean, standard deviation, minimum, and maximum values for numerical columns.
3. **Data Preprocessing:**

- Text Preprocessing: A function named preprocess_text() is defined to pre-process the text data in the "tweet" column. The function applies the following steps:
    - Convert the text to lowercase using the lower() function.
    - Remove URLs using regular expressions (re.sub() function) to replace URLs starting with "http", "www", or "https" with an empty string.
    - Remove special characters and numbers using regular expressions to keep only alphabets and spaces.
    - Remove stopwords using the NLTK library's stopwords corpus. Stopwords are common words like "the", "is", "and" that do not contribute much to the overall meaning of the text.
    - Return the pre-processed text.

4. **Applying Data Preprocessing:**
    - The preprocess_text() function is applied to the "tweet" column of both the train and test datasets using the apply() function from pandas. The pre-processed text is then stored in a new column named "clean_text" in both datasets.
5. **Data Visualization:**
    - The length distribution of tweets in the train dataset is visualized using a histogram using the hist() function from matplotlib.
    - The word frequency distribution of the tweets is visualized using a bar chart using the bar() function from matplotlib.
    - The sentiment distribution of the tweets is also visualized using a bar chart.
    - The hashtag analysis is performed by extracting hashtags from tweets using regular expressions and counting their occurrences. The top 10 most used hashtags are visualized using a bar chart.
6. **Feature Engineering:**

- TF-IDF Vectorization: The TfidfVectorizer function from scikit-learn is used to convert the pre-processed text data into a numerical representation. The fit_transform() function is then applied to the "clean_text" column of the train dataset in order to obtain the TF-IDF feature matrix. The transform() function is then applied to the "clean_text" column of the test dataset to obtain the corresponding TF-IDF feature matrix.
- Bag-of-Words (BoW) Representation: The CountVectorizer class from scikit-learn is used to convert the pre-processed text data into a bag-of-words representation. The fit_transform() function is applied to the "clean_text" column of the train dataset to obtain the BoW feature matrix. The transform() function is then applied to the "clean_text" column of the test dataset to obtain the corresponding BoW feature matrix.

## MODEL BUILDING AND HYPERPARAMETER TUNING

1. **Support Vector Machine (SVM) Model:**
   - Created an SVM model with TF-IDF vectorization using scikit-learn's TfidfVectorizer and SVC.
   - Utilized a pipeline to combine TF-IDF vectorization and SVM classifier.
   - Performed a grid search for hyperparameter tuning, exploring C values and linear/rbf kernels.
   - Evaluated the SVM model before tuning with an accuracy of approximately 95.8%.
   - After hyperparameter tuning, accuracy improved to 95.6%, with precision of 78.7% and recall of 57.7% for class 1.
2. **Recurrent Neural Network (RNN) Model with Word Embeddings:**

- Designed an RNN model with word embeddings to capture sequential patterns using Keras.
- Utilised pre-trained GloVe word embeddings for text representation.
- Evaluated the RNN model before tuning with an accuracy of around 95.2%.
- After hyperparameter tuning, accuracy improved to 95.9%, with precision and recall of 78.7% and 57.7%, respectively, for class 1.

3. **XGBoost Model wit Bag-of-Words:**
    - Developed an XGBoost model with Bag-of-Words (BoW) representation using scikit-learn's CountVectorizer and XGBClassifier.
    - Assessed the XGBoost model before tuning, achieving an accuracy of about 95.1%.
    - After hyperparameter tuning, the model's accuracy remained approximately the same at 95.1%, with precision of 74.6% and recall of 48.2% for class 1.

## MODEL PERFORMANCE COMPARISON

- The SVM and RNN models showed significant improvements in performance after hyperparameter tuning, with balanced precision and recall for class 1 (hate speech).
- The RNN model slightly outperformed the SVM model in terms of accuracy and F1-score after tuning, indicating better hate speech classification.

## CONCLUSION

The project addressed hate speech detection using machine learning to promote safer online communities using three models: SVM with TF-IDF, RNN with word embeddings, and XGBoost with Bag-of-Words.

Before hyperparameter tuning, the SVM model achieved 95.8% accuracy, while RNN had an accuracy of 95.2%, and XGBoost an accuracy of 95.1%.

After hyperparameter tuning, SVM improved to 95.6% accuracy, RNN to 95.9%, and XGBoost showed almost no improvements with a constant accuracy of 95.1%. SVM and RNN exhibited balanced precision and recall for class 1, with F1-scores of 0.67. RNN outperformed SVM in accuracy and F1-score after hyperparameter tuning due to its ability to capture sequential patterns and dependencies in text. In conclusion, RNN with word embeddings showed the best performance in hate speech classification after hyperparameter tuning.