Deliverables

Name : Noura Alsahli
Email: fnouraa@gmail.com
Country: Saudi Arabia
College/company: Data glacier
Specialization: NPL

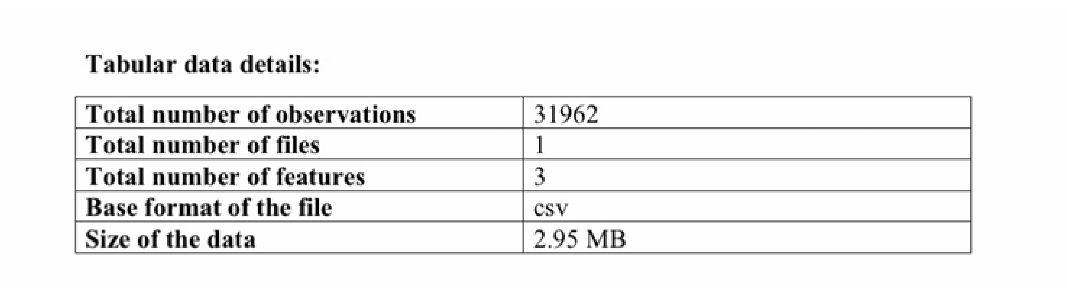
Problem Description & Business Understanding:

The term hate speech is understood as any type of verbal, written or behavioural communication that attacks or uses derogatory or discriminatory language against a person or group based on what they are, in other words, based on their religion, ethnicity, nationality, race, colour, ancestry, sex or another identity factor. In this problem, We will take you through a hate speech detection model with Machine Learning and Python.

Hate Speech Detection is generally a task of sentiment classification. So for training, a model that can classify hate speech from a certain piece of text can be achieved by training it on a data that is generally used to classify sentiments. So for the task of hate speech detection model, We will use the Twitter tweets to identify tweets containing Hate speech.

Project lifecycle along with deadline:

• week 1: week of 19 March 2024 - Problem Understanding and Data Cleaning
• week 2: week of 26 March 2024 - Data preprocessing and data cleaning with different methods
• week 3: Week of 2 April 2024 - Making recommendations based on data analysis and selecting the model that will be applied after.
• Week 4: week of 9 April 2024 October 2022 - Create a presentation for the past week
• Week 5: Week of 16 April 2024 - EDA presentation and proposed technique for modeling.
• Week 6: Week of 23 April 2024 - Select a base model and then try a different model and compare results.
• Week 7: week of 20 April 2024 - Write a final report for the project and make a presentationte



Tabular data details:

Total number of observations	31962
Total number of files	1
Total number of features	3
Base format of the file	csv
Size of the data	2.95 MB

Data Preprocessing: Text Cleaning

In this section, I outlined the data preprocessing techniques applied to my text data.

Lowercasing

I begin by converting all words to lowercase. For instance, the words “Racism” and “racism” are treated as equivalent. By doing so, I avoided representing them as separate words in the vector space model, which would increase the dimensionality.

Remove Punctuation

Punctuation marks are not essential for our analysis. Therefore, I remove them using regular expressions.

Remove URLs

Since I am working on a hate speech detection application, I focus solely on the text content. URLs are irrelevant for my purpose, so I removed them.

Remove @tags

@tags are typically used to mention specific individuals. However, they do not pertain to my application. Hence, I eliminated them using regular expressions.

Remove Special Characters

Special characters such as [!”#$%&’()*+,./:;<=>?@[]^_`{|}~] lack meaningful information. I removed them from the text using Python’s `isalnum` method.

preprocessing operations:

In this section, we carry out the necessary preprocessing steps for our text data. Here are the key operations:

Tokenization:

Tokenization involves breaking raw text into smaller chunks. In this case, the text data is organized into paragraphs. To tokenize it, I utilize the NLTK word_tokenize library. These tokens play a crucial role in understanding context and developing natural language processing (NLP) models. By analyzing the sequence of words, tokenization helps interpret the meaning of the text.

Removing Stop Words:

Stop words such as “a,” “is,” “the,” and “are” don’t carry significant meaning and are unnecessary for building a hate speech detection application. To remove stop words from a sentence, I followed these steps:

First, I divided the text into words (as done during tokenization).
Next, I checked if each word exists in the list of NLTK-provided stop words. If it does, I removed it.
I achieved this by importing the StopWords collection from NLTK.

Lemmatization:

Lemmatization groups together different inflected forms of a word so that they can be analyzed as a single item. Unlike stemming, which simply truncates words, lemmatization considers context. It links words with similar meanings to their root form.
For example, the words “intelligently” and “intelligence” both convert to the root form “intelligent.”

Word Cloud:

**A word cloud visually represents text data.
It is commonly used to depict keyword metadata on websites or to visualize
free-form text.
Each tag (usually a single word) is displayed with font size or color indicating its importance.**

Feature Extraction:

**I used the Term Frequency-Inverse Document Frequency (TF-IDF) model after preparing the dictionary.
Specifically, I selected the 2000 most frequent words from the dictionaries
for each category (Hate/Free Speech)
in the entire dataset.
Each word count vector captures the frequency of these 2000 words across the entire dataset file.**

Data Splitting:

**In this step, I divide the data into training and test sets.
Typically, an 80% split is used for training, while the
remaining 20% is reserved for testing.
Data splitting is crucial for creating robust data science models.**

Model Construction In this section, I constructed the CNN with LSTM Model using Tensorflow.

CNN with LSTM The CNN LSTM architecture combines Convolutional Neural Network (CNN) layers for feature extraction on input data with Long Short-Term Memory (LSTM) units to facilitate sequence prediction. Originally known as a Long-term Recurrent Convolutional Network (LRCN) model, I will use the more general term "CNN LSTM" to describe LSTMs that utilize a CNN as their front end.

Evaluating Results and Discussion:

In this chapter, I assessed my results and establish evaluation criteria for measuring the performance of the top classification model.

Evaluation Criteria:

**During the training process, I employed the confusion matrix to evaluate the classification models.
The confusion matrix is a tabular representation that compares predicted outcomes with actual outcomes.
It serves as a valuable tool for assessing a classification model's performance on a given test dataset.**

In addition to the overall classification accuracy, I considered several other essential metrics for evaluating the classification models. These metrics include:

True Positive Rate (TPR): Also known as sensitivity or recall, TPR measures the proportion of actual positive cases correctly predicted by the model.

True Negative Rate (TNR): Also called specificity, TNR represents the proportion of actual negative cases correctly predicted by the model.

False Positive Rate (FPR): FPR quantifies the rate of false alarms, indicating how often the model incorrectly predicts a positive outcome when the actual outcome is negative.

False Negative Rate (FNR): FNR reflects the rate of missed positive cases, indicating how often the model fails to predict a positive outcome when it is actually positive.

Precision: Precision assesses the accuracy of positive predictions made by the model.

F1 Score: The F1 score balances precision and recall, providing a single metric that considers both false positives and false negatives.

**Misclassification Rate: This metric quantifies the overall error rate of the model.
These metrics collectively offer a comprehensive view of our classification model's performance.**

Creating a Web Application for the Trained Model

Model Training:

I begin by using the Twitter hate speech dataset to build a prediction model capable of accurately classifying hate speech.
the chosen model is a Convolutional Neural Network (CNN).
After training the model, I faced the challenge of persisting it for future use without the need for retraining.

Model Persistence:

To achieve this, I created a pickle file containing our trained model. This file allows to store the model's parameters and architecture, making it accessible for future predictions without retraining.
By saving the model as a pickle file, I ensured its longevity and ease of use.

Web Application Development:

the next step involves developing a web application. The application consists of a straightforward web page with a form field.
Users can input a message into the form field and submit it to the web application.
Upon submission, the application processes the message and renders the result, indicating whether it contains hate speech or qualifies as free speech.
In summary, the web application provides a user-friendly interface for evaluating messages and determining their hate speech status.

Conclusion:

The objective of this project was to identify effective methods and configurations for detecting hate speech and free speech on Twitter. While the model is not error-free, some misclassifications may still occur. Looking ahead, I planned to enhance the work by incorporating the following techniques:

Temporal Convolutional Network (TCN): I aim to explore TCNs, which can capture temporal dependencies in sequential data, potentially improving our model's performance.

Random Multimodel Deep Learning (RMDL): By leveraging RMDL techniques, I hope to further enhance the classification capabilities.
In summary, the ongoing efforts will contribute to more robust and accurate hate speech detection in online platforms.