Automatic recognition of violence-related content on social media

Realized by :

Boufous Bouchra

Charaf Oualid

El-Ouardi Yassine

Supervised by :

Pr. Hajar Mousannif

Master : Data Science

Introduction

The project "Automatic Recognition of Violence-Related Content on Social Media (Text Mining)" focuses on using text mining techniques to detect and classify violent content in messages posted on social media. The main objective of this project is to develop a predictive model that can identify potentially violent messages, enabling quick and effective intervention to prevent harmful online behaviors.

The project aims to apply natural language processing (NLP) techniques to analyze and understand messages posted on social media. Specifically, it focuses on detecting violence-related content, such as hate speech, threats, bullying, and offensive messages. The goal is to develop an automatic classification model that can accurately identify toxic and non-toxic messages.

1. Data Collection

In this step, relevant data is collected from various sources, such as social media platforms, online forums, or public databases. The data include textual messages, associated metadata, and toxicity or no toxicity annotations

Datasets Links

Description	Link Data
An Arabic Levantine Twitter Dataset for Misogynistic Language	https://drive.google.com/file/d/1mM2vnjsy7QfUmdVUpKqHRJjZyQobhTrW/view
A Levantine Twitter Dataset for Hate Speech and Abusive Language	https://github.com/Hala-Mulki/L-HSAB-First-Arabic-Levantine-HateSpeech-Dataset
Abusive Language Detection on Arabic Social Media (Twitter)	http://alt.qcri.org/~hmubarak/offensive/TweetClassification-Summary.xlsx
Abusive Language Detection on Arabic Social Media (Al Jazeera)	http://alt.qcri.org/~hmubarak/offensive/AJCommentsClassification-CF.xlsx
arabic-toxic-dataset	https://www.kaggle.com/datasets/assiri/arabic-toxic-dataset
Arabic-Offensive-Multi-Platform-SocialMedia-Comment-Dataset	https://github.com/shammur/Arabic-Offensive-Multi-Platform-SocialMedia-Comment-Dataset/blob/master/data/Arabic_offensive_comment_detection_annotation_4000_selected.xlsx
Hate-Speech-Detection-on-Arabic-Tweets	https://github.com/aehabV/Hate-Speech-Detection-on-Arabic-Tweets/blob/main/Dataset/subTask-B-HSNOT_HS.csv

2. Dataset Construction

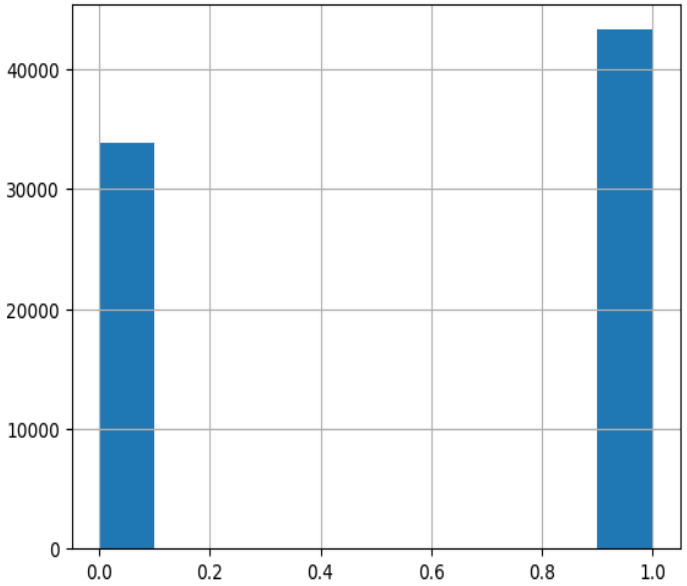
After collecting the data, we merged it into a single file and the preprocessed data is divided into two main columns: the "**Text**" column containing the messages posted on social media, and the "**Toxic/Not Toxic**" column indicating whether the message is toxic(**1**) or non-toxic(**0**).

The dataset consists of **77,176** instances.

```
fusion = pd.read_csv("fusionne.csv")
fusion
```

	Text	Toxic/Not Toxic
0	...-يبغي التنبيه على ان السعودية تستخدم صواريخ جو	1
1	...أمريكا قتلت بالامس معوق رفض رفع يديه فماذا تري	1
2	...هذا الشخص هو من كان مؤيد لاحتلال العراق وضرب ا	1
3	...الى جمال ريان مذيع الجزيرة نحن من رعاك فى المه	1
4	...عزم لكفالة\nخيبة الأمل ليست تشاؤما ولا تقولا ٠	0
...	...	...
77171	...أستاذنا العزيز بلال فضل يحزنني أن يتم حذف حلقا	0
77172	... برنامج يدعو للتفكير أكيد يمنعوه لأنهم خوافون،	0
77173	...أزهى_عصور_المسخره هل ضيعت "شيما" محافظ بورس#	1
77174	الله يلعنك و يلعن اللى جابوك يا رمه جايفه	1
77175	أنت رجل حيوان	1

77176 rows × 2 columns



3. Data Preprocessing

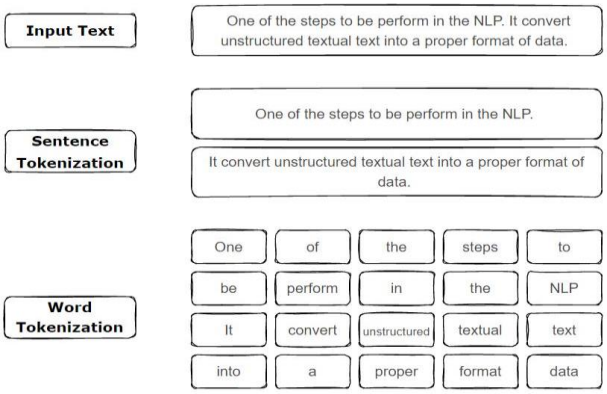
The collected data needs to be cleaned and prepared for analysis. This involves removing unwanted characters, normalizing texts, removing stop words, and other common preprocessing techniques to enhance data quality.

Tokenization

Tokenization is the process of breaking down a text into smaller units, such as words or subwords. It is an essential step in natural language processing as it helps to extract meaningful information from the text.

The simple_word_tokenize method in CamelTools follows a simple approach for tokenization. It splits the text based on whitespace characters and punctuation marks, separating words and other tokens.

By using this tokenization method, the text is split into individual Arabic words, which allows for further analysis and processing on a word-level basis.



The preprocessing function

- ✓ **Converting non-string data to string:** If the data is not already a string, it is converted to a string.
- ✓ **Removing English words**: Any English words present in the text are removed.
- ✓ **Removing usernames**: Usernames starting with '@' are removed from the text.
- ✓ **Removing URLs**: Any URLs present in the text are removed.
- ✓ **Removing special characters**: All characters that are not Arabic letters are replaced with spaces.
- ✓ **Reducing sequences of repeated letters**: Sequences of repeated letters are reduced to a single letter.
- ✓ **Tokenizing text**: The text is tokenized into individual words.
- ✓ **Removing single-letter words**: Any words with a length of one character are removed.
- ✓ **Converting tokens back to a string**: The tokens are joined back together to form a preprocessed text string.

```python
# Function to preprocess text
def preprocess_text(text):
    # Convert non-string data to string
    if not isinstance(text, str):
        text = str(text)
    # Remove English words
    text = re.sub(r'[a-zA-Z]+', '', text)
    # Remove usernames
    text = re.sub(r'@\w+', '', text)
    # Remove URLs
    text = re.sub(r'http.?://[^\s]+[\s]?', '', text)
    # Remove special characters
    text = re.sub('[^ي-ا]', ' ', text)
    # Reduce sequences of repeated letters to a single letter
    text = re.sub(r'(.)\1+', r'\1', text)
    # Tokenize text
    text = simple_word_tokenize(text)
    # Remove single-letter words
    text = [word for word in text if len(word) > 1]
    # Convert tokens back into a string
    text = ' '.join(text)
    return text
```

4. Feature Extraction

Once the data has been preprocessed, the next step is to convert the text into a format that can be understood by the machine learning algorithm. This process is known as feature extraction. There are several techniques that can be used for feature extraction:

A. **TF-IDF (Term Frequency-Inverse Document Frequency)**

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document),
Now Lets jump into the example part of it:
Let's Consider these Three sentences:

1.He is a Good Boy

2. She is a Good Girl, and,

3. Both are Good Boy, and Girl, respectively.

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Now, We have Values for both, TF(Term Frequency) as well as IDF(Inverse Document Frequency) for each word, for each Sentence we have,

So,Finally the TF-IDF Value for each word wouldbe=TF(Value)*IDF(Value).

Let's Present TF-IDF Value of each word in a tabular form given below,

Sentence\Word	Word 1: Good TF-IDF Value	Word 2: Boy TF-IDF Value	Word 3: Girl TF-IDF Value
Sentence 1	1/2*Log (3/3) = 0	1/2*Log (3/2) = 0.088	0*Log (3/3) = 0
Sentence 2	1/2*Log (3/3) = 0	0*Log (3/2) = 0	1/2*Log (3/2) = 0.088
Sentence 3	1/2*Log (3/3) = 0	1/3*Log (3/2) = 0.058	1/3*Log (3/2) = 0.058

As a **Conclusion,** we can see that, the word “Good”, appears in each of these 3 sentences, as a result the Value of the word “Good” is Zero, while the Word “Boy” appears only 2 times, in each of these 3 sentences, As a results, we can see, in Sentence 1, the Value(Importance) of word “Boy” is more than the Word “Good”.As a result, we can see that, **TF-IDF,** gives **Specific Value** or **Importance** to each Word, in any paragraph, The terms with higher weight scores are considered to be more importance, as a result TF-IDF has replaced the concept of “**Bag-Of-Words**” which gives the Same Value to each word, when occurred in any sentence of Paragraph, which is **Major Disadvantage** of **Bag-Of-Words**.

```
In [65]: vect = TfidfVectorizer(max_features=500)
         X = vect.fit_transform(splittedSentences)
         vect.get_feature_names_out()
```

```
Out[65]: array(['ابن', 'ابو', 'ات', 'احد', 'احسن', 'احنا', 'اخر', 'اذا', 'اسرا',
                'اسم', 'اصلا', 'افضل', 'اقول', 'اكبر', 'اكثر', 'اكيد', 'ال', 'الا',
                'الاتحاد', 'الاخوان', 'الارض', 'الارهاب', 'الاسد', 'الاسلام',
                'الاسلامية', 'الاسلاميه', 'الان', 'البلد', 'البنات', 'التاريخ',
                'التحالف', 'التي', 'الثورة', 'الجزا', 'الجزيرة', 'الجزيره',
                'الجيش', 'الحرب', 'الحق', 'الحمد', 'الحيا', 'الحين', 'الخير',
                'الخلافة', 'الخليج', 'الخير', 'الدنيا', 'الدول', 'الدولة',
                'الدوله', 'الدين', 'الذي', 'الذين', 'الر', 'الرجال', 'الرجل',
                'السعودي', 'السعودية', 'السعوديه', 'السلام', 'السنة', 'السوري',
                'السورين', 'السيد', 'السيسي', 'الشباب', 'الشعب', 'الشعوب',
                'الشيطان', 'الشيعة', 'الصهاينة', 'الطا', 'العالم', 'العالمين',
                'العراق', 'العرب', 'العربي', 'العربية', 'العظيم', 'الغرب', 'الف',
                'الكل', 'الكلام', 'الكلب', 'الكليجا', 'الم', 'المجرم', 'المسلمين',
                'المصري', 'الملك', 'المنطقة', 'الموت', 'الموضوع', 'الناس', 'النصر',
                'النظام', 'اله', 'الهم', 'الوطن', 'الوكيل', 'الى', 'الي', 'اليمن',
                'اليهود', 'اليوم', 'ام', 'اما', 'امريكا', 'امك', 'امين', 'ان',
                'انا', 'انت', 'انتم', 'انتو', 'انتي', 'انك', 'انه', 'انها', 'انهم',
                'انو', 'اني', 'اهل', 'او', 'اول', 'اي', 'ايران', 'ايش', 'اين',
                'ايه', 'ايها', 'با', 'باسيل', 'بال', 'باله', 'بابع', 'بد', 'بدك',
```

```
print(X)
```

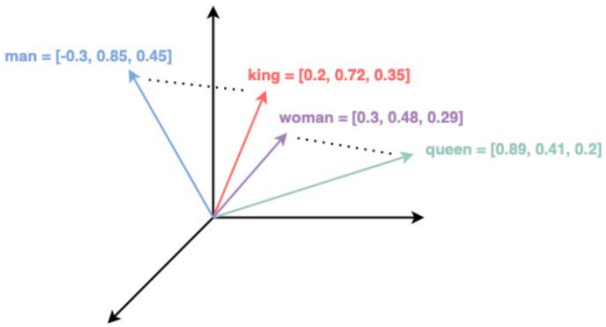
```
  (0, 102)      0.4137803319399968
  (0, 304)      0.6572356099043231
  (0, 283)      0.30906844405202233
  (0, 57)       0.39792276338598465
  (0, 110)      0.29255364702913533
  (0, 271)      0.2395281295668509
  (1, 394)      0.43389099359193195
```

B. Word Embeddings

Word embedding is a technique used in natural language processing (NLP) to represent words as dense numerical vectors. These vectors capture the semantic and syntactic information of words, allowing for the measurement of similarities and relationships between words.

Word embedding is often performed using algorithms such as **Word2Vec**, which learn from large text corpora to predict neighboring words given a specific word. By adjusting the model's parameters, word vectors are optimized so that words with similar contexts have similar vectors.

When words are represented as vectors, words with similar syntactic or semantic relationships are represented by vectors that are close to each other in the vector space.



$$\begin{bmatrix} W_{11} \\ W_{12} \\ \\ W_{1n} \end{bmatrix}_{W_1} + \begin{bmatrix} W_{21} \\ W_{22} \\ \\ W_{2n} \end{bmatrix}_{W_2} + \dots + \begin{bmatrix} W_{n1} \\ W_{n2} \\ \\ W_{nn} \end{bmatrix}_{W_n} = \begin{bmatrix} \frac{W_{11}+W_{21}+\dots+W_{n1}}{n} \\ \\ \frac{W_{1n}+W_{2n}+\dots+W_{nn}}{n} \end{bmatrix}_{D}$$

average word vectors

Sentence preparation:

Each text is split into individual words using the **split()** method, resulting in a list of words for each text. All these word lists are grouped into a list called **sentences**.

Word2Vec model creation:

The Word2Vec class from the **gensim** library is used to create the word embedding model.
The parameters for Word2Vec include:

- **sentences**: The prepared sentences from the previous step.
- **vector_size**: The dimensionality of the word vectors you want to obtain. In your case, you specified 100.
- **window**: The size of the context window for predicting neighboring words.
- **min_count**: The minimum number of occurrences a word must have to be considered in the model.

```
[4]: sentences = [text.split() for text in df['Text']]
     embedding_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1)
```

```
Entrée [5]: sentences

Out[5]:  'يبغي']]
         'التنبيه'
         'على'
         'ان'
         'السعودية'
         'تستخدم'
         'صواريخ'
         'جو'
         'ارض'
         'مزودة'
         'بتقنيات'
         'بوليس'
         'النهي'
         'عن'
         'المنكر'
         'في'
         'الجلد'
         'في'
         'الساحات'
```

Conversion of texts to word vectors:

the code converts texts to word vectors using the Word2Vec model. It defines a function called "**text_to_vector**" that iterates over each word in a text, checks if the word exists in the embedding model, and adds the corresponding vector to a list. After processing all the words, the function calculates the mean of the vectors to obtain an average vector representation of the text. If no words are found in the embedding model, a zero vector is created. This process allows for the conversion of texts into numerical representations that can be used for various natural language processing tasks.

```
def text_to_vector(text):
    vector = []
    for word in text.split():
        if word in embedding_model.wv:
            vector.append(embedding_model.wv[word])
    if vector:
        vector = np.mean(vector, axis=0)
    else:
        vector = np.zeros(embedding_model.vector_size)
    return vector
```

Applying the "text_to_vector" function to your dataframe:

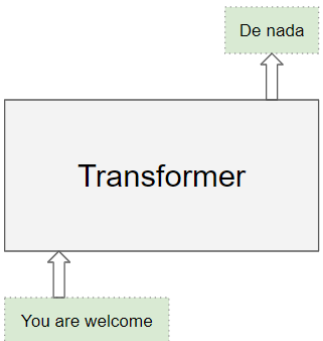
```
vector = text_to_vector(text)
print("Text:", text)
print("Vector:", vector)
```

```
Text: مريكا قتلت بالامس معوق رفض رفع يديه فماذا تريد من الشرطة المصرية تفعل عندما تعامل مع مسلحين معهم متفجرات
Vector: [-0.36112776  0.30494243  0.24558525 -0.4735253   0.24384551 -0.3850021
  0.2506579   1.0850585   0.13199401 -0.01537475 -0.02701964 -1.0646343
 -0.2907199   0.26681003  0.0011454  -0.5830876   0.1603785  -0.18098271
 -0.15568072 -0.74646485 -0.00845524 -0.09191674  0.53472453 -0.52977324
  0.20772298 -0.0582713  -0.4002066  -0.39734614 -0.34492794  0.27451724
  0.62213135 -0.07906406 -0.11100897 -0.49478194 -0.16031073  0.14285915
  0.17543772 -0.04585664 -0.16069268 -0.67187226  0.723916   -0.5203179
 -0.56017286  0.2660966   0.46549296 -0.23078915  0.05115719  0.08693587
 -0.27859375  0.3943416   0.4321132  -0.7685168   0.03478753 -0.38199148
 -0.40422627  0.34009734  0.4178826  -0.23654373 -0.15537012  0.07007518
 -0.20095162 -0.39382875  0.398158   -0.13022335 -0.33680424  0.58813936
  0.40047592  0.23796111 -0.4905924   0.46194935 -0.3342154  -0.0229991
  0.6700082   0.366135    0.22751537  0.17804162  0.11343076 -0.23936261
 -0.48840332  0.19389772 -0.40280995  0.14937289  0.13223748  0.18812509
  0.11646964 -0.12649877  0.42864403  0.36847085  0.5122186   0.3397115
  0.61455274 -0.06365775  0.24165216 -0.38985893  0.70568943  0.78354275
  0.45089304 -0.20301713 -0.0250103  -0.28659096]
```

C. **Transformers**

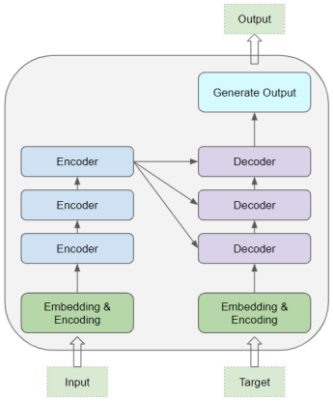
What is a Transformer

The Transformer architecture excels at handling text data which is inherently sequential. They take a text sequence as input and produce another text sequence as output. eg. to translate an input English sentence to Spanish.

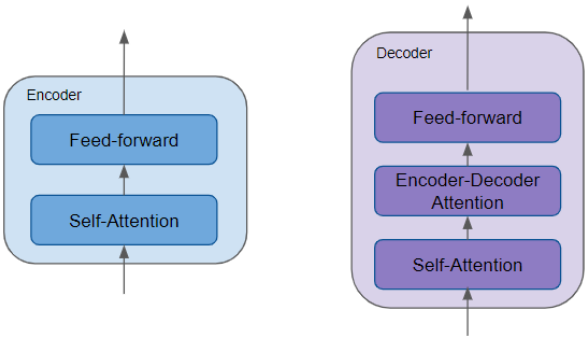


At its core, it contains a stack of Encoder layers and Decoder layers. To avoid confusion we will refer to the individual layer as an Encoder or a Decoder and will use Encoder stack or Decoder stack for a group of Encoder layers.

The Encoder stack and the Decoder stack each have their corresponding Embedding layers for their respective inputs. Finally, there is an Output layer to generate the final output.



All the Encoders are identical to one another. Similarly, all the Decoders are identical.



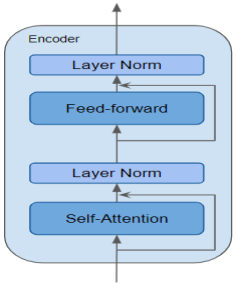
The Encoder contains the all-important Self-attention layer that computes the relationship between different words in the sequence, as well as a Feed-forward layer.

The Decoder contains the Self-attention layer and the Feed-forward layer, as well as a second Encoder-Decoder attention layer.

Each Encoder and Decoder has its own set of weights.

The Encoder is a reusable module that is the defining component of all Transformer architectures. In addition to the above two layers, it also has Residual skip connections around both layers along with two LayerNorm layers.

There are many variations of the Transformer architecture. Some Transformer architectures have no Decoder at all and rely only on the Encoder.



5. Model training

After extracting the features, you need to split the data into a **training set (80%)** and a **test set (20%)**. Then, you can train your model using various algorithms, including:

- ✓ Support Vector Machines (SVM)
- ✓ Random Forests
- ✓ Naive Bayes

SVM (Support Vector Machines):

- Supervised learning algorithm used for classification and regression.
- Finds an optimal hyperplane that separates data points of different classes.
- Maximizes the margin between the hyperplane and the nearest samples.
- Can use kernel functions to handle non-linear classification problems.
- Effective for small to medium-sized datasets.

```
#SVM
svm_model = SVC(verbose=1)
svm_model.fit(X_train, y_train)
y_preds = svm_model.predict(X_test)
svm_scores = svm_model.decision_function(X_test)
```

```
[LibSVM]
```

```
SVC
SVC(verbose=1)
```

Random Forest:

- Supervised learning algorithm used for classification and regression.
- Builds an ensemble of decision trees and combines their predictions.
- Each tree is built from a random subset of the training data.
- Final predictions are obtained through majority voting or averaging of tree results.
- Resistant to overfitting and suitable for large datasets.

```
# Random Forest
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_predf = rf.predict(X_test)

rf_pred_prob = rf.predict_proba(X_test)[:,1]
```

Naive Bayes:

- Supervised learning algorithm based on Bayes' theorem.
- Assumes conditional independence between features.
- Used for text and document classification.
- Fast to train and predict, even with large amounts of data.
- Can handle binary and multiclass classification problems.

```
# Naive Bayes
nb = GaussianNB()
nb.fit(X_train, y_train)
y_predn = nb.predict(X_test)

nb_pred_prob = nb.predict_proba(X_test)[:,1]
```

6. Model Evaluation

Once the model has been trained, its performance needs to be evaluated to assess how well it is performing. There are various metrics that can be used to evaluate the model's performance, including:

- **Accuracy**: Measures the overall correctness of the model by calculating the ratio of correctly predicted instances to the total number of instances. It indicates how often the model predicts correctly, regardless of the class.

- **Precision**: Measures the proportion of correctly predicted positive instances (true positives) out of the total instances predicted as positive (true positives + false positives). It indicates the model's ability to avoid false positives, meaning correctly identifying toxic instances without falsely labeling non-toxic instances.

- **Recall**: Also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances (true positives) out of the total actual positive instances (true positives + false negatives). It indicates the model's ability to identify all actual positive instances without missing any.

- **F1 Score**: The F1 score is the harmonic mean of precision and recall. It provides a single metric that combines both precision and recall, giving a balanced measure of the model's performance. It is useful when both precision and recall are important, as it takes into account both false positives and false negatives in its calculation.

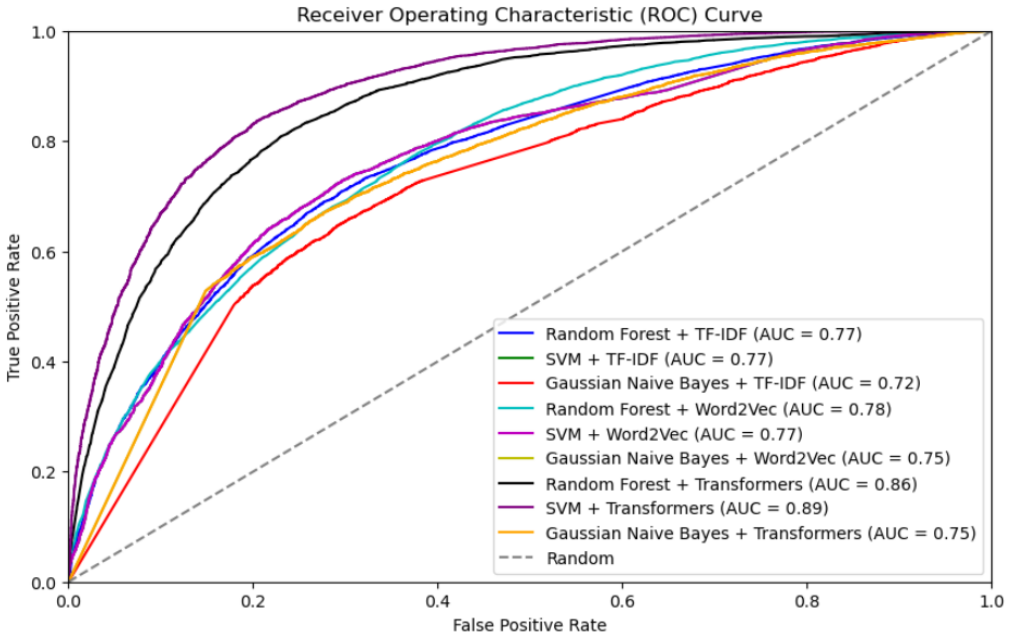
```
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_preds)
precision = precision_score(y_test, y_preds)
recall = recall_score(y_test, y_preds)
f1 = f1_score(y_test, y_preds)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

ROC (Receiver Operating Characteristic) curve

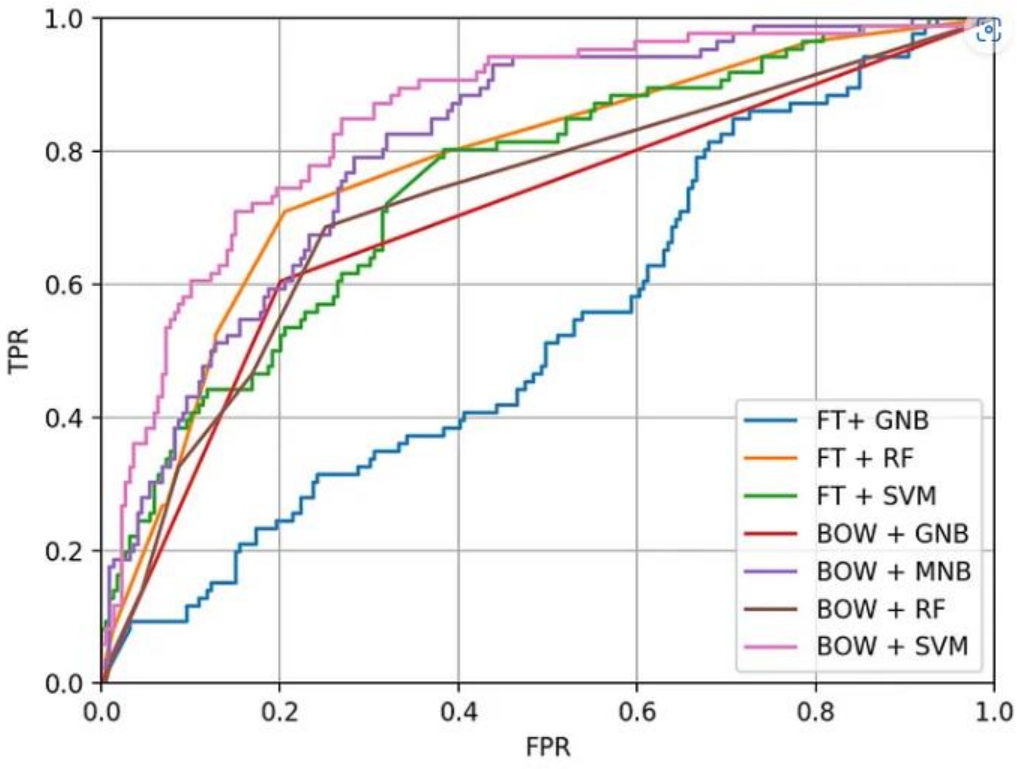
The ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a binary classification model. It displays the relationship between the true positive rate (TPR) and the false positive rate (FPR) at various classification thresholds.It provides an intuitive visualization of the model's ability to discriminate between classes and helps in selecting the optimal classification threshold based on specific problem requirements.

True Positive Rate (TPR) focuses on correctly identifying positive instances.
False Positive Rate (FPR) focuses on incorrectly classifying negative instances as positive.
These rates provide insights into the model's performance in terms of identifying true positives and avoiding false positives.

➔ The ROC curve of our Model with Arabic dataset:



➔ The ROC curve with English dataset:



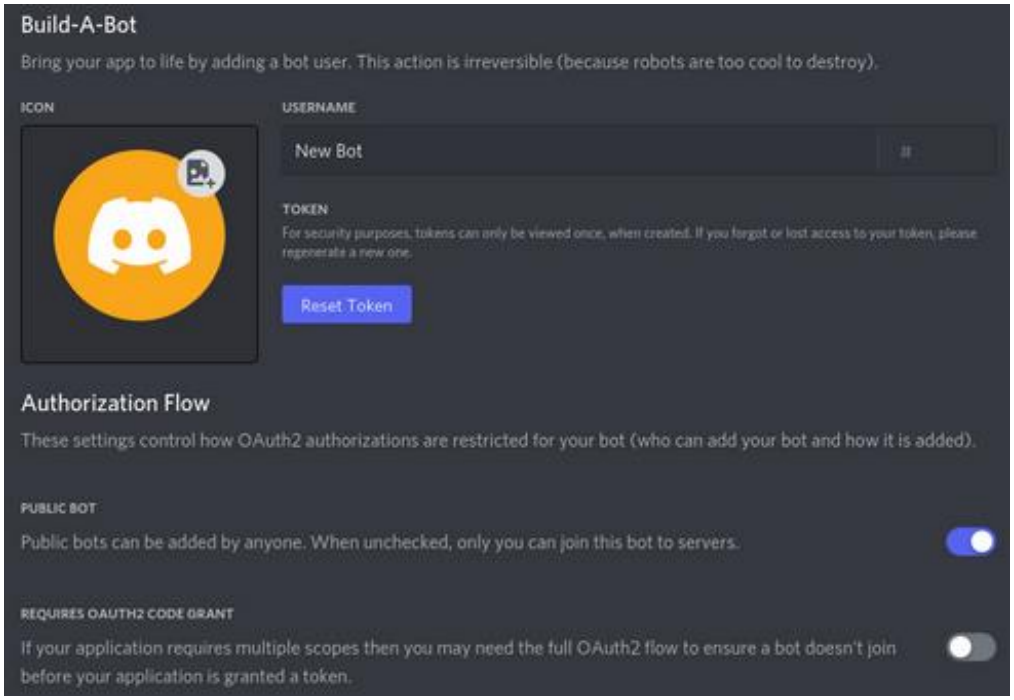
Hate Speech Detection. (Foreword by Teemu Turunen, Corporate… | by Futurice | Medium

7. Model deployment

This report documents the deployment of a machine learning model, specifically designed for hate speech detection, using a Discord bot on a social media platform. This innovative approach allows for real-time analysis and feedback on potentially toxic content, enhancing the safety and user experience of the community.

Discord Bot Creation

The deployment process began with the creation of a Discord bot. This process was conducted via Discord's developer portal, where a new application was registered and a bot user was added. This step resulted in the generation of a bot token, a unique identifier that allows the bot to log into Discord servers.



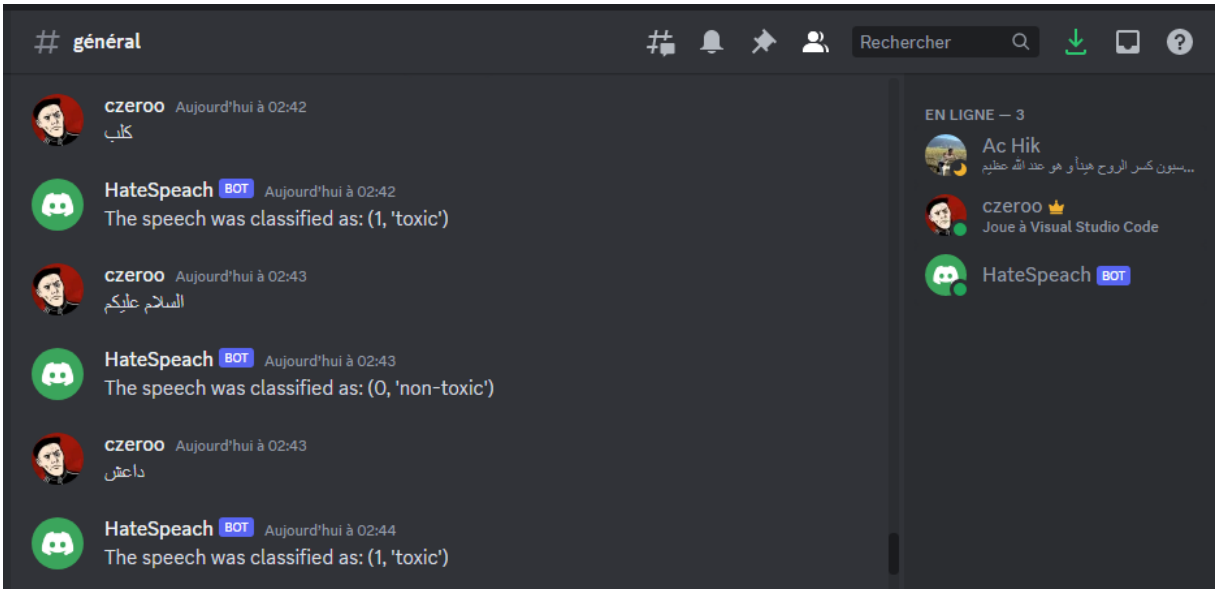
Integration of Bot and Model

The bot was coded in Python, using the discord.py library, which interacts with Discord's API. It was configured to listen for messages in the server, process the text, and use the machine learning model to determine the toxicity of the content.

The model, trained with an extensive dataset of text messages labeled for toxicity, was integrated into the bot's code. Upon detection of a new message, the bot processes the text and feeds it to the model, which then outputs a classification of "toxic" or "non-toxic".

Following the successful integration, the bot was added to the Discord server. This step was achieved by generating an invite link for the bot from the developer portal and using this link to add the bot to the desired server.

With the bot now active on the server, it continuously monitors incoming messages, classifies them, and provides real-time feedback on their toxicity. This process occurs in the background, with users only seeing the bot's feedback if a message is deemed toxic.



The deployment of this machine learning model via a Discord bot has proven to be an effective method for monitoring and addressing hate speech on social media platforms. This method offers significant potential for enhancing digital safety and promoting healthier, more respectful online interactions.

Technologies used

Discord: An instant messaging and digital distribution platform designed for creating communities. It offers voice, video, and text communications.

Discord Developer Portal: This is a part of Discord's website where users can create and manage bots, OAuth applications, and game SDK applications. You likely used this to create your bot and get the necessary tokens for it to operate.

Discord Bot Token: A unique identifier assigned to a bot that authenticates it when it connects to Discord's servers. It's like a password that should be kept secure, as anyone with the token can control the bot.

Discord API: This API allows you to interact with Discord's service programmatically. You likely used it to receive messages from the chat, check user permissions, send responses, and more.

discord.py: A Python library that simplifies the use of the Discord API for Python developers. It allows you to create bots with less code and offers high-level abstractions for many of Discord's features.

Python: The language you used to code the bot. Python is a high-level, interpreted programming language that is known for its readability and simplicity.