



## **School of Science and Engineering**

CSC 5354 NLP for Big Data

Spring 2023

## **Online Grooming Detection to Protect Children Using NLP**

Realized by :

Oumaima Laghzaoui

Soufiane Lamchoudi

Sofya Sellak

Supervised by:

Dr. Violetta Cavalli-Sforza

# Table of Contents

Table of Contents .....	2
I. Statement of the Problem: .....	4
Proposal Statement: .....	5
Domain Research: .....	6
II. Code Organization: .....	7
Data Collection and EDA Task: .....	7
Model Building Task: .....	7
Model Deployment Task: .....	7
III. Implementation Details: .....	8
Data Pre-processing: .....	8
Model Selection: .....	9
Fine Tuning Using Native TensorFlow: .....	10
Model Saving and Loading: .....	11
Model Inference: .....	11
Model Evaluation: .....	11
IV. Proof of Concept: .....	13
V. Code Description: .....	14
Transformers: .....	14
Import Libraries: .....	14
NLTK Downloads: .....	16
Preprocess the Text: .....	16
Most Frequent Words: .....	17
Define Features and Labels: .....	18
Tokenizing with DistilBert Tokenizer: .....	19
Create Dataset Object for TensorFlow: .....	20
Fine Tuning Using Native TensorFlow: .....	21
Deployment of the Model: .....	22
VI. Challenges: .....	24
VII. Future Work: .....	25
VIII. Conclusion: .....	25

## Table of Figures:

Figure 1: Model Evaluation .....	13
Figure 2: Import Transformers.....	14
Figure 3: Import Libraries.....	15
Figure 4: The Preprocessing Phase .....	17
Figure 5: Count the Most Frequent Words.....	17
Figure 6: Cloud Chart for Most Frequent Words .....	18
Figure 7: Define Features and Labels.....	19
Figure 8: DistilBert Tokenizer.....	20
Figure 9: Dataset Object for TensorFlow .....	21
Figure 10: Model Building.....	22
Figure 11: Helper Functions for Model Deployment .....	23
Figure 12: Main Function for Model Deployment .....	23

## Abstract

“Online Grooming Detection to Protect Children Using NLP” presents the collaborative efforts of three students, Oumaima Laghzaoui, a graduate student majoring in Big Data Analytics, Sofya Sellak, an undergraduate student majoring in General Engineering and Thematic Areas in AI, and Soufiane Lamchoudi, a graduate student majoring in Big Data Analytics. This project aimed to tackle the critical issue of online grooming by utilizing Natural Language Processing (NLP) techniques. The team worked diligently to define the problem scope, collect relevant sentences from social media falling under the category of 'hate,' preprocess and annotate the data, and develop a machine learning algorithm for effective classification based on predefined labels. The culmination of their efforts resulted in the successful deployment of a Proof of Concept (POC) model using Streamlit. The project was executed smoothly, achieving its objectives, and will be followed by future work focused on enhancing both the model's performance and its deployment methodologies. This project underscores the importance of leveraging NLP techniques to safeguard children online and lays the foundation for ongoing advancements in protecting children from online grooming.

## I. Statement of the Problem:

Online violence has become a widespread and growing concern in recent years. The main problem is when it comes to identifying grooming behavior in online chats with children. Adults who have a sexual interest in children or intend to harm them often use online grooming to establish contact with them. Grooming, in general, is a complex and multidimensional process, where an adult seeks to engage a child in a seemingly voluntary interaction to sexually abuse them. However, in French-speaking regions, such as Morocco, the problem is particularly acute due to the complexity of the French language and the idiosyncrasy in grooming expressed. This makes it challenging to monitor and control, leading to the propagation of harmful or abusive content, cyberbullying, and other forms of online hate violence that can target children on online platforms.

Furthermore, the challenge is exacerbated by the rapid growth in the volume of user-generated content, which makes manual moderation impractical and inefficient. Online platforms often lack the resources to consistently and effectively moderate content, allowing online violence/grooming to slip through the cracks. Moreover, research suggests that children who are targeted by online grooming may not recognize the danger until it is too late, which makes it challenging for children.

The need to address this problem is underscored by the potential harm that unchecked online violence can cause. It can lead to the marginalization of certain groups, contribute to the spread of misinformation and discrimination, and negatively impact the mental health of individuals, especially children. Additionally, it can lead to real-world violence and conflict, making it a serious societal issue.

## **Proposal Statement:**

The solution proposed is the development of a French text classifier that utilizes Large Language Models to detect online grooming. This tool sought to analyze and categorize French-language text content in real-time, identifying, and flagging potential online violence for further review. This would provide a scalable solution to the challenge of moderating online content and help mitigate the harm caused by online violence. A few examples where this French Language Classifier (FLC) could be particularly useful for detecting online grooming and protecting children are as follow:

- **Social Media Monitoring:** Children spend significant time on social media platforms, where they might engage in conversations with strangers. An FLC could be integrated into these platforms to monitor public and private conversations for signs of grooming. For example, if an adult user frequently sends messages to minors, uses overly affectionate language, asks for personal information, or suggests private meetings, the LLM could flag these interactions for further review.
- **Online Gaming Communities:** Online multiplayer games often have chat functionalities, which can unfortunately be exploited for grooming. An FLC could monitor these chats in real time, flagging suspicious behavior or conversations. For instance, if a user consistently singles out a particular player for private chats or uses language indicative of grooming, the system could alert the platform's moderators.
- **Educational Platforms:** Children increasingly use online platforms for learning and homework help. An LLM could monitor these platforms to ensure they remain safe and focused on learning. If a user tries to shift the conversation away from academics towards personal topics, or attempts to establish contact outside the platform, the FLC could detect this behavior.
- **Parental Control Applications:** Parents could use an FLC-powered application to oversee their children's online activity without infringing on their privacy. The application could alert parents if it detects potential grooming in their children's online conversations, enabling them to intervene if necessary.
- **Law Enforcement:** An FLC could assist law enforcement and child protection agencies in identifying and investigating potential cases of online grooming. By analyzing large amounts of

online content and flagging suspicious conversations, the FLC could help these agencies proactively identify threats and take appropriate action.

In all these cases, it's important to note that while an FLC can be a powerful tool for detecting online grooming, it should be part of a larger strategy that includes educating children about online safety, promoting safe online behaviors, and encouraging open communication about their online experiences. In conclusion, our proposed solution not only has the potential for immediate practical application but also contributes to a broader discussion on how technology can be leveraged to create safer, more inclusive online spaces.

## **Domain Research:**

At the initial stages of the project, various classes of hate speech were identified, and five key classes were defined to serve as a framework for data collection and annotation. These five classes of hate encompass different forms of online hate and provide a comprehensive understanding of the types of harmful content encountered in the digital sphere. The first class, sexism, refers to acts of violence that occur or persist in the online realm and are characterized by sexism or sexual nature. It targets girls and boys by reinforcing dominant gender norms, tarnishing the reputation of girls, and threatening the masculinity of boys. The second class, racism, focuses on cyber racism, which encompasses racism manifesting in online spaces. This includes racist websites, images, blogs, videos, online comments, as well as racist content in text messages, emails, and social networking sites. The broader definition encompasses any use of information and communication technologies to transmit racist attitudes and behaviors, intending to harm or distress others. Homophobia, the third class, explores the experiences of sexual minorities in online environments. Online spaces often serve as refuge from offline discrimination, enabling individuals to seek socialization and connection. However, paradoxically, this increased reliance on the internet can expose sexual minorities to online sexual victimization and risks (OSVR). Online platforms, mirroring societal prejudices, can perpetuate homophobia and discrimination, contributing to higher rates of OSVR and negative mental health outcomes among sexual minorities. The fourth class, hate speech, encompasses offensive discourse targeting individuals or groups based on inherent characteristics such as race, religion, or gender. Hate speech is characterized by pejorative or discriminatory language that threatens social harmony. The United Nations Strategy and Plan of Action on Hate Speech defines hate speech as any form of communication, whether spoken, written, or behavioral, that employs discriminatory or derogatory language against individuals or groups based on their religion, ethnicity, nationality, race, color, descent, gender, or other identity factors. Finally, bullying, both in physical and cyber forms, is addressed. Bullying involves intentional and repeated acts that cause harm or discomfort to another person. Cyberbullying specifically

encompasses threatening or harassing behavior conducted through electronic technology, such as cell phones, email, social media platforms, or text messaging. These five classes of hate served as the basis for data collection and subsequently acted as labels for data annotation. By employing these classes, our project aimed to capture and analyze a broad spectrum of harmful content in the French language, providing a comprehensive understanding of online violence against children and facilitating targeted interventions and prevention strategies.

## **II. Code Organization:**

### **Data Collection and EDA Task:**

- The data is loaded into a pandas DataFrame from a CSV file. Pandas DataFrames are highly efficient, 2-dimensional data structures that can handle complex operations on large datasets.
- Then we perform exploratory data analysis (EDA), which is a crucial step in understanding the data, its structure, and the potential features that could influence the outcome. The unique labels are identified, and their frequency is counted.
- The data cleaning step involves the conversion of all text to lowercase (to ensure uniformity), removal of stop words (commonly used words like 'is', 'in', 'the' etc.), tokenization (breaking up the text into individual words), and lemmatization (reducing words to their base or root form).

### **Model Building Task:**

- The model used in this project is DistilBERT, a transformer-based model that's a lighter and faster version of BERT. It's used for the task of sequence classification, a common task in NLP where the goal is to classify text into predefined categories.
- We first define the model and then set up the Adam optimizer with a learning rate of  $5e-5$ . The model is then compiled with the optimizer and accuracy as the metric. Following this, then we trained the model using the fit method on the training dataset.
- Then we used the train\_test\_split function from the sklearn library to split the data into training and validation sets, which is essential for understanding the model's performance and ensuring that it is not overfitting to the training data.

### **Model Deployment Task:**

- After training the model, we saved it to disk for later use. This allows us to load the pre-trained model without needing to train it again.

- Then we load the model from the saved file, and it is ready to make predictions on new, unseen data. The prediction process involves tokenizing the input text, feeding it into the model, and then taking the argmax of the output to get the predicted label.
- The predict sentiment function is a helper function that maps the model's output (which is a number) to the corresponding sentiment class (which is a string). This makes the output more understandable for humans.
- Then We test the model on a sample text to validate that it is working as expected

### III. Implementation Details:

The idea of this project is to create a French Language Classifier that can detect text and classify it into different categories such as "bullying", "hate speech", "homophobia", "none", "racism", and "sexism". The goal is to build a text classifier using DistilBERT Language Model to predict the category of a given text. the following, is the explanation of the design choice and how it works:

#### Data Pre-processing:

Data pre-processing plays a vital role in the development of any NLP application, and its significance cannot be overstated. Textual data in its raw form often contains various elements that can hinder the performance and accuracy of NLP models. By pre-processing the data, we can refine and transform it into a suitable format for analysis. This step is essential as it helps in reducing the dimensionality of the data, enhancing the signal-to-noise ratio, and allowing the model to focus on the most relevant information. Moreover, pre-processing aids in standardizing the text, such as converting all characters to lowercase or handling common abbreviations, ensuring consistency in the data. Additionally, it can include techniques like tokenization, stemming, or lemmatization, which help in normalizing words and reducing their inflectional forms. By performing these pre-processing steps, we create a clean and well-structured dataset that enables NLP models to extract meaningful patterns, relationships, and insights from the text, ultimately improving the overall performance and accuracy of the application.

In the initial stages of our project, the annotated data was subjected to pre-processing to ensure its readiness for the subsequent model development phase. This pre-processing step played a crucial role in preparing the textual data for further analysis. The process involved several essential steps, such as the removal of stop words, punctuation marks, URLs, emojis, and other irrelevant information from the text. By eliminating these elements, we aimed to refine the data and eliminate noise that could potentially hinder



the accuracy of the hate speech detection model. Pre-processing is a fundamental step in any machine-learning pipeline that deals with textual data. Its purpose is to enable the AI algorithm to focus on the most relevant information, thereby improving the overall accuracy of the model. However, it is worth noting that some of the aforementioned steps may not be necessary if the hate speech detection models employed are based on advanced techniques such as Transformers or Language Models. These models possess the inherent ability to handle certain aspects of pre-processing, eliminating the need for manual removal of stop words, punctuation, and other similar elements. This highlights the adaptability and efficiency of modern approaches in the field of natural language processing for hate speech detection. To prepare the textual data for our French Text Classifier, the team implemented a set of pre-processing functions. These functions, written in Python, are designed to enhance the quality and relevance of the text for subsequent analysis. The code includes functions to remove URLs, accents, punctuation marks, emojis, digits, and extra whitespaces from the text. Additionally, there is a function to remove common stop words, such as frequently occurring words in the French language, which can often be noise in the analysis. Furthermore, the code offers a stemming function using the FrenchStemmer to reduce words to their root form. These pre-processing steps significantly contribute to refining the textual data and reducing noise, thereby improving the accuracy and performance of the NLP model. The functions can be easily integrated into the data pipeline, allowing for efficient data pre-processing in our NLP application.

## **Model Selection:**

The DistilBERT model is chosen for its effectiveness in text classification tasks. DistilBERT is a powerful pre-trained language model that can be fine-tuned for various natural language processing tasks, including text classification. One of the main advantages of using DistilBERT is its smaller size and faster training time compared to its predecessor, BERT. This makes it more efficient for deployment in real-world applications with limited computational resources. Additionally, DistilBERT has shown competitive performance on various benchmark datasets, indicating its effectiveness for text classification tasks. Its ability to capture contextual relationships between words and phrases in text also makes it well-suited for tasks that require a deep understanding of language, such as sentiment analysis or named entity recognition. However, our classifier model was based on the DistilBERT language model, in which the team used fine-tuning techniques for the multi-class text classification problem that involves detecting online violence/grooming. Fine-tuning in the HuggingFace's transformers library involved leveraging a pre-trained model and a compatible tokenizer tailored to the model's architecture and input requirements. The team selected the DistilBertTokenizer's tokenizer class to tokenize the texts and subsequently utilized the TFDistilBertForSequenceClassification model class for fine-tuning the pre-trained DistilBert model using the tokenized outputs. The DistilBertTokenizer played a pivotal role in generating the necessary input\_ids

and attention\_mask for the DistilBert model, as these were the expected inputs. The team instantiated the tokenizer object using the from\_pretrained() method, which facilitated the downloading and caching of the required tokenizer files associated with the DistilBert model. Padding and truncation techniques were employed to ensure uniform vector sizes across the dataset.

Before initiating the fine-tuning process, it was imperative to convert the encoded texts and labels into a TensorFlow Dataset object. The team accomplished this by employing the *from\_tensor\_slices* constructor method, enabling seamless integration into the TensorFlow framework.

The team discussed two distinct fine-tuning methods for the DistilBert model, but the main focus was one native fine-tuning using TensorFlow which is outlined in detail below. These methods served as crucial components of the overall fine-tuning strategy, enhancing the model's ability to capture meaningful patterns and optimize its classification performance.

*Training and Validation:* The dataset is split into training and validation sets using the *train\_test\_split function*. This division ensures that the model learns from a large portion of the data during training and evaluates its performance on unseen data during validation. The split ratio can be adjusted based on the available data size and the desired balance between training and validation performance.

*Tokenization and Encoding:* The text data is tokenized and encoded using the DistilBERT tokenizer. This process converts the text into numerical representations that the model can process. By truncating and padding the tokenized sequences, the input data is made consistent in length, allowing it to be efficiently processed by the model.

## Fine Tuning Using Native TensorFlow:

The initialization of a pre-trained model in our project was achieved through the utilization of the *from\_pretrained()* method. This method played a vital role in loading the pre-existing weights and initializing the model based on the predefined configurations. The DistilBert model, along with other models available in the transformers library, conforms to the standard tf. keras.Model class (or torch.nn.Module in the case of PyTorch), enabling seamless integration with the native TensorFlow and Keras APIs. As a result, the pre-trained model could be employed in a manner consistent with other models in the TensorFlow ecosystem. During the training phase conducted by the team, significant progress was made in optimizing the model's performance. The outcomes of the training process, including the model's accuracy, loss, and other relevant metrics, are visually presented in the accompanying image. This visualization provides a comprehensive overview of the training results and offers valuable insights into the efficacy of the fine-tuning process. By analyzing the training outcomes, we can assess the model's ability

to learn from the Twitter dataset and perform accurate multi-class text classification. These results serve as a testament to the effectiveness of the chosen approach and highlight the model's performance and potential for real-world applications.

## Model Saving and Loading:

To ensure the reusability and easy deployment of our trained model, a design choice was made to save both the model and tokenizer to a specified directory. This was accomplished using the *save\_pretrained* method provided by the Hugging Face's transformers library. By saving the model and tokenizer, we enable future inference without the need for retraining, which significantly enhances efficiency and convenience. This design choice is particularly valuable in scenarios where the model needs to be deployed in production systems or shared with others. It provides a straightforward mechanism to package and distribute the trained model, allowing for seamless integration into different environments or collaborative projects. The saved model and tokenizer serve as valuable assets that can be readily utilized in various applications, highlighting the scalability and versatility of our trained model.

## Model Inference:

Upon loading the saved model and tokenizer, the *predict\_sentiment* function plays a crucial role in mapping the predicted label index to the corresponding sentiment category. This mapping process is essential to ensure that the model's output is easily interpretable by humans and provides meaningful insights. By associating the predicted label index with its sentiment category, we enable a clear understanding of the sentiment expressed in the text. This feature enhances the model's usability and allows users to derive actionable information from the model's predictions. To demonstrate the model's text classification capabilities, a sample text is utilized, and predictions are made using the loaded model. The predicted sentiment is then presented, showcasing the model's ability to accurately classify text and provide valuable insights into the sentiment expressed. This demonstration reinforces the model's effectiveness in capturing and interpreting sentiment, thereby validating its utility in real-world applications.

## Model Evaluation:

Accuracy serves as a fundamental metric for evaluating the performance of our model. In the context of our text classification task, accuracy measures the proportion of correctly classified instances out of the total number of instances. By considering the accuracy metric, we gain valuable insights into the overall effectiveness of our model in correctly predicting the sentiment of the text. Throughout the training and evaluation process, we monitored the accuracy values to gauge the model's performance. The recorded accuracy values demonstrate the model's ability to make accurate predictions on both the training and

validation datasets. Higher accuracy values indicate a stronger alignment between the predicted sentiment labels and the ground truth labels. It is important to note that while accuracy provides a useful measure of overall model performance, it may not capture the full picture, especially in cases where the dataset is imbalanced, or the cost of misclassification varies across different sentiment categories. Therefore, it is essential to consider additional evaluation metrics such as precision, recall, and F1-score to gain a comprehensive understanding of the model's performance in different aspects. However, accuracy remains a valuable and widely used metric for assessing the overall predictive capability of our text classification model. The training progress of our model can be observed through the recorded values of loss, accuracy, validation loss, and validation accuracy. These metrics provide insights into the model's performance and its ability to learn from the training data. The table presents the recorded values for each epoch during the training process. In the initial epoch, the model achieved a loss of 0.918537 and an accuracy of 0.733333, while the validation loss and validation accuracy were 0.817960 and 0.740260, respectively. As the training progressed, improvements in the model's performance were evident. By the fifth epoch, the model achieved a significantly reduced loss of 0.296667 and an improved accuracy of 0.908401. The validation loss and validation accuracy also showed improvements, with values of 0.726989 and 0.783550, respectively. These results demonstrate the model's ability to effectively learn from the training data and generalize well to unseen validation data. The progressive decrease in loss and increase in accuracy indicates that the model successfully adapted to the dataset, showcasing its potential for accurate text classification and sentiment analysis tasks.

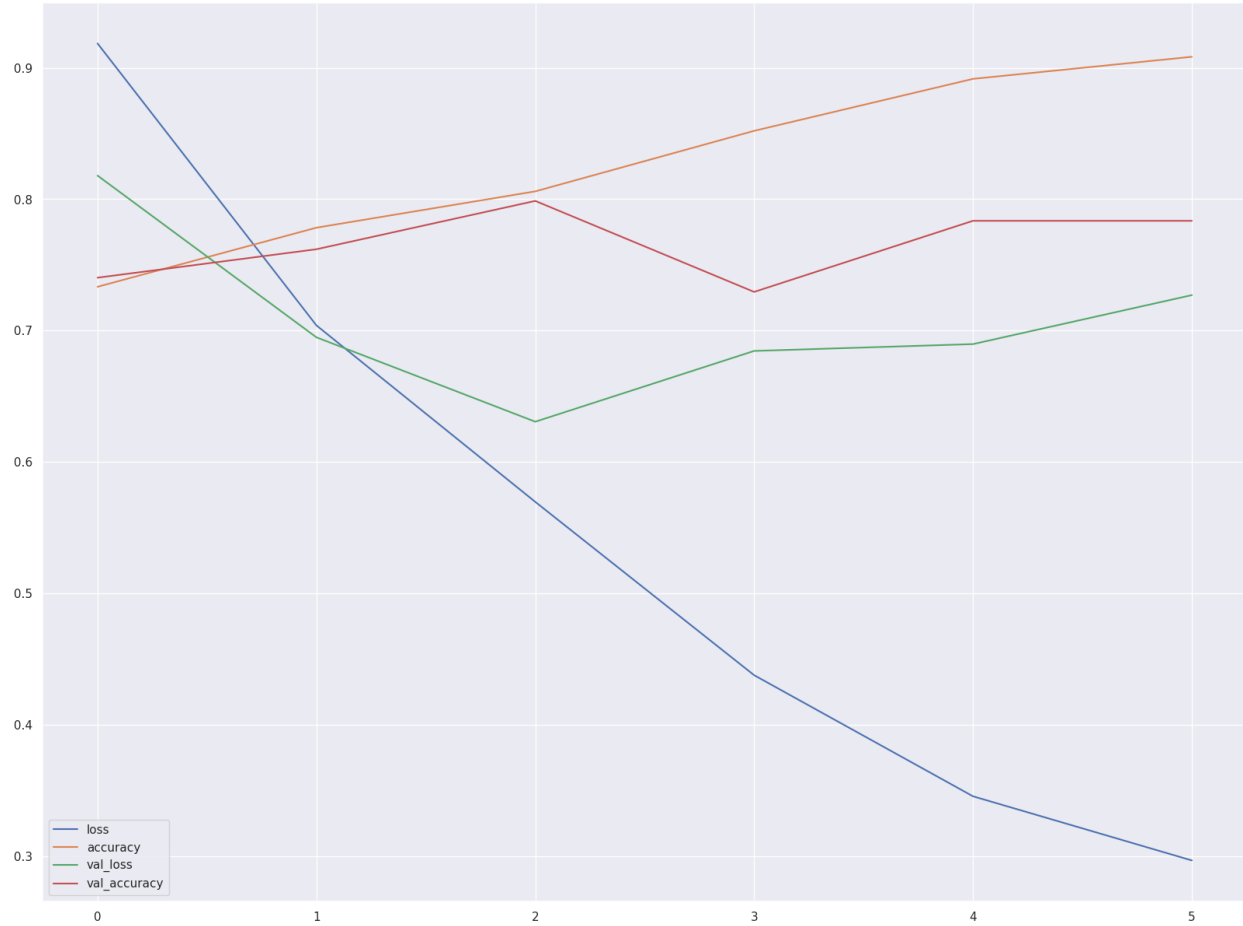


Figure 1: Model Evaluation

## IV. Proof of Concept:

To demonstrate the feasibility and real-world applicability of our ML model, a proof of concept (POC) was developed. A POC serves as an experimental demonstration, showcasing the successful deployment of a program, product, or system in practical scenarios. To create a user-friendly and interactive interface for our POC, we utilized Streamlit, an open-source app framework specifically designed for Machine Learning and Data Science applications. Streamlit facilitates the rapid development and deployment of web apps, without requiring extensive front-end development knowledge. It seamlessly integrates with popular Python libraries, including scikit-learn, Keras, PyTorch, NumPy, pandas, and Matplotlib, enabling seamless integration with our ML model.

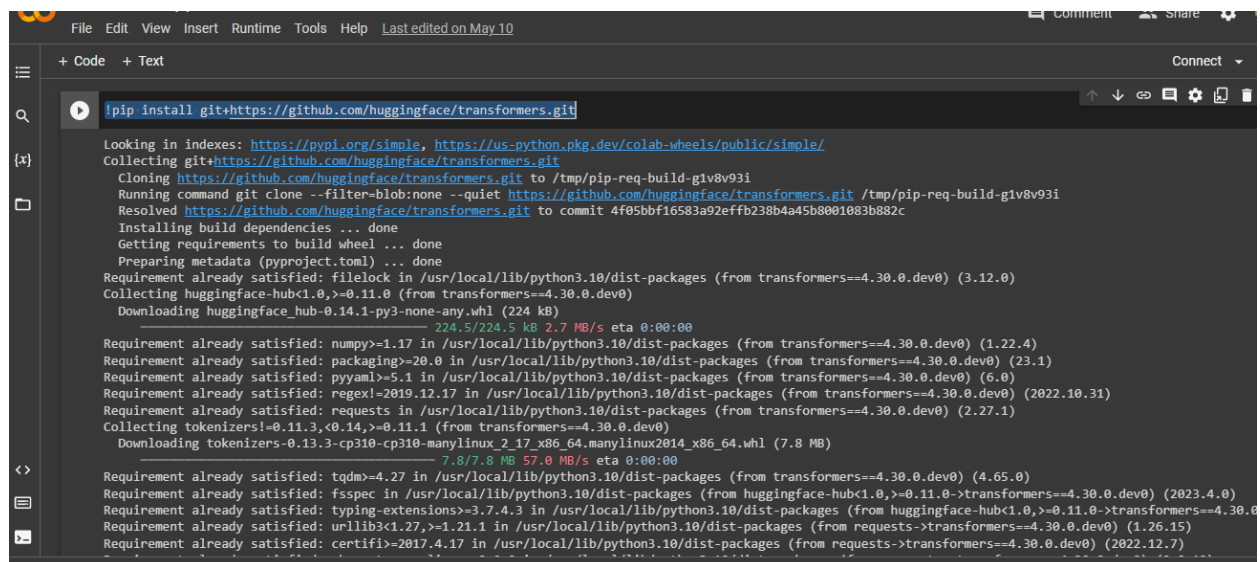
The POC implemented in our project takes the form of a user interface where the user can input a sentence in French. The ML model then classifies the sentence based on its content. The interface provides real-time feedback on the classification results. As an example, prompts such as 'hate speech' and 'homophobic' were tested, and the model accurately classified them as such. This POC demonstrates the practical

implementation of our ML model in a user-friendly manner, showcasing its ability to effectively classify French sentences based on their content. This serves as a foundation for potential future development and deployment of the model in real-world applications.

## V. Code Description:

### Transformers:

The command provided in the figure serves the purpose of installing the transformers library from the GitHub repository of Hugging Face. Widely acclaimed and utilized in the field of natural language processing, this library offers an extensive range of cutting-edge models, including BERT, GPT, and many others. By executing this installation command, researchers and practitioners gain access to a comprehensive suite of pre-trained models, tokenizers, and utilities tailored for various language-related tasks such as text classification, named entity recognition, and question answering. This installation step proves crucial in harnessing the diverse functionalities and capabilities provided by the transformers library, empowering users to advance their NLP projects with state-of-the-art tools and techniques.



```
File Edit View Insert Runtime Tools Help Last edited on May 10
+ Code + Text
Connect
pip install git+https://github.com/huggingface/transformers.git

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/huggingface/transformers.git
  Cloning https://github.com/huggingface/transformers.git to /tmp/pip-req-build-g1v8v93i
  Running command git clone --filter=blob:none --quiet https://github.com/huggingface/transformers.git /tmp/pip-req-build-g1v8v93i
  Resolved https://github.com/huggingface/transformers.git to commit 4f05bbf16583a92effb238b4a45b8001083b882c
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers==4.30.0.dev0) (3.12.0)
Collecting huggingface-hub<1.0,>=0.11.0 (from transformers==4.30.0.dev0)
  Downloading huggingface_hub-0.14.1-py3-none-any.whl (224 kB)
    224.5/224.5 kB 2.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers==4.30.0.dev0) (1.22.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers==4.30.0.dev0) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers==4.30.0.dev0) (6.0)
Requirement already satisfied: regex<=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers==4.30.0.dev0) (2019.12.17)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers==4.30.0.dev0) (2.27.1)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers==4.30.0.dev0)
  Downloading tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
    7.8/7.8 MB 57.0 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers==4.30.0.dev0) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.11.0->transformers==4.30.0.dev0) (2023.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.11.0->transformers==4.30.0.dev0) (4.3.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers==4.30.0.dev0) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers==4.30.0.dev0) (2022.12.7)
```

Figure 2: Import Transformers

### Import Libraries:

The code in the figure includes several import statements that are crucial for various aspects of the project. The "from transformers import DistilBertTokenizer" statement imports the DistilBertTokenizer from the

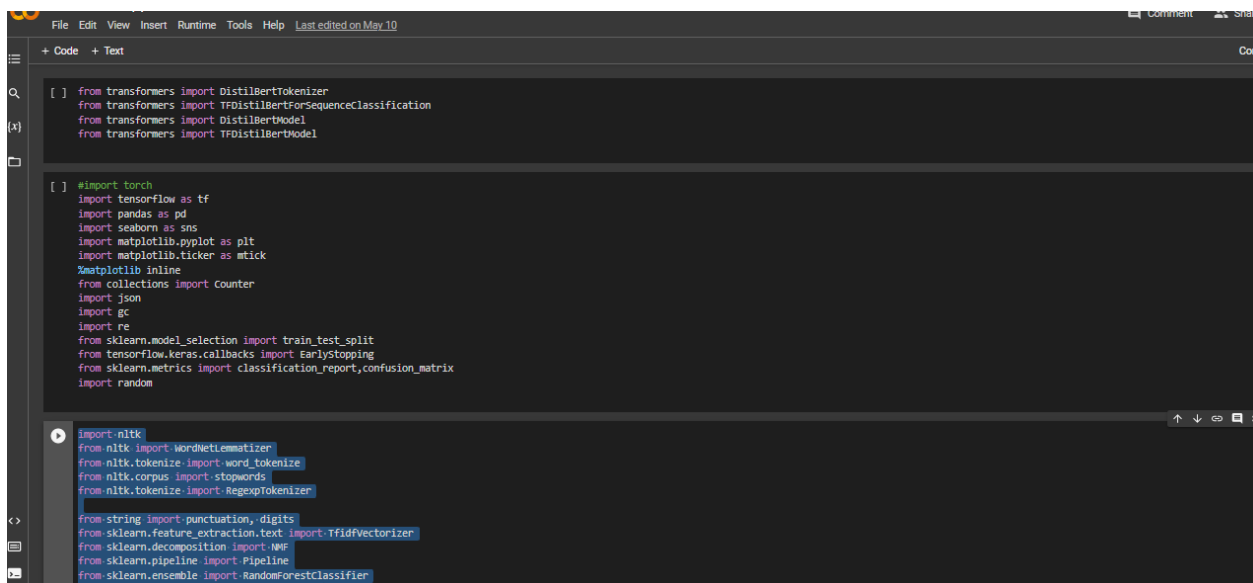
transformers library. This tokenizer is used to preprocess and tokenize the input text data before feeding it into the DistilBERT model, ensuring effective language representation.

Similarly, the "from transformers import TFDistilBertForSequenceClassification" import statement brings in the TFDistilBertForSequenceClassification model from the transformers library. This pre-trained DistilBERT model is specifically fine-tuned for sequence classification tasks, making it a suitable choice for the project's classification requirements.

The "from transformers import DistilBertModel" statement imports the DistilBertModel from the transformers library. By utilizing this model, you can leverage the pre-trained DistilBERT model for various tasks, such as extracting contextualized word embeddings, which can enhance the model's understanding of the input text and improve its performance.

Moreover, the "from transformers import TFDistilBertModel" import statement brings in the TFDistilBertModel from the transformers library. This model is specifically designed for TensorFlow and can be used for tasks like sequence classification or sequence-to-sequence tasks within the TensorFlow framework, providing seamless integration with TensorFlow-based models and workflows.

The remaining import statements, such as importing TensorFlow, pandas, seaborn, matplotlib, and nltk libraries, along with their respective modules or classes, bring in essential tools and functionalities. These libraries offer a wide range of capabilities, including data manipulation, visualization, statistical analysis, and natural language processing tasks. Overall, these important statements provide a strong foundation for developing and implementing the project's machine-learning models and data processing pipelines.



```
[ ] from transformers import DistilBertTokenizer
[ ] from transformers import TFDistilBertForSequenceClassification
[ ] from transformers import DistilBertModel
[ ] from transformers import TFDistilBertModel

[ ] #import torch
[ ] import tensorflow as tf
[ ] import pandas as pd
[ ] import seaborn as sns
[ ] import matplotlib.pyplot as plt
[ ] import matplotlib.ticker as mtk
[ ] %matplotlib inline
[ ] from collections import Counter
[ ] import json
[ ] import gc
[ ] import re
[ ] from sklearn.model_selection import train_test_split
[ ] from tensorflow.keras.callbacks import EarlyStopping
[ ] from sklearn.metrics import classification_report, confusion_matrix
[ ] import random

[ ] import nltk
[ ] from nltk import WordNetLemmatizer
[ ] from nltk.tokenize import word_tokenize
[ ] from nltk.corpus import stopwords
[ ] from nltk.tokenize import RegexpTokenizer

[ ] from string import punctuation, digits
[ ] from sklearn.feature_extraction.text import TfidfVectorizer
[ ] from sklearn.decomposition import NMF
[ ] from sklearn.pipeline import Pipeline
[ ] from sklearn.ensemble import RandomForestClassifier
```

Figure 3: Import Libraries

## NLTK Downloads:

The figure includes three `nltk.download` statement that downloads specific resources and datasets required by the NLTK library.

***nltk.download('omw-1.4')***: This statement downloads the Open Multilingual Wordnet (OMW) dataset version 1.4. WordNet is a lexical database that provides semantic relationships between words, and the OMW dataset specifically includes multilingual wordnets. This dataset is useful for tasks related to word sense disambiguation and cross-lingual applications.

***nltk.download('stopwords')***: This statement downloads the stopwords corpus. Stopwords are common words (e.g., "and," "the," and "is") that are often removed from text data as they carry little semantic meaning and can be noise in natural language processing tasks. The stopwords corpus provides a collection of stopwords for different languages, allowing you to remove them from text data efficiently.

***nltk.download('wordnet')***: This statement downloads the WordNet corpus. WordNet is a lexical database that groups words into sets of synonyms called synsets, providing information about their semantic relationships, such as hypernyms (superordinate terms) and hyponyms (subordinate terms). The WordNet corpus is widely used for various natural language processing tasks, including word sense disambiguation, semantic similarity, and lexical knowledge exploration.

By downloading these resources, you ensure that the required datasets and language resources are available for your project, enabling you to perform tasks such as lemmatization, stopword removal, and semantic analysis effectively using the NLTK library.

## Preprocess the Text:

The figure defines a function called ***clean\_text()*** that performs text cleaning and preprocessing on a given dataframe. The purpose of this function is to prepare the text data for further analysis or modelling tasks. In the function, the input dataframe is copied, and only the 'text' and 'encoded\_labels' columns are selected for processing. The French stopwords are loaded using the NLTK library to filter out common words that do not contribute much to the overall meaning of the text. The function then applies a series of cleaning operations to each text entry in the dataframe. This includes converting the text to lowercase, removing stopwords, tokenizing the text into individual words, and lemmatizing the words to their base form. By performing these cleaning steps, the function aims to reduce noise in the text data and improve the quality of subsequent analysis or modelling tasks. The cleaned text is stored in a new column of the dataframe, and the updated dataframe is returned as the output of the function.



```

3      3      4477      Sun Dec 18 08:31:26 +0000 2022      15856141      @ajplusfrancais Comment un migrant arrivant d...      racism      4
4      4      4761      Sun Dec 18 08:21:48 +0000 2022      17654549      @Charonoria Gros cadeau ga l      none      3

```

Cleaning the Data

```

[ ] def clean_text(data):
    """
    ---Input: data: a dataframe containing texts to be cleaned
    ---return: the same dataframe with an added column of clean_text
    """
    clean_data = data.copy()
    clean_data = clean_data[['text', 'encoded_labels']]
    stop_words = stopwords.words('french')
    clean_text = []
    tokenizer = RegexpTokenizer(r'\w+')
    lemmatizer = WordNetLemmatizer()

    for idx in range(len(data)):
        text = clean_data['text'][idx]
        label = clean_data['encoded_labels'][idx]
        text_lowercase = text.lower()
        text_no_stopwords = " ".join(word for word in text_lowercase.split() if word not in stop_words)
        text_tokenized = tokenizer.tokenize(text_no_stopwords)
        text_lemmatized = [lemmatizer.lemmatize(token) for token in text_tokenized]
        clean_text = " ".join(text_lemmatized)
        clean_data['text'][idx] = clean_text

    return clean_data

[ ] clean_data = clean_text(df)

```

Figure 4: The Preprocessing Phase

## Most Frequent Words:

The code in the figure performs word frequency analysis and generates a word cloud visualization based on the frequency of words in a given dataset. In the code, the **Counter()** function from the collections module is used to count the occurrences of each word in the word\_list. The **most\_common(100)** method is then applied to retrieve the 100 most common words and their respective frequencies. These results are stored in a pandas DataFrame named all\_words\_df, with columns labeled as 'word' and 'frequency'. Next, the word cloud is created using the WordCloud class from the wordcloud library. The text variable is created by joining all the words from the 'word' column of the all\_words\_df DataFrame. This text is passed as an argument to the **generate()** method of the WordCloud class to generate the word cloud visualization. Various parameters are specified for the word cloud, such as the size, colormap, maximum font size, and background color. The **imshow()** function from the matplotlib.pyplot module is then used to display the generated word cloud, and the **axis("off")** method is called to remove the axis labels. Finally, the **plt.show()** function is called to show the word cloud visualization.

```

(x) count_all = Counter(word_list).most_common(100)
all_words_df = pd.DataFrame(count_all)
all_words_df.columns = ['word', 'frequency']

# Creating word_cloud with text as argument in .generate() method
text = " ".join(all_words_df.word)

# Creating word_cloud with text as argument in .generate() method
word_cloud = WordCloud(collocations=False, width=1000, height=500,
                      colormap='dark2', max_font_size=150,
                      background_color='black').generate(text)

# Display the generated Word Cloud
plt.rcParams['figure.figsize'] = [20, 15]
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.show()

```

Figure 5: Count the Most Frequent Words

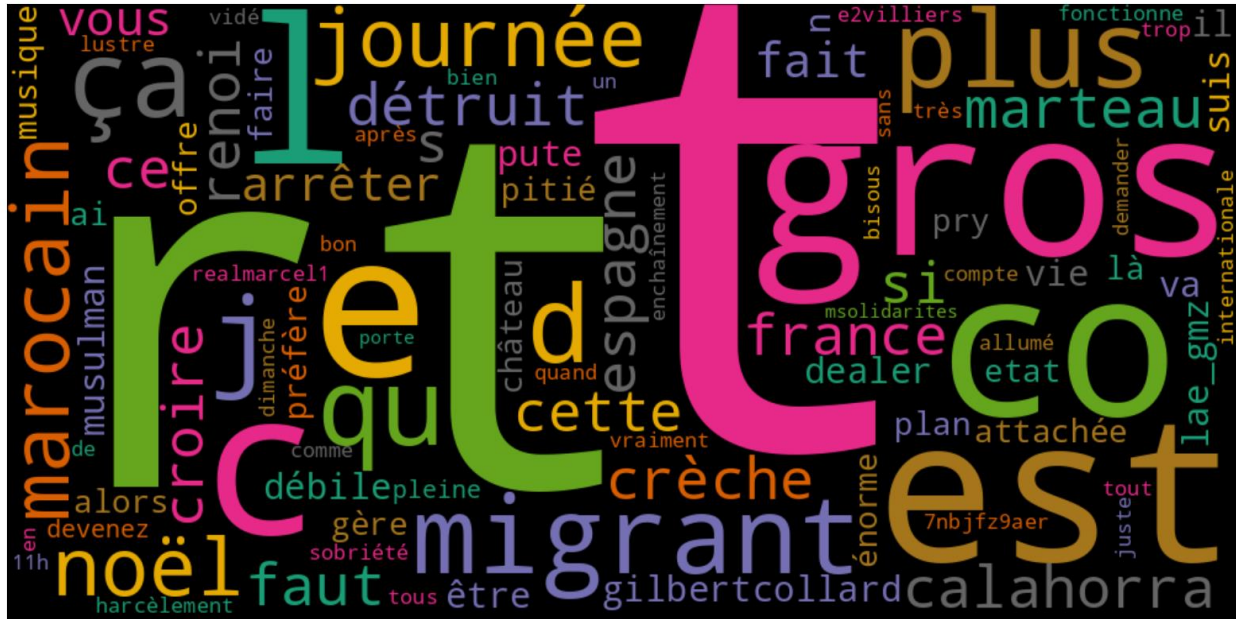


Figure 6: Cloud Chart for Most Frequent Words

## Define Features and Labels:

The provided code in the figure is responsible for splitting the dataset into training, validation, and testing subsets for model training and evaluation.

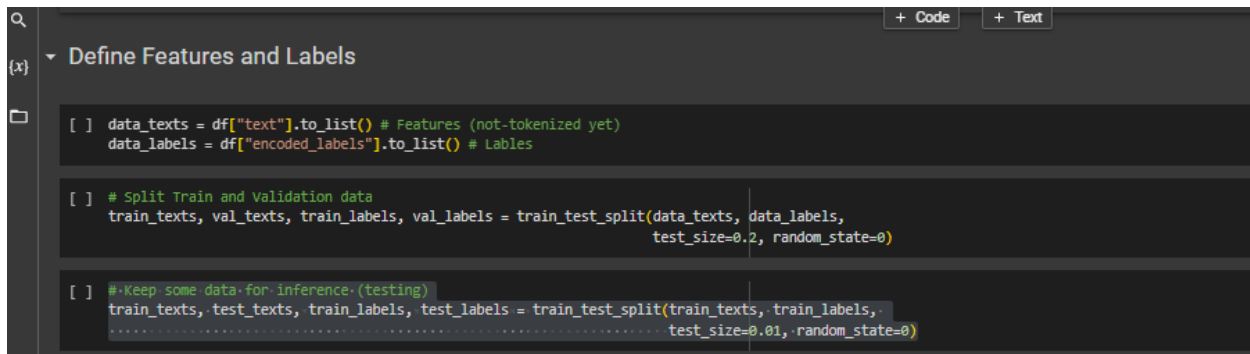
In the code, the text data is extracted from the 'text' column of the dataframe `df` using the `to_list()` method and stored in the `data_texts` variable. Similarly, the encoded labels are extracted from the 'encoded\_labels' column and stored in the `data_labels` variable.

The dataset is then split into training and validation data using the *train\_test\_split()* function from the `sklearn.model_selection` module. The `data_texts` and `data_labels` are passed as the input features and labels, respectively. The `test_size` parameter is set to 0.2, indicating that 20% of the data will be allocated for validation. The `random_state` parameter is set to 0 to ensure the reproducibility of the split.

The split is performed again on the training data to reserve a small portion for inference or testing purposes. This time, the ***train\_test\_split()*** function is applied to train\_texts and train\_labels. The test-size is set to 0.01, indicating that 1% of the training data will be used for testing. Again, the random\_state is set to 0 to maintain consistency in the split.

By splitting the data into training, validation, and testing subsets, the code enables the model to be trained on a portion of the data, validated on a separate subset to assess its performance, and finally tested on an

independent subset to evaluate its generalization capabilities. This division ensures a comprehensive evaluation of the model's effectiveness and provides insights into its performance on unseen data.



```
{x} ▾ Define Features and Labels

[ ] data_texts = df["text"].to_list() # Features (not-tokenized yet)
    data_labels = df["encoded_labels"].to_list() # Labels

[ ] # Split Train and Validation data
    train_texts, val_texts, train_labels, val_labels = train_test_split(data_texts, data_labels,
                                                                    test_size=0.2, random_state=0)

[ ] # Keep some data for inference (testing)
    train_texts, test_texts, train_labels, test_labels = train_test_split(train_texts, train_labels,
                                                                    test_size=0.01, random_state=0)
```

Figure 7: Define Features and Labels

## Tokenizing with DistilBert Tokenizer:

The provided code in the figure focuses on the tokenization process using the DistilBertTokenizer from the Hugging Face transformers library. In the code, the DistilBertTokenizer is instantiated by calling the *from\_pretrained()* method with the argument 'distilbert-base-uncased'. This retrieves the pre-trained tokenizer associated with the DistilBERT model, which has been trained on a large corpus of uncased English text. The tokenization is performed on the train\_texts and val\_texts datasets using the instantiated tokenizer. The **tokenizer ()** method is applied to each dataset, with the additional parameters truncation=True and padding=True. Setting truncation=True ensures that the texts are truncated to a specified maximum length if they exceed it, while padding=True adds padding tokens to make all input sequences equal in length. The resulting tokenized encodings are stored in the variables train\_encodings and val\_encodings, respectively. These encodings represent the transformed input texts, where each word is mapped to a numerical token ID suitable for processing by the DistilBERT model. By utilizing the DistilBertTokenizer, the code enables the conversion of raw text data into a numerical representation that can be understood by the DistilBERT model. This tokenization step is crucial for preparing the data to be fed into the model for training and evaluation, allowing for efficient processing and interpretation of the text data.

```
[ ] test_size=0.01, random_state=0)

Tokenizing with DistilBert Tokenizer

Fine-tuning in the HuggingFace's transformers library involves using a pre-trained model and a tokenizer that is compatible with that model's architecture and input requirements. Each pre-trained model in transformers can be accessed using the right model class and be used with the associated tokenizer class. Since we want to use DistilBert for a classification task, we will use the DistilBertTokenizer's tokenizer class to tokenize our texts and then use TFDistilBertForSequenceClassification model class in a later section to fine-tune the pre-trained model using the output from the tokenizer.

The DistilBertTokenizer generates input_ids and attention_mask as outputs. This is what is required by a DistilBert model as its inputs.

[ ] tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

Downloading (...)solve/main/vocab.txt: 100% 232k/232k [00:00<00:00, 3.70MB/s]
Downloading (...)tokenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 1.12kB/s]
Downloading (...)ve/main/config.json: 100% 483/483 [00:00<00:00, 25.9kB/s]

[ ] train_encodings = tokenizer(train_texts, truncation=True, padding=True)

[ ] val_encodings = tokenizer(val_texts, truncation=True, padding=True)
```

Figure 8: DistilBert Tokenizer

## Create Dataset Object for TensorFlow:

To proceed with the fine-tuning phase of our model, it is necessary to convert our input encodings and labels into a TensorFlow Dataset object. This conversion step is crucial for the efficient processing and training of our model. To achieve this, we utilize the *from\_tensor\_slices* constructor method provided by TensorFlow. This method allows us to create a Dataset object directly from our input encodings and labels. Bypassing the encodings and labels as arguments to this method, the data is sliced into individual elements, with each element representing a pair of input encoding and corresponding labels. The conversion to a TensorFlow Dataset object offers several benefits. It enables seamless integration with the TensorFlow framework, providing enhanced performance and flexibility during training. Additionally, it allows for convenient batch processing and parallelization, optimizing the utilization of computational resources. By transforming our input encodings and labels into a TensorFlow Dataset object, we ensure that our data is in a suitable format for efficient training of our model. This step lays the foundation for the subsequent fine-tuning process, where the model will learn from the data and adapt its parameters to improve its performance on the specific task at hand.

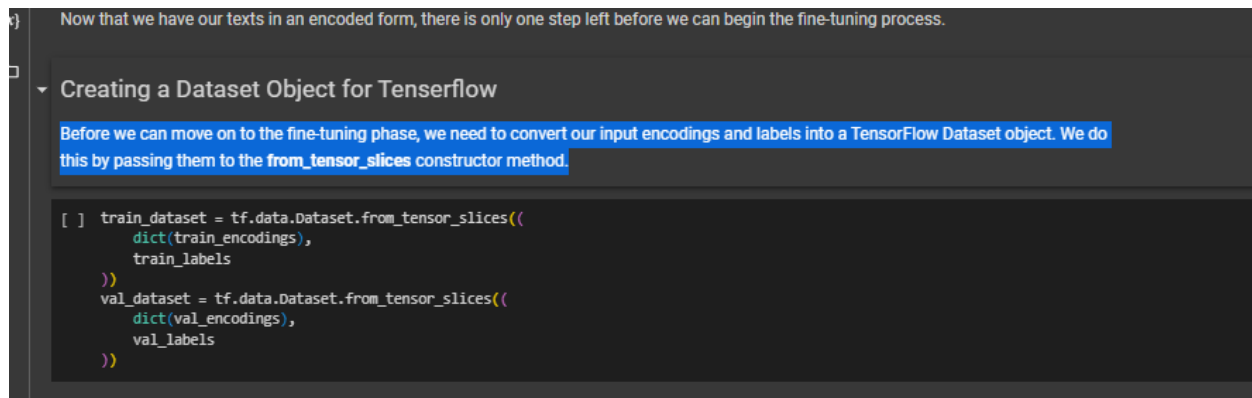
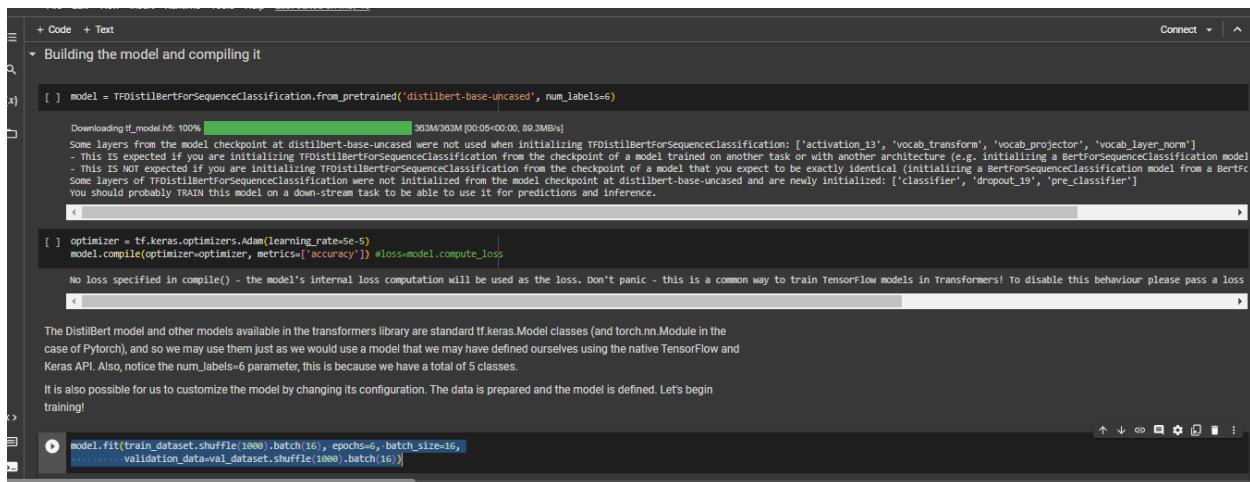


Figure 9: Dataset Object for TensorFlow

## Fine Tuning Using Native TensorFlow:

The provided code in the figure focuses on the model initialization, optimization, compilation, and training phases using the `TFDistilBertForSequenceClassification` model. In the code, the `TFDistilBertForSequenceClassification` model is instantiated using the `from_pretrained()` method with the argument 'distilbert-base-uncased'. This loads the pre-trained DistilBERT model with an uncased configuration. Additionally, the `num_labels` parameter is set to 6, indicating the number of labels in our classification task. Next, the Adam optimizer is initialized with a learning rate of  $5e-5$ . This optimizer is widely used for gradient-based optimization algorithms, and the learning rate determines the step size during parameter updates. The model is compiled using the `compile()` method, with the optimizer specified as the argument. Additionally, the desired evaluation metric, 'accuracy', is specified using the `metrics` parameter. The compilation step prepares the model for training by configuring the loss function and optimizer settings. The `fit()` method is then used to train the model. The training dataset, `train_dataset`, is shuffled and batched with a batch size of 16. The `epochs` parameter is set to 6, indicating the number of complete passes through the training dataset. The validation dataset, `val_dataset`, is also shuffled and batched for evaluation during training. By executing this code, the model undergoes the training process, where it learns from the training data to improve its performance on the classification task. The training progress is monitored, and the model's performance is evaluated on the validation dataset after each epoch.



```
[ ] model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=6)

Downloading tf_model_h5: 100% 303M/303M [00:05<00:00, 89.3MB/s]

Some layers from the model checkpoint at distilbert-base-uncased were not used when initializing TFDistilBertForSequenceClassification: ['activation_13', 'vocab_transform', 'vocab_projector', 'vocab_layer_norm']
- This is expected if you are initializing TFDistilBertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model)
- This is NOT expected if you are initializing TFDistilBertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model)
Some layers of TFDistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier', 'dropout_19', 'pre_classifier']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[ ] optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, metrics=['accuracy']) #loss=model.compute_loss

No loss specified in compile() - the model's internal loss computation will be used as the loss. Don't panic - this is a common way to train TensorFlow models in Transformers! To disable this behaviour please pass a loss

The DistilBert model and other models available in the transformers library are standard tf.keras.Model classes (and torch.nn.Module in the case of PyTorch), and so we may use them just as we would use a model that we may have defined ourselves using the native TensorFlow and Keras API. Also, notice the num_labels=6 parameter, this is because we have a total of 5 classes.

It is also possible for us to customize the model by changing its configuration. The data is prepared and the model is defined. Let's begin training!

model.fit(train_dataset.shuffle(1000).batch(16), epochs=6, batch_size=16,
        validation_data=val_dataset.shuffle(1000).batch(16))
```

Figure 10: Model Building

## Deployment of the Model:

The deployment of our model for real-time classification is achieved using Streamlit, an open-source app framework for machine learning and data science. Streamlit allows us to create and publish web applications with ease, enabling users to interact with our model in real-time.

To deploy our model using Streamlit, we create a Streamlit application that takes user input and provides the predicted sentiment category and probability scores as output. We start by importing the necessary libraries, including Streamlit and the functions and classes required for model inference.

Within the Streamlit application, we define the user interface, which typically includes input elements such as text fields or buttons. Users can input their text in the provided interface, and upon submission, the application triggers the model to classify the text.

Using the classify function we previously implemented, the application passes the user input to the model for prediction. The predicted sentiment category and probability scores are then displayed on the web interface.

Streamlit provides an interactive and user-friendly experience, allowing users to input various texts and receive immediate predictions. This deployment approach enables users to utilize our model in real-time scenarios, such as monitoring online conversations or flagging potentially harmful content.

Deploying the model with Streamlit streamlines the process of making predictions accessible to users without requiring any front-end development expertise. It enhances the practicality and usability of our model, empowering users to apply it effectively in tackling online violence against children and promoting a safer online environment.

```

1  import tensorflow as tf
2  from transformers import DistilBertTokenizer, TFDistilBertForSequenceClassification
3  import numpy as np
4  save_directory = './saved_models'
5
6  #Function that changes the encoded_label back to category
7  def predict_sentiment(num):
8      if num==0:
9          return 'bullying'
10         elif num==1:
11             return 'hate speech'
12         elif num==2:
13             return 'homophobia'
14         elif num==3:
15             return 'none'
16         elif num==4:
17             return 'racism'
18         else:
19             return 'sexism'
20
21 def classify(input_text):
22     """
23     This method takes in the input text and returns the predicted class and the probability score of each class.
24     """
25     loaded_tokenizer = DistilBertTokenizer.from_pretrained(save_directory)
26     loaded_model = TFDistilBertForSequenceClassification.from_pretrained(save_directory)
27     classes = ['bullying', 'hate_speech', 'homophobia', 'none', 'racism', 'sexism']
28     if(input_text):
29         # inference on input text
30         tokens=loaded_tokenizer(input_text, padding=True,truncation=True, return_tensors='tf')
31         logits=loaded_model.predict(dict(tokens), verbose=1).logits
32         probab=t.nn.softmax(logits, axis=1).numpy()
33         predictions=np.argmax(prob, axis=1)
34
35         probab_val = [round(p,4) for p in probab]
36         probab_values = dict(map(lambda i,j : (i,j),classes,probab_val))
37
38     return predict_sentiment(predictions), probab_values

```

Figure 11: Helper Functions for Model Deployment

```

1  import streamlit as st
2  from helper import *
3  st.title("French text classifier to detect online hate")
4
5  st.markdown(
6      """
7      The classifier detects whether the text falls under the following categories:
8      The classifier detects which of the below classes the text belongs to:
9      - bullying
10     - hate-speech
11     - homophobia
12     - racism
13     - non-toxic
14     """
15 )
16 st.header("Enter a text in French")
17
18 user_input = st.text_area("Enter a french text to classify")
19
20 if st.button('Classify'):
21     prediction, probab_val = classify(user_input)
22     st.subheader("Classification")
23     st.write("The text is classified as ",prediction)
24     st.write("The probability score of each class",probab_val)

```

Figure 12: Main Function for Model Deployment

## VI. Work Distribution:

For this project, the team works on both individual and group tasks. However, in the following you will find the tasks that were assigned to each one of the team members.

### Oumaima Laghzaoui Tasks:

- The main tasks were preprocessing the dataset and cleaning the tweets to make them ready to process them.

### Sofya Sellak Tasks:

- The main task was model evaluation and literature review.

### Soufiane Lamchoudi Tasks:

- The main tasks were model building and deployment.

### Team Tasks:

- Collecting the dataset and doing domain research.

## VI. Challenges:

The development of a language classifier for online child protection in the French language presented several notable challenges. Firstly, the scarcity of labelled training data specific to online violence against children in French posed a significant obstacle. Collecting a comprehensive and diverse dataset that accurately captured the nuances of harmful content in the French language required extensive effort and meticulous annotation. Additionally, ensuring the quality and reliability of the collected data proved to be a challenge, as the interpretation and labelling of sensitive and harmful text can be subjective. Another challenge encountered was the inherent complexity of detecting and classifying various forms of online violence against children. The French language, like any other language, exhibits a wide range of expressions, slang, and cultural references that can be employed in harmful content. Adapting the classifier to effectively handle these linguistic intricacies, including understanding sarcasm, implicit messages, or regional dialects, required continuous refinement and tuning of the model. Furthermore, the ever-evolving nature of online violence against children posed a challenge in terms of keeping the classifier up to date. As new forms of online abuse emerge and the tactics employed by perpetrators evolve, the classifier must be regularly updated to maintain its relevance and effectiveness. This necessitates ongoing monitoring and continuous training of the model to ensure it remains robust and adaptable. Lastly, deployment of the classifier in real-world online platforms also presented challenges. Integrating the classifier into existing platforms while respecting privacy and security considerations, ensuring efficient scalability, and addressing potential legal and ethical concerns required careful planning and collaboration with relevant



stakeholders. Addressing these challenges required a multidisciplinary approach, encompassing expertise in linguistics, data annotation, machine learning, and child protection. Through a combination of dedicated efforts, innovative solutions, and iterative improvements, we successfully developed a language classifier tailored for online child protection in the French language, contributing to the broader goal of creating a safer online environment for children.

## **VII. Future Work:**

Looking ahead, there is potential for further enhancements to both the model itself and its deployment. The team is currently planning a follow-up phase in which a custom extension will be developed to enable real-time prediction of text content. This extension will be deployed within an EC2 instance on Amazon Web Services (AWS). The envisioned application will operate by setting a threshold based on the similarity of suspicious text to the training data. Once this threshold is surpassed, the system will trigger an appropriate action tailored to the platform and intervention objectives. For instance, it could involve discreetly alerting the child through a chatbot, contacting a moderator, or even suspending the chat entirely. Furthermore, future expansion of the project to encompass languages other than French is highly desirable and should be actively encouraged. The idea is to include also the Moroccan Dialect Darija. By broadening the scope of language support, we can extend the reach and impact of our classification model, ensuring its relevance and effectiveness in diverse linguistic contexts. Such an expansion would reinforce our commitment to safeguarding children from online violence on a global scale.

## **VIII. Conclusion:**

The project has been a resounding success, resulting in the development and publication of a proof of concept (POC) for our classification model on the Streamlit platform. In terms of addressing the primary goal of combating online violence against children, our classifier has demonstrated satisfactory performance. The results obtained are highly promising and represent a significant initial milestone towards our collective objective of eradicating online violence against minors.

## References

- [1] K. Sørensen, “Grooming – a strategic process,” in *Is it that bad? An anthology of online sexual abuse of children and young people*. Save the Children Denmark, 2015. Online. Available: <https://resourcecentre.savethechildren.net/node/12241/pdf/is-it-really-that-bad-an-anthology-of-online-sexual-abuse-of-children-and-young-people.pdf>
- [2] S. Greijer and J. Doek, “Terminology Guidelines for the Protection of Children from Sexual Exploitation and Sexual Abuse,” *ECPAT International and ECPAT Luxembourg*, 2016. Online. Available: <http://luxembourgguidelines.org/>
- [3] “Grooming in the Eyes of a Child,” Online. Available: [https://pelastakaalapset.s3.eu-west-1.amazonaws.com/main/2021/08/03151159/grooming\\_in\\_the\\_eyes\\_of\\_a\\_child\\_2021.pdf](https://pelastakaalapset.s3.eu-west-1.amazonaws.com/main/2021/08/03151159/grooming_in_the_eyes_of_a_child_2021.pdf)
- [4] “Cyber Racism,” *Racism. No Way!*, Online. Available: <https://racismnoway.com.au/about-racism/cyber-racism/>
- [5] “Online Sexual Victimization and Risks (OSVR) and Mental Health Outcomes among Sexual Minorities,” *ScienceDirect*, 2021. Online. Available: <https://www.sciencedirect.com/science/article/pii/S0747563221000509>
- [6] “What is Hate Speech?” *United Nations*. Online. Available: <https://www.un.org/en/hate-speech/understanding-hate-speech/what-is-hate-speech>
- [7] “Bullying,” *American Psychological Association*. Online. Available: <https://www.apa.org/topics/bullying>
- [8] “Exploratory data analysis,” *Wikipedia*. Online. Available: [https://en.wikipedia.org/wiki/Exploratory\\_data\\_analysis](https://en.wikipedia.org/wiki/Exploratory_data_analysis)