lab08

June 17, 2023

1 LAB08 - NLP2

1.1 Introduction

We need to choose between `hate` and `offensive` datasets, use a specifical need for commercial use. Thus, we will use the `offensive` dataset, as HuggingFace indicates we need permissions to use the `hate` dataset for commercial use. On the datasets page on HuggingFace, we can see the following terms:

- hate (HateEval): Need permission here
- irony: Undefined
- Offensive: Undefined

The HatEval dataser is released under the Crea

The `hate` dataset is under the CC-BY-NC-4.0 license, which state the following:

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

Thus, we will use the `offensive` dataset, which is under no license.

1.2 Evaluating the dataset

1. Describe the dataset. Look at the splits, proportion of classes, and see what you can figure out by just looking at the text.

[ ]:
```
%pip install datasets
%pip install bertopic
%pip install sentence_transformers
```

[2]:
```
import numpy as np
import json
import pandas as pd
import shap
import statsmodels.stats.inter_rater as ir
import random

from typing import List, Tuple

from datasets import load_dataset
from bertopic import BERTopic
from sentence_transformers import SentenceTransformer
from umap import UMAP

from transformers import AutoModelForSequenceClassification
from transformers import pipeline
from transformers import AutoTokenizer

from sklearn.metrics import classification_report
```

```
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/utils/_clustering.py:35: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _pt_shuffle_rec(i, indexes, index_mask, partition_tree, M, pos):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/utils/_clustering.py:54: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
```

```
  def delta_minimization_order(all_masks, max_swap_size=100, num_passes=2):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/utils/_clustering.py:63: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window(order, start, length):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/utils/_clustering.py:69: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window_score_gain(masks, order, start, length):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/utils/_clustering.py:77: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _mask_delta_score(m1, m2):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/links.py:5: NumbaDeprecationWarning: The 'nopython' keyword
argument was not supplied to the 'numba.jit' decorator. The implicit default
value for this argument is currently False, but it will be changed to True in
Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def identity(x):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
```

```
packages/shap/links.py:10: NumbaDeprecationWarning: The 'nopython' keyword
argument was not supplied to the 'numba.jit' decorator. The implicit default
value for this argument is currently False, but it will be changed to True in
Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _identity_inverse(x):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/links.py:15: NumbaDeprecationWarning: The 'nopython' keyword
argument was not supplied to the 'numba.jit' decorator. The implicit default
value for this argument is currently False, but it will be changed to True in
Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def logit(x):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/links.py:20: NumbaDeprecationWarning: The 'nopython' keyword
argument was not supplied to the 'numba.jit' decorator. The implicit default
value for this argument is currently False, but it will be changed to True in
Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _logit_inverse(x):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/utils/_masked_model.py:363: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _build_fixed_single_output(averaged_outs, last_outs, outputs,
batch_positions, varying_rows, num_varying_rows, link, linearizing_weights):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
```

```
packages/shap/utils/_masked_model.py:385: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _build_fixed_multi_output(averaged_outs, last_outs, outputs,
batch_positions, varying_rows, num_varying_rows, link, linearizing_weights):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/utils/_masked_model.py:428: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _init_masks(cluster_matrix, M, indices_row_pos, indptr):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/utils/_masked_model.py:439: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _rec_fill_masks(cluster_matrix, indices_row_pos, indptr, indices, M, ind):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/maskers/_tabular.py:186: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _single_delta_mask(dind, masked_inputs, last_mask, data, x, noop_code):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
```

```
packages/shap/maskers/_tabular.py:197: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _delta_masking(masks, x, curr_delta_inds, varying_rows_out,
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/maskers/_image.py:175: NumbaDeprecationWarning: The 'nopython'
keyword argument was not supplied to the 'numba.jit' decorator. The implicit
default value for this argument is currently False, but it will be changed to
True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _jit_build_partition_tree(xmin, xmax, ymin, ymax, zmin, zmax,
total_ywidth, total_zwidth, M, clustering, q):
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/shap/explainers/_partition.py:676: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def lower_credit(i, value, M, values, clustering):
The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but
it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but
it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
```

```
The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but
it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but
it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but
it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but
it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
```

[34]:
```
dataset = load_dataset("tweet_eval", "offensive")
```

```
Found cached dataset tweet_eval (/Users/quentinfisch/.cache/huggingface/datasets
/tweet_eval/offensive/1.1.0/12aee5282b8784f3e95459466db4cdf45c6bf49719c25cdb0743
d71ed0410343)

  0%|          | 0/3 [00:00<?, ?it/s]
```

[169]:
```
print(dataset)
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 11916
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 860
    })
    validation: Dataset({
```

```
        features: ['text', 'label'],
        num_rows: 1324
    })
})
```

The dataset has 3 splits: `train`, `validation` and `test`. The `train` split contains 11916 samples, the `validation` split contains 1324 samples and the `test` split contains 860 samples. Let's take a look at some samples

[170]:
```
NB_PRINT_SAMPLES = 5
for i in range(NB_PRINT_SAMPLES):
    print(dataset['train'][i])
```

```
{'text': '@user Bono… who cares. Soon people will understand that they gain
nothing from following a phony celebrity. Become a Leader of your people instead
or help and support your fellow countrymen.', 'label': 0}
{'text': '@user Eight years the republicans denied obama's picks. Breitbarters
outrage is as phony as their fake president.', 'label': 1}
{'text': '@user Get him some line help. He is gonna be just fine. As the game
went on you could see him progressing more with his reads. He brought what has
been missing. The deep ball presence. Now he just needs a little more time',
'label': 0}
{'text': '@user @user She is great. Hi Fiona!', 'label': 0}
{'text': "@user She has become a parody unto herself? She has certainly taken
some heat for being such an…well idiot. Could be optic too  Who know with
Liberals  They're all optics.  No substance", 'label': 1}
```

This data is a collection of tweets, with a label indicating if the tweet is offensive or not. The label is a boolean, `0` for non-offensive and `1` for offensive. Tweets that are classified as offensive seem to be mostly insults, or tweets that are not politically correct. Some tweets are also classified as offensive because they are not politically correct, but are not insults. The last tweet printed above is classified as offensive because it contains the word "idiot".

2. Use BERTopic to extract the topics within the data, and the main topics within each class.

[35]:
```
SEED = 42
umap_model = UMAP(random_state=SEED)

model = SentenceTransformer('all-MiniLM-L6-v2')

topic_model = BERTopic(language="english", calculate_probabilities=True,␣
 ↪embedding_model=model, umap_model=umap_model)
topics, probs = topic_model.fit_transform(dataset['train']['text'])
```

[172]:
```
topic_model.get_topic_freq().head(10)
```

[172]:

	Topic	Count
3	0	3939
4	-1	3069

```
5        1    991
28       2    382
13       3    317
27       4    262
21       5    195
14       6    161
25       7    153
17       8    113
```

Let's visualize the topics within the data, and the main topics within each class (20 topics)

[173]:
```
topic_per_class = topic_model.topics_per_class(dataset['train']['text'], topics)
topic_model.visualize_topics_per_class(topic_per_class, top_n_topics=20)
```

3. What do you think about the results? How do you think it could impact a model trained on these data? The topic with the highest frequency is `she_you_is_he`, which is just a list of pronouns. It makes sense that this topic is the most frequent, as pronouns are very common in the english language. Then, the topics are mainly related to diverse subjects around politics. Some subjects are surprising such as `nfl_football_he_game`, but it makes sense that controversial tweets might appear on this topic.

The model trained on these data might be biased towards politics, as the topics are mainly related to politics. This might be a problem if the model is used to classify tweets that are not related to politics.

4. (Bonus) By default, BERTopic extracts single keywords. Play with the model to extract bigrams or more. See if you can go deeper in your analysis.

[174]:
```
topic_model.visualize_term_rank()
```

The c-TF-IDF score is a measure of how important a word is in a topic. The higher the score, the more important the word is in the topic. We see that the topic with the highest score is topic 50 (bono_u2_asshold_his_tax_dublin_inc), followed by topics related to sexual words, popular brands or classic words.

Let's now see what topics are extracted when we use bigrams.

[175]:
```
topic_model = BERTopic(language="english", calculate_probabilities=True,␣
↪embedding_model=model, umap_model=umap_model, n_gram_range=(1, 2))
topics, probs = topic_model.fit_transform(dataset['train']['text'])
topic_per_class = topic_model.topics_per_class(dataset['train']['text'], topics)
topic_model.visualize_topics_per_class(topic_per_class, top_n_topics=20)
```

There is not a big difference using bigrams, topics names are more repetitive, but the topics are still in the same domain as before.

1.3 Evaluate a model

1. Evaluate their model on the test split of the dataset you picked, using precision, recall, and F1-score. So we will pick the RoBERTa model that has been fine-tuned on the offensive dataset. Let's use it through the HugginFace library

[176]:
```
!rm -rf cardiffnlp
```

[27]:
```
MODEL = f"cardiffnlp/twitter-roberta-base-offensive"

tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
```

1. Evaluate their model on the test split of the dataset you picked, using precision, recall, and F1-score.

[28]:
```
roberta_pipeline = pipeline(
    "text-classification",
    model=model,
    tokenizer=tokenizer
)
```

```
Xformers is not installed correctly. If you want to use
memorry_efficient_attention to accelerate training use the following command to
install Xformers
pip install xformers.
```

[179]:
```
predictions = roberta_pipeline(dataset['test']['text'])
predictions
```

[179]:
```
[{'label': 'offensive', 'score': 0.856173038482666},
 {'label': 'offensive', 'score': 0.6463530659675598},
 {'label': 'non-offensive', 'score': 0.5957751274108887},
 {'label': 'non-offensive', 'score': 0.7610461711883545},
 {'label': 'non-offensive', 'score': 0.9149481654167175},
 {'label': 'non-offensive', 'score': 0.9536920189857483},
 {'label': 'non-offensive', 'score': 0.7719836831092834},
 {'label': 'non-offensive', 'score': 0.9647331237792969},
 {'label': 'offensive', 'score': 0.8978868722915649},
 {'label': 'non-offensive', 'score': 0.505090594291687},
 {'label': 'offensive', 'score': 0.7437807321548462},
 {'label': 'non-offensive', 'score': 0.9143494963645935},
 {'label': 'non-offensive', 'score': 0.9382737278938293},
 {'label': 'non-offensive', 'score': 0.6899848580360413},
 {'label': 'non-offensive', 'score': 0.5975456237792969},
 {'label': 'offensive', 'score': 0.8460323810577393},
 {'label': 'non-offensive', 'score': 0.7833716869354248},
 {'label': 'non-offensive', 'score': 0.951020359992981},
 {'label': 'offensive', 'score': 0.6954761743545532},
 {'label': 'offensive', 'score': 0.6148088574409485},
 {'label': 'non-offensive', 'score': 0.9211805462837219},
 {'label': 'non-offensive', 'score': 0.9613785743713379},
 {'label': 'non-offensive', 'score': 0.9272322654724121},
 {'label': 'offensive', 'score': 0.859533965587616},
```

{'label': 'offensive', 'score': 0.6659108996391296},
{'label': 'non-offensive', 'score': 0.8005006313323975},
{'label': 'non-offensive', 'score': 0.888978123664856},
{'label': 'non-offensive', 'score': 0.8529670238494873},
{'label': 'non-offensive', 'score': 0.8805304765701294},
{'label': 'non-offensive', 'score': 0.832657516002655},
{'label': 'non-offensive', 'score': 0.9255490303039551},
{'label': 'non-offensive', 'score': 0.5920920372009277},
{'label': 'offensive', 'score': 0.6086819767951965},
{'label': 'non-offensive', 'score': 0.7895367741584778},
{'label': 'non-offensive', 'score': 0.7168017029762268},
{'label': 'non-offensive', 'score': 0.5830867290496826},
{'label': 'non-offensive', 'score': 0.8905478119850159},
{'label': 'non-offensive', 'score': 0.6870761513710022},
{'label': 'non-offensive', 'score': 0.9002664685249329},
{'label': 'non-offensive', 'score': 0.9598767757415771},
{'label': 'offensive', 'score': 0.7929825782775879},
{'label': 'offensive', 'score': 0.8564584255218506},
{'label': 'offensive', 'score': 0.8085554242134094},
{'label': 'non-offensive', 'score': 0.9607175588607788},
{'label': 'offensive', 'score': 0.7872814536094666},
{'label': 'offensive', 'score': 0.8807374238967896},
{'label': 'non-offensive', 'score': 0.8646060824394226},
{'label': 'offensive', 'score': 0.8747128248214722},
{'label': 'non-offensive', 'score': 0.7228529453277588},
{'label': 'non-offensive', 'score': 0.7152253985404968},
{'label': 'non-offensive', 'score': 0.8877006769180298},
{'label': 'non-offensive', 'score': 0.9071815609931946},
{'label': 'non-offensive', 'score': 0.8986267447471619},
{'label': 'non-offensive', 'score': 0.6094287633895874},
{'label': 'non-offensive', 'score': 0.9176159501075745},
{'label': 'non-offensive', 'score': 0.8822653293609619},
{'label': 'non-offensive', 'score': 0.8496428728103638},
{'label': 'non-offensive', 'score': 0.9567338824272156},
{'label': 'non-offensive', 'score': 0.6381656527519226},
{'label': 'non-offensive', 'score': 0.9627326130867004},
{'label': 'non-offensive', 'score': 0.8923089504241943},
{'label': 'non-offensive', 'score': 0.8799210786819458},
{'label': 'non-offensive', 'score': 0.6779547929763794},
{'label': 'offensive', 'score': 0.7261376976966858},
{'label': 'offensive', 'score': 0.6632963418960571},
{'label': 'non-offensive', 'score': 0.5622185468673706},
{'label': 'non-offensive', 'score': 0.7111819982528687},
{'label': 'non-offensive', 'score': 0.6981507539749146},
{'label': 'non-offensive', 'score': 0.6280423402786255},
{'label': 'offensive', 'score': 0.6128137707710266},
{'label': 'non-offensive', 'score': 0.9325283169746399},

{'label': 'offensive', 'score': 0.8811163902282715},
{'label': 'non-offensive', 'score': 0.5353884100914001},
{'label': 'non-offensive', 'score': 0.9125734567642212},
{'label': 'non-offensive', 'score': 0.9384405016899109},
{'label': 'non-offensive', 'score': 0.7982913255691528},
{'label': 'non-offensive', 'score': 0.7532820701599121},
{'label': 'offensive', 'score': 0.9006553888320923},
{'label': 'non-offensive', 'score': 0.9413447976112366},
{'label': 'non-offensive', 'score': 0.7884467244148254},
{'label': 'non-offensive', 'score': 0.6244893670082092},
{'label': 'offensive', 'score': 0.5826718807220459},
{'label': 'non-offensive', 'score': 0.7817951440811157},
{'label': 'non-offensive', 'score': 0.9468815326690674},
{'label': 'non-offensive', 'score': 0.8110015988349915},
{'label': 'non-offensive', 'score': 0.7197837233543396},
{'label': 'non-offensive', 'score': 0.8770596385002136},
{'label': 'non-offensive', 'score': 0.7743412852287292},
{'label': 'non-offensive', 'score': 0.9729042649269104},
{'label': 'non-offensive', 'score': 0.9490767121315002},
{'label': 'non-offensive', 'score': 0.902915358543396},
{'label': 'non-offensive', 'score': 0.7529154419898987},
{'label': 'non-offensive', 'score': 0.9693130850791931},
{'label': 'non-offensive', 'score': 0.8198955655097961},
{'label': 'non-offensive', 'score': 0.8196569085121155},
{'label': 'offensive', 'score': 0.5213666558265686},
{'label': 'non-offensive', 'score': 0.9338217973709106},
{'label': 'offensive', 'score': 0.8366519808769226},
{'label': 'non-offensive', 'score': 0.9089363217353821},
{'label': 'offensive', 'score': 0.5521014928817749},
{'label': 'non-offensive', 'score': 0.7843078970909119},
{'label': 'non-offensive', 'score': 0.9632238745689392},
{'label': 'non-offensive', 'score': 0.5262848138809204},
{'label': 'non-offensive', 'score': 0.9495053291320801},
{'label': 'non-offensive', 'score': 0.7394158840179443},
{'label': 'non-offensive', 'score': 0.963688313961029},
{'label': 'non-offensive', 'score': 0.9206315875053406},
{'label': 'offensive', 'score': 0.8659908175468445},
{'label': 'non-offensive', 'score': 0.8537524342536926},
{'label': 'offensive', 'score': 0.811662495136261},
{'label': 'non-offensive', 'score': 0.8177697658538818},
{'label': 'non-offensive', 'score': 0.7034445405006409},
{'label': 'non-offensive', 'score': 0.817930281162262},
{'label': 'non-offensive', 'score': 0.8469158411026001},
{'label': 'non-offensive', 'score': 0.7968279123306274},
{'label': 'offensive', 'score': 0.8153446316719055},
{'label': 'non-offensive', 'score': 0.5297676920890808},
{'label': 'non-offensive', 'score': 0.8634473085403442},

{'label': 'offensive', 'score': 0.7103620171546936},
{'label': 'non-offensive', 'score': 0.5440486073493958},
{'label': 'non-offensive', 'score': 0.8350384831428528},
{'label': 'non-offensive', 'score': 0.6329460144042969},
{'label': 'non-offensive', 'score': 0.895766019821167},
{'label': 'non-offensive', 'score': 0.8325799107551575},
{'label': 'non-offensive', 'score': 0.9437342286109924},
{'label': 'non-offensive', 'score': 0.9381805062294006},
{'label': 'non-offensive', 'score': 0.8568535447120667},
{'label': 'non-offensive', 'score': 0.956928014755249},
{'label': 'non-offensive', 'score': 0.7905188202857971},
{'label': 'non-offensive', 'score': 0.6041426062583923},
{'label': 'non-offensive', 'score': 0.7086198329925537},
{'label': 'offensive', 'score': 0.649199903011322},
{'label': 'offensive', 'score': 0.871320366859436},
{'label': 'non-offensive', 'score': 0.8954464793205261},
{'label': 'non-offensive', 'score': 0.8920491337776184},
{'label': 'non-offensive', 'score': 0.5658277273178101},
{'label': 'non-offensive', 'score': 0.5189424157142639},
{'label': 'non-offensive', 'score': 0.6660752892494202},
{'label': 'non-offensive', 'score': 0.774082362651825},
{'label': 'non-offensive', 'score': 0.7921082973480225},
{'label': 'offensive', 'score': 0.5211134552955627},
{'label': 'non-offensive', 'score': 0.7442581653594971},
{'label': 'non-offensive', 'score': 0.9638006687164307},
{'label': 'non-offensive', 'score': 0.9419043660163879},
{'label': 'non-offensive', 'score': 0.9113001227378845},
{'label': 'non-offensive', 'score': 0.8180261850357056},
{'label': 'non-offensive', 'score': 0.7933775782585144},
{'label': 'non-offensive', 'score': 0.5861366987228394},
{'label': 'non-offensive', 'score': 0.5466554164886475},
{'label': 'non-offensive', 'score': 0.9063020944595337},
{'label': 'offensive', 'score': 0.5894610285758972},
{'label': 'non-offensive', 'score': 0.7297394871711731},
{'label': 'offensive', 'score': 0.8354783058166504},
{'label': 'non-offensive', 'score': 0.8747234344482422},
{'label': 'non-offensive', 'score': 0.9477459192276001},
{'label': 'non-offensive', 'score': 0.8575080633163452},
{'label': 'non-offensive', 'score': 0.8984280228614807},
{'label': 'offensive', 'score': 0.6692493557929993},
{'label': 'non-offensive', 'score': 0.9403613209724426},
{'label': 'non-offensive', 'score': 0.9420992136001587},
{'label': 'non-offensive', 'score': 0.7596739530563354},
{'label': 'non-offensive', 'score': 0.7340707182884216},
{'label': 'offensive', 'score': 0.8357967734336853},
{'label': 'non-offensive', 'score': 0.970574140548706},
{'label': 'offensive', 'score': 0.7510614395141602},

{'label': 'non-offensive', 'score': 0.796035647392273},
{'label': 'non-offensive', 'score': 0.9665788412094116},
{'label': 'offensive', 'score': 0.5260827541351318},
{'label': 'non-offensive', 'score': 0.6838005185127258},
{'label': 'offensive', 'score': 0.7566713094711304},
{'label': 'offensive', 'score': 0.7728422284126282},
{'label': 'non-offensive', 'score': 0.8088419437408447},
{'label': 'offensive', 'score': 0.812073826789856},
{'label': 'offensive', 'score': 0.5438273549079895},
{'label': 'non-offensive', 'score': 0.8925508856773376},
{'label': 'offensive', 'score': 0.5793178081512451},
{'label': 'non-offensive', 'score': 0.8900401592254639},
{'label': 'non-offensive', 'score': 0.919756293296814},
{'label': 'non-offensive', 'score': 0.7605836391448975},
{'label': 'non-offensive', 'score': 0.8877191543579102},
{'label': 'non-offensive', 'score': 0.7382698655128479},
{'label': 'non-offensive', 'score': 0.9481533169746399},
{'label': 'non-offensive', 'score': 0.9678386449813843},
{'label': 'non-offensive', 'score': 0.8794602155685425},
{'label': 'offensive', 'score': 0.5213223695755005},
{'label': 'offensive', 'score': 0.8659946918487549},
{'label': 'non-offensive', 'score': 0.9194746613502502},
{'label': 'offensive', 'score': 0.8206168413162231},
{'label': 'non-offensive', 'score': 0.7786996960639954},
{'label': 'non-offensive', 'score': 0.7998613715171814},
{'label': 'non-offensive', 'score': 0.66825270652771},
{'label': 'non-offensive', 'score': 0.7131672501564026},
{'label': 'non-offensive', 'score': 0.8758537173271179},
{'label': 'non-offensive', 'score': 0.8071776032447815},
{'label': 'non-offensive', 'score': 0.6661297678947449},
{'label': 'non-offensive', 'score': 0.9424570202827454},
{'label': 'non-offensive', 'score': 0.7717088460922241},
{'label': 'non-offensive', 'score': 0.7224788665771484},
{'label': 'non-offensive', 'score': 0.8906568884849548},
{'label': 'non-offensive', 'score': 0.7085135579109192},
{'label': 'non-offensive', 'score': 0.9688556790351868},
{'label': 'non-offensive', 'score': 0.5352303981781006},
{'label': 'non-offensive', 'score': 0.5357815623283386},
{'label': 'non-offensive', 'score': 0.8583065271377563},
{'label': 'offensive', 'score': 0.6699379682540894},
{'label': 'offensive', 'score': 0.701668918132782},
{'label': 'non-offensive', 'score': 0.9596728682518005},
{'label': 'non-offensive', 'score': 0.9133543372154236},
{'label': 'non-offensive', 'score': 0.8600544333457947},
{'label': 'offensive', 'score': 0.7671756744384766},
{'label': 'offensive', 'score': 0.7014821767807007},
{'label': 'non-offensive', 'score': 0.7130832076072693},

{'label': 'non-offensive', 'score': 0.7439190745353699},
{'label': 'offensive', 'score': 0.5096213221549988},
{'label': 'non-offensive', 'score': 0.706321656703949},
{'label': 'non-offensive', 'score': 0.5330220460891724},
{'label': 'offensive', 'score': 0.8115618228912354},
{'label': 'offensive', 'score': 0.5494672060012817},
{'label': 'non-offensive', 'score': 0.9458969831466675},
{'label': 'non-offensive', 'score': 0.9572929739952087},
{'label': 'non-offensive', 'score': 0.8219975233078003},
{'label': 'non-offensive', 'score': 0.7878559231758118},
{'label': 'non-offensive', 'score': 0.9180790781974792},
{'label': 'non-offensive', 'score': 0.7705109119415283},
{'label': 'non-offensive', 'score': 0.9781582355499268},
{'label': 'non-offensive', 'score': 0.9604655504226685},
{'label': 'non-offensive', 'score': 0.6272556185722351},
{'label': 'non-offensive', 'score': 0.826004147529602},
{'label': 'non-offensive', 'score': 0.9081719517707825},
{'label': 'non-offensive', 'score': 0.9443882703781128},
{'label': 'offensive', 'score': 0.6546138525009155},
{'label': 'offensive', 'score': 0.6115046143531799},
{'label': 'non-offensive', 'score': 0.5467252731323242},
{'label': 'offensive', 'score': 0.864190399646759},
{'label': 'non-offensive', 'score': 0.8991748690605164},
{'label': 'non-offensive', 'score': 0.9626563787460327},
{'label': 'offensive', 'score': 0.7211799621582031},
{'label': 'non-offensive', 'score': 0.7961703538894653},
{'label': 'non-offensive', 'score': 0.8196569085121155},
{'label': 'non-offensive', 'score': 0.7071813941001892},
{'label': 'non-offensive', 'score': 0.7421274781227112},
{'label': 'non-offensive', 'score': 0.9747382402420044},
{'label': 'non-offensive', 'score': 0.9220367074012756},
{'label': 'offensive', 'score': 0.7846319079399109},
{'label': 'non-offensive', 'score': 0.9598767757415771},
{'label': 'offensive', 'score': 0.9212592244148254},
{'label': 'non-offensive', 'score': 0.9634412527084351},
{'label': 'offensive', 'score': 0.5068149566650391},
{'label': 'non-offensive', 'score': 0.7883318662643433},
{'label': 'offensive', 'score': 0.8967992067337036},
{'label': 'offensive', 'score': 0.5644499659538269},
{'label': 'non-offensive', 'score': 0.574522078037262},
{'label': 'non-offensive', 'score': 0.8695482015609741},
{'label': 'offensive', 'score': 0.7471998333930969},
{'label': 'non-offensive', 'score': 0.9544573426246643},
{'label': 'non-offensive', 'score': 0.7230839729309082},
{'label': 'offensive', 'score': 0.657132089138031},
{'label': 'non-offensive', 'score': 0.5831230282783508},
{'label': 'offensive', 'score': 0.8674396276473999},

{'label': 'non-offensive', 'score': 0.8422994017601013},
{'label': 'non-offensive', 'score': 0.8638830184936523},
{'label': 'non-offensive', 'score': 0.8327128887176514},
{'label': 'non-offensive', 'score': 0.9295574426651001},
{'label': 'offensive', 'score': 0.5390884280204773},
{'label': 'offensive', 'score': 0.7727183699607849},
{'label': 'non-offensive', 'score': 0.5419759154319763},
{'label': 'non-offensive', 'score': 0.8944764137268066},
{'label': 'non-offensive', 'score': 0.8971509337425232},
{'label': 'offensive', 'score': 0.8106285333633423},
{'label': 'non-offensive', 'score': 0.8308777213096619},
{'label': 'offensive', 'score': 0.9143260717391968},
{'label': 'non-offensive', 'score': 0.896653413772583},
{'label': 'non-offensive', 'score': 0.747501015663147},
{'label': 'non-offensive', 'score': 0.6010663509368896},
{'label': 'offensive', 'score': 0.8638688325881958},
{'label': 'non-offensive', 'score': 0.9200100898742676},
{'label': 'non-offensive', 'score': 0.7578940391540527},
{'label': 'non-offensive', 'score': 0.9469349384307861},
{'label': 'offensive', 'score': 0.8744546175003052},
{'label': 'non-offensive', 'score': 0.8753796219825745},
{'label': 'non-offensive', 'score': 0.940129280090332},
{'label': 'non-offensive', 'score': 0.9618083834648132},
{'label': 'non-offensive', 'score': 0.8775813579559326},
{'label': 'non-offensive', 'score': 0.9466975927352905},
{'label': 'offensive', 'score': 0.8881096839904785},
{'label': 'non-offensive', 'score': 0.9543339610099792},
{'label': 'offensive', 'score': 0.5587500929832458},
{'label': 'non-offensive', 'score': 0.8736361265182495},
{'label': 'offensive', 'score': 0.6735706329345703},
{'label': 'non-offensive', 'score': 0.9327573776245117},
{'label': 'offensive', 'score': 0.6854352951049805},
{'label': 'non-offensive', 'score': 0.5194258689880371},
{'label': 'non-offensive', 'score': 0.5959472060203552},
{'label': 'offensive', 'score': 0.8108363747596741},
{'label': 'non-offensive', 'score': 0.776725709438324},
{'label': 'non-offensive', 'score': 0.5828918814659119},
{'label': 'offensive', 'score': 0.8937872648239136},
{'label': 'non-offensive', 'score': 0.8735195994377136},
{'label': 'non-offensive', 'score': 0.7431633472442627},
{'label': 'non-offensive', 'score': 0.8211655616760254},
{'label': 'non-offensive', 'score': 0.933017373085022},
{'label': 'non-offensive', 'score': 0.8592702150344849},
{'label': 'non-offensive', 'score': 0.6876612901687622},
{'label': 'non-offensive', 'score': 0.9458333253860474},
{'label': 'non-offensive', 'score': 0.8215388059616089},
{'label': 'non-offensive', 'score': 0.942115306854248},

{'label': 'non-offensive', 'score': 0.7306233644485474},
{'label': 'non-offensive', 'score': 0.7003610134124756},
{'label': 'non-offensive', 'score': 0.6231130957603455},
{'label': 'non-offensive', 'score': 0.686599612236023},
{'label': 'non-offensive', 'score': 0.7677751779556274},
{'label': 'offensive', 'score': 0.6484770774841309},
{'label': 'offensive', 'score': 0.5975080728530884},
{'label': 'non-offensive', 'score': 0.9717153906822205},
{'label': 'non-offensive', 'score': 0.8573783040046692},
{'label': 'non-offensive', 'score': 0.9600428938865662},
{'label': 'non-offensive', 'score': 0.8168665766716003},
{'label': 'non-offensive', 'score': 0.9308887124061584},
{'label': 'non-offensive', 'score': 0.7957576513290405},
{'label': 'offensive', 'score': 0.5696908831596375},
{'label': 'non-offensive', 'score': 0.9598820805549622},
{'label': 'offensive', 'score': 0.7215139865875244},
{'label': 'non-offensive', 'score': 0.8578736782073975},
{'label': 'non-offensive', 'score': 0.9244564771652222},
{'label': 'non-offensive', 'score': 0.9284777641296387},
{'label': 'non-offensive', 'score': 0.6267338991165161},
{'label': 'offensive', 'score': 0.6692473888397217},
{'label': 'non-offensive', 'score': 0.8970034718513489},
{'label': 'non-offensive', 'score': 0.957027792930603},
{'label': 'offensive', 'score': 0.8328858613967896},
{'label': 'offensive', 'score': 0.6654389500617981},
{'label': 'non-offensive', 'score': 0.8302890062332153},
{'label': 'non-offensive', 'score': 0.7953725457191467},
{'label': 'non-offensive', 'score': 0.7666193246841431},
{'label': 'non-offensive', 'score': 0.8905996680259705},
{'label': 'offensive', 'score': 0.5180891752243042},
{'label': 'non-offensive', 'score': 0.9284366369247437},
{'label': 'offensive', 'score': 0.6543797850608826},
{'label': 'non-offensive', 'score': 0.9126635193824768},
{'label': 'offensive', 'score': 0.6509085893630981},
{'label': 'offensive', 'score': 0.7710650563240051},
{'label': 'offensive', 'score': 0.7159867286682129},
{'label': 'non-offensive', 'score': 0.6982142925262451},
{'label': 'non-offensive', 'score': 0.8875365257263184},
{'label': 'non-offensive', 'score': 0.9508486390113831},
{'label': 'non-offensive', 'score': 0.9334331750869751},
{'label': 'non-offensive', 'score': 0.957865118980407 7},
{'label': 'non-offensive', 'score': 0.8654608130455017},
{'label': 'non-offensive', 'score': 0.9177231192588806},
{'label': 'non-offensive', 'score': 0.9027013778686523},
{'label': 'non-offensive', 'score': 0.8740304112434387},
{'label': 'non-offensive', 'score': 0.8444955348968506},
{'label': 'non-offensive', 'score': 0.9475903511047363},

{'label': 'non-offensive', 'score': 0.7827322483062744},
{'label': 'offensive', 'score': 0.5314677953720093},
{'label': 'non-offensive', 'score': 0.9316120147705078},
{'label': 'non-offensive', 'score': 0.8890028595924377},
{'label': 'offensive', 'score': 0.8847941160202026},
{'label': 'non-offensive', 'score': 0.9181578755378723},
{'label': 'non-offensive', 'score': 0.8407604694366455},
{'label': 'non-offensive', 'score': 0.9297388792037964},
{'label': 'non-offensive', 'score': 0.9071236848831177},
{'label': 'non-offensive', 'score': 0.885219931602478},
{'label': 'non-offensive', 'score': 0.7988125681877136},
{'label': 'non-offensive', 'score': 0.7710421085357666},
{'label': 'non-offensive', 'score': 0.914920449256897},
{'label': 'non-offensive', 'score': 0.8186503648757935},
{'label': 'non-offensive', 'score': 0.9257184863090515},
{'label': 'non-offensive', 'score': 0.9582757353782654},
{'label': 'non-offensive', 'score': 0.9010458588600159},
{'label': 'non-offensive', 'score': 0.8828783631324768},
{'label': 'non-offensive', 'score': 0.8380753397941589},
{'label': 'non-offensive', 'score': 0.9118924140930176},
{'label': 'offensive', 'score': 0.5982977151870728},
{'label': 'offensive', 'score': 0.5962234139442444},
{'label': 'non-offensive', 'score': 0.7622970342636108},
{'label': 'non-offensive', 'score': 0.7860060334205627},
{'label': 'non-offensive', 'score': 0.8565376996994019},
{'label': 'non-offensive', 'score': 0.9321754574775696},
{'label': 'non-offensive', 'score': 0.9527190327644348},
{'label': 'offensive', 'score': 0.8947945833206177},
{'label': 'non-offensive', 'score': 0.5827414989471436},
{'label': 'non-offensive', 'score': 0.9683616161346436},
{'label': 'non-offensive', 'score': 0.8715022206306458},
{'label': 'non-offensive', 'score': 0.9445766806602478},
{'label': 'non-offensive', 'score': 0.9338706731796265},
{'label': 'non-offensive', 'score': 0.663296639919281},
{'label': 'non-offensive', 'score': 0.8624588847160339},
{'label': 'non-offensive', 'score': 0.8689831495285034},
{'label': 'non-offensive', 'score': 0.8545988202095032},
{'label': 'non-offensive', 'score': 0.7707483768463135},
{'label': 'non-offensive', 'score': 0.8498957753181458},
{'label': 'non-offensive', 'score': 0.5546286702156067},
{'label': 'non-offensive', 'score': 0.8966789245605469},
{'label': 'non-offensive', 'score': 0.9206033945083618},
{'label': 'offensive', 'score': 0.6490565538406372},
{'label': 'non-offensive', 'score': 0.8422994017601013},
{'label': 'non-offensive', 'score': 0.8912426829338074},
{'label': 'offensive', 'score': 0.8317632079124451},
{'label': 'non-offensive', 'score': 0.8028572201728821},

{'label': 'non-offensive', 'score': 0.8184941411018372},
{'label': 'non-offensive', 'score': 0.652527391910553},
{'label': 'non-offensive', 'score': 0.899795413017273},
{'label': 'non-offensive', 'score': 0.6901146769523621},
{'label': 'offensive', 'score': 0.8893686532974243},
{'label': 'non-offensive', 'score': 0.9110732674598694},
{'label': 'non-offensive', 'score': 0.9634975790977478},
{'label': 'offensive', 'score': 0.7273526787757874},
{'label': 'non-offensive', 'score': 0.9074219465255737},
{'label': 'non-offensive', 'score': 0.7024166584014893},
{'label': 'non-offensive', 'score': 0.9598140716552734},
{'label': 'non-offensive', 'score': 0.943040668964386},
{'label': 'non-offensive', 'score': 0.8882394433021545},
{'label': 'offensive', 'score': 0.5667027831077576},
{'label': 'offensive', 'score': 0.590703010559082},
{'label': 'non-offensive', 'score': 0.6273155808448792},
{'label': 'offensive', 'score': 0.628393828868866},
{'label': 'offensive', 'score': 0.8847941160202026},
{'label': 'non-offensive', 'score': 0.9112220406532288},
{'label': 'non-offensive', 'score': 0.9258099794387817},
{'label': 'non-offensive', 'score': 0.9740973711013794},
{'label': 'non-offensive', 'score': 0.9297930002212524},
{'label': 'non-offensive', 'score': 0.9342532753944397},
{'label': 'non-offensive', 'score': 0.7214885950088501},
{'label': 'non-offensive', 'score': 0.9610843062400818},
{'label': 'non-offensive', 'score': 0.9441562294960022},
{'label': 'non-offensive', 'score': 0.7759115695953369},
{'label': 'non-offensive', 'score': 0.8192125558853149},
{'label': 'non-offensive', 'score': 0.8420127630233765},
{'label': 'non-offensive', 'score': 0.7843393087387085},
{'label': 'non-offensive', 'score': 0.8660610914230347},
{'label': 'offensive', 'score': 0.8386447429656982},
{'label': 'non-offensive', 'score': 0.9591578841209412},
{'label': 'non-offensive', 'score': 0.7760999798774719},
{'label': 'non-offensive', 'score': 0.9715202450752258},
{'label': 'non-offensive', 'score': 0.9361696243286133},
{'label': 'non-offensive', 'score': 0.9054701924324036},
{'label': 'offensive', 'score': 0.6855992674827576},
{'label': 'offensive', 'score': 0.8033919334411621},
{'label': 'non-offensive', 'score': 0.8251733183860779},
{'label': 'non-offensive', 'score': 0.9306417107582092},
{'label': 'offensive', 'score': 0.9518308639526367},
{'label': 'non-offensive', 'score': 0.913827657699585},
{'label': 'non-offensive', 'score': 0.9300767779350281},
{'label': 'non-offensive', 'score': 0.6459757089614868},
{'label': 'offensive', 'score': 0.6931162476539612},
{'label': 'non-offensive', 'score': 0.6203073263168335},

{'label': 'non-offensive', 'score': 0.69371497631073},
{'label': 'non-offensive', 'score': 0.8553194403648376},
{'label': 'non-offensive', 'score': 0.8734447360038757},
{'label': 'non-offensive', 'score': 0.9338760375976562},
{'label': 'non-offensive', 'score': 0.8783687949180603},
{'label': 'non-offensive', 'score': 0.698531985282898},
{'label': 'non-offensive', 'score': 0.9310998320579529},
{'label': 'non-offensive', 'score': 0.9581574201583862},
{'label': 'offensive', 'score': 0.8939998745918274},
{'label': 'non-offensive', 'score': 0.6092102527618408},
{'label': 'offensive', 'score': 0.5508459806442261},
{'label': 'non-offensive', 'score': 0.8069246411323547},
{'label': 'non-offensive', 'score': 0.8802856206893921},
{'label': 'non-offensive', 'score': 0.771723210811615},
{'label': 'non-offensive', 'score': 0.5214255452156067},
{'label': 'non-offensive', 'score': 0.8381596207618713},
{'label': 'non-offensive', 'score': 0.9505134224891663},
{'label': 'non-offensive', 'score': 0.9742013216018677},
{'label': 'non-offensive', 'score': 0.7872870564460754},
{'label': 'non-offensive', 'score': 0.947471559047699},
{'label': 'non-offensive', 'score': 0.5463524460792542},
{'label': 'offensive', 'score': 0.6892694234848022},
{'label': 'non-offensive', 'score': 0.860987663269043},
{'label': 'non-offensive', 'score': 0.7477929592132568},
{'label': 'non-offensive', 'score': 0.7527275681495667},
{'label': 'non-offensive', 'score': 0.9601522088050842},
{'label': 'non-offensive', 'score': 0.9241985082626343},
{'label': 'non-offensive', 'score': 0.9228795766830444},
{'label': 'non-offensive', 'score': 0.8849829435348511},
{'label': 'non-offensive', 'score': 0.7943196296691895},
{'label': 'non-offensive', 'score': 0.9641134142875671},
{'label': 'non-offensive', 'score': 0.6359685063362122},
{'label': 'non-offensive', 'score': 0.9335331916809082},
{'label': 'non-offensive', 'score': 0.9404245018959045},
{'label': 'non-offensive', 'score': 0.9733466506004333},
{'label': 'non-offensive', 'score': 0.8974725604057312},
{'label': 'non-offensive', 'score': 0.9217658042907715},
{'label': 'non-offensive', 'score': 0.564075231552124},
{'label': 'non-offensive', 'score': 0.941900908946991},
{'label': 'non-offensive', 'score': 0.8131815195083618},
{'label': 'offensive', 'score': 0.5652127265930176},
{'label': 'non-offensive', 'score': 0.8472980856895447},
{'label': 'non-offensive', 'score': 0.5997186899185181},
{'label': 'non-offensive', 'score': 0.7992671728134155},
{'label': 'non-offensive', 'score': 0.8553436994552612},
{'label': 'non-offensive', 'score': 0.9085837006568909},
{'label': 'non-offensive', 'score': 0.7697728872299194},

{'label': 'offensive', 'score': 0.6305913329124451},
{'label': 'non-offensive', 'score': 0.5145313739776611},
{'label': 'non-offensive', 'score': 0.8902519345283508},
{'label': 'non-offensive', 'score': 0.9592536687850952},
{'label': 'non-offensive', 'score': 0.7007097005844116},
{'label': 'non-offensive', 'score': 0.9651986360549927},
{'label': 'non-offensive', 'score': 0.8049811720848083},
{'label': 'offensive', 'score': 0.8090773820877075},
{'label': 'offensive', 'score': 0.8524211049079895},
{'label': 'non-offensive', 'score': 0.8975100517272949},
{'label': 'offensive', 'score': 0.6889671683311462},
{'label': 'non-offensive', 'score': 0.9441065788269043},
{'label': 'offensive', 'score': 0.6391233801841736},
{'label': 'offensive', 'score': 0.8211570382118225},
{'label': 'non-offensive', 'score': 0.9268559217453003},
{'label': 'non-offensive', 'score': 0.7526609897613525},
{'label': 'offensive', 'score': 0.5146870017051697},
{'label': 'non-offensive', 'score': 0.8820255398750305},
{'label': 'non-offensive', 'score': 0.9607264399528503},
{'label': 'non-offensive', 'score': 0.8656893968582153},
{'label': 'non-offensive', 'score': 0.6839671730995178},
{'label': 'non-offensive', 'score': 0.7310650944709778},
{'label': 'non-offensive', 'score': 0.7176061868667603},
{'label': 'offensive', 'score': 0.6504815220832825},
{'label': 'offensive', 'score': 0.6041460037231445},
{'label': 'offensive', 'score': 0.8247111439704895},
{'label': 'non-offensive', 'score': 0.8512121438980103},
{'label': 'offensive', 'score': 0.8576400279998779},
{'label': 'non-offensive', 'score': 0.9364951252937317},
{'label': 'non-offensive', 'score': 0.965849757194519},
{'label': 'offensive', 'score': 0.9010688662528992},
{'label': 'non-offensive', 'score': 0.9255746603012085},
{'label': 'non-offensive', 'score': 0.8996019959449768},
{'label': 'offensive', 'score': 0.6523518562316895},
{'label': 'non-offensive', 'score': 0.8830508589744568},
{'label': 'offensive', 'score': 0.7986152768135071},
{'label': 'non-offensive', 'score': 0.7232412099838257},
{'label': 'non-offensive', 'score': 0.8879839181900024},
{'label': 'non-offensive', 'score': 0.9503480195999146},
{'label': 'non-offensive', 'score': 0.9254708290100098},
{'label': 'non-offensive', 'score': 0.8842161297798157},
{'label': 'offensive', 'score': 0.5054827332496643},
{'label': 'non-offensive', 'score': 0.6346677541732788},
{'label': 'non-offensive', 'score': 0.840191662311554},
{'label': 'non-offensive', 'score': 0.8553398847579956},
{'label': 'non-offensive', 'score': 0.9599411487579346},
{'label': 'non-offensive', 'score': 0.9629791975021362},

{'label': 'non-offensive', 'score': 0.5147514939308167},
{'label': 'offensive', 'score': 0.8532840013504028},
{'label': 'non-offensive', 'score': 0.8670883178710938},
{'label': 'offensive', 'score': 0.7944380640983582},
{'label': 'offensive', 'score': 0.6178374886512756},
{'label': 'non-offensive', 'score': 0.5496169328689575},
{'label': 'non-offensive', 'score': 0.8832625150680542},
{'label': 'non-offensive', 'score': 0.7727338075637817},
{'label': 'non-offensive', 'score': 0.8880078792572021},
{'label': 'non-offensive', 'score': 0.854920506477356},
{'label': 'non-offensive', 'score': 0.5111041069030762},
{'label': 'offensive', 'score': 0.7042728662490845},
{'label': 'non-offensive', 'score': 0.8765092492103577},
{'label': 'non-offensive', 'score': 0.9583364129066467},
{'label': 'non-offensive', 'score': 0.5639200806617737},
{'label': 'non-offensive', 'score': 0.9701842665672302},
{'label': 'non-offensive', 'score': 0.9034126996994019},
{'label': 'non-offensive', 'score': 0.9278426766395569},
{'label': 'offensive', 'score': 0.8761103749275208},
{'label': 'offensive', 'score': 0.5954149961471558},
{'label': 'offensive', 'score': 0.8101827502250671},
{'label': 'non-offensive', 'score': 0.7798418402671814},
{'label': 'non-offensive', 'score': 0.9412974715232849},
{'label': 'non-offensive', 'score': 0.5859498381614685},
{'label': 'non-offensive', 'score': 0.9153278470039368},
{'label': 'offensive', 'score': 0.7442778944969177},
{'label': 'non-offensive', 'score': 0.8805034160614014},
{'label': 'non-offensive', 'score': 0.8920629620552063},
{'label': 'offensive', 'score': 0.8418039083480835},
{'label': 'non-offensive', 'score': 0.9282906651496887},
{'label': 'non-offensive', 'score': 0.8455625772476196},
{'label': 'non-offensive', 'score': 0.9598767757415771},
{'label': 'non-offensive', 'score': 0.5911389589309692},
{'label': 'offensive', 'score': 0.656711757183075},
{'label': 'offensive', 'score': 0.9067296981811523},
{'label': 'offensive', 'score': 0.71621173620224},
{'label': 'non-offensive', 'score': 0.8127477765083313},
{'label': 'non-offensive', 'score': 0.9545506238937378},
{'label': 'offensive', 'score': 0.6244956851005554},
{'label': 'non-offensive', 'score': 0.8573060631752014},
{'label': 'non-offensive', 'score': 0.7661344408988953},
{'label': 'non-offensive', 'score': 0.9172436594963074},
{'label': 'non-offensive', 'score': 0.5856847167015076},
{'label': 'non-offensive', 'score': 0.9734447002410889},
{'label': 'non-offensive', 'score': 0.7104761600494385},
{'label': 'offensive', 'score': 0.5828077793121338},
{'label': 'offensive', 'score': 0.614445686340332},

{'label': 'non-offensive', 'score': 0.6870855093002319},
{'label': 'non-offensive', 'score': 0.5623599290847778},
{'label': 'offensive', 'score': 0.8855676651000977},
{'label': 'offensive', 'score': 0.8876639604568481},
{'label': 'non-offensive', 'score': 0.6772134304046631},
{'label': 'non-offensive', 'score': 0.9568246603012085},
{'label': 'offensive', 'score': 0.8663862347602844},
{'label': 'non-offensive', 'score': 0.9489739537239075},
{'label': 'non-offensive', 'score': 0.5711002945899963},
{'label': 'non-offensive', 'score': 0.7447052597999573},
{'label': 'offensive', 'score': 0.7933859825134277},
{'label': 'non-offensive', 'score': 0.5663027763366699},
{'label': 'non-offensive', 'score': 0.7980839610099792},
{'label': 'offensive', 'score': 0.693128228187561},
{'label': 'offensive', 'score': 0.6588349938392639},
{'label': 'non-offensive', 'score': 0.8766363859176636},
{'label': 'non-offensive', 'score': 0.9420992136001587},
{'label': 'non-offensive', 'score': 0.9043669700622559},
{'label': 'offensive', 'score': 0.5194647908210754},
{'label': 'non-offensive', 'score': 0.826751708984375},
{'label': 'offensive', 'score': 0.5548930764198303},
{'label': 'non-offensive', 'score': 0.6667755842208862},
{'label': 'offensive', 'score': 0.6066457033157349},
{'label': 'non-offensive', 'score': 0.9098477363586426},
{'label': 'non-offensive', 'score': 0.9518694877624512},
{'label': 'non-offensive', 'score': 0.5722610950469971},
{'label': 'non-offensive', 'score': 0.9042643308639526},
{'label': 'non-offensive', 'score': 0.9493448138237},
{'label': 'non-offensive', 'score': 0.9543051719665527},
{'label': 'non-offensive', 'score': 0.8174348473548889},
{'label': 'non-offensive', 'score': 0.7703944444656372},
{'label': 'non-offensive', 'score': 0.8066638708114624},
{'label': 'non-offensive', 'score': 0.9716463685035706},
{'label': 'non-offensive', 'score': 0.6805480718612671},
{'label': 'non-offensive', 'score': 0.9721750617027283},
{'label': 'non-offensive', 'score': 0.9358510971069336},
{'label': 'non-offensive', 'score': 0.8673626780509949},
{'label': 'non-offensive', 'score': 0.9806139469146729},
{'label': 'non-offensive', 'score': 0.891538143157959},
{'label': 'non-offensive', 'score': 0.815442681312561},
{'label': 'offensive', 'score': 0.5177029967308044},
{'label': 'offensive', 'score': 0.6714714169502258},
{'label': 'non-offensive', 'score': 0.7777089476585388},
{'label': 'non-offensive', 'score': 0.900209903717041},
{'label': 'offensive', 'score': 0.5448732972145081},
{'label': 'offensive', 'score': 0.7883957624435425},
{'label': 'non-offensive', 'score': 0.5834346413612366},

{'label': 'offensive', 'score': 0.6621526479721069},
{'label': 'non-offensive', 'score': 0.8583590388298035},
{'label': 'non-offensive', 'score': 0.6641010046005249},
{'label': 'non-offensive', 'score': 0.844746470451355},
{'label': 'non-offensive', 'score': 0.9374734163284302},
{'label': 'non-offensive', 'score': 0.8089913129806519},
{'label': 'non-offensive', 'score': 0.9416565299034119},
{'label': 'non-offensive', 'score': 0.8864585757255554},
{'label': 'non-offensive', 'score': 0.9671632051467896},
{'label': 'non-offensive', 'score': 0.9556635022163391},
{'label': 'non-offensive', 'score': 0.6419916152954102},
{'label': 'non-offensive', 'score': 0.9024519920349121},
{'label': 'offensive', 'score': 0.5668319463729858},
{'label': 'non-offensive', 'score': 0.6347986459732056},
{'label': 'non-offensive', 'score': 0.9622607231140137},
{'label': 'non-offensive', 'score': 0.5192092657089233},
{'label': 'non-offensive', 'score': 0.7500264644622803},
{'label': 'non-offensive', 'score': 0.9441766142845154},
{'label': 'non-offensive', 'score': 0.7643901109695435},
{'label': 'non-offensive', 'score': 0.7194973230361938},
{'label': 'non-offensive', 'score': 0.6228901147842407},
{'label': 'non-offensive', 'score': 0.6957954168319702},
{'label': 'non-offensive', 'score': 0.7453330755233765},
{'label': 'non-offensive', 'score': 0.9372200965881348},
{'label': 'non-offensive', 'score': 0.9360722899436951},
{'label': 'non-offensive', 'score': 0.8643784523010254},
{'label': 'non-offensive', 'score': 0.6532891988754272},
{'label': 'non-offensive', 'score': 0.602623462677002},
{'label': 'non-offensive', 'score': 0.948958158493042},
{'label': 'offensive', 'score': 0.8069931268692017},
{'label': 'offensive', 'score': 0.8161106705665588},
{'label': 'non-offensive', 'score': 0.7892906665802002},
{'label': 'non-offensive', 'score': 0.5153287649154663},
{'label': 'non-offensive', 'score': 0.954807460308075},
{'label': 'non-offensive', 'score': 0.6196908950805664},
{'label': 'non-offensive', 'score': 0.8665497303009033},
{'label': 'non-offensive', 'score': 0.9156047701835632},
{'label': 'non-offensive', 'score': 0.9057238698005676},
{'label': 'offensive', 'score': 0.8836828470230103},
{'label': 'non-offensive', 'score': 0.8978877663612366},
{'label': 'non-offensive', 'score': 0.8049733638763428},
{'label': 'offensive', 'score': 0.8733487129211426},
{'label': 'non-offensive', 'score': 0.8855512142181396},
{'label': 'non-offensive', 'score': 0.830785870552063},
{'label': 'non-offensive', 'score': 0.9557346105575562},
{'label': 'non-offensive', 'score': 0.9321025609970093},
{'label': 'non-offensive', 'score': 0.9512830376625061},

{'label': 'offensive', 'score': 0.6694768667221069},
{'label': 'offensive', 'score': 0.5230463743209839},
{'label': 'non-offensive', 'score': 0.9541545510292053},
{'label': 'non-offensive', 'score': 0.6916643977165222},
{'label': 'non-offensive', 'score': 0.9328643083572388},
{'label': 'non-offensive', 'score': 0.7018622159957886},
{'label': 'offensive', 'score': 0.8005268573760986},
{'label': 'non-offensive', 'score': 0.9101123809814453},
{'label': 'non-offensive', 'score': 0.9777984023094177},
{'label': 'non-offensive', 'score': 0.7393465042114258},
{'label': 'non-offensive', 'score': 0.9479533433914185},
{'label': 'offensive', 'score': 0.7362173795700073},
{'label': 'non-offensive', 'score': 0.9617113471031189},
{'label': 'non-offensive', 'score': 0.8205026984214783},
{'label': 'non-offensive', 'score': 0.8920491337776184},
{'label': 'offensive', 'score': 0.9068023562431335},
{'label': 'non-offensive', 'score': 0.8109164834022522},
{'label': 'non-offensive', 'score': 0.5252447724342346},
{'label': 'non-offensive', 'score': 0.6327130198478699},
{'label': 'offensive', 'score': 0.8978598713874817},
{'label': 'non-offensive', 'score': 0.6433027386665344},
{'label': 'non-offensive', 'score': 0.950530469417572},
{'label': 'non-offensive', 'score': 0.944420576095581},
{'label': 'offensive', 'score': 0.5235204696655273},
{'label': 'non-offensive', 'score': 0.9545952677726746},
{'label': 'non-offensive', 'score': 0.9531868100166321},
{'label': 'offensive', 'score': 0.6940492391586304},
{'label': 'offensive', 'score': 0.7354618310928345},
{'label': 'offensive', 'score': 0.9266201853752136},
{'label': 'offensive', 'score': 0.6227362751960754},
{'label': 'offensive', 'score': 0.699895977973938},
{'label': 'non-offensive', 'score': 0.9618956446647644},
{'label': 'non-offensive', 'score': 0.905529797077179},
{'label': 'non-offensive', 'score': 0.915638267993927},
{'label': 'non-offensive', 'score': 0.6067402958869934},
{'label': 'non-offensive', 'score': 0.9106586575508118},
{'label': 'non-offensive', 'score': 0.8283422589302063},
{'label': 'non-offensive', 'score': 0.8180041313171387},
{'label': 'non-offensive', 'score': 0.8629913926124573},
{'label': 'non-offensive', 'score': 0.9492982625961304},
{'label': 'non-offensive', 'score': 0.7508594989776611},
{'label': 'non-offensive', 'score': 0.9335468411445618},
{'label': 'non-offensive', 'score': 0.6175878643989563},
{'label': 'offensive', 'score': 0.9186577200889587},
{'label': 'non-offensive', 'score': 0.8346035480499268},
{'label': 'non-offensive', 'score': 0.5564724802970886},
{'label': 'offensive', 'score': 0.8492959141731262},

{'label': 'non-offensive', 'score': 0.9089798927307129},
{'label': 'non-offensive', 'score': 0.847247302532196},
{'label': 'non-offensive', 'score': 0.8166351914405823},
{'label': 'non-offensive', 'score': 0.7531733512878418},
{'label': 'non-offensive', 'score': 0.8556636571884155},
{'label': 'offensive', 'score': 0.8203002214431763},
{'label': 'non-offensive', 'score': 0.9355129599571228},
{'label': 'offensive', 'score': 0.8533977270126343},
{'label': 'non-offensive', 'score': 0.9766384959220886},
{'label': 'non-offensive', 'score': 0.7236941456794739},
{'label': 'offensive', 'score': 0.7585206031799316},
{'label': 'non-offensive', 'score': 0.8702991604804993},
{'label': 'non-offensive', 'score': 0.9319545030593872},
{'label': 'non-offensive', 'score': 0.9220424890518188},
{'label': 'non-offensive', 'score': 0.7004123330116272},
{'label': 'non-offensive', 'score': 0.6033399105072021},
{'label': 'non-offensive', 'score': 0.9681439399719238},
{'label': 'non-offensive', 'score': 0.9625951647758484},
{'label': 'non-offensive', 'score': 0.9641011953353882},
{'label': 'non-offensive', 'score': 0.6946874260902405},
{'label': 'non-offensive', 'score': 0.8520649075508118},
{'label': 'offensive', 'score': 0.9172079563140869},
{'label': 'non-offensive', 'score': 0.9380196928977966},
{'label': 'non-offensive', 'score': 0.9170390963554382},
{'label': 'non-offensive', 'score': 0.8501089215278625},
{'label': 'non-offensive', 'score': 0.7347993850708008},
{'label': 'non-offensive', 'score': 0.919245183467865},
{'label': 'offensive', 'score': 0.9391582012176514},
{'label': 'non-offensive', 'score': 0.8315045833587646},
{'label': 'offensive', 'score': 0.6347769498825073},
{'label': 'offensive', 'score': 0.6722444295883179},
{'label': 'non-offensive', 'score': 0.7066056728363037},
{'label': 'offensive', 'score': 0.6878634691238403},
{'label': 'non-offensive', 'score': 0.7335588932037354},
{'label': 'non-offensive', 'score': 0.9413359761238098},
{'label': 'non-offensive', 'score': 0.9435192346572876},
{'label': 'offensive', 'score': 0.891071081161499},
{'label': 'non-offensive', 'score': 0.9542928338050842},
{'label': 'offensive', 'score': 0.6509085893630981},
{'label': 'offensive', 'score': 0.5737082958221436},
{'label': 'offensive', 'score': 0.6403729915618896},
{'label': 'non-offensive', 'score': 0.881782054901123},
{'label': 'non-offensive', 'score': 0.9142394065856934},
{'label': 'non-offensive', 'score': 0.761211097240448},
{'label': 'offensive', 'score': 0.6153733134269714},
{'label': 'non-offensive', 'score': 0.5134412050247192},
{'label': 'offensive', 'score': 0.6931162476539612},

{'label': 'non-offensive', 'score': 0.909564733505249},
{'label': 'offensive', 'score': 0.8422945141792297},
{'label': 'non-offensive', 'score': 0.9350215196609497},
{'label': 'non-offensive', 'score': 0.891211748123169},
{'label': 'offensive', 'score': 0.5229942202568054},
{'label': 'non-offensive', 'score': 0.8584861159324646},
{'label': 'non-offensive', 'score': 0.939052939414978},
{'label': 'non-offensive', 'score': 0.8066257834434509},
{'label': 'non-offensive', 'score': 0.6496645212173462},
{'label': 'non-offensive', 'score': 0.9715701937675476},
{'label': 'non-offensive', 'score': 0.9190877676010132},
{'label': 'non-offensive', 'score': 0.5936459302902222},
{'label': 'offensive', 'score': 0.535834550857544},
{'label': 'non-offensive', 'score': 0.5043784379959106},
{'label': 'non-offensive', 'score': 0.5824353098869324},
{'label': 'non-offensive', 'score': 0.9724574089050293},
{'label': 'non-offensive', 'score': 0.7268106937408447},
{'label': 'non-offensive', 'score': 0.8452096581459045},
{'label': 'non-offensive', 'score': 0.7755357027053833},
{'label': 'offensive', 'score': 0.5538263320922852},
{'label': 'offensive', 'score': 0.6478709578514099},
{'label': 'non-offensive', 'score': 0.9191747903823853},
{'label': 'non-offensive', 'score': 0.9763135313987732},
{'label': 'non-offensive', 'score': 0.5566112995147705},
{'label': 'offensive', 'score': 0.6997122168540955},
{'label': 'offensive', 'score': 0.5832380056381226},
{'label': 'non-offensive', 'score': 0.8445264101028442},
{'label': 'non-offensive', 'score': 0.6064488291740417},
{'label': 'non-offensive', 'score': 0.8130218386650085},
{'label': 'offensive', 'score': 0.6660960912704468},
{'label': 'non-offensive', 'score': 0.8123977780342102},
{'label': 'non-offensive', 'score': 0.7957322597503662},
{'label': 'non-offensive', 'score': 0.8343762159347534},
{'label': 'non-offensive', 'score': 0.7806297540664673},
{'label': 'non-offensive', 'score': 0.6809470653533936},
{'label': 'offensive', 'score': 0.6802361011505127},
{'label': 'non-offensive', 'score': 0.9467289447784424},
{'label': 'non-offensive', 'score': 0.9598767757415771},
{'label': 'non-offensive', 'score': 0.562699556350708},
{'label': 'non-offensive', 'score': 0.626956045627594},
{'label': 'non-offensive', 'score': 0.7280963659286499},
{'label': 'non-offensive', 'score': 0.9264888763427734},
{'label': 'non-offensive', 'score': 0.9616975784301758},
{'label': 'non-offensive', 'score': 0.9645494818687439},
{'label': 'non-offensive', 'score': 0.896135151386261},
{'label': 'non-offensive', 'score': 0.9762118458747864},
{'label': 'non-offensive', 'score': 0.9321606755256653},

```
{'label': 'offensive', 'score': 0.7734279632568359},
{'label': 'non-offensive', 'score': 0.8792917132377625},
{'label': 'non-offensive', 'score': 0.8584861159324646},
{'label': 'non-offensive', 'score': 0.9502109885215759},
{'label': 'non-offensive', 'score': 0.9683456420898438},
{'label': 'non-offensive', 'score': 0.7083724737167358},
{'label': 'non-offensive', 'score': 0.8514422178268433},
{'label': 'offensive', 'score': 0.5353519916534424},
{'label': 'non-offensive', 'score': 0.9574507474899292},
{'label': 'non-offensive', 'score': 0.9499408006668091},
{'label': 'non-offensive', 'score': 0.9243402481079102},
{'label': 'non-offensive', 'score': 0.5347045660018921},
{'label': 'non-offensive', 'score': 0.7769464254379272},
{'label': 'non-offensive', 'score': 0.8930925130844116},
{'label': 'non-offensive', 'score': 0.7793471813201904},
{'label': 'non-offensive', 'score': 0.9498886466026306},
{'label': 'non-offensive', 'score': 0.9442235827445984},
{'label': 'non-offensive', 'score': 0.874133288860321},
{'label': 'non-offensive', 'score': 0.6802535653114319},
{'label': 'non-offensive', 'score': 0.6958889961242676},
{'label': 'non-offensive', 'score': 0.7828267812728882},
{'label': 'non-offensive', 'score': 0.7042650580406189},
{'label': 'non-offensive', 'score': 0.9741178154945374},
{'label': 'non-offensive', 'score': 0.7758073806762695},
{'label': 'offensive', 'score': 0.9251351356506348},
{'label': 'non-offensive', 'score': 0.510896623134613},
{'label': 'non-offensive', 'score': 0.6726999282836914},
{'label': 'offensive', 'score': 0.8587868213653564},
{'label': 'non-offensive', 'score': 0.5923421382904053},
{'label': 'non-offensive', 'score': 0.8264032602310181},
{'label': 'non-offensive', 'score': 0.7675051689147949},
{'label': 'non-offensive', 'score': 0.9622277021408081},
{'label': 'non-offensive', 'score': 0.6721819043159485},
{'label': 'non-offensive', 'score': 0.7396516799926758},
{'label': 'non-offensive', 'score': 0.6119992733001709},
{'label': 'non-offensive', 'score': 0.7643154263496399},
{'label': 'non-offensive', 'score': 0.6847031116485596}]
```

[25]:
```python
def evaluate(predictions: List, labels: List) -> None:
    """
    Evaluate the predictions of a model.

    ## Parameters
    predictions: List
        The predictions of a model.
    labels: List
        The labels of the test set.
```

```
    """
    predictions = [0 if p['label'] == "non-offensive" else 1 for p in␣
↪predictions]
    print(classification_report(labels, predictions))
```

[181]:
```
evaluate(predictions, dataset['test']['label'])
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.91       620
           1       0.80      0.67      0.73       240

    accuracy                           0.86       860
   macro avg       0.84      0.80      0.82       860
weighted avg       0.86      0.86      0.86       860
```

The global average F1 score is 0.82, which is pretty good. The model seems to be better at classifying non-offensive tweets than offensive tweets, as the F1 score for non-offensive tweets is 0.91, and the F1 score for offensive tweets is 0.73. Also, the dataset is unbalanced, as there are more non-offensive tweets than offensive tweets. Thus, the model is better at classifying the majority class, which is non-offensive tweets.

2. Look for prediction failures. Extract the top 5 misclassified tweets (highest score in wrong class) for each class and discuss what could be wrong with the model.

[182]:
```
def get_top_misclassified(predictions: List, labels: List, nb_samples: int = 5)␣
↪-> Tuple[List, List]:
    """
    Extract nb_samples misclassified samples per class.

    ## Parameters
    predictions: List
        The predictions of a model.
    labels: List
        The labels of the test set.
    nb_samples: int
        The number of samples to extract per class.

    ## Returns
    offensive_missclassified: List
        The top nb_samples offensive samples that were missclassified.
    non_offensive_missclassified: List
        The top nb_samples non-offensive samples that were missclassified.
    """
    offensive_missclassified = []
    non_offensive_missclassified = []

    for i, p in enumerate(predictions):
```

```
        if p['label'] == "non-offensive" and labels[i] == 1:
            offensive_missclassified.append((i, p['score']))
        elif p['label'] == "offensive" and labels[i] == 0:
            non_offensive_missclassified.append((i, p['score']))
    offensive_missclassified.sort(key=lambda x: x[1], reverse=True)
    non_offensive_missclassified.sort(key=lambda x: x[1], reverse=True)

    return offensive_missclassified[:nb_samples], non_offensive_missclassified[:
↪nb_samples]
```

[183]:
```
offensive_missclassified, non_offensive_missclassified =␣
↪get_top_misclassified(predictions, dataset['test']['label'])
```

[184]:
```
for i, score in offensive_missclassified:
    print(score, dataset['test']["text"][i])
```

```
0.9338217973709106 #Liberals / #Democrats THIS is what you stand for. If not,
then #WalkAway
0.919756293296814 #Liberals Are Reaching Peak Desperation To Call On
#PhillipRuddock To Talk With #Turnbull To Convince Him To Help with
#WentworthVotes 18 Sept 2018  @user #Auspol #LNP #NSWpol @user  @user @user
#LNPMemes
0.9112220406532288 #NoPasaran: Unity demo to oppose the far-right in #London -
#antifa #Oct13 - Enough is Enough!
0.9081719517707825 #BREXIT deal HAS been reached - and will be unveiled at
special summit in NOVEMBER, Has @user sold out the #UK to the eu??? She better
have not or the @user are finished!! @user
0.895766019821167 #America  … tear down that #Wall! #tcot #partisanship #Trump
#thewall #Borderwall #liberty #civilsociety #think #Conservatives #Democrats
#Progressives #liberals #Independent #libertarians #GOP #DNC #CriticalThinking
```

[185]:
```
for i, score in non_offensive_missclassified:
    print(score, dataset['test']["text"][i])
```

```
0.9010688662528992 Are you fucking serious?
0.8939998745918274 @user I guess that’s where swamp ass originated
0.8576400279998779 An American Tail really is one of the most underrated
animations ever ever ever. Fuck I cried in this scene
0.8492959141731262 @user @user Bull crap. You know she doesn't care.  She is
trying to get attention for her Presidential run.  Do you see any other Senator
giving nonsense?  Nope.
0.8418039083480835 #Room25 is actually incredible, Noname is the shit, always
has been,  and I’m seein her in like 5 days in Melbourne. Life is good. Have a
nice day.
```

Regarding offensive tweets misclassified as non-offensive, the model seems to have a hard time classifying tweets that are not insults, but are not politically correct. For example, the second tweet is classified as non-offensive, but seems to have strong opinion about Liberals. Also, the model is really confident about these classifications, with almost all scores above 0.90.

Regarding non-offensive tweets misclassified as offensive, it looks like the model is classifying these example as offensive because they contain insults. But actually, these insults do not seem to be put in an offensive way, more as an accent on the sentence. So this is very understandable that the model is misclassifying these examples.

3. Extract the top 10 tweets your model is most confident about in the target class (offensive or hateful), the top 10 in the neutral class, and the top 10 your model is most uncertain about. Do you believe the model is doing a great job?

[29]:
```
with open('tweets.json') as f:
    data = json.load(f)

df = pd.DataFrame(data)
df = df.dropna()
df = df.reset_index(drop=True)
df = df.drop_duplicates(subset=['text'])

df
```

[29]:

	id	id_str	text	lang	created_at
0	1410492618790817793	1410492618790817793	YOU BETTER SUCK HIS DICK KOZY I SEE YOU WITH K…	en	Thu Jul 01 06:57:00 +0000 2021
1	1410492618769780742	1410492618769780742	I still canr believe it.	en	Thu Jul 01 06:57:00 +0000 2021
2	1410492618790686720	1410492618790686720	You should raise the webform…how would they…	en	
3	1410492618803335174	1410492618803335174	im tired too but this is so entertaining i cant	en	
4	1410492618778157059	1410492618778157059	Fuckof	en	
…	…	…	…	…	
9995	1410721732642492418	1410721732642492418	Because It’s My Business: Hear Tabitha Brown’s…	en	
9996	1410721732659322881	1410721732659322881	comer pipoca enquanto assisto girl from nowher…	en	
9997	1410721736841056259	1410721736841056259	They will be mad with me if they’re not 504Boy…	en	
9998	1410721736828411905	1410721736828411905	Omg so beautiful	en	
9999	1410721736849379328	1410721736849379328	Aight I'll try better next time. Was fun!	en	

```
2     Thu Jul 01 06:57:00 +0000 2021
3     Thu Jul 01 06:57:00 +0000 2021
4     Thu Jul 01 06:57:00 +0000 2021
…                                 …
9995  Thu Jul 01 22:07:25 +0000 2021
9996  Thu Jul 01 22:07:25 +0000 2021
9997  Thu Jul 01 22:07:26 +0000 2021
9998  Thu Jul 01 22:07:26 +0000 2021
9999  Thu Jul 01 22:07:26 +0000 2021

[9740 rows x 5 columns]
```

[30]:
```
tweets_preds = roberta_pipeline(df['text'].tolist())
```

[188]:
```
def extract_top_tweets(predictions: List, nb_samples: int = 10) -> Tuple[List,␣
↪List, List]:
    """
    Extract the top nb_samples offensive, non-offensive and uncertain tweets.

    ## Parameters
    predictions: List
        The predictions of a model.
    nb_samples: int
        The number of samples to extract per class.

    ## Returns
    top_offensive_tweets: List
        The top nb_samples offensive tweets.
    top_non_offensive_tweets: List
        The top nb_samples non-offensive tweets.
    top_uncertain_tweets: List
        The top nb_samples uncertain tweets.
    """
    top_offensive_tweets = []
    top_non_offensive_tweets = []
    top_uncertain_tweets = []

    for i, p in enumerate(predictions):
        if p['label'] == "non-offensive":
            top_non_offensive_tweets.append((i, p['score']))
        elif p['label'] == "offensive":
            top_offensive_tweets.append((i, p['score']))
        top_uncertain_tweets.append((i, p['score']))
    top_offensive_tweets.sort(key=lambda x: x[1], reverse=True)
    top_non_offensive_tweets.sort(key=lambda x: x[1], reverse=True)
    top_uncertain_tweets.sort(key=lambda x: x[1])
```

```
    return top_offensive_tweets[:nb_samples], top_non_offensive_tweets[:
↪nb_samples], top_uncertain_tweets[:nb_samples]
```

[189]:
```
top_offensive_tweets, top_non_offensive_tweets, top_uncertain_tweets =␣
↪extract_top_tweets(tweets_preds)
```

[190]:
```
for i, score in top_offensive_tweets:
    print(score, df['text'][i])
```

```
0.9484737515449524 Stop with the slow mo it make it look bad
0.9465802907943726 morningggggg
0.9437395930290222 YOU GET ITTT and same omg, i think the last time i had one of
those was in 2018 but its so good
0.9423408508300781 Me too. Buck up; you are not alone. Good people agree, and we
are all in this together.
0.9394720792770386 sexy
0.9366845488548279 she should pay attention more omg it’s so annoying :/
0.9325129985809326 Or how about rather than playing a game and tweeting about it
you pull your finger out and reply to your backers you are letting down every
single day. Shame on you
0.9313321113586426 Finally a good take
0.9305409789085388 desoff
0.9289026856422424 mans is lifting two of you.
```

[191]:
```
for i, score in top_non_offensive_tweets:
    print(score, df['text'][i])
```

```
0.9816755652427673 Cool
0.9815455079078674 Man you guys really know how to make a mofo feel totally
socially inept.
0.9814001321792603  LITERALLY  WAKE  UP  RN  WHERE  R  U
0.9809073805809021 I was referred to her by a friend online and I thought is a
scam …but I was moved to try and here I earned.. just want to share this to
people too. @user
0.9806414246559143 Deja vu

BLINKS U KNOW WHAT 2 DO
#PremiosMTVMIAW
#MTVLAKPOPROSE
#MTVLAFANDOMBLINKS
@user
0.9802667498588562 Who doesn’t love u
0.9802438020706177 one person followed me and 3 people unfollowed me //
automatically checked by http
0.979927659034729 Just the way I make money and would not marry or even date who
doesn’t make her own money, same applies to cooking. I know how to cook and I
won’t settle with a woman who can’t cook unless she’s Rich/wealthy or sum. You
can’t be broke and still be lazy
```

```
0.9798141121864319 July
0.9793971180915833 Good afternoon&lt;333
```

[192]:
```
for i, score in top_uncertain_tweets:
    print(score, df['text'][i])
```

```
0.5004608631134033 Johor recorded most suicide cases for two consecutive years - 2019, 2020. As of May 2021, Selangor recorded the highest number of suicide cases, with 117 or 25% of 468 cases reported this year
0.500519871711731 the part about this that scares me the most is that i drink heavily i take vyvanse i occasionally smoke and i use retinol every single night. like that baby would be gambling with its life keeping me out of the loop bruh
0.5005561709403992 STOP HE WAITED TILL 3:25 HUH
0.50067538022995  stay safe.
0.500801682472229 too scared to soend my own money bc my mom gets notifs if i buy something
0.5008642673492432 Having my nipples pierced again makes me feel closer to the person i was.
0.5010842084884644 And you didn't need the Americans at all this time!
0.5010976195335388 You feel like that, perhaps something in your past may have informed that. So maybe try understand what about sharing a milestone with loved ones is deeply making you feel like you're attention seeking because I'm pretty sure those that really care about you wont read it that way
0.5011819005012512 LOL, you're excused Tequilla. x
0.5011822581291199 oh Brent how i've missed you
```

[193]:
```
print(df['text'][0], tweets_preds[0])
```

```
YOU BETTER SUCK HIS DICK KOZY I SEE YOU WITH KNUCKLES GET EM GYAAAAL {'label': 'offensive', 'score': 0.8737722039222717}
```

Looking at the tweets that the model is most confident about, we can see that it's very confident when classifying non-offensive tweets as offensive. For example, it classified *"morningggggg"* as offensive with a score of 0.946, which makes no sense. Top non-offensive tweets looks better classified than top offensive tweets. The uncertain ones also make sense, as some are about drinking, sex related vocabulary, or strange formulations. Overall, we can say it's not doing a great job, but in this case it seems better to have false positives than false negatives, so it's not that bad. It can still classify very offensive tweets such as the last example, which is good.

4. (Bonus) Use SHAP on the provided tweets, or manually written texts, to see if you can find topics on which the model is biased.

[194]:
```
some_tweets = df['text'].tolist()[:10]
some_tweets
```

[194]:
```
['YOU BETTER SUCK HIS DICK KOZY I SEE YOU WITH KNUCKLES GET EM GYAAAAL',
 'I still canr believe it.   ',
 'You should raise the webform…how would they know then that you completed ur medicals',
```

```
 'im tired too but this is so entertaining i cant',
 'Fuckof',
 'People  ',
 'Even if they didn't exploit people to acquire their riches, how are you gonna
be okay literally wasting thousands and thousands of dollars while there are
still people who are homeless? While there are people skipping life saving
medical treatments bc of the cost?',
 'He rather have used the cash to buy some clothes that dont resemble a duvet.',
 'Baret',
 'Bloody awesome!']
```

[195]:
```
explainer = shap.Explainer(roberta_pipeline)
shap_values = explainer(some_tweets)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[195], line 2
      1 explainer = shap.Explainer(roberta_pipeline)
----> 2 shap_values = explainer(some_tweets)

File ~/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-packages/shap
 ↪explainers/_partition.py:136, in Partition.__call__(self, max_evals,␣
 ↪fixed_context, main_effects, error_bounds, batch_size, outputs, silent, *args
    132 def __call__(self, *args, max_evals=500, fixed_context=None,␣
 ↪main_effects=False, error_bounds=False, batch_size="auto",
    133              outputs=None, silent=False):
    134     """ Explain the output of the model on the given arguments.
    135     """
--> 136     return super().__call__(
    137         *args, max_evals=max_evals, fixed_context=fixed_context,␣
 ↪main_effects=main_effects, error_bounds=error_bounds, batch_size=batch_size,
    138         outputs=outputs, silent=silent
    139     )

File ~/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-packages/shap
 ↪explainers/_explainer.py:266, in Explainer.__call__(self, max_evals,␣
 ↪main_effects, error_bounds, batch_size, outputs, silent, *args, **kwargs)
    264     feature_names = [[] for _ in range(len(args))]
    265 for row_args in show_progress(zip(*args), num_rows, self.__class__.
 ↪__name__+" explainer", silent):
--> 266     row_result = self.explain_row(
    267         *row_args, max_evals=max_evals, main_effects=main_effects,␣
 ↪error_bounds=error_bounds,
    268         batch_size=batch_size, outputs=outputs, silent=silent, **kwargs
    269     )
    270     values.append(row_result.get("values", None))
    271     output_indices.append(row_result.get("output_indices", None))
```

```
File ~/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-packages/shap
 ↪explainers/_partition.py:154, in Partition.explain_row(self, max_evals,␣
 ↪main_effects, error_bounds, batch_size, outputs, silent, fixed_context,␣
 ↪*row_args)
    151     raise ValueError("Unknown fixed_context value passed (must be 0, 1␣
 ↪or None): %s" %fixed_context)
    153 # build a masked version of the model for the current input sample
--> 154 fm = MaskedModel(self.model, self.masker, self.link, self.
 ↪linearize_link, *row_args)
    156 # make sure we have the base value and current value outputs
    157 M = len(fm)

File ~/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-packages/shap
 ↪utils/_masked_model.py:28, in MaskedModel.__init__(self, model, masker, link,
 ↪linearize_link, *args)
     26 # if the masker supports it, save what positions vary from the backgrou d
     27 if callable(getattr(self.masker, "invariants", None)):
---> 28     self._variants = ~self.masker.invariants(*args)
     29     self._variants_column_sums = self._variants.sum(0)
     30     self._variants_row_inds = [
     31         self._variants[:,i] for i in range(self._variants.shape[1])
     32     ]

File ~/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-packages/shap
 ↪maskers/_text.py:297, in Text.invariants(self, s)
    293 """ The names of the features for each mask position for the given inpu␣
 ↪string.
    294 """
    295 self._update_s_cache(s)
--> 297 invariants = np.zeros(len(self._tokenized_s), dtype=np.bool)
    298 if self.keep_prefix > 0:
    299     invariants[:self.keep_prefix] = True

File ~/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-packages/numpy/
 ↪__init__.py:305, in __getattr__(attr)
    300     warnings.warn(
    301         f"In the future `np.{attr}` will be defined as the "
    302         "corresponding NumPy scalar.", FutureWarning, stacklevel=2)
    304 if attr in __former_attrs__:
--> 305     raise AttributeError(__former_attrs__[attr])
    307 # Importing Tester requires importing all of UnitTest which is not a
    308 # cheap import Since it is mainly used in test suits, we lazy import it
    309 # here to save on the order of 10 ms of import time for most users
    310 #
    311 # The previous way Tester was imported also had a side effect of adding
    312 # the full `numpy.testing` namespace
    313 if attr == 'testing':
```

```
AttributeError: module 'numpy' has no attribute 'bool'.
`np.bool` was a deprecated alias for the builtin `bool`. To avoid this error in existing code, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
The aliases was originally deprecated in NumPy 1.20; for more details and guidance see the original release note at:
    https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

[ ]:
```
shap.plots.text(shap_values[:10])
```

SHAP seems to not be working anymore…

5. What are the advantages of using a pre-trained transformer vs naive Bayes? Think about training, and usage in production. The advantages of using a pre-trained transformer are the following: - The model is already trained, so we don't need to train it again. This is a huge advantage because training a model can take a lot of time, and a lot of data. - The model can capture a lot of information, and can be used for a lot of different tasks. It can be used on tasks that are not related to the task it has been trained on, which is not the case for a Naive Bayes model.

The advantages of using a Naive Bayes model are the following: - The model is very simple, and can be trained very quickly. It can also be trained on a small amount of data. - Since the model is very simple, it is way easier to understand how it works, and to debug it.

In term of efficiency, the pre-trained transformer is way more efficient than the Naive Bayes model. However, the Naive Bayes model is way easier to understand and to debug, and it will be more efficient on a small dataset. It will also require less resources to run. So this is a trade-off between efficiency and simplicity.

6. Train a naive Bayes model on the data, and compare its results with this model.

[196]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report

# define a pipeline
pipeline = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('classifier', MultinomialNB())
])

# train the model
pipeline.fit(dataset['train']['text'], dataset['train']['label'])

# evaluate the model
predictions = pipeline.predict(dataset['test']['text'])
print(classification_report(dataset['test']['label'], predictions))
```

```
              precision    recall  f1-score   support

           0       0.84      0.87      0.85       620
           1       0.62      0.56      0.59       240

    accuracy                           0.78       860
   macro avg       0.73      0.71      0.72       860
weighted avg       0.78      0.78      0.78       860
```

The F1-score of the Naive Bayes model is 0.72, which is way lower than the F1-score of the RoBERTa model, which is 0.82. This makes sense and confirms what we said in the previous question: the RoBERTa model is way more efficient than the Naive Bayes model.

1.4 Annotate data

1. Extract about 100 tweets containing at least 20% of your target class (offensive/hateful), from the 10K tweets provided. You can use the pretrained model to help you find tweets in the target class.

[36]:
```
offensive_tweets = []
for i, p in enumerate(tweets_preds):
    if p['label'] == "offensive":
        offensive_tweets.append((df["text"][i], p["label"]))
    if len(offensive_tweets) == 30:
        break

non_offensive_tweets = []
for i, p in enumerate(tweets_preds):
    if p['label'] == "non-offensive":
        non_offensive_tweets.append((df["text"][i], p["label"]))
    if len(non_offensive_tweets) == 70:
        break

extracted_tweets = offensive_tweets + non_offensive_tweets

# shuffle the tweets
random.seed(SEED)
random.shuffle(extracted_tweets)
extracted_tweets[:10]
```

[36]:
```
[('agree.', 'non-offensive'),
 ('I guess looking at interannotator metrics such as fleiss kappa may be useful
to understand how consistent they are? You could look at some kind of document
similarity to infer information about the related pages?',
  'non-offensive'),
 ('Tried and tested  ', 'non-offensive'),
 ('setiap lihat teman2ku baru main hades and they're feeing horny for no reason
yeah that's the point babe this game and the fandom are so horny',
```

```
  'offensive'),
 ('Free Britney', 'non-offensive'),
 ('Correction: Eastern WA will see *smoke from the #LavaFire in CA',
  'non-offensive'),
 ('Fuckof', 'offensive'),
 ('literally', 'non-offensive'),
 ('feeling so sexy and principled at my horrifically corrupt and unfulfilling
blue collar job',
  'offensive'),
 ('VOTE FOR CHANGBIN RN', 'non-offensive')]
```

[37]:
```
extracted_tweets_without_labels = [t[0] for t in extracted_tweets]
```

[90]:
```
extracted_tweets_df = pd.DataFrame(extracted_tweets_without_labels,␣
↪columns=["text"])
extracted_tweets_df["label"] = ""
extracted_tweets_df.to_csv("extracted_tweets.csv", index=False)
```

2. Write down an annotation guildeline. Annotation Guideline for Tweet Classification

Objective: The aim of this annotation guideline is to provide clear instructions for annotating tweets into three distinct classes: "neutral," "offensive," and "can't tell." The guidelines ensure consistent annotation across different annotators and help define the target classes, provide examples for ambiguous cases, and clarify the meaning of the "can't tell" class.

1. Target Classes:
    a. Neutral: Tweets that do not contain offensive or biased language, and express a neutral or non-controversial sentiment.
    b. Offensive: Tweets that contain offensive, abusive, derogatory, or inappropriate language targeting individuals or groups.
    c. Can't tell: Use this class when the tweet is too ambiguous, lacks context, or the annotator cannot confidently determine whether it belongs to the "neutral" or "offensive" class.
2. Characteristics of Each Class:
    a. Neutral:
        - The tweet presents a non-controversial or unbiased opinion.
        - It does not contain any offensive language, personal attacks, or discriminatory content.
        - The sentiment expressed in the tweet is neither positive nor negative.
        - The tweet express an opinion that is not likely to provoke any strong reactions.
        - It does not contain any profanity, vulgar language, or sexually explicit content.
    b. Offensive:
        - The tweet includes explicit or implicit offensive language, hate speech, or derogatory remarks targeting individuals or groups based on attributes such as race, gender, religion, ethnicity, etc.
        - It contains personal attacks, threats, or intends to demean or harm others.
        - The tweet may provoke anger, disgust, or be considered inappropriate or disrespectful.

- It takes party in political or social discussions that are controversial or sensitive in nature.
- Contains profanity, vulgar language, or sexually explicit content.
- It contains offensive or abusive terms that are used to insult others.
- The tweet expresses extreme political or religious views that are likely to provoke strong reactions.

c. Can't tell:
- Select this class if the tweet is ambiguous, lacks sufficient context, or contains language that makes it difficult to confidently assign it to "neutral" or "offensive."
- The tweet might be written in an unclear or sarcastic tone, making it hard to discern the true intent.
- The tweet could be in a language or cultural context that is unfamiliar to the annotator.

3. Examples of Ambiguous Cases:

a. Ambiguous "neutral" cases:
- Tweets that contain mild sarcasm or irony that might be mistaken for offensive language without proper context.
- Statements that mention controversial topics without expressing a clear opinion (e.g. "People talk out more and more about racism these days", this example should not be considered offensive because it does not express a clear opinion about racism, it just mentions that people talk about it more and more)
- Tweets with ambiguous humor that could be perceived as offensive without further clarification.

b. Ambiguous "offensive" cases:
- Tweets that mention sensitive topics but do not directly contain offensive language (e.g. "Nazis were the enemy of Europe in WWII", this example does not contain direct offensive language, but the topic is sensitive and could be considered offensive by some people).
- Statements that criticize public figures or institutions without crossing the line into offensive territory (e.g. "The president is not doing a good job", this example is not offensive because it does not contain any offensive language, it just expresses an opinion about the president, but should be considered offensive nonetheless because it criticizes a public figure).
- Tweets that include euphemisms, coded language, or implicit offensive content (e.g. "I don't like people who are not like me", this example does not contain any offensive language, but it could be considered offensive by some people because it implies that the author does not like people who are different from him/her)

4. Handling "Can't Tell" Class:

- The "can't tell" class should be used sparingly when there is genuine uncertainty or lack of information to make a clear determination.
- Annotators should strive to provide clear and well-supported annotations for as many tweets as possible.
- Whenever possible, annotators should seek additional context, or utilize external resources to aid in the classification process.

Consistency is key in maintaining high-quality annotations. Annotators should review and famil-

iarize themselves with this guideline thoroughly before starting the annotation process.

3. Every person in your group is going to annotate these tweets separately. So if you are 3, annotate them 3 times Follow this link to find the Google Sheets used for this question: https://docs.google.com/spreadsheets/d/1K44X43JfVI8CpPVSIKrZjdk1wrGfWduXWFaYfKXp7V0/edit?usp=sh

4. Evaluate your inter-annotaor agreement using Fleiss Kappa

[58]:
```
df_annotated_1 = pd.read_csv("annotations/FISCH.csv")
df_annotated_2 = pd.read_csv("annotations/RIPOLL.csv")
df_annotated_3 = pd.read_csv("annotations/FIDEL.csv")
df_annotated_1
```

[58]:
```
                                                 text       label
0                                              agree.     neutral
1   I guess looking at interannotator metrics such…     neutral
2                                 Tried and tested        neutral
3   setiap lihat teman2ku baru main hades and they…   offensive
4                                       Free Britney  can't tell
..                                                 …         …
95  He rather have used the cash to buy some cloth…     neutral
96                        Come one come all into 6988     neutral
97  Make up, dress up, like a princess \nfor him s…   offensive
98    "Tell me then, YOU BROKEDICK SON OF A BITCH…   offensive
99  This is a man thing? I keep finding out I'm a …     neutral

[100 rows x 2 columns]
```

[59]:
```
df_annotated = pd.concat([df_annotated_1, df_annotated_2, df_annotated_3],␣
 ↪axis=1)
df_annotated.columns = ["text", "label_1", "text_2", "label_2", "text_3",␣
 ↪"label_3"]
df_annotated.drop(columns=["text_2", "text_3"], axis=1, inplace=True)
df_annotated
```

[59]:
```
                                                 text     label_1     label_2  \
0                                              agree.     neutral     neutral
1   I guess looking at interannotator metrics such…     neutral     neutral
2                                 Tried and tested        neutral     neutral
3   setiap lihat teman2ku baru main hades and they…   offensive   offensive
4                                       Free Britney  can't tell     neutral
..                                                 …         …         …
95  He rather have used the cash to buy some cloth…     neutral     neutral
96                        Come one come all into 6988     neutral  can't tell
97  Make up, dress up, like a princess \nfor him s…   offensive   offensive
98    "Tell me then, YOU BROKEDICK SON OF A BITCH…   offensive   offensive
99  This is a man thing? I keep finding out I'm a …     neutral     neutral
```

```
       label_3
0      neutral
1      neutral
2      neutral
3    offensive
4   can't tell
..         …
95     neutral
96  can't tell
97     neutral
98   offensive
99   offensive

[100 rows x 4 columns]
```

[60]:
```
df_annotated_with_text = df_annotated.copy()
df_annotated = df_annotated.drop(columns=["text"])
df_annotated = df_annotated.applymap(lambda x: 1 if x == "offensive" else 0 if␣
 ↪x == "neutral" else 2)
df_annotated
```

[60]:
```
    label_1  label_2  label_3
0         0        0        0
1         0        0        0
2         0        0        0
3         1        1        1
4         2        0        2
..      …        …        …
95        0        0        0
96        0        2        2
97        1        1        0
98        1        1        1
99        0        0        1

[100 rows x 3 columns]
```

[61]:
```
all_agreed = df_annotated.apply(lambda x: 1 if x["label_1"] == x["label_2"] ==␣
 ↪x["label_3"] else 0, axis=1).sum()
some_agreed = df_annotated.apply(lambda x: 1 if x["label_1"] == x["label_2"] or␣
 ↪x["label_1"] == x["label_3"] or x["label_2"] == x["label_3"] else 0, axis=1).
 ↪sum()
print(f"Percentage of all agreed: {all_agreed / len(df_annotated) * 100:.2f}%")
print(f"Percentage of some agreed: {some_agreed / len(df_annotated) * 100:.
 ↪2f}%")
```

```
Percentage of all agreed: 52.00%
Percentage of some agreed: 96.00%
```

Out of 100 samples, we all agreed on 52 samples, and 2/3 agreed on 96 samples.

[62]:
```
vals, _ = ir.aggregate_raters(df_annotated.values)
ir.fleiss_kappa(vals, method='fleiss')
```

[62]:
```
0.3263876414585947
```

We obtained a Fleiss Kappa score of 0.32 for the 3 annotators. This score is considered as a "Fair agreement" according to Wikipedia's table. Some of the differences seem to be: - Where sex related words are considered as offensive or not - Where insults that are expressing a strong opinion are considered as offensive or not (e.g. "That's fucking good" is not negative, but contains an insult) - Do we classify random words as "Can't tell" or "neutral" ? (e.g. "literally" was one of the examples were it was classified as "can't tell" and "neutral")

6. (Bonus) Evaluate the model your data. Use a majority vote for labels (remove majority "can't tell") and compute the precision, recall, and F1-score.

[63]:
```
# merge the labels (majority vote) and remove when the majority is 2 (uncertain)
df_annotated_final = df_annotated_with_text.copy()
df_annotated_final[["label_1", "label_2", "label_3"]] =␣
 ↪df_annotated_final[["label_1", "label_2", "label_3"]].applymap(lambda x: 1␣
 ↪if x == "offensive" else 0 if x == "neutral" else 2)
df_annotated_final["label"] = df_annotated_final.apply(lambda x: x["label_1"]␣
 ↪if x["label_1"] == x["label_2"] else x["label_2"] if x["label_2"] ==␣
 ↪x["label_3"] else x["label_1"] if x["label_1"] == x["label_3"] else 2,␣
 ↪axis=1)
df_annotated_final = df_annotated_final[df_annotated_final["label"] != 2]
df_annotated_final = df_annotated_final.drop(columns=["label_1", "label_2",␣
 ↪"label_3"])
df_annotated_final
```

[63]:

	text	label
0	agree.	0
1	I guess looking at interannotator metrics such…	0
2	Tried and tested	0
3	setiap lihat teman2ku baru main hades and they…	1
5	Correction: Eastern WA will see *smoke from th…	0
..	…	…
94	You should raise the webform…how would they…	0
95	He rather have used the cash to buy some cloth…	0
97	Make up, dress up, like a princess \nfor him s…	1
98	"Tell me then, YOU BROKEDICK SON OF A BITCH…	1
99	This is a man thing? I keep finding out I'm a …	0

[91 rows x 2 columns]

[65]:
```
annotated_preds = roberta_pipeline(df_annotated_final["text"].tolist())
evaluate(annotated_preds, df_annotated_final["label"].tolist())
```

	precision	recall	f1-score	support
0	0.97	0.89	0.93	70
1	0.70	0.90	0.79	21
accuracy			0.89	91
macro avg	0.84	0.90	0.86	91
weighted avg	0.91	0.89	0.89	91

Interestingly, the F1 score is better on our annotated data with a F1 score of 0.86 than with all the tweets. An explanation can be that we have 100x less samples on our annotated data than on the tweets dataset.