

```
# !pip install transformers datasets evaluate scikit-learn torch
```

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
True
```

```
# import kagglehub
# waalbannyantudre_hate_speech_detection_curated_dataset_path = kagglehub.dataset_download('waalbannyantudre/hate-speech-detection-curated-dataset')

print('Data source import complete.')
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
import random
from tensorflow.keras.layers import Input,Dense,Embedding, LSTM,Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
import re
from sklearn.metrics import accuracy_score
from transformers import DistilBertTokenizer, TFDistilBertForSequenceClassification
from tensorflow.keras.optimizers import Adam
from nltk.corpus import stopwords
import nltk
# nltk.download('stopwords')
```

```
Data source import complete.
```

```
dataset_path = "/kaggle/input/hate-speech-detection-curated-dataset/HateSpeechDatasetBalanced.csv"
df = pd.read_csv(dataset_path)
df.head()
```

```
Content Label
0 denial of normal the con be asked to comment o... 1
1 just by being able to tweet this insufferable ... 1
2 that is retarded you too cute to be single tha... 1
3 thought of a real badass mongol style declarat... 1
4 afro american hasho 1
```

## DATA CLEANING

```
stop_words = set(stopwords.words('english'))

#Text Preprocessing Function
def clean_text(text):
    text = re.sub(r'^a-zA-Z|', ' ', text)
    text = text.lower()
    words = text.split()
    filtered_words = [word for word in words if word not in stop_words]
    return " ".join(filtered_words)

df['CleanedContent'] = df['Content'].apply(clean_text)
```

## TEST TRAIN SPLIT

```
#LIMITED DF
limited_df = df.sample(n=200, random_state=42)
X_train_texts, X_test_texts, y_train, y_test = train_test_split(
    limited_df['CleanedContent'].values,
    limited_df['Label'].values,
    test_size=0.2,
    random_state=42
)
```

```
# X_train_texts, X_test_texts, y_train, y_test = train_test_split(
#     df['CleanedContent'].values,
#     df['Label'].values,
#     test_size=0.2,
#     random_state=42
# )
```

```
print("Training set size:", len(X_train_texts))
print("Test set size:", len(X_test_texts))
```

↗ Training set size: 160  
Test set size: 40

## HUGGINGFACE TRANSFORMERS

```
from transformers import DistilBertTokenizer, TFDistilBertForSequenceClassification
```

```
# Load pretrained model and tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=1)
```

↗ /usr/local/lib/python3.11/dist-packages/huggingface\_hub/utils/\_auth.py:94: UserWarning:  
The secret `HF\_TOKEN` does not exist in your Colab secrets.  
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>), set it as :  
You will be able to reuse this secret in all of your notebooks.  
Please note that authentication is recommended but still optional to access public models or datasets.  
warnings.warn(  
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFDistilBertForSequenceClassification: ['vocab\_l  
- This IS expected if you are initializing TFDistilBertForSequenceClassification from a PyTorch model trained on another task or wif  
- This IS NOT expected if you are initializing TFDistilBertForSequenceClassification from a PyTorch model that you expect to be exac  
Some weights or buffers of the TF 2.0 model TFDistilBertForSequenceClassification were not initialized from the PyTorch model and ar  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Double-click (or enter) to edit

## PRE PROCESSING

```
# Tokenize texts (use max_length=512 for typical BERT-based models)
train_encodings = tokenizer(X_train_texts.tolist(), truncation=True, padding=True, max_length=512, return_tensors="tf")
test_encodings = tokenizer(X_test_texts.tolist(), truncation=True, padding=True, max_length=512, return_tensors="tf")
```

```
import tensorflow as tf
```

```
# Convert to TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    y_train
)).batch(128)
```

```
test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    y_test
)).batch(128)
```

```
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
```

```
# Compile the model
# optimizer = Adam(learning_rate=5e-5)
# print(optimizer)
loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
metrics = ['accuracy']
```

```
model.compile(optimizer='adam', loss=loss, metrics=metrics)
```

## TRAINING

```
history = model.fit(
    train_dataset,
```

```
validation_data=test_dataset,
epochs=5
)
```

```
↔ Epoch 1/5
2/2 [=====] - 183s 48s/step - loss: 0.7150 - accuracy: 0.3875 - val_loss: 0.6363 - val_accuracy: 0.6750
Epoch 2/5
2/2 [=====] - 145s 40s/step - loss: 0.8860 - accuracy: 0.3938 - val_loss: 0.7106 - val_accuracy: 0.6750
Epoch 3/5
2/2 [=====] - 142s 40s/step - loss: 0.6983 - accuracy: 0.4125 - val_loss: 0.6980 - val_accuracy: 0.6750
Epoch 4/5
2/2 [=====] - 142s 39s/step - loss: 0.6932 - accuracy: 0.4187 - val_loss: 0.6885 - val_accuracy: 0.6750
Epoch 5/5
2/2 [=====] - 142s 39s/step - loss: 0.6985 - accuracy: 0.4187 - val_loss: 0.6994 - val_accuracy: 0.6750
```

## PREDICTIONS

```
y_pred_logits = model.predict(test_dataset).logits
y_pred_probs = tf.sigmoid(y_pred_logits).numpy().squeeze()
```

```
# Convert probabilities to labels (threshold > 0.5)
y_pred_labels = (y_pred_probs > 0.5).astype(int)
```

```
↔ 2/2 [=====] - 15s 3s/step
```

```
import random
```


```
# Test a sentence
for i in range(20):
    num = random.randint(1, len(X_test_texts))
    print(X_test_texts[num],f"Prob: {y_pred_probs[num]:.4f} -> Label: {y_pred_labels[num]}")
```

```
↔ protection expire three months end time left wikipedia alone created additional accounts problems maybe someone open hearing say sir
note click line might need clean obvious intent line Prob: 0.5946 -> Label: 1
censorship article constantly censored zionist hawks supposed even handed analysis phony propagandists like jay jg espouse bullshit
love https www linkedin com pub wayne weber got air force went straight head hk wow Prob: 0.5946 -> Label: 1
love https www linkedin com pub wayne weber got air force went straight head hk wow Prob: 0.5946 -> Label: 1
znali itk dobro doctor squad exertion jude priporuci za license bill biam roditelji galvani cell pivot Prob: 0.5946 -> Label: 1
afd nomination zambian wolpertinger nominated zambian wolpertinger article created deletion feel article satisfies wikipedia criteri:
sucks ass Prob: 0.5945 -> Label: 1
worthwhile keeping old images display article well new images utc Prob: 0.5946 -> Label: 1
newsletter draft ready hi dank prepared initial draft drive wrap newsletter wikipediawikiproject guild copy could find time check gc
censorship article constantly censored zionist hawks supposed even handed analysis phony propagandists like jay jg espouse bullshit
newsletter draft ready hi dank prepared initial draft drive wrap newsletter wikipediawikiproject guild copy could find time check gc
man put false words mouth finish whatever edit conflicted read whole thing first Prob: 0.5946 -> Label: 1
quite welcome glad could help Prob: 0.5946 -> Label: 1
protection expire three months end time left wikipedia alone created additional accounts problems maybe someone open hearing say sir
women simply whores detest race Prob: 0.5945 -> Label: 1
cannot believe calls favorable treatment eu migrants brexit Prob: 0.5946 -> Label: 1
mls outrage listen conference xbox next gen fifa article screwed fix information technology operating theater logging away Prob: 0.5
tom lazovi jedi get nice rumors djevojci Prob: 0.5946 -> Label: 1
man put false words mouth finish whatever edit conflicted read whole thing first Prob: 0.5946 -> Label: 1
```

## GETTING TOXICITY SCORES

```
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
import tensorflow as tf
import numpy as np
```

```
# Load tokenizer and model
model_name = "unitary/toxic-bert"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name)
```

	tokenizer_config.json: 100%	174/174 [00:00<00:00, 3.27kB/s]
	config.json: 100%	811/811 [00:00<00:00, 17.8kB/s]
	vocab.txt: 100%	232k/232k [00:00<00:00, 2.00MB/s]
	special_tokens_map.json: 100%	112/112 [00:00<00:00, 3.05kB/s]
	model.safetensors: 100%	438M/438M [00:06<00:00, 117MB/s]

Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertForSequenceClassification: ['bert.embedding  
- This IS expected if you are initializing TFBertForSequenceClassification from a PyTorch model trained on another task or with another  
- This IS NOT expected if you are initializing TFBertForSequenceClassification from a PyTorch model that you expect to be exactly identical  
All the weights of TFBertForSequenceClassification were initialized from the PyTorch model.

If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertForSequenceClassification

```
def get_toxicity(text):
    # Tokenize
    inputs = tokenizer(text, return_tensors="tf", truncation=True, padding=True)

    # Predict logits
    outputs = model(**inputs)
    logits = outputs.logits

    # Apply sigmoid to get probabilities
    probs = tf.sigmoid(logits)[0].numpy()

    # The model has multiple classes: toxic, severe toxic, obscene, threat, insult, identity hate
    labels = [
        "toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"
    ]

    # Convert to % and return as dict
    toxicity_scores = {label: round(prob * 100, 2) for label, prob in zip(labels, probs)}
    return toxicity_scores
```

```
import random
```

```
# Test a sentence
for i in range(20):
    num = random.randint(1, len(X_test_texts))
    scores = get_toxicity(X_test_texts[num])
    print("\n", X_test_texts[num])
    formatted_scores = {k: round(float(v), 2) for k, v in scores.items()}
    for label, score in formatted_scores.items():
        print(f"{label.title()}: {score}%")
```

```
# print("Toxicity Levels (%)ate", scores)
```



Inreac: 0.01%  
Insult: 0.06%  
Identity\_Hate: 0.02%

cannot believe calls favorable treatment eu migrants brexit  
Toxic: 0.12%  
Severe\_Toxic: 0.01%  
Obscene: 0.02%  
Threat: 0.01%  
Insult: 0.02%  
Identity\_Hate: 0.02%

let us go memory lane rapist terrorist trump anti trump putinsbitch traitor liarinchief dementedinchief predator justiceforkatic  
Toxic: 73.78%  
Severe\_Toxic: 0.76%  
Obscene: 21.25%  
Threat: 0.73%  
Insult: 17.46%  
Identity\_Hate: 6.36%

newsletter draft ready hi dank prepared initial draft drive wrap newsletter wikipediawikiproject guild copy could find time che  
Toxic: 0.08%  
Severe\_Toxic: 0.01%  
Obscene: 0.02%  
Threat: 0.01%  
Insult: 0.02%