Toxicity Detection

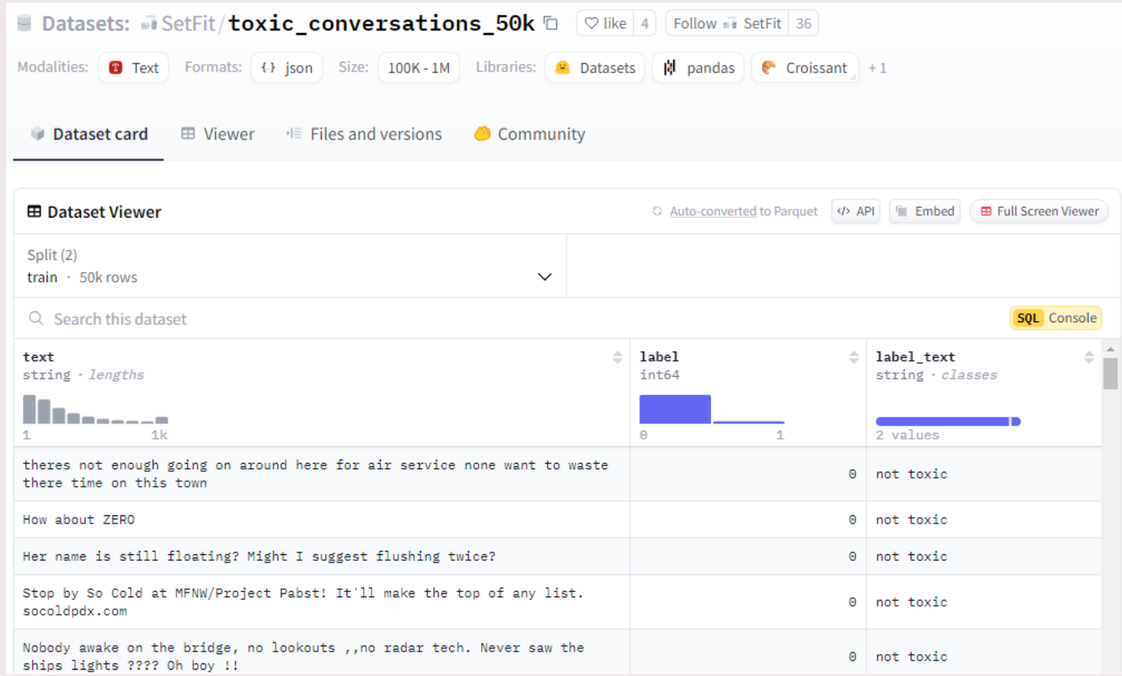
Guillermo David
James Latanna
Matthias Mangold

Dataset Selection & Preprocessing

Dataset Selection:

The dataset is a subset of the Jigsaw Unintended Bias in Toxicity Classification dataset, containing the first 50,000 training examples curated for toxicity classification tasks.

Toxicity is defined when the target value is 0.5 or higher.



Features:

- text: The conversational text
- label: Binary label indicating toxicity
- label_text: Textual representation of the label

TOXIC

```
import pandas as pd
splits = {'train': 'train.jsonl', 'test': 'test.jsonl'}
df = pd.read_json("hf://datasets/SetFit/toxic_conversations_50k/" + splits["train"], lines=True)
```

Quality of dataset selection:

```
# General dataset information
print("Dataset Shape:", df.shape)
print("Columns:", df.columns)

# Check for missing values
print("Missing Values:\n", df.isnull().sum())

# Check for duplicates
print("Number of Duplicate Rows:", df.duplicated().sum())
```

```
Dataset Shape: (50000, 3)
Columns: Index(['text', 'label', 'label_text'], dtype='object')
Missing Values:
 text          0
label         0
label_text    0
dtype: int64
Number of Duplicate Rows: 100
```

- 50000 rows, 3 columns
- no missing values
- 100 duplicates

Removing duplicates:

```python
# Remove duplicate rows
df = df.drop_duplicates().reset_index(drop=True)
print("Dataset Shape After Removing Duplicates:", df.shape)
print("Number of Duplicate Rows:", df.duplicated().sum())
```

```
Dataset Shape After Removing Duplicates: (49900, 3)
Number of Duplicate Rows: 0
```

- Remove duplicates
- Check result

Dataset Selection & Preprocessing

Preprocessing the Data

- Convert to lowercase
- Remove special characters
- Check result

```python
import re

# Function to preprocess text
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation and special characters
    text = re.sub(r'[^a-z\s]', '', text)
    return text

# Apply preprocessing
df['cleaned_text'] = df['text'].apply(preprocess_text)

# Preview cleaned text
df[['text', 'cleaned_text']].head()
```

	text	cleaned_text
0	theres not enough going on around here for air...	theres not enough going on around here for air...
1	How about ZERO	how about zero
2	Her name is still floating? Might I suggest f...	her name is still floating might i suggest fl...
3	Stop by So Cold at MFNW/Project Pabst! It'll m...	stop by so cold at mfnwproject pabst itll make...
4	Nobody awake on the bridge, no lookouts ,,no r...	nobody awake on the bridge no lookouts no rada...

EDA

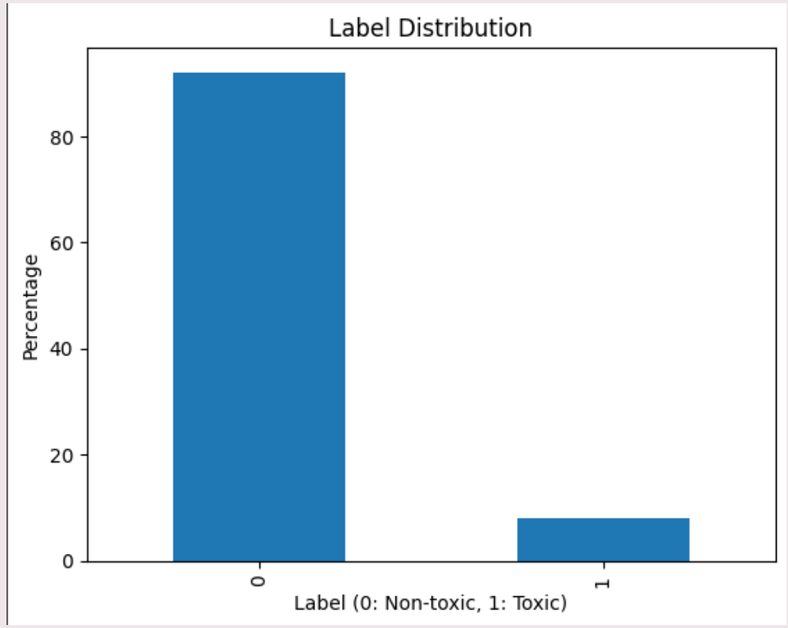
Around 8% of the dataset is labeled as toxic, making it highly imbalanced.

```python
# Label distribution
label_distribution = df['label'].value_counts(normalize=True) * 100
print("Label Distribution:\n", label_distribution)

# Text length statistics
df['text_length'] = df['text'].apply(len)
print("Text Length Statistics:\n", df['text_length'].describe())
```

```
Label Distribution:
 label
0    92.056112
1     7.943888
Name: proportion, dtype: float64
Text Length Statistics:
 count    49900.000000
mean       299.286273
std        270.072584
min          1.000000
25%         95.000000
50%        205.000000
75%        415.000000
max       1000.000000
Name: text_length, dtype: float64
```

Label distribution:



Non-toxic: 92.06%

Toxic: 7.94%

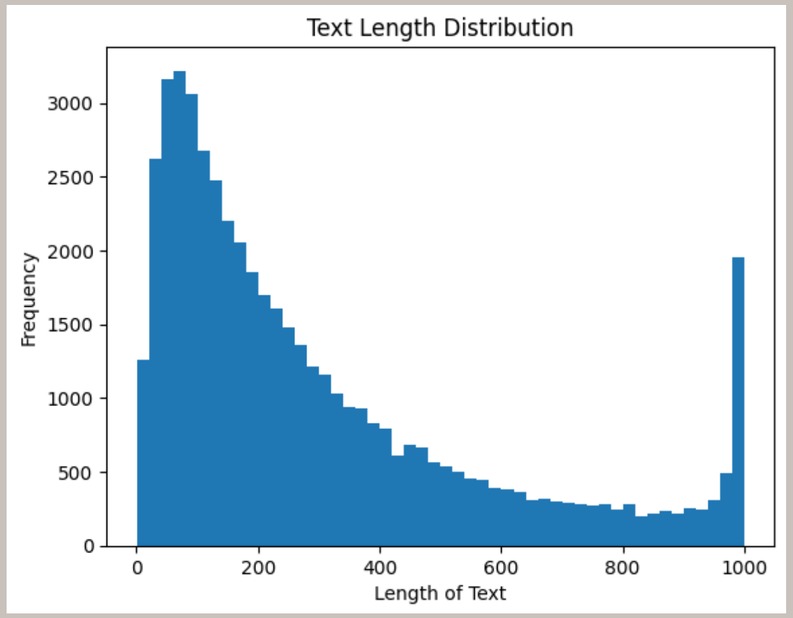
EDA

```python
# Label distribution
label_distribution = df['label'].value_counts(normalize=True) * 100
print("Label Distribution:\n", label_distribution)

# Text length statistics
df['text_length'] = df['text'].apply(len)
print("Text Length Statistics:\n", df['text_length'].describe())
```

```
Label Distribution:
 label
0    92.056112
1     7.943888
Name: proportion, dtype: float64
Text Length Statistics:
 count    49900.000000
mean       299.286273
std        270.072584
min          1.000000
25%         95.000000
50%        205.000000
75%        415.000000
max       1000.000000
Name: text_length, dtype: float64
```

The average comment length is 299 characters, with a median of 204 characters. Some comments are as short as 1 character, while others are up to 1,000 characters.



Text length distribution

Model Description

This model we used is a fine-tuned version of the s-nlp/roberta_toxicity_classifier model to classify toxic comments.

Limitations and Bias

This model is intended to be used for classifying toxic online classifications. However, one limitation of the model is that immediately classifies a sentence that includes profanity a toxic one

Implementation & Use of Pre-trained Model using Hugging Face

- Successful implementation of a relevant pre -trained model.
- Proper application of the model on your dataset.

check if the dataset will load

perform the toxic model
for the said dataset

Implement the model on the

see the first 10 rows

```
df_5000.head(10)
```

	text	label	label_text
0	theres not enough going on around here for air...	0	not toxic
1	how about zero	0	not toxic
2	her name still floating might i suggest flush...	0	not toxic
3	stop by so cold at mfnwproject pabst itll make...	0	not toxic
4	nobody awake on bridge no lookouts no radar te...	0	not toxic
5	pessimists might say that we us will have a go...	0	not toxic
6	im sure hawaiian can put southwest out of busi...	0	not toxic
7	states dont grow economies cities grow economy...	0	not toxic
8	what makes magnitsky so special countless thou...	0	not toxic
9	the mayor said we need more skin in game that...	0	not toxic

perform the toxic classifier model on the dataset

```
#turn it into a list
result = pipe(df_5000['text'].tolist())
```

```
result
```

```
[{'label': 'neutral', 'score': 0.9999489784240723},
 {'label': 'neutral', 'score': 0.9999027252197266},
 {'label': 'neutral', 'score': 0.9998712539672852},
 {'label': 'neutral', 'score': 0.9994361996650696},
 {'label': 'neutral', 'score': 0.9999505281448364},
 {'label': 'neutral', 'score': 0.9993759989738464},
 {'label': 'neutral', 'score': 0.9999643564224243},
 {'label': 'neutral', 'score': 0.9998916387557983},
 {'label': 'neutral', 'score': 0.9999550580978394},
 {'label': 'neutral', 'score': 0.9998138546943665},
 {'label': 'neutral', 'score': 0.9999642372131348},
 {'label': 'neutral', 'score': 0.9948996901512146},
 {'label': 'neutral', 'score': 0.9997825026512146},
 {'label': 'neutral', 'score': 0.9991825222969055},
 {'label': 'neutral', 'score': 0.957952082157135},
 {'label': 'neutral', 'score': 0.9999587535858154},
 {'label': 'neutral', 'score': 0.9998555183410645},
 {'label': 'neutral', 'score': 0.9970824122428894},
 {'label': 'neutral', 'score': 0.9999492168426514},
 {'label': 'neutral', 'score': 0.9959064722061157},
 {'label': 'neutral', 'score': 0.9999669790267944},
 {'label': 'neutral', 'score': 0.999620795249939},
 {'label': 'neutral', 'score': 0.9999626874923706},
 {'label': 'toxic', 'score': 0.9770503044128418},
 {'label': 'neutral', 'score': 0.9905894994735718},
 {'label': 'neutral', 'score': 0.9999531507492065},
 {'label': 'neutral', 'score': 0.9814943075180054},
 {'label': 'neutral', 'score': 0.9997186064720154},
 {'label': 'neutral', 'score': 0.9996829032897949},
```

```python
# Assuming 'result' is your list of dictionaries
import pandas as pd

data = []
for entry in result:
    # Check if 'score' is iterable
    if isinstance(entry['score'], (list, tuple)):
        # Find the label with score > 0.5
        label = None
        score = None
        for l, s in zip(entry['label'], entry['score']):
            if s > 0.5:
                label = l
                score = s
                break  # Stop after finding the first label with score > 0.5

        # Append to data list
        data.append({
            'label': label,
            'score': score
        })
    else:  # Handle cases where 'score' is a single float
        label = entry['label']
        score = entry['score'] if entry['score'] > 0.5 else None  # Assign score if > 0.5, else None
        data.append({
            'label': label,
            'score': score
        })

# Convert to DataFrame
predicteddata = pd.DataFrame(data)

predicteddata['text'] = df_5000['text']

predicteddata
```

	label	score	text
0	neutral	0.999949	theres not enough going on around here for air...
1	neutral	0.999903	how about zero
2	neutral	0.999871	her name still floating might i suggest flush...
3	neutral	0.999436	stop by so cold at mfnwproject pabst itll make...
4	neutral	0.999951	nobody awake on bridge no lookouts no radar te...
...	...	...	...
4995	neutral	0.999645	i would bet that judge has insurance in witch ...
4996	neutral	0.999793	the end game for him to undo everything obama ...
4997	neutral	0.982541	like ayatollah khomenei hes gone where goblins...
4998	neutral	0.999964	this should be headline\n\nthe neighborhood bo...
4999	neutral	0.525550	hillary clinton did make donald trump possible...

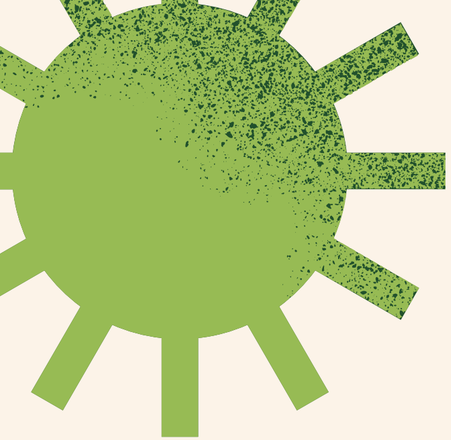
5000 rows × 3 columns

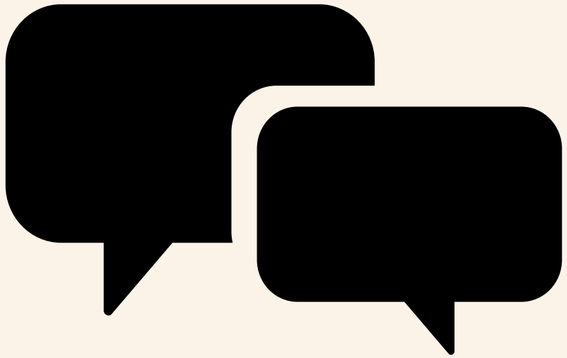
If score is iterable: It selects the first label with a score > 0.5.

If score is a single float: It checks if the score > 0.5 and keeps the label and score if true.

Creates a DataFrame predicteddata with label and score columns.

Adds a text column from an existing DataFrame (df_5000).

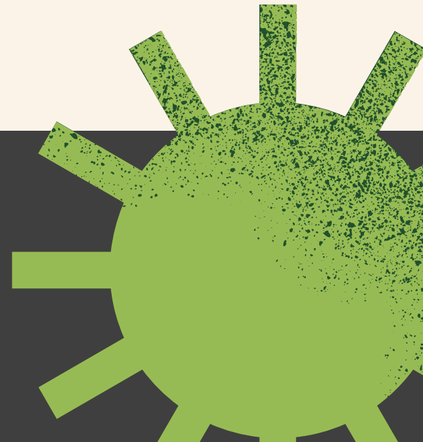




```
dataset = pd.DataFrame()
dataset['true_label'] = df_5000['label_text']
dataset['label'] = predicteddata['label']
dataset['score'] = predicteddata['score']
dataset
```

	true_label	label	score
0	not toxic	neutral	0.999949
1	not toxic	neutral	0.999903
2	not toxic	neutral	0.999871
3	not toxic	neutral	0.999436
4	not toxic	neutral	0.999951
...	...	...	...
4995	not toxic	neutral	0.999645
4996	not toxic	neutral	0.999793
4997	not toxic	neutral	0.982541
4998	not toxic	neutral	0.999964
4999	toxic	neutral	0.525550

5000 rows × 3 columns



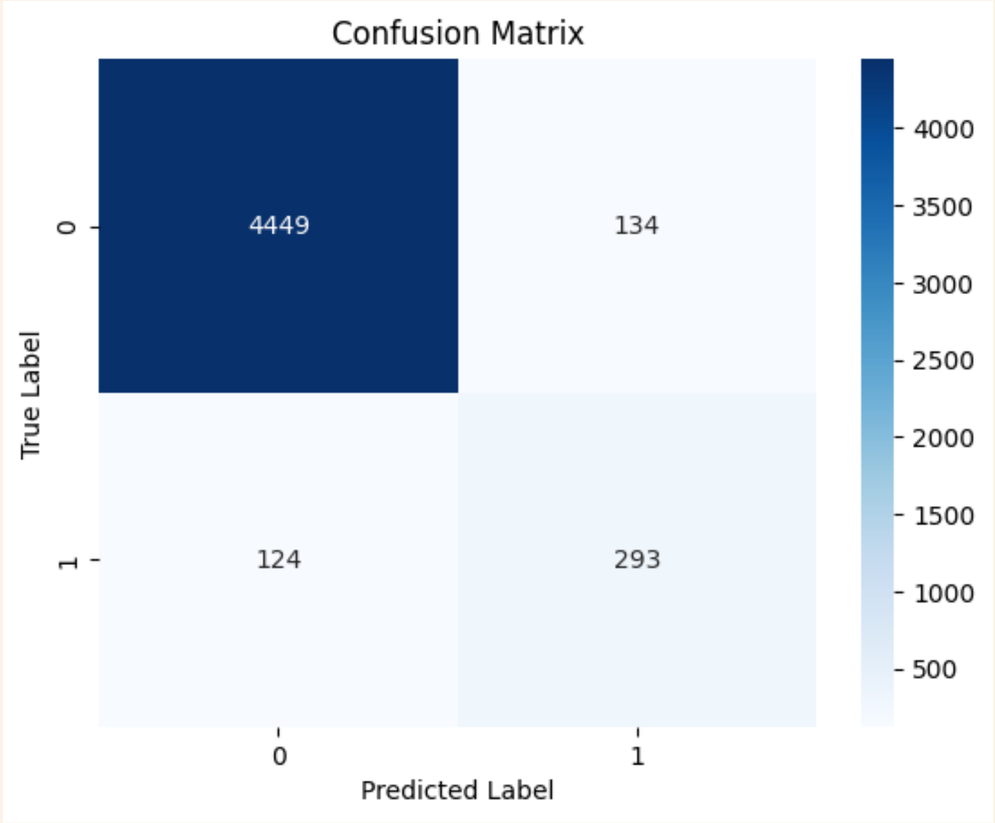
Compare

Analysis & Presentation Quality

- Clear explanation of results and insights.
- Well-organized and concise presentation or report.

confusion matrix

confusion matrix



True Positive (TP): 293

False Negative (FN): 124

False Positive (FP): 134

True Negative(TN): 4449

In summary

- True Positive (TP): 293
- False Negative (FN): 124
- False Positive (FP): 134
- True Negative(TN): 4449

Specificity: TN/(TN+FP)
4449/(4449+134) = 0.97

Accuracy: Measures overall correctness.

```
Accuracy: 0.9484
```

Precision: Measures how many predicted toxic comments are actually toxic.

```
Precision: 0.6862
```

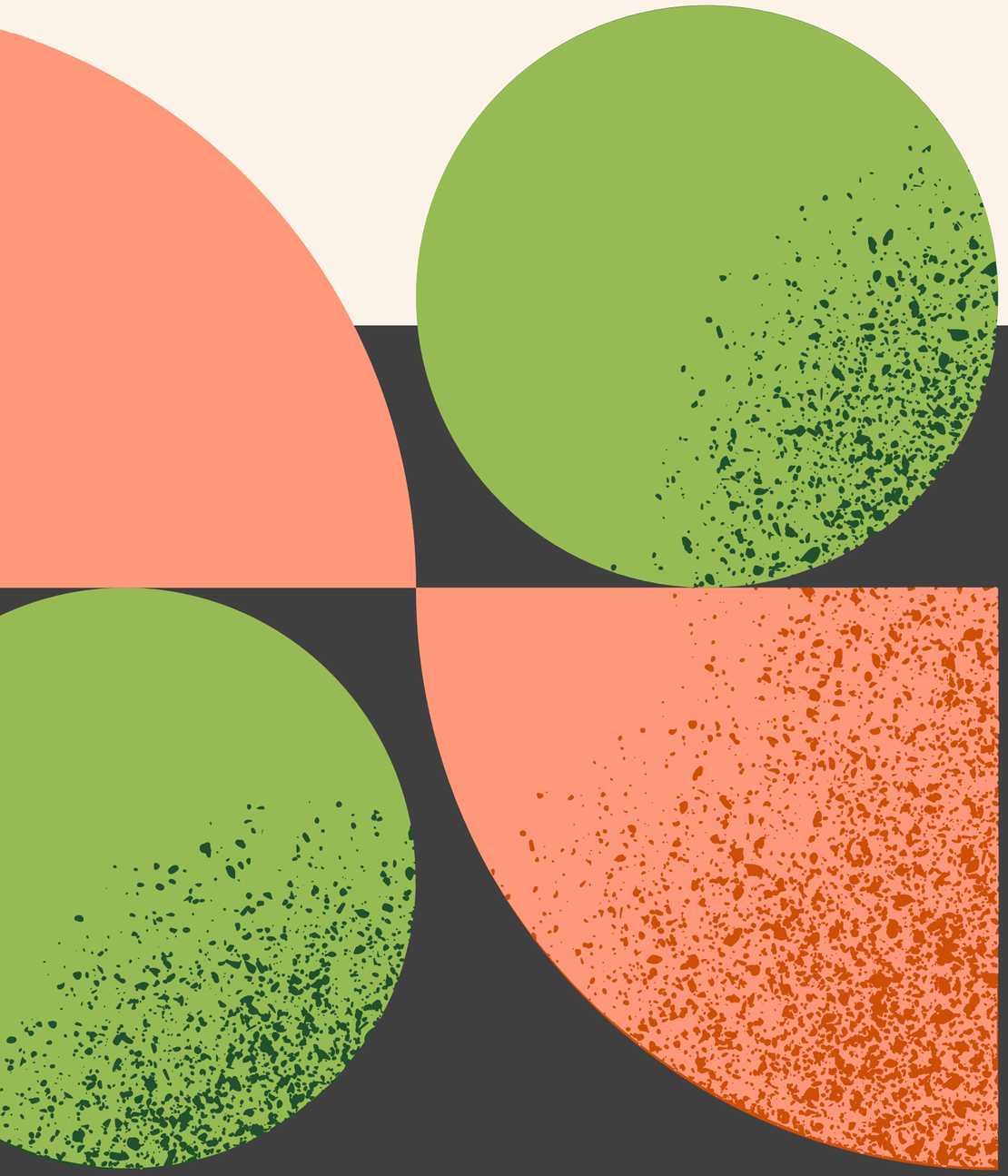
Recall: Measures how many actual toxic comments are identified as toxic.

```
Recall: 0.7026
```

F1 score: F1 Score is a metric that balances precision and recall.

```
F1-Score: 0.6943
```

Insights with key metrics



Detection of toxic comments

- High accuracy (94.84%) indicates good overall performance.
- The precision (68.61%) and recall (70.26%) for toxic comments are moderate, suggesting the model struggles to identify and isolate toxic comments effectively.
- The specificity (to identify neutral comments correctly) we can infer that the model is better for avoiding false positives (wrongly flagging neutral as toxic) than catching all toxic comments.

Fine-tuned model

```python
from datasets import load_dataset

# Load the dataset
dataset = load_dataset("SetFit/toxic_conversations_50k", split="train")

# Use a subset of the first 1000 instances
subset = dataset.select(range(1000))

# Split into train and validation sets (80-20 split)
split = subset.train_test_split(test_size=0.2, seed=42)
train_data = split["train"]
val_data = split["test"]
```

- Loads the dataset: Fetches the SetFit/toxic_conversations_50k dataset from Hugging Face, specifically the train split.
- Creates a subset: Selects the first 1000 instances of the dataset.
- Splits into train and validation sets: Divides the subset into an 80 -20 split, creating train_data and val_data.

```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("s-nlp/roberta_toxicity_classifier")

def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True, padding="max_length", max_length=128)

train_data = train_data.map(tokenize_function, batched=True)
val_data = val_data.map(tokenize_function, batched=True)
```

```
Map: 100% 800/800 [00:00<00:00, 1014.05 examples/s]
Map: 100% 200/200 [00:00<00:00, 1116.39 examples/s]
```

- Loads the tokenizer: It uses the AutoTokenizer to load the pre -trained tokenizer
- Defines a tokenize_function: This function tokenizes the text, applying truncation and padding to a maximum length of 128 tokens.
- Applies tokenization: The train_data and val_data datasets are tokenized

```python
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained("s-nlp/roberta_toxicity_classifier", num_labels=2)
```

- This loads a pre -trained RoBERTa toxicity classifier with two output labels

```python
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    push_to_hub=False,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=val_data,
    tokenizer=tokenizer,
)
```

- TrainingArguments: Configures training parameters
- Trainer: Combines the model, datasets, and tokenizer for training and evaluation.

```
trainer.train()
```

```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run name by setting the `Traini
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit: ··········
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
Tracking run with wandb version 0.18.7
Run data is saved locally in /content/wandb/run-20241126_191138-a7l7z4bk
Syncing run ./results to Weights & Biases (docs)
View project at https://wandb.ai/james13112004-/huggingface
View run at https://wandb.ai/james13112004-/huggingface/runs/a7l7z4bk
[150/150 1:02:21, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss
1	0.085100	0.147689
2	0.052500	0.277617
3	0.026100	0.314203

```
TrainOutput(global_step=150, training_loss=0.1035738064845403, metrics={'train_runtime': 3809.8016, 'train_samples_per_second': 0.63, 'train_steps_per_second': 0.039, 'total_flos':
157866633216000.0, 'train_loss': 0.1035738064845403, 'epoch': 3.0})
```

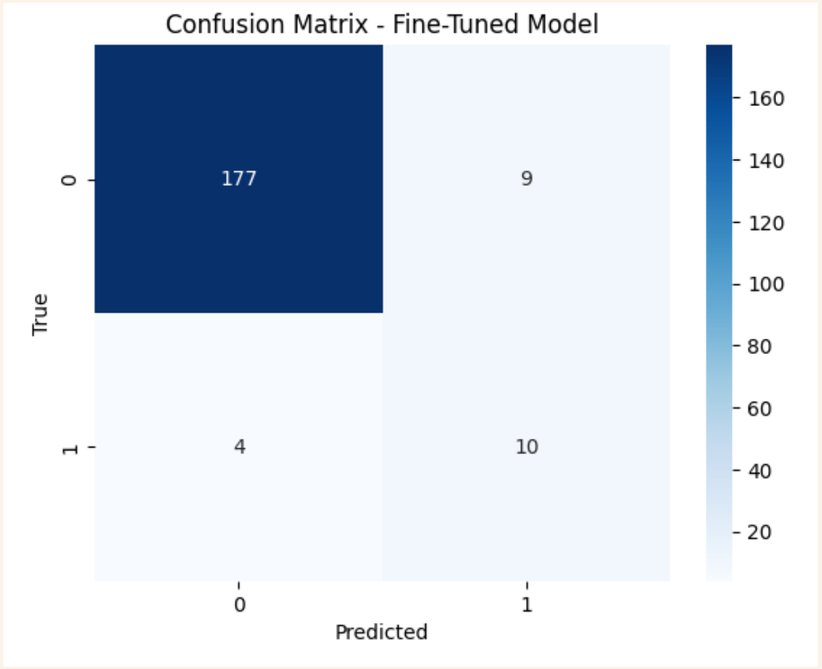
- starts the training process using the provided datasets and training parameters. It will train the model on train_data for the specified number of epochs and evaluate it on val_data as defined in training_args

```
model.save_pretrained("./fine_tuned_toxicity_model")
tokenizer.save_pretrained("./fine_tuned_toxicity_model")
```

```
('./fine_tuned_toxicity_model/tokenizer_config.json',
 './fine_tuned_toxicity_model/special_tokens_map.json',
 './fine_tuned_toxicity_model/vocab.json',
 './fine_tuned_toxicity_model/merges.txt',
 './fine_tuned_toxicity_model/added_tokens.json',
 './fine_tuned_toxicity_model/tokenizer.json')
```

save the fine-tuned model and tokenizer to the specified directory (./fine_tuned_toxicity_model).

Accuracy: 0.9350
Precision: 0.5263
Recall: 0.7143
F1-Score: 0.6061

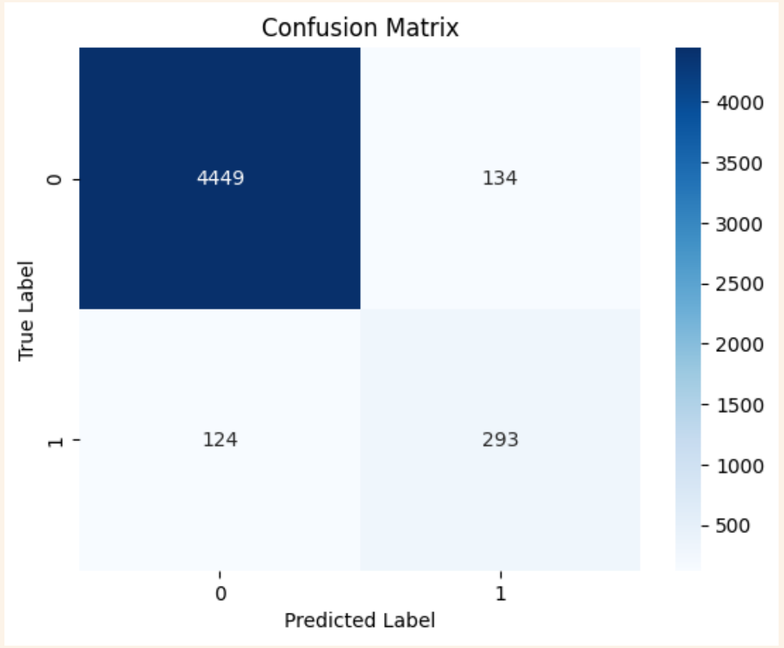


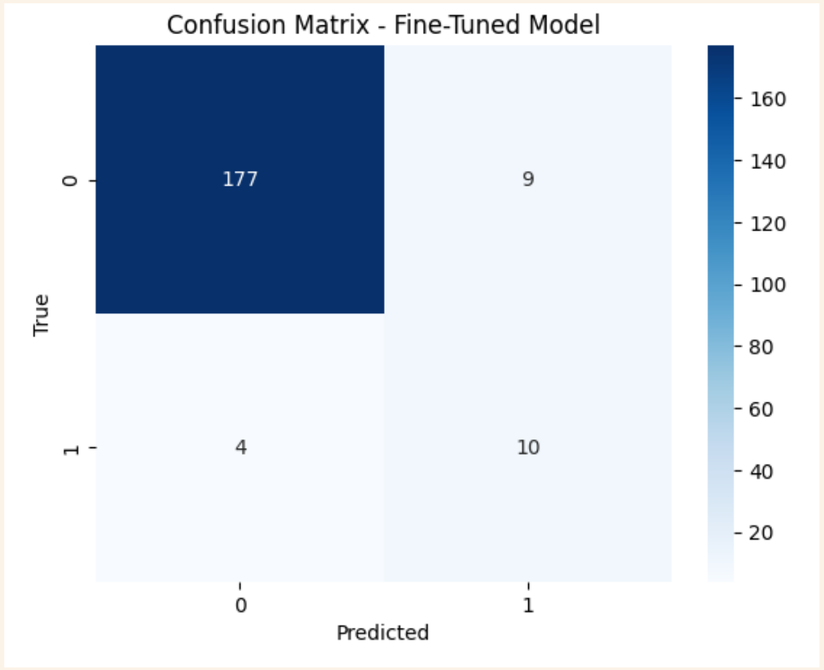
side by side

The pre-trained model has slightly higher accuracy and recall, while the fine-tuned model improves precision but lowers recall and F1-score.

Metric	Pre-trained Model	Fine-tuned Model
Accuracy	0.9484	0.935
Precision	0.6861826697892272	0.5263157894736842
Recall	0.7026378896882494	0.7142857142857143
F1-Score	0.6943127962085308	0.6060606060606061

pre trained





fine tuned

- The accuracy is good (94%).
- However, it fails to identify 30% of toxic comments (recall 70%), meaning these comments are incorrectly classified as non-toxic (false negatives).
- The model prioritizes specificity over sensitivity, meaning it is better at avoiding false positives than catching all toxic comments.
- Toxic comments harm individuals, degrade community quality, and undermine platform integrity. Addressing this issue is critical for creating inclusive, engaging, and sustainable online spaces.
- Implementing robust moderation systems and fostering a culture of respect are essential steps toward mitigating the impact of toxicity on online communities.

Conclusion

Conclusion

thank

you

reference

“Martin -Ha/Toxic -Comment -Model · Hugging Face.” Huggingface.co, 2024, huggingface.co/martin -ha/toxic -comment -model. Accessed 27 Nov. 2024.

SetFit. “Toxic_conversations_50k.” Huggingface.co, 2015, huggingface.co/datasets/SetFit/toxic_conversations_50k. Accessed 27 Nov. 2024.

“Weights & Biases – Developer Tools for ML.” Wandb.ai, wandb.ai.

chatgpt