finetuning-bert-for-classification

July 7, 2025

[1]:
```
!pip install transformers -U
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-
packages (4.52.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-
packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.33.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-
packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-
packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-
packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-
packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.30.0->transformers) (2025.3.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.30.0->transformers) (4.14.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.30.0->transformers) (1.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
```

```
/usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers)
(2025.6.15)
```

[4]:
```
import pandas as pd
```

```
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 4, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.11/dist-packages/kaggle/__init__.py", line 6, in
<module>
    api.authenticate()
  File "/usr/local/lib/python3.11/dist-
packages/kaggle/api/kaggle_api_extended.py", line 434, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in
/root/.config/kaggle. Or use the environment method. See setup instructions at
https://github.com/Kaggle/kaggle-api/
```

[6]:
```
# https://www.kaggle.com/competitions/
  ↪jigsaw-toxic-comment-classification-challenge/data
data = pd.read_csv("/content/train.csv", engine="python")
data.head()
```

[6]:

	id	comment_text	toxic	\
0	0000997932d777bf	Explanation\nWhy the edits made under my usern…	0	
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s…	0	
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It…	0	
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on …	0	
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember…	0	

	severe_toxic	obscene	threat	insult	identity_hate
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

[7]:
```
data['toxic'].value_counts()
```

[7]:
```
toxic
0    144277
1     15294
Name: count, dtype: int64
```

[8]:
```
data = data[['comment_text','toxic']]
data = data[0:1000]
data.head()
```

[8]:

	comment_text	toxic
0	Explanation\nWhy the edits made under my usern…	0
1	D'aww! He matches this background colour I'm s…	0
2	Hey man, I'm really not trying to edit war. It…	0
3	"\nMore\nI can't make any real suggestions on …	0
4	You, sir, are my hero. Any chance you remember…	0

[9]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score,␣
 ↪f1_score
import torch
from transformers import TrainingArguments, Trainer
from transformers import BertTokenizer, BertForSequenceClassification
```

[10]:
```
from transformers import BertTokenizer, BertForSequenceClassification
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.
 ↪from_pretrained('bert-base-uncased',num_labels=2)
```

```
/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 secret in your Google Colab
and restart your session.
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(

tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[11]:
```
model
```

[11]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

[12]:
```python
model = model.to('cuda')
```

[14]:
```python
sample_data = ["I am eating","I am playing "]
tokenizer(sample_data, padding=True, truncation=True, max_length=512)
```

[14]:
```
{'input_ids': [[101, 1045, 2572, 5983, 102], [101, 1045, 2572, 2652, 102]],
'token_type_ids': [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0]], 'attention_mask': [[1, 1,
1, 1, 1], [1, 1, 1, 1, 1]]}
```

[15]:
```python
X = list(data["comment_text"])
y = list(data["toxic"])
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,stratify=y)
X_train_tokenized = tokenizer(X_train, padding=True, truncation=True, max_length=512)
X_val_tokenized = tokenizer(X_val, padding=True, truncation=True, max_length=512)
```

[16]:
```python
X_train_tokenized.keys()
```

[16]:
```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
```

[18]:
```python
print(X_train_tokenized['attention_mask'][0])
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[19]:
```python
len(X_train),len(X_val)
```

[19]:
```
(800, 200)
```

[20]:
```
# Create dataset
class Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.
 ↪items()}
        if self.labels:
            item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.encodings["input_ids"])
```

[21]:
```
train_dataset = Dataset(X_train_tokenized, y_train)
val_dataset = Dataset(X_val_tokenized, y_val)
```

[22]:
```
train_dataset[5]
```

[22]:
```
{'input_ids': tensor([  101,  1000,  5034,  1013, 18856,  1024,  1057, 13007,
8026,  1037,
          1011,  2126,  1012,  2066,  2146,  2051,  2066,  2051,  2041,  2050,
          2568,  2146,  1012, 27571, 17139,  4485,  2021,  9587,  1005,  2474,
          2696,  2232,  2019, 23755,  1012,  2574, 17207,  6289, 21025,  5753,
          2067,  2002,  4430,  7680,  1037,  2139,  2567,  2480,  1005,  1050,
         24761,  2696,  7898,  7367,  2480,  1000,  1000,  7658,  3401,  2757,
          2030,  4142,  1029,  2057,  1005,  2128,  2055,  2000,  4604,  1037,
         19999, 11505,  2000,  4638,  2006,  2010,  4650,  1012,  1000,  1000,
          6289,  7367,  2480,  1000,  1000, 23281,   999,  2419,  2033, 21025,
          2497,  2032,  1037,  7570, 10820,  1012,  1000,  1000,  4487,  2015,
          2033,  7570, 10820,  2378, 22953,  1012,  2139, 19999, 11505,  2480,
          2022, 17183, 23755,  2079,  2087,  1037,  1005,  2139,  3328,  2378,
         19817,  2226,  2813,  2480,  1012,  2025,  2035,  2021,  2087,  1012,
          2139,  2813,  3328,  2378, 12043,  7110,  6069,  2079,  1054, 22953,
          2053,  7386,  1010,  2002, 15333,  2015,  2022,  4638,  2378,  1012,
         25212,  1011,  3841,  2061,  1037,  7132, 10825,  4430, 13970,  2213,
         11865,  7898,  2102,  1012,  1057,  2022,  3191,  2378,  8909,  1012,
          1037,  3389, 10362,  2695,  2378,  1037,  4487,  2015,  1005,  1050,
         23755,  2175,  2091,  2204,  1012,  2079,  2319,  1042, 11631, 18150,
          2000,  3013,  7680,  1062, 13213,  2480, 16638,  4430,  1037, 18819,
          1037,  2695,  2378,  1012,  3637,  2204,  1005,  1050,  2079,  2319,
         10768, 10623,  4183, 11937,  3959,  1012,  7680, 13876,  2378,  2453,
         13970,  2213, 15333,  2015, 11937,  1038, 16425,  2035, 10236,  2102,
         20996,  4609, 15536,  2094,  7160,  4400,  1005,  1050,  1037, 13970,
         24759,  2050, 27178,  3270,  1005,  6839,  2480,  2054,  2057,  4487,
```

```
         2480,  1011, 12731,  4757,  2474,  2696,  2232,  1012, 20209,  1010,
        22953,  1012,  1000,   102,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0]),
 'token_type_ids': tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]),
 'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]),
 'labels': tensor(1)}
```

[23]:
```python
def compute_metrics(p):
    print(type(p))
    pred, labels = p
    pred = np.argmax(pred, axis=1)

    accuracy = accuracy_score(y_true=labels, y_pred=pred)
    recall = recall_score(y_true=labels, y_pred=pred)
    precision = precision_score(y_true=labels, y_pred=pred)
    f1 = f1_score(y_true=labels, y_pred=pred)

    return {"accuracy": accuracy, "precision": precision, "recall": recall,␣
↪"f1": f1}
```

[24]:
```python
# Define Trainer
args = TrainingArguments(
    output_dir="output",
    num_train_epochs=1,
    per_device_train_batch_size=8
```

```
)
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)
```

[25]:
```
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 `TrainingArguments.run_name`
parameter.
```

```
<IPython.core.display.Javascript object>
```

```
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?ref=models
wandb: Paste an API key from your profile and hit enter:
 ..........
wandb: WARNING If you're specifying your api key in code,
ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY
environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
wandb: Currently logged in as: johnprabhasith
(johnprabhasith-cmr-college-of-engineering-technology) to
https://api.wandb.ai. Use `wandb login --relogin` to force
relogin
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[25]:
```
TrainOutput(global_step=100, training_loss=0.2390480613708496,
metrics={'train_runtime': 630.766, 'train_samples_per_second': 1.268,
```

```
'train_steps_per_second': 0.159, 'total_flos': 210488844288000.0, 'train_loss':
0.2390480613708496, 'epoch': 1.0})
```

[26]:
```
trainer.evaluate()
```

```
<IPython.core.display.HTML object>

<class 'transformers.trainer_utils.EvalPrediction'>
```

[26]:
```
{'eval_loss': 0.08775264769792557,
 'eval_accuracy': 0.97,
 'eval_precision': 0.7777777777777778,
 'eval_recall': 1.0,
 'eval_f1': 0.875,
 'eval_runtime': 6.036,
 'eval_samples_per_second': 33.135,
 'eval_steps_per_second': 4.142,
 'epoch': 1.0}
```

[27]:
```
np.set_printoptions(suppress=True)
```

[38]:
```
text = "That was good point"
# text = "go to hell"
inputs = tokenizer(text,padding = True, truncation = True, return_tensors='pt').
 ↪to('cuda')
outputs = model(**inputs)
print(outputs)
predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
print(predictions)
predictions = predictions.cpu().detach().numpy()
if predictions[0][0] > 0.5:
  print("Not Toxic")
else:
  print("Toxic")
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[ 2.0817, -1.9117]],
device='cuda:0', grad_fn=<AddmmBackward0>), hidden_states=None, attentions=None)
tensor([[0.9819, 0.0181]], device='cuda:0', grad_fn=<SoftmaxBackward0>)
Not Toxic
```

[29]:
```
trainer.save_model('CustomModel')
```

[30]:
```
model_2 = BertForSequenceClassification.from_pretrained("CustomModel")
model_2.to('cuda')
```

[30]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
```

```
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

[37]:
```
# text = "That was good point"
inputs = tokenizer(text,padding = True, truncation = True, return_tensors='pt').
  ↪to('cuda')
outputs = model_2(**inputs)
```

```
predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
predictions = predictions.cpu().detach().numpy()
if predictions[0][0] > 0.5:
  print("Not Toxic")
else:
  print("Toxic")
```

```
Not Toxic
```

[ ]: