```
!pip install --quiet wandb
!pip install --quiet transformers
!pip install --quiet datasets
!pip install --quiet emoji
!pip install --quiet kaggle
!pip install --quiet torchinfo
!pip install --quiet imbalanced-learn
!pip install --quiet gdown
!pip install --quiet clean-text
!pip install --quiet accelerate -U
!pip install --quiet transformers[torch]
!pip install --quiet huggingface_hub[tensorflow]
!pip install --quiet huggingface_hub[cli,torch]
!pip install --quiet openai

from huggingface_hub import HfApi
import huggingface_hub

#transformers
import transformers
from transformers import AutoTokenizer
from transformers import GPT2Tokenizer
from transformers import BertTokenizer
from transformers import OpenAIGPTTokenizer
from transformers import AutoConfig
from transformers import GPT2Config
from transformers.models.bert.modeling_bert import BertModel
from transformers.models.gpt2.modeling_gpt2 import GPT2Model
from transformers.models.openai.modeling_openai import OpenAIGPTModel
from transformers.models.bert.modeling_bert import BertPreTrainedModel
from transformers.models.gpt2.modeling_gpt2 import GPT2PreTrainedModel
from transformers.models.openai.modeling_openai import OpenAIGPTPreTrainedModel
from transformers import BertForSequenceClassification
from transformers import GPT2ForSequenceClassification
from transformers import OpenAIGPTForSequenceClassification
from transformers.modeling_outputs import SequenceClassifierOutput
from transformers import Trainer
from transformers import TrainingArguments
from transformers import get_scheduler
from transformers import TextDataset
from transformers.modeling_utils import PreTrainedModel

#torch
import torch
import torch.nn as nn
from torch.nn import BCEWithLogitsLoss, MSELoss, CrossEntropyLoss
from torch.optim import AdamW
from torch.utils import data
from torch.utils.data import Dataset as ds, DataLoader

# dataset
import datasets
from datasets import Dataset
from datasets import Sequence
from datasets import Value
from datasets import Features
from datasets import ClassLabel
from datasets import DatasetDict

# sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

#open ai
import openai

# others
import os
import wandb
import re, string
import emoji
import pandas as pd
import seaborn as sns
```

```
import matplotlib.pyplot as plt
import numpy as np
import pickle
import joblib
import traceback
from tqdm import tqdm
from collections import defaultdict

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
api = HfApi()


! huggingface-cli login --token <TOKEN>
```

```
Token will not been saved to git credential helper. Pass `add_to_git_credential=True` if you want to set the git credential as well.
Token is valid (permission: write).
Your token has been saved to /root/.cache/huggingface/token
Login successful
```

```
huggingface_hub.login()
```

Copy a token from your Hugging Face tokens page and paste it below.

Immediately click login after copying your token or it might be stored in plain text in this notebook file.

```
class BertForClassification(BertForSequenceClassification):

    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels
        self.config = config

        # Load model body > return all og the HS
        self.bert = BertModel(config)
        # Set up token classification head
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)

        # Initialize weights and apply final processing
        self.post_init()


    def forward(self, input_ids=None, attention_mask=None, token_type_ids=None,
                labels=None, **kwargs):
        # Use model body to get encoder representations
        outputs = self.bert(input_ids, attention_mask=attention_mask,
                            token_type_ids=token_type_ids, **kwargs)

        # Apply classifier to encoder representation > [cls]
        sequence_output = self.dropout(outputs[1])
        logits = self.classifier(sequence_output)

        # Calculate losses
        loss = None
        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()
            loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
            #outputs = (loss,) + outputs # can comment

        # return outputs  # (loss), logits, (hidden_states), (attentions)


        # Return model output object
```

```
        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


model_names = ["bert-base-uncased", "openai-gpt"]

id2label = {0: 'non-toxic', 1: 'mild', 2: 'toxic'}
label2id = { v:k for (k,v) in id2label.items()}
labels = ['non-toxic','mild', 'toxic']

tokenizers = {"bert-base-uncased": BertTokenizer.from_pretrained("bert-base-uncased")}
configs = {"bert-base-uncased": AutoConfig.from_pretrained("bert-base-uncased", num_labels=3, id2label=id2label, label2id=label2id)}
models = {"bert-base-uncased": BertForClassification.from_pretrained("bert-base-uncased", config=configs["bert-base-uncased"])}
```

```
Downloading                                232k/232k
(…)solve/main/vocab.txt:                   [00:00<00:00,
100%                                       5.39MB/s]
Downloading                                28.0/28.0
```

```
def clean_text(text):
  text = re.sub('#', '', text) # Removing '#' hashtag
  text = re.sub('\w+:\/\/\S+', '', text) # Removing hyperlink
  text = re.sub('[^a-zA-Z]', ' ', text) # Remove punctuation
  return text.lower()

def compute_metrics(pred):
  labels = pred.label_ids
  preds = pred.predictions.argmax(-1)
  f1 = f1_score(labels, preds, average="weighted")
  acc = accuracy_score(labels, preds)
  return {"accuracy": acc, "f1": f1}

def create_dataset(df, text, label):
  class_names = ['non-toxic','mild', 'toxic']
  data_dict = {'text':df[text], 'labels':df[label]}
  tags = ClassLabel(num_classes=3 , names=class_names)
  feature_set = Features({'text':Value(dtype='string'), 'labels':tags})

  return Dataset.from_dict(mapping = data_dict, features = feature_set)


def get_dataset():
  train = pd.read_csv('/content/train.csv')
  validation = pd.read_csv('/content/validation.csv')

  dataset_train = create_dataset(train,"message","target")
  dataset_val = create_dataset(validation,"message","target")

  train_convert = train.copy()
  train_convert['target'] = train_convert['target'].astype(str)
  train_convert['message'] = (train_convert['message'] + ' ->')
  train_convert['target'] = (' ' + train_convert['target'])
  train_convert.rename(columns={'message':'prompt', 'target':'completion'}, inplace=True)
  train_convert.to_json("train_pandas.jsonl", orient='records', lines=True)

  validation_convert = validation.copy()
  validation_convert['target'] = validation_convert['target'].astype(str)
  validation_convert['message'] = (validation_convert['message'] + ' ->')
  validation_convert['target'] = (' ' + validation_convert['target'])
  validation_convert.rename(columns={'message':'prompt', 'target':'completion'}, inplace=True)
  validation_convert.to_json("validation_pandas.jsonl", orient='records', lines=True)

  dataset = DatasetDict()
  dataset["train"] = dataset_train
  dataset["validation"] = dataset_val
  return dataset
```

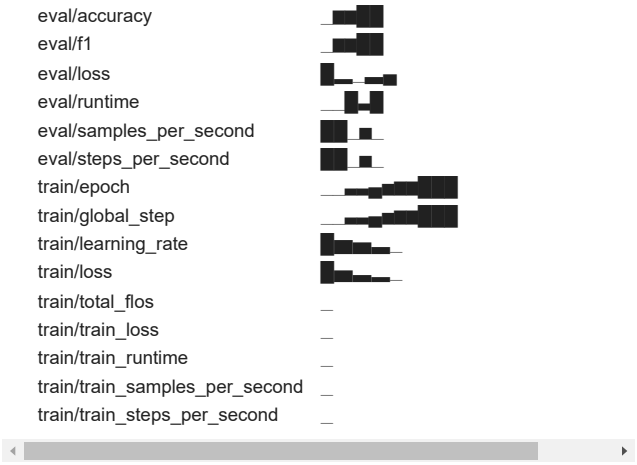
```
! wandb login <TOKEN>
```

```
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

```
wandb.login()
wandb.init(project="bert-toxic-chat-detection", entity="dani-squad")
```

```
wandb: WARNING Calling wandb.login() after wandb.init() has n
Finishing last run (ID:rb8xpvt3) before initializing another...
Waiting for W&B process to finish... (success).
0.010 MB of 0.027 MB uploaded (0.000 MB deduped)
```

Run history:




```
def finetune_bert(model_name, dataset, max_len=64, train_batch_size=16, num_epochs=5):

  tokenizer = tokenizers[model_name]
  model = models[model_name]

  tokenize = lambda batch: tokenizer(batch["text"], padding=True, truncation=True, max_length=max_len)

  encoded = dataset.map(tokenize, batched=True, batch_size=32)

  model.to(device)
  optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
  logging_steps = len(encoded['train']) // train_batch_size
  num_training_steps = num_epochs * logging_steps

  scheduler = get_scheduler(
      name="linear",
      optimizer=optimizer,
      num_warmup_steps=0,
      num_training_steps=num_training_steps)

  training_args = TrainingArguments(
      output_dir=f"{model_name}-toxic-detector",
      num_train_epochs=num_epochs,
      per_device_train_batch_size=train_batch_size,
      per_device_eval_batch_size=train_batch_size,
      weight_decay=0.01,
      evaluation_strategy="epoch",
      save_steps=1e6,
      disable_tqdm=False,
      logging_steps=logging_steps,
      push_to_hub=False,
      log_level="error"
```

```
        log_level= error ,
        report_to="wandb",
        run_name=model_name)

    optimizers = (optimizer, scheduler)

    model_trainer = Trainer(
        model=model,
        args=training_args,
        compute_metrics=compute_metrics,
        train_dataset=encoded["train"],
        eval_dataset=encoded["validation"],
        tokenizer=tokenizer,
        optimizers=optimizers)

    model_trainer.train()
    model.eval()

    return model_name, model_trainer, encoded


def save_model(model_name, model_trainer):
    model_trainer.save_model(f'/content/{model_name}-saved')
    api.upload_folder(
        folder_path=f"/content/{model_name}-saved",
        repo_id=f"dffesalbon/{model_name}-dota-toxic",
        repo_type="model")


dataset = get_dataset()
```

BERT (Base-uncased)

```
model_name, model_trainer, encoded = finetune_bert(model_names[0], dataset)
```

```
Map: 100%                                                 1435/1435 [00:00<00:00, 2544.48 examples/s]
Map: 100%                                                 479/479 [00:00<00:00, 2292.94 examples/s]
[450/450 01:03, Epoch 5/5]
```

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.211800	0.348512	0.891441	0.893533
2	0.120900	0.382619	0.901879	0.903323
3	0.081300	0.428387	0.903967	0.904764
4	0.033100	0.445785	0.903967	0.904166
5	0.027100	0.464333	0.897704	0.898949

```
try:
  save_model(model_name, model_trainer)
except Exception as ex:
  print(ex)
```

```
Upload 2 LFS files:                                       2/2 [00:18<00:00,
100%                                                      18.84s/it]
pytorch_model.bin:                                        438M/438M
```

GPT

```
openai.api_key = "<TOKEN>"
!export OPENAI_API_KEY="<TOKEN>"
!pip install --upgrade openai
```

```
    Requirement already satisfied: openai in /usr/local/lib/python3.10/dist-packages (0.28.0)
    Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.10/dist-packages (from openai) (2.31.0)
    Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from openai) (4.66.1)
    Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from openai) (3.8.5)
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (3.2.0
```

```
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (2023.7.22)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (23.1.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (6.0.4)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (4.0.3)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (1.3.1)
```

```
training_file_name = '/content/train_pandas.jsonl'
validation_file_name = '/content/validation_pandas.jsonl'

train_res = openai.File.create(file=open(training_file_name, "rb"), purpose="fine-tune")
train_file_id = train_res["id"]

validation_res = openai.File.create(file=open(validation_file_name, "rb"), purpose="fine-tune")
validation_file_id = validation_res["id"]

print("Training file id:", train_file_id)
print("Validation file id:", validation_file_id)
```

```
Training file id: file-hG3ZYg8CCaz9DnhTfpJDxSng
Validation file id: file-JFTTnUS84ihVBR2cTCsuTvlZ
```

```
suffix_name = "gpt-dota-toxic"

res = openai.FineTuningJob.create(
    training_file=train_file_id,
    validation_file=validation_file_id,
    model="davinci-002",
    suffix=suffix_name,
)

job_id = res["id"]

print(res)
```


```
{
  "object": "fine_tuning.job",
  "id": "ftjob-AGkXY8Lge85cZcOdnR1mix3n",
  "model": "davinci-002",
  "created_at": 1693659681,
  "finished_at": null,
  "fine_tuned_model": null,
  "organization_id": "org-TQ05ulctnaTm4vtkVxU9ZIrM",
  "result_files": [],
  "status": "created",
  "validation_file": "file-JFTTnUS84ihVBR2cTCsuTvlZ",
  "training_file": "file-hG3ZYg8CCaz9DnhTfpJDxSng",
  "hyperparameters": {
    "n_epochs": 3
  },
  "trained_tokens": null
}
```


```
print(job_id)
```

```
ftjob-AGkXY8Lge85cZcOdnR1mix3n
```

```
job_res = openai.FineTuningJob.retrieve(job_id)
gpt_model = job_res['model']
print(job_res)
```


```
{
  "object": "fine_tuning.job",
  "id": "ftjob-AGkXY8Lge85cZcOdnR1mix3n",
  "model": "davinci-002",
  "created_at": 1693659681,
  "finished_at": 1693660291,
  "fine_tuned_model": "ft:davinci-002:personal:gpt-dota-toxic:7uKhQqc5",
  "organization_id": "org-TQ05ulctnaTm4vtkVxU9ZIrM",
  "result_files": [
    "file-C3BNXEdIwEEfGv7xEi07WqnI"
  ],
  "status": "succeeded",
  "validation_file": "file-JFTTnUS84ihVBR2cTCsuTvlZ",
```

```
          "training_file": "file-hG3ZYg8CCaz9DnhTfpJDxSng",
          "hyperparameters": {
            "n_epochs": 3
          },
          "trained_tokens": 34224
        }
```

```
event_res = openai.FineTuningJob.list_events(id=job_id, limit=5)

events = event_res["data"]
events.reverse()

for event in events:
    print(event["message"])
```

```
     Step 1900/2153: training loss=0.00
     Step 2000/2153: training loss=0.00
     Step 2100/2153: training loss=0.00
     New fine-tuned model created: ft:davinci-002:personal:gpt-dota-toxic:7uKhQqc5
     Fine-tuning job successfully completed
```

```
job_res = openai.FineTuningJob.retrieve(job_id)
fine_tuned_model_id = job_res["fine_tuned_model"]
print("\nFine-tuned model id:", fine_tuned_model_id)
```

```
     Fine-tuned model id: ft:davinci-002:personal:gpt-dota-toxic:7uKhQqc5
```

```
file_id = job_res['result_files'][0]
content = openai.File.download(file_id)
contents = content.decode()
with open('results.csv', 'w') as f:
    f.write(contents)
```

```
print(job_res)
```

```
     {
       "object": "fine_tuning.job",
       "id": "ftjob-AGkXY8Lge85cZcOdnR1mix3n",
       "model": "davinci-002",
       "created_at": 1693659681,
       "finished_at": 1693660291,
       "fine_tuned_model": "ft:davinci-002:personal:gpt-dota-toxic:7uKhQqc5",
       "organization_id": "org-TQ05ulctnaTm4vtkVxU9ZIrM",
       "result_files": [
         "file-C3BNXEdIwEEfGv7xEi07WqnI"
       ],
       "status": "succeeded",
       "validation_file": "file-JFTTnUS84ihVBR2cTCsuTvlZ",
       "training_file": "file-hG3ZYg8CCaz9DnhTfpJDxSng",
       "hyperparameters": {
         "n_epochs": 3
       },
       "trained_tokens": 34224
     }
```

✓ 22s completed at 9:15 PM