```
!pip install transformers datasets pandas
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packages (3.0.
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.1.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: tokenizers<0.20,>=0.19 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from d
Requirement already satisfied: multiprocess in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: fsspec<=2024.6.1,>=2023.1.0 in /usr/local/lib/python3.10/
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/d
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pack
```

```
from transformers import AutoModelForCausalLM, AutoTokenizer, Trainer, TrainingArguments
import pandas as pd
from datasets import Dataset
import os

os.environ['HF_TOKEN'] = 'your_huggingface_token_here'

model_name = "gpt2"
model = AutoModelForCausalLM.from_pretrained(model_name, token=os.environ['HF_TOKEN'])
tokenizer = AutoTokenizer.from_pretrained(model_name, token=os.environ['HF_TOKEN'])
```

```
tokenizer.pad_token = tokenizer.eos_token

df = pd.read_csv('/content/impression_300_llm.csv')
dataset = Dataset.from_pandas(df)

def tokenize_function(examples):
    inputs = tokenizer(examples['Report Name'], examples['History'], examples['Observation']
    inputs['labels'] = inputs['input_ids'].copy()
    return inputs

tokenized_dataset = dataset.map(tokenize_function, batched=True)

training_args = TrainingArguments(
    output_dir='./results',
    per_device_train_batch_size=8,
    num_train_epochs=3
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset.train_test_split(test_size=30)['train'],
    eval_dataset=tokenized_dataset.train_test_split(test_size=30)['test']
)

trainer.train()

model.save_pretrained("./fine-tuned-model")
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: Fu
  warnings.warn(
```

Map: 100% 330/330 [00:00<00:00, 515.96 examples/s]

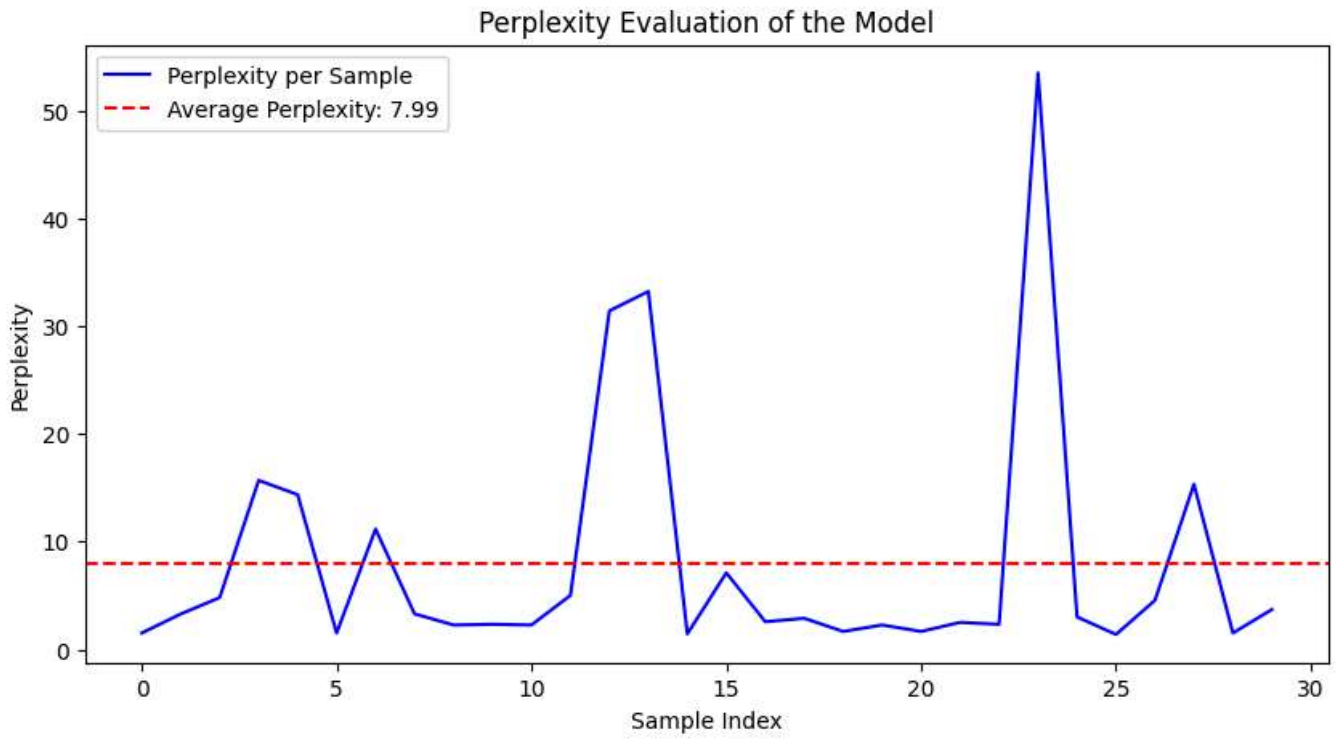
[114/114 36:32, Epoch 3/3]

Step	Training Loss

```
from transformers import AutoModelForCausalLM, AutoTokenizer, Trainer, TrainingArguments
import pandas as pd
from datasets import Dataset
import os
import numpy as np
import matplotlib.pyplot as plt
import torch
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk
```

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

model = AutoModelForCausalLM.from_pretrained("./fine-tuned-model")
tokenizer = AutoTokenizer.from_pretrained(model_name)

def calculate_perplexity(text):
    tokenizer.pad_token = tokenizer.eos_token
    inputs = tokenizer(text, return_tensors='pt', padding='longest', truncation=True)
    with torch.no_grad():
        outputs = model(**inputs, labels=inputs['input_ids'])
        log_likelihood = outputs.loss.item()
    perplexity = np.exp(log_likelihood)
    return perplexity


perplexities = []
eval_texts = tokenized_dataset.train_test_split(test_size=30)['test']['Report Name']

for text in eval_texts:
    perplexity = calculate_perplexity(text)
    perplexities.append(perplexity)

average_perplexity = np.mean(perplexities)

plt.figure(figsize=(10, 5))
plt.plot(perplexities, label='Perplexity per Sample', color='blue')
plt.axhline(y=average_perplexity, color='red', linestyle='--', label=f'Average Perplexity: {
plt.title('Perplexity Evaluation of the Model')
plt.xlabel('Sample Index')
plt.ylabel('Perplexity')
plt.legend()
plt.show()

text_analysis_pipeline = pipeline("text-classification", model=model, tokenizer=tokenizer)

sample_texts = eval_texts[:5]
# for text in sample_texts:
#     try:
#         analysis = text_analysis_pipeline(text)
#         print(f'Text: {text}\nAnalysis: {analysis}\n')
#     except Exception as e:
        # print(f'Error analyzing text: {text}\nError: {str(e)}\n')
```

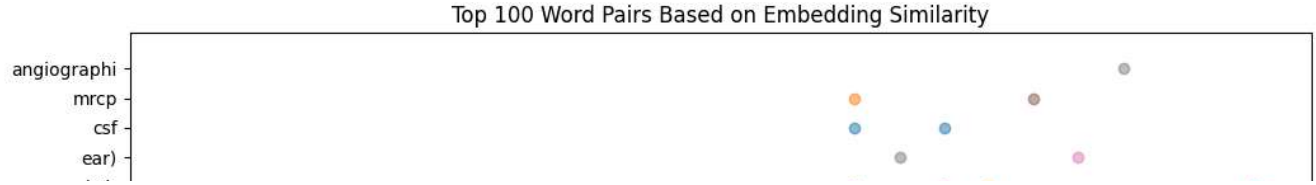
```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```



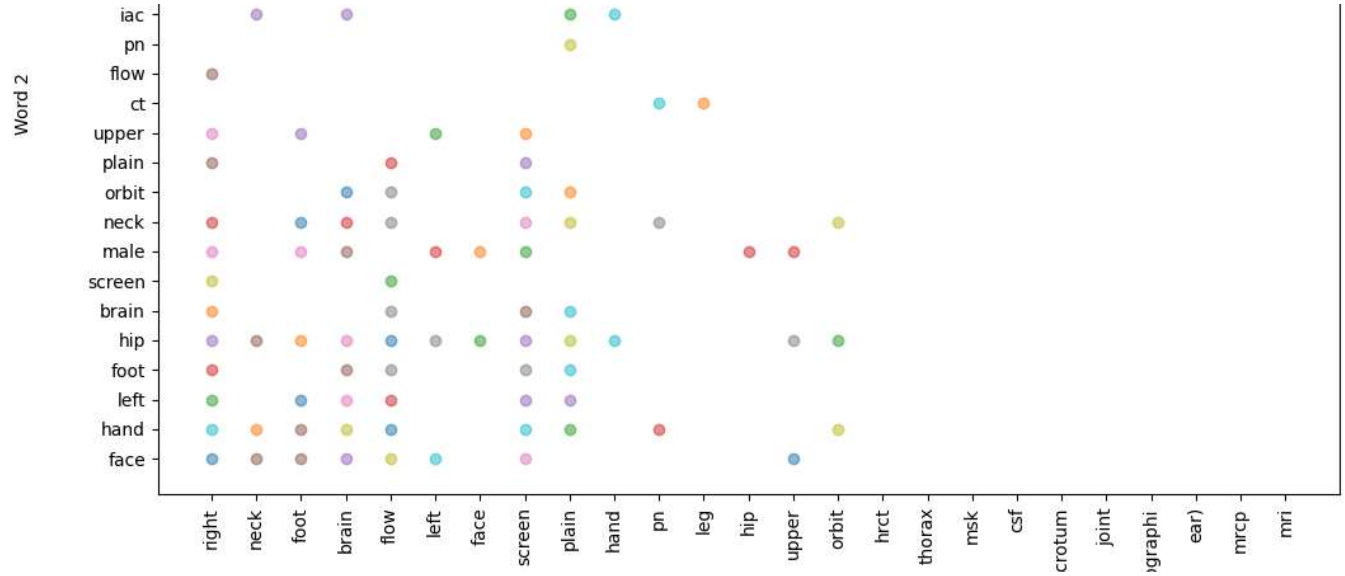

```
The model 'GPT2LMHeadModel' is not supported for text-classification. Supported models a
```

```
def preprocess_text(text):
    if not isinstance(text, str) or not text.strip():
        return []
    stop_words = set(stopwords.words('english'))
    stemmer = PorterStemmer()
    lemmatizer = WordNetLemmatizer()
    words = text.split()
    words = [lemmatizer.lemmatize(stemmer.stem(word.lower())) for word in words if word.lower
    return words

all_words = []
for text in df['Report Name']:
    processed = preprocess_text(text)
    all_words.extend(processed)

if all_words:
    unique_words = list(set(all_words))
else:
    unique_words = []

if unique_words:
    word_embeddings = []
    for word in unique_words:
```

```
            inputs = tokenizer(word, return_tensors='pt')
            with torch.no_grad():
                outputs = model(**inputs)
                # Get the hidden states or the logits from the last layer
                embeddings = outputs.logits.mean(dim=1).squeeze().numpy()  # Squeeze to flatten
            word_embeddings.append((word, embeddings))

        word_embeddings = np.array([embed for _, embed in word_embeddings])
        similarity_matrix = cosine_similarity(word_embeddings)
        word_pairs = []

        for i in range(len(unique_words)):
            for j in range(i + 1, len(unique_words)):
                word_pairs.append((unique_words[i], unique_words[j], similarity_matrix[i][j]))

        word_pairs.sort(key=lambda x: x[2], reverse=True)
        top_100_pairs = word_pairs[:100]

        plt.figure(figsize=(12, 8))
        for pair in top_100_pairs:
            plt.scatter(pair[0], pair[1], alpha=0.5)
        plt.title('Top 100 Word Pairs Based on Embedding Similarity')
        plt.xlabel('Word 1')
        plt.ylabel('Word 2')
        plt.xticks(rotation=90)
        plt.show()
    else:
        print("No unique words found.")
```


Top 100 Word Pairs Based on Embedding Similarity
angiographi
mrcp
csf
ear)

New Section


Word 2
iac
pn
flow
ct
upper
plain
orbit
neck
male
screen
brain
hip
foot
left
hand
face
right
neck
foot
brain
flow
left
face
screen
plain
hand
pn
leg
hip
upper
orbit
hrct
thorax
msk
csf
crotum
joint
ear)
mrcp
mri