BUSINESS INTELLIGENCE AND DATA ANALYTICS

Institute of Analytics

Open in Colab

Context Aware Word Embeddings - BERT

```
%pip install transformers torch
```

```
Requirement already satisfied: transformers in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (4.51.3)
Requirement already satisfied: tqdm>=4.27 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from transformers) (
Requirement already satisfied: safetensors>=0.4.3 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from transfo
Requirement already satisfied: packaging>=20.0 in c:\users\me\appdata\roaming\python\python310\site-packages (from transformers) (24.2)
Requirement already satisfied: numpy>=1.17 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from transformers)
Requirement already satisfied: tokenizers<0.22,>=0.21 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from tra
Requirement already satisfied: filelock in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from transformers) (3.
Requirement already satisfied: regex!=2019.12.17 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from transfor
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (fr
Requirement already satisfied: requests in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from transformers) (2.
Requirement already satisfied: pyyaml>=5.1 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from transformers)
Requirement already satisfied: fsspec>=2023.5.0 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from huggingfa
Requirement already satisfied: typing-extensions>=3.7.4.3 in c:\users\me\appdata\roaming\python\python310\site-packages (from huggingfac
Requirement already satisfied: colorama in c:\users\me\appdata\roaming\python\python310\site-packages (from tqdm>=4.27->transformers) (0
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from r
Requirement already satisfied: idna<4,>=2.5 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from requests->tra
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from request
Requirement already satisfied: certifi>=2017.4.17 in c:\users\me\appdata\local\programs\python\python310\lib\site-packages (from request
Note: you may need to restart the kernel to use updated packages.
WARNING: You are using pip version 21.2.3; however, version 25.0.1 is available.
You should consider upgrading via the 'c:\Users\me\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip' comm
```

Display BERT Embeddings

```python
from transformers import BertTokenizer, BertModel
import torch

# Load pretrained BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Sentence
sentence = "He went to the bank to deposit money."

# Tokenize
inputs = tokenizer(sentence, return_tensors='pt')
tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])

# Get outputs
with torch.no_grad():
    outputs = model(**inputs)

# Get hidden states (embeddings)
embeddings = outputs.last_hidden_state.squeeze(0)  # shape: (seq_len, hidden_size)
# Print tokens and their embeddings
for token, embedding in zip(tokens, embeddings):
    print(f"Token: {token}\n Embedding(First 10 Numbers):\n{embedding[:10]}\nShape: {embedding.shape}\n")


# # Find index of "bank"
# try:
#     idx = tokens.index("bank")
#     bank_embedding = embeddings[idx]
#     print(f"Embedding for 'bank':\n{bank_embedding}\n\nShape: {bank_embedding.shape}")
# except ValueError:
#     print("'bank' not found in tokenized input:", tokens)

```

```
Embedding for 'bank':
tensor([ 4.7019e-01, -1.9835e-01, -1.0122e-01, -1.3519e-01,  1.2612e+00,
        -9.6139e-03, -4.9014e-02,  1.0147e+00, -4.5361e-02,  1.7432e-01,
         1.2800e-01, -3.2356e-01, -1.3227e-01,  3.6582e-02, -7.8302e-01,
        -6.2770e-01,  5.2776e-01,  3.5693e-01,  1.3597e+00,  2.3784e-01,
        -3.0995e-01,  4.3136e-02,  3.2358e-01,  3.2144e-01,  3.3207e-01,
         4.5470e-01,  6.8660e-01,  5.2037e-01, -2.8076e-01, -5.2107e-01,
         5.3412e-01,  9.5313e-01,  3.6960e-01,  4.9074e-01,  1.0348e-01,
        -1.2543e-01,  1.8115e-01,  3.9604e-02, -1.1310e+00,  2.2161e-02,
        -4.4877e-01, -8.1382e-01, -6.2421e-01,  3.5284e-01, -2.4929e-01,
        -6.1539e-01,  1.9276e-01,  2.8171e-01, -7.0082e-01, -8.2422e-01,
        -3.0416e-01,  1.0278e+00,  4.3732e-01, -5.0054e-01,  1.1097e-01,
         4.7545e-01, -1.0476e+00, -4.6538e-01, -5.3300e-01, -2.1977e-01,
         7.0954e-01,  3.1443e-01,  5.0420e-01, -7.7659e-01,  2.3119e-01,
        -1.6568e-01,  4.8205e-01, -5.2181e-03, -1.1804e-01, -9.9157e-02,
        -2.4582e-01,  1.9177e+00, -8.5401e-01, -9.5023e-01, -4.6143e-01,
         5.0577e-01, -2.2459e-01,  2.9306e-01, -2.2287e-01, -9.2279e-03,
        -5.4351e-01, -7.2676e-01, -1.5370e-01,  3.8075e-01,  3.8973e-01,
        -6.8711e-01,  1.1629e-01,  1.9948e-01, -4.7043e-01,  1.2130e+00,
        -1.4237e-01,  2.6370e-01, -9.1200e-01, -2.1185e-01, -1.3493e+00,
        -2.0093e-02, -8.5553e-02,  7.4500e-02, -2.7861e-01,  5.7108e-01,
         6.0045e-01, -1.1580e+00,  2.2457e-01,  5.2492e-02,  7.7377e-02,
         2.1038e-01, -1.0968e+00, -1.8806e-01,  1.6193e-04,  3.0690e-01,
         1.5269e-01, -1.2186e+00,  4.4192e-01, -8.6209e-01, -3.1611e-01,
        -5.5617e-01,  2.4480e-02, -6.4080e-01, -8.4688e-01, -3.2271e-01,
         5.6128e-01,  6.9319e-01,  3.6391e-01,  1.4688e+00,  3.0880e-01,
         3.2401e-03,  5.1110e-02,  9.8396e-01, -5.8512e-01, -1.1058e-01,
         5.6782e-01,  5.1704e-01,  4.8607e-02,  3.8416e-02,  1.0723e-01,
        -5.8880e-01, -3.2970e-03, -4.8399e-01, -1.0776e+00,  3.2018e-01,
         8.1521e-01,  8.5122e-01,  5.0063e-01,  1.1447e+00, -1.8171e-01,
         4.7930e-01,  6.8499e-02, -9.1607e-01, -2.6628e-01, -1.0907e-01,
         1.1996e-01, -3.1764e-01, -3.9093e-01, -5.3363e-01,  2.1021e-01,
        -1.9328e-01, -4.6874e-01,  1.2553e-01, -5.8689e-01,  1.6500e-02,
         2.6675e-01,  4.8762e-03, -4.1349e-01, -1.0107e-01, -5.9281e-01,
        -3.6083e-01,  9.8077e-02,  4.3204e-01, -3.1348e-01,  2.6281e-01,
        -4.4020e-01,  9.3242e-02,  1.1335e+00, -3.1242e-01,  9.5865e-01,
         4.3642e-01, -9.1704e-01, -5.5271e-01,  5.3453e-01,  8.2077e-01,
        -3.4234e-02,  9.8388e-01,  5.4295e-01,  6.9819e-01,  1.0401e+00,
         5.3207e-01,  1.7786e-01,  1.1376e-01,  1.5848e-01, -2.3289e-01,
        -4.0887e-01, -4.7423e-01, -3.7506e-01,  3.0031e-02, -3.0034e-01,
        -3.0897e-01, -3.0131e-01, -2.1677e-01, -8.5815e-01, -3.3625e-01,
         1.7334e-01, -1.0664e+00,  2.9628e-01,  1.5440e-01, -8.2941e-02,
         1.9345e-01,  4.2606e-01,  3.3704e-01,  1.4883e+00, -1.4554e-01,
        -7.5085e-01,  5.5163e-01, -9.2985e-01,  1.2159e-01, -2.5228e-01,
         4.6588e-01, -1.2934e-02, -8.3803e-01,  3.8504e-02,  4.2223e-01,
         1.9769e-01, -4.9039e-02,  7.5554e-02,  7.1144e-01,  1.0876e-01,
         6.0351e-01,  3.1833e-01, -9.8136e-02,  8.2948e-02, -1.1635e-01,
         1.1131e-01,  1.5795e-01,  1.0441e+00, -7.2768e-02, -2.5971e-01,
        -4.1198e-01, -1.6680e+00, -7.8462e-01, -4.9429e-01, -2.1480e-02,
        -4.3348e-01, -3.0725e-01, -2.0169e-01, -3.1562e-01,  5.5264e-01,
         8.7739e-01, -2.5813e-01, -4.6994e-01,  2.2665e-01, -6.8825e-01,
        -1.0124e-01, -8.5311e-01, -4.8196e-01, -8.7491e-01,  8.3805e-01,
        -2.4941e-01, -4.3674e-02,  1.3940e-02, -7.0811e-01,  1.0864e+00,
         9.5805e-01, -1.4053e-01,  1.4219e-01,  5.0391e-01, -5.8840e-01,
        -6.1770e-01, -4.2367e-02,  2.9746e-01,  3.9472e-01,  6.5948e-01,
         6.4870e-01,  2.8621e-01, -5.4719e-01,  1.5666e+00,  7.9899e-01,
        -4.2446e-01,  1.0406e-01,  4.4579e-01,  3.1292e-01,  3.9760e-01,
```

Use BERT to Create Context-Aware Word Embeddings

```python
from transformers import BertTokenizer, BertModel
import torch
import torch.nn.functional as F

# Load pretrained BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Function to extract contextual embedding for a word (handles subwords)
def get_token_embedding(sentence, target_word):
    # Tokenize the sentence and get embeddings
    inputs = tokenizer(sentence, return_tensors='pt')
    outputs = model(**inputs)

    # Get tokens and embeddings
    tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])
    embeddings = outputs.last_hidden_state.squeeze(0)

    # Tokenize the target word the same way BERT does
    target_tokens = tokenizer.tokenize(target_word)

    # Search for the position of the target word (handling subwords)
    matches = []
    for i in range(len(tokens) - len(target_tokens) + 1):
        if tokens[i:i + len(target_tokens)] == target_tokens:
            matches = list(range(i, i + len(target_tokens)))
            break

    if not matches:
        raise ValueError(f"'{target_word}' not found in tokens: {tokens}")

    # Average the embeddings over all subword tokens
    return embeddings[matches].mean(dim=0)

# Contextual sentences
sentence_fruit = "He ate a fresh apple and enjoyed the fruit."
sentence_company = "Apple released a new product in the computer market."
sentence_orange = "An orange is a juicy fruit."
sentence_microsoft = "Microsoft computer was running the latest software."

# Get embeddings
apple_fruit = get_token_embedding(sentence_fruit, "apple")
apple_company = get_token_embedding(sentence_company, "apple")
orange = get_token_embedding(sentence_orange, "orange")
microsoft = get_token_embedding(sentence_microsoft, "Microsoft")

# Cosine similarity comparisons
sim_fruit = F.cosine_similarity(apple_fruit, orange, dim=0)
sim_company = F.cosine_similarity(apple_company, microsoft, dim=0)

# Results
print(f"Similarity between 'apple' (fruit) and 'orange': {sim_fruit.item():.4f}")
print(f"Similarity between 'apple' (company) and 'Microsoft': {sim_company.item():.4f}")
```

```
Similarity between 'apple' (fruit) and 'orange': 0.5839
Similarity between 'apple' (company) and 'Microsoft': 0.8549
```

Pipelines

Basic Pipeline Usage

1. Text Classification (Sentiment Analysis)

```
from transformers import pipeline

# Create a sentiment analysis pipeline
classifier = pipeline("sentiment-analysis")

# Analyze single text
result = classifier("I love using Hugging Face!")
print(result)
# Output: [{'label': 'POSITIVE', 'score': 0.9998}]

# Analyze multiple texts
texts = [
    "I hate this product",
    "This is amazing!",
    "It's okay, nothing special"
]
results = classifier(texts)
for text, result in zip(texts, results):
    print(f"Text: {text}")
    print(f"Sentiment: {result['label']}, Score: {result['score']:.4f}\n")
```

2. Named Entity Recognition (NER)

```
# NER pipeline
ner = pipeline("ner", aggregation_strategy="simple")

text = "My name is John and I live in New York. I work at Google."
entities = ner(text)

for entity in entities:
    print(f"Entity: {entity['word']}")
    print(f"Label: {entity['entity_group']}")
    print(f"Score: {entity['score']:.4f}")
    print(f"Start: {entity['start']}, End: {entity['end']}\n")
```

3. Question Answering

```
# Question answering pipeline
qa = pipeline("question-answering")

context = """
Hugging Face is a company that develops tools for building applications using machine learning.
They are especially known for their work in natural language processing. The company was founded in 2016
and is headquartered in New York.
"""

questions = [
    "When was Hugging Face founded?",
    "Where is Hugging Face headquartered?",
    "What is Hugging Face known for?"
]

for question in questions:
    result = qa(question=question, context=context)
    print(f"Question: {question}")
    print(f"Answer: {result['answer']}")
    print(f"Score: {result['score']:.4f}\n")
```

4. Text Generation

```
# Text generation pipeline
generator = pipeline("text-generation", model="gpt2")

# Generate text with custom parameters
prompts = [
    "The future of artificial intelligence is",
    "In a world where robots exist,"
]

for prompt in prompts:
    generated = generator(
        prompt,
        max_length=50,
        num_return_sequences=2,
        temperature=0.7,
        do_sample=True,
        pad_token_id=generator.tokenizer.eos_token_id
    )

    print(f"Prompt: {prompt}")
    for i, gen in enumerate(generated):
        print(f"Generation {i+1}: {gen['generated_text']}\n")
```

5. Text Summarization

```
# Summarization pipeline
summarizer = pipeline("summarization")

article = """
Machine learning is a subset of artificial intelligence that enables computers to learn and improve
from experience without being explicitly programmed. It focuses on the development of computer programs
that can access data and use it to learn for themselves. The process of learning begins with observations
or data, such as examples, direct experience, or instruction, in order to look for patterns in data and
make better decisions in the future based on the examples that we provide. The primary aim is to allow
the computers to learn automatically without human intervention or assistance and adjust actions accordingly.
"""

summary = summarizer(article, max_length=50, min_length=25, do_sample=False)
print("Original length:", len(article.split()))
print("Summary:", summary[0]['summary_text'])
print("Summary length:", len(summary[0]['summary_text'].split()))
```

6. Translation

```
# Translation pipeline
translator = pipeline("translation", model="Helsinki-NLP/opus-mt-en-fr")

texts = [
    "Hello, how are you today?",
    "Machine learning is fascinating.",
    "I would like to order a coffee."
]

for text in texts:
    translated = translator(text)
    print(f"English: {text}")
    print(f"French: {translated[0]['translation_text']}\n")
```

Use specific model e.g. BERT to Create Questions Answering Pipeline

```
# Import required libraries
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
from transformers import pipeline
import torch

# Using pipeline (High-level API)
qa_pipeline = pipeline( "question-answering",
model="bert-large-uncased-whole-word-masking-finetuned-squad",
tokenizer="bert-large-uncased-whole-word-masking-finetuned-squad" )

# Example usage
context = """ BERT is a method of pre-training language representations,
meaning that it trains a general-purpose language understanding
model on a large text corpus (like Wikipedia),
and then uses that model for downstream NLP tasks like question answering. """

question = "What is BERT?"
result = qa_pipeline(question=question, context=context)
print(f"Answer: {result['answer']}")
print(f"Confidence: {result['score']:.4f}")

```

```
c:\Users\me\myenv310\lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See http
  from .autonotebook import tqdm as notebook_tqdm
c:\Users\me\myenv310\lib\site-packages\huggingface_hub\file_download.py:144: UserWarning: `huggingface_hub` cache-system uses symlinks b
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to activate de
  warnings.warn(message)
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfo
Some weights of the model checkpoint at bert-large-uncased-whole-word-masking-finetuned-squad were not used when initializing BertForQue
- This IS expected if you are initializing BertForQuestionAnswering from the checkpoint of a model trained on another task or with anoth
- This IS NOT expected if you are initializing BertForQuestionAnswering from the checkpoint of a model that you expect to be exactly ide
Device set to use cpu
Answer: a method of pre-training language representations
Confidence: 0.6874
```

Fine Tuning Large Language Models

Tutorial: Fine-tuning a Language Model and Deploying with Hugging Face Spaces - Sentiment Analysis IMDB Reviews

Step 1: Install Required Libraries

```
! pip install transformers datasets huggingface_hub gradio
```

Step 2: Load and Prepare the Dataset

We will use a small portion of the IMDb dataset for binary sentiment classification.

```
from datasets import load_dataset

# Load a small subset for quicker training
dataset = load_dataset("imdb", split="train[:2000]")
dataset = dataset.train_test_split(test_size=0.2)
```

Step 3: Load the Tokenizer and Model

We use `distilbert-base-uncased`, a lightweight version of BERT.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

Step 4: Tokenize the Dataset

Tokenization prepares the text for input to the model.

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

tokenized_dataset = dataset.map(tokenize_function, batched=True)
tokenized_dataset.set_format("torch", columns=["input_ids", "attention_mask", "label"])
```

Step 5: Define Training Arguments and Trainer

```
from transformers import TrainingArguments, Trainer
import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    save_total_limit=1,
    logging_dir="./logs",
)

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, predictions, average='binary')
    acc = accuracy_score(labels, predictions)
    return {"accuracy": acc, "f1": f1, "precision": precision, "recall": recall}

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["test"],
    compute_metrics=compute_metrics,
)
```

Step 6: Train the Model

```
trainer.train()
```

Step 7: Log in to Hugging Face Hub

Do this only when you're ready to push your model.

```
from huggingface_hub import notebook_login

```

```
notebook_login()
```

After running this cell, you'll be prompted to enter your Hugging Face access token. You can create one here: https://huggingface.co/settings/tokens

Step 8: Push Model and Tokenizer to Hugging Face Hub

Replace `"your-username/model-name"` with your actual username and desired model name.

```
model_name = "your-username/distilbert-sentiment-imdb-small"

model.push_to_hub(model_name)
tokenizer.push_to_hub(model_name)
```

This makes your model available for download and use in a Hugging Face Space.

Step 9: (Optional) Test with Gradio Locally in Colab

This is useful for debugging before deploying.

```
import gradio as gr
from transformers import pipeline

classifier = pipeline("sentiment-analysis", model=model_name)

def predict_sentiment(text):
    result = classifier(text)[0]
    return f"Label: {result['label']}, Confidence: {round(result['score'], 3)}"

interface = gr.Interface(
    fn=predict_sentiment,
    inputs="text",
    outputs="text",
    title="Sentiment Analysis",
    description="Enter a movie review to classify as POSITIVE or NEGATIVE."
)

interface.launch()
```

Step 10: Create a Hugging Face Space for Deployment

1. Go to https://huggingface.co/spaces
2. Click "Create New Space"
3. Choose:
    - **SDK**: Gradio
    - **Visibility**: Public or Private
    - Name: e.g. `sentiment-analyzer-student`

Add these two files to your Space:

1. `app.py`

```
import gradio as gr
from transformers import pipeline

model_name = "your-username/distilbert-sentiment-imdb-small"
classifier = pipeline("sentiment-analysis", model=model_name)

def predict_sentiment(text):
    result = classifier(text)[0]
    return f"Label: {result['label']}, Confidence: {round(result['score'], 3)}"

interface = gr.Interface(
    fn=predict_sentiment,
    inputs="text",
    outputs="text",
    title="Sentiment Analysis",
    description="Enter a movie review to classify as POSITIVE or NEGATIVE."
)

interface.launch()
```

2. `requirements.txt`

```
transformers
torch
gradio
```

After uploading both files, Hugging Face will automatically build and deploy your Space.

Conclusion

This complete workflow demonstrates how to:

- Fine-tune a transformer model on a small dataset
- Save and share the model using Hugging Face Hub
- Deploy the model as a web app with Hugging Face Spaces and Gradio

This structure is optimized for educational use, minimal setup, and reproducibility. If you would like a Colab version or a GitHub template, I can generate those for you as well.

Full Fine Tuning Code - IMDB Reviews

```
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

# Step 1: Load dataset
dataset = load_dataset("imdb", split='train[:2000]')
dataset = dataset.train_test_split(test_size=0.2)

# Step 2: Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=2)

# Step 3: Tokenize
def tokenize(example):
    return tokenizer(example["text"], truncation=True, padding="max_length")

tokenized_data = dataset.map(tokenize, batched=True)
tokenized_data.set_format("torch", columns=["input_ids", "attention_mask", "label"])

# Step 4: Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
)

# Step 5: Evaluation function
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc, "f1": f1, "precision": precision, "recall": recall}

# Step 6: Train
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_data["train"],
    eval_dataset=tokenized_data["test"],
    compute_metrics=compute_metrics,
)

trainer.train()

```

Full Fine Tuning Code - Sentiment Analysis Amazon Reviews

```
# Required installations (uncomment if not already installed)
# !pip install transformers datasets scikit-learn

from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from datasets import load_dataset, Dataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

# Load your CSV file (replace with your actual path)
df = pd.read_csv("amazon_reviews.csv")  # Columns: 'title', 'content', 'label'

# Combine title and content for input
df["text"] = df["title"] + " " + df["content"]
df = df[["text", "label"]]

# Split into train and validation
train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)

# Convert to Hugging Face Dataset format
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)

# Load pre-trained BERT tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Tokenize the text
def tokenize_function(example):
    return tokenizer(example["text"], padding="max_length", truncation=True, max_length=256)

train_dataset = train_dataset.map(tokenize_function, batched=True)
val_dataset = val_dataset.map(tokenize_function, batched=True)

# Set format for PyTorch
train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
val_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])

# Load BERT model for binary classification
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)

# Define evaluation metrics
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = logits.argmax(axis=1)
    return {"accuracy": accuracy_score(labels, preds)}

# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    logging_dir="./logs",
    logging_steps=10,
)

# Trainer for training and evaluation
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
)

# Start fine-tuning
trainer.train()
# Save the model
trainer.save_model("fine_tuned_bert_amazon_reviews")
```

More Fine Tuning Examples

1. Text Classification: News Topic Classification

Task: Classify news articles into topics (e.g., business, sports, politics)

- **Dataset**: AG News (4-class classification)
- **Model**: `distilbert-base-uncased`
- **Why it's good**: Multiclass instead of binary; introduces students to topic classification.

```
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

# Load data
dataset = load_dataset("ag_news")
dataset = dataset["train"].train_test_split(test_size=0.2)

# Tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=4)

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

tokenized_data = dataset.map(tokenize_function, batched=True)
tokenized_data.set_format("torch", columns=["input_ids", "attention_mask", "label"])

# Training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    per_device_train_batch_size=8,
    num_train_epochs=3,
)

# Evaluation
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc}

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_data["train"],
    eval_dataset=tokenized_data["test"],
    compute_metrics=compute_metrics,
)

trainer.train()
```

Then follow the same login + push + deploy steps.

---

2. Text Generation: Simple Story Completion (using GPT-2)

Task: Given a prompt, generate the next few sentences of a story.

- **Model**: `gpt2`
- **Dataset**: A small set of fairy tales or a pre-tokenized open dataset like `wikitext`

> GPT-based fine-tuning takes longer and needs GPU memory, so keep the dataset very small.

```
from datasets import load_dataset
from transformers import GPT2Tokenizer, GPT2LMHeadModel, Trainer, TrainingArguments

# Load small dataset
dataset = load_dataset("wikitext", "wikitext-2-raw-v1", split="train[:1%]")

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token  # GPT2 has no padding token

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

tokenized_dataset = dataset.map(tokenize_function, batched=True)
tokenized_dataset.set_format("torch", columns=["input_ids"])

# Load model
model = GPT2LMHeadModel.from_pretrained("gpt2")

training_args = TrainingArguments(
    output_dir="./gpt2-finetuned",
    overwrite_output_dir=True,
    per_device_train_batch_size=2,
    num_train_epochs=1,
    save_total_limit=1,
    prediction_loss_only=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
)

trainer.train()
```

You can then build a Gradio app with a text input (prompt) and text output (generated continuation).

---

3. Named Entity Recognition (NER)

Task: Identify entities like person names, locations, etc.

- **Dataset**: `conll2003`
- **Model**: `bert-base-cased`

NER gives students exposure to **token-level** classification.

```
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForTokenClassification, TrainingArguments, Trainer
from transformers import DataCollatorForTokenClassification
import numpy as np

dataset = load_dataset("conll2003")
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

# Tokenize
```

```
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(examples["tokens"], truncation=True, is_split_into_words=True, padding="max_length")
    labels = []
    for i, label in enumerate(examples["ner_tags"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        previous_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            if word_idx is None:
                label_ids.append(-100)
            elif word_idx != previous_word_idx:
                label_ids.append(label[word_idx])
            else:
                label_ids.append(label[word_idx])
            previous_word_idx = word_idx
        labels.append(label_ids)
    tokenized_inputs["labels"] = labels
    return tokenized_inputs

tokenized_data = dataset.map(tokenize_and_align_labels, batched=True)
tokenized_data.set_format("torch")

model = AutoModelForTokenClassification.from_pretrained("bert-base-cased", num_labels=9)

training_args = TrainingArguments(
    output_dir="./ner-model",
    evaluation_strategy="epoch",
    per_device_train_batch_size=8,
    num_train_epochs=2,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_data["train"].select(range(1000)),
    eval_dataset=tokenized_data["validation"].select(range(200)),
    data_collator=DataCollatorForTokenClassification(tokenizer),
)

trainer.train()
```