DEVELOPING A HIGHLY ACCURATE TRANSCRIPTION MODEL FOR MARATHI LANGUAGE

Overview

This documentation provides a comprehensive guide to understanding and implementing an audio transcription system using Python. The system consists of multiple steps, including data collection, audio and text preprocessing, model development, and model training.

TABLE OF CONTENTS

S.NO	TOPICS
1.	Introduction
2.	Libraries and Dependencies
3.	Initialization and Usage of Models • BERT Model Initialization: • Whisper-Large-v3 Model Implementation:
4.	Hyperparameter Tuning
5.	Model Evaluation: WER Calculation
6.	Conclusion
7.	Code snippet

I. INTRODUCTION

The code snippet demonstrates the implementation of automatic speech recognition (ASR) using state-of-the-art pre-trained models available in the Hugging Face Transformers library. It covers the initialization of models, hyperparameter tuning, dataset handling, and error rate calculations for evaluating ASR performance metrics like Word Error Rate (WER) and Character Error Rate (CER).

II. LIBRARIES AND DEPENDENCIES:

The script begins by installing necessary Python packages using pip, including transformers, datasets, accelerate, and tensorflow. These packages provide access to various pre-trained models, datasets, and utilities required for ASR tasks.

PRE- REQUIREMENTS:

```
! pip3 install transformers
! pip3 install datasets
! pip install accelerate
```

- Transformers - Transformers have emerged as a groundbreaking architecture in natural language processing (NLP) and various related tasks, including automatic speech recognition (ASR).
- Dataset - The datasets library is used to access and handle datasets, specifically for the ASR task in conjunction with the Hugging Face Transformers library.
- Accelerate - The accelerate library is utilized to enhance the performance of the code, particularly concerning training and inference speed when working with machine learning models. In this project focused on automatic speech recognition (ASR) using the Hugging Face Transformers library, accelerate offers specific advantages.

III. INITIALIZATION AND USAGE OF MODELS:

- BERT Model Initialization:

The code initializes a BERT model and tokenizer ('bert-base-uncased') from the Hugging Face Transformers library. This model is commonly used for natural language processing tasks.

- Whisper-Large-v3 Model Implementation:

The script utilizes the Whisper-Large-v3 model, designed specifically for speech-to-text conversion. It loads the model and its processor, configuring it for optimal performance based on available hardware resources.

The script uses a pipeline to perform automatic speech recognition on audio samples from a 'LibriSpeech' dataset for validation purposes. It also demonstrates ASR on a Marathi language audio file.

IV. HYPERPARAMETER TUNING

- The code conducts hyperparameter tuning for the Whisper-Large-v3 model by iterating through different combinations of max_new_tokens and chunk_length_s values.
- For each combination, it evaluates the Word Error Rate (WER) using a reference text and the recognized text generated by the ASR model.
- The best hyperparameters (resulting in the lowest WER) are tracked to optimize model performance for the ASR task.

V. WER CALCULATION

Error Rate Calculation Functions:

- The script defines functions to calculate Character Error Rate (CER) and Sentence Error Rate (SER) based on reference and hypothesis text or sentences.
- These functions compute the error rates by comparing the characters or sentences between the reference and the ASR-generated hypotheses.

<u>Usage Examples:</u>

- The code snippet demonstrates the use of CER and SER calculations by providing specific reference and hypothesis sentences and computing the error rates.

<u>Documentation Enhancement:</u>

To further improve code comprehension and maintainability:

- Consider adding descriptive comments within the code to explain key steps, variable roles, and logical segments.
- Incorporate docstrings for functions to provide detailed information about their purpose, parameters, and return values.
- Utilize inline comments to clarify complex operations or unusual coding practices.

VI. CONCLUSION

The code snippet showcases a comprehensive ASR workflow utilizing Hugging Face's Transformers library, focusing on model initialization, hyperparameter tuning, and error rate evaluation. Through this, it enables users to understand, implement, and fine-tune ASR models for various speech recognition tasks.

VII. CODE SNIPPET:

```
#installing requirements
! pip3 install transformers
! pip3 install datasets
! pip install accelerate
! pip install --upgrade tensorflow


#using predefined model
from transformers import BertTokenizer, BertModel
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained("bert-base-uncased")
text = "hello guys"
encoded_input = tokenizer(text, return_tensors='pt')
output = model(**encoded_input)
print(output)


#using whisper -large -v3 model
import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline
from datasets import load_dataset


device = "cuda:0" if torch.cuda.is_available() else "cpu"
torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32

model_id = "openai/whisper-large-v3"

model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, low_cpu_mem_usage=True, use_safetensors=True
)
model.to(device)

processor = AutoProcessor.from_pretrained(model_id)

pipe = pipeline(
    "automatic-speech-recognition",
    model=model,
    tokenizer=processor.tokenizer,
```

```
    feature_extractor=processor.feature_extractor,
    max_new_tokens=128,
    chunk_length_s=30,
    batch_size=16,
    return_timestamps=True,
    torch_dtype=torch_dtype,
    device=device,
)

dataset = load_dataset("distil-whisper/librispeech_long", "clean", split="validation")
sample = dataset[0]["audio"]

result = pipe(sample)
print(result["text"])
```

#adding marathi audio data

```
sample = '/content/common_voice_mr_31917739.wav'
result = pipe(sample)
print(result["text"])
```

#adding text for the marathi audio data

```
reference = 'त्यानुसार कृषी विद्यापीठातील शिक्षणक्रम राबविले जातात'

def calculate_wer(reference, hypothesis):
    ref_chars = list(reference)
    hyp_chars = list(hypothesis)


    substitutions = sum(
    1 for ref, hyp in zip(ref_chars, hyp_chars)
    if ref != hyp
    )

    deletions = max(len(ref_chars) - len(hyp_chars), 0)
    insertions = max(len(hyp_chars) - len(ref_chars), 0)

    total_chars = max(len(ref_chars), 1)  # Avoid division by zero
    cer = (substitutions + deletions + insertions) / total_chars
    return cer
```

#Hyperparameter tuning:

```
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline
from datasets import load_dataset

# Hyperparameters for tuning
max_new_tokens_values = [64, 128, 256]
chunk_length_s_values = [20, 30, 40]

best_wer = float('inf')
best_hyperparams = {}

device = "cuda:0" if torch.cuda.is_available() else "cpu"
torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32
```

```
model_id = "openai/whisper-large-v3"

for max_tokens in max_new_tokens_values:
    for chunk_length in chunk_length_s_values:
        model = AutoModelForSpeechSeq2Seq.from_pretrained(
            model_id, torch_dtype=torch_dtype, low_cpu_mem_usage=True, use_safetensors=True
        )
        model.to(device)

        processor = AutoProcessor.from_pretrained(model_id)

        pipe = pipeline(
            "automatic-speech-recognition",
            model=model,
            tokenizer=processor.tokenizer,
            feature_extractor=processor.feature_extractor,
            max_new_tokens=max_tokens,
            chunk_length_s=chunk_length,
            batch_size=16,
            return_timestamps=True,
            torch_dtype=torch_dtype,
            device=device,
        )

        dataset = load_dataset("distil-whisper/librispeech_long", "clean", split="validation")
        sample = dataset[0]["audio"]

        result = pipe(sample)
        print(f"WER for max_tokens={max_tokens}, chunk_length={chunk_length}:
{calculate_wer(reference, result['text'])}")

        current_wer = calculate_wer(reference, result["text"])
        if current_wer < best_wer:
            best_wer = current_wer
            best_hyperparams = {'max_tokens': max_tokens, 'chunk_length': chunk_length}

print(f"Best hyperparameters: {best_hyperparams}, Best WER: {best_wer}")


# Calculating Character Error Rate (CER)
def calculate_cer(reference, hypothesis):
    ref_chars = list(reference)
    hyp_chars = list(hypothesis)

    # Counting the number of substitutions, deletions, and insertions
    substitutions = sum(1 for ref, hyp in zip(ref_chars, hyp_chars) if ref != hyp)
    deletions = max(len(ref_chars) - len(hyp_chars), 0)
    insertions = max(len(hyp_chars) - len(ref_chars), 0)

    total_chars = max(len(ref_chars), 1)  # Avoid division by zero
    cer = (substitutions + deletions + insertions) / total_chars
    return cer

# Calculating Sentence Error Rate (SER)
```

```
def calculate_ser(reference_sentences, hypothesis_sentences):
    total_sentences = len(reference_sentences)
    if total_sentences != len(hypothesis_sentences):
        raise ValueError("Number of reference and hypothesis sentences should match.")

    # Calculating CER for each sentence
    error_sentences = sum(1 for ref, hyp in zip(reference_sentences, hypothesis_sentences)
                          if calculate_cer(ref, hyp) > threshold)

    # Calculating SER for each sentence
    ser = error_sentences / total_sentences if total_sentences > 0 else 0
    return ser

reference_sentences = ['त्यानुसार कृषी विद्यापीठातील शिक्षणक्रम राबविले जातात']
hypothesis_sentences = ['त्या नुसार कृषिव विद्यापितातील शिक्षन क्रम राभवले जातात।',]
threshold = 0.1

# for CER
cer = calculate_cer(reference_sentences[0], hypothesis_sentences[0])
print(f"Character Error Rate (CER): {cer}")

# for SER
ser = calculate_ser(reference_sentences, hypothesis_sentences)
print(f"Sentence Error Rate (SER): {ser}")
```