Documentation for developing fine tuned transformer model for ASR in Marathi Language

Project done by:

Jagadeesh Busayavalasa

mail id : jagadeeshbusayavalasa@gmail.com

LinkedIn profile

Objective:

The primary objective of this project is to create finetuned ASR model for the Marathi language. The ASR model must able to transcript the spoken Marathi content Marathi text.

Approach:

Created a fine tuned transformer model which transcribes spoken audio file and transcribes into Marathi text. Took the help of pretrained "'openai/whisper-small' model to train the dataset taken from external site "'mozilla-foundation/common_voice_14_0" for improved performance of model.

Resources used:

1. Hugging face
2. Transformers
3. Tensorsflow

> ** Dataset used: ** mozilla-foundation/common_voice_14_0 with 4008 marathi audio files and text for training the model.

> **Pre-trained model used**: openai/whisper-small' for feature extraction and tokenization of training data

> ** Steps in code execution:**

1. Loading data and Processing of data
2. Training of data
3. Compute evaluation metrics
4. Pushing model to huggingface
5. Load model to transcribe audio files

```
** Major   Code Execution Steps**
```

Loading data and Processing of data

Loaded dataset from "mozilla-foundation/common_voice_14_0" by importing load_dataset.

```
common_voice["train"] = load_dataset(
    "mozilla-foundation/common_voice_14_0", "mr", split="train+validation"
)
common_voice["test"] = load_dataset(
    "mozilla-foundation/common_voice_14_0", "mr", split="test"
)
```

** Loading pretrained whisper processor:**

whisper processor is used in the code for feature extraction and tokenization purposes.

```
from transformers import WhisperProcessor

processor = WhisperProcessor.from_pretrained(
    "openai/whisper-small", language="marathi", task="transcribe"

)
```

Casting audio column to match sampling rate of whisper model from 48 KHz to 16 KHz:

from datasets import Audio

sampling_rate = processor.feature_extractor.sampling_rate common_voice = common_voice.cast_column("audio", Audio(sampling_rate=sampling_rate))

```
# This is formatted as code
```

Data Preparation:

1. We use the feature extractor of pre-trained whisper-small model to compute the *log-Mel spectrogram* input features from our 1-dimensional audio array from dataset.

2. We encode the transcriptions(text in Marathi) to label ids through the use of the tokenizer of whisper model processor.
3. we will map the dataset to a function "prep _dataset" which performs feature extraction and encoding of text/transcription to label id numbers, and returns dataset with input features and labelids of text.

```
# This is formatted as code
def prepare_dataset(example):
    audio = example["audio"]

    example = processor(
        audio=audio["array"],
        sampling_rate=audio["sampling_rate"],
        text=example["sentence"],
    )

    # compute input length of audio sample in seconds
    example["input_length"] = len(audio["array"]) / audio["sampling_rate"]

    return example


# This is formatted as code
common_voice = common_voice.map(
    prepare_dataset, remove_columns=common_voice.column_names["train"], num_proc=1
)
```

Now, after mapping common_voice dataset like above and removing previous semtence and audio columns,finally our dataset contains:

1. input features column (log-mel spect)
2. labels (ids genertated for text by whisper tokenizer)
3. input_length column (audio length returned by prepare_dataset function)

Training and Evaluation:

Now we all set with data with input_features as input and labels as output/target to train our data. we use Trainer() from transformers to train our data. Before that we need to define multiple arguments that Trainer method consumes.

The important arguments that Train() will consume are:

1. data collator
2. Evaluation metrics
3. pre-trained checkpoint
4. training arguments

Defining Data collator:

data collator takes our pre-processed data and prepares PyTorch tensors ready for the model.

Below snippet is used for defining collator:

```
# This is formatted as code
import torch

from dataclasses import dataclass
from typing import Any, Dict, List, Union


@dataclass
class DataCollatorSpeechSeq2SeqWithPadding:
    processor: Any

    def __call__(
        self, features: List[Dict[str, Union[List[int], torch.Tensor]]]
    ) -> Dict[str, torch.Tensor]:
        # split inputs and labels since they have to be of different lengths and need different padd
        # first treat the audio inputs by simply returning torch tensors
        input_features = [
            {"input_features": feature["input_features"][0]} for feature in features
        ]
        batch = self.processor.feature_extractor.pad(input_features, return_tensors="pt")

        # get the tokenized label sequences
        label_features = [{"input_ids": feature["labels"]} for feature in features]
        # pad the labels to max length
        labels_batch = self.processor.tokenizer.pad(label_features, return_tensors="pt")

        # replace padding with -100 to ignore loss correctly
        labels = labels_batch["input_ids"].masked_fill(
            labels_batch.attention_mask.ne(1), -100
        )
```

```
        # if bos token is appended in previous tokenization step,
        # cut bos token here as it's append later anyways
        if (labels[:, 0] == self.processor.tokenizer.bos_token_id).all().cpu().item():
            labels = labels[:, 1:]

        batch["labels"] = labels

        return batch
```

```
# This is formatted as code
data_collator = DataCollatorSpeechSeq2SeqWithPadding(processor=processor)
```

The **input features** column which contains *log-mel spectrum of audio* are converted to batched PyTorch tensors by feauture extractor **'pad' method of whishper processor** which returns tensors 'pt' as discussed in above code snippet.

The **labels column** which contains label ids encoded by tokenizer of whisper processor are un-padded after padding the sequences to maximum length in batch using *tokenizer.pad() *method and then differed by -100 to avoid computing loss during training.

Evaluation Metrics:

We will use "word error rate" (WER) as our evaluation metric to test performance of the model. WER is calculated by comparing predicted text list and actual reference text list word by word.

WER looks for substitution, removal and addition of words and calculates wer. It is most useful metric in case of ASR model performance evaluation.

Ideally, WER with '0' score is indication of perfect ASR model.

```
import evaluate
```

```
metric = evaluate.load("wer")
```

Defining compute_metrics:

This function 1st replaces -100 with the pad_token_id in the label_ids column undoing the step which we applied in the data collator step to ignore padded tokens correctly in the loss.Then it decodes the p label ids to strings to calculate WER between predictions list and references list as below.

```
def compute_metrics(pred):
    pred_ids = pred.predictions
    label_ids = pred.label_ids

    # replace -100 with the pad_token_id
    label_ids[label_ids == -100] = tokenizer.pad_token_id

    # we do not want to group tokens when computing the metrics
    pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    label_str = tokenizer.batch_decode(label_ids, skip_special_tokens=True)

    wer = 100 * metric.compute(predictions=pred_str, references=label_str)

    return {"wer": wer}
```

Loading a Pre-Trained Checkpoint:

```
from transformers import WhisperForConditionalGeneration

model = WhisperForConditionalGeneration.from_pretrained("openai/whisper-small")
```

Training Arguments:

Here is the place where we define all fine tuning parameters to improve model performance like learning_rate, gradient accumulation steps, batch size, save steps, metrics, save steps, report to, output directory to push the model evaluationcheckpoingt to hugging face hub etc. Below are metrics selected for finetuning of our model.

```
from transformers import Seq2SeqTrainingArguments

training_args = Seq2SeqTrainingArguments(
    output_dir="./whisper-small-mr",  # name on the HF Hub
    per_device_train_batch_size=16,
    gradient_accumulation_steps=1,  # increase by 2x for every 2x decrease in batch size
    learning_rate=1e-5,
    lr_scheduler_type="constant_with_warmup",
    warmup_steps=50,
```

```
    max_steps=500,
    gradient_checkpointing=True,
    fp16=True,
    fp16_full_eval=True,
    evaluation_strategy="steps",
    per_device_eval_batch_size=16,
    predict_with_generate=True,
    generation_max_length=225,
    save_steps=500,
    eval_steps=500,
    logging_steps=25,
    report_to=["tensorboard"],
    load_best_model_at_end=True,
    metric_for_best_model="wer",
    greater_is_better=False,
    push_to_hub=True,
)
```

Selction of hyper parameters:

1. output_dir="./whisper-small-mr". This is the directory of hugging face hub repo to store model.
2. batch size of 16 and gradient accumulation of 1 is taken to increase training speed to get effective batch size of 16 as per GPU availability.
3. max_steps is set to 500 to make total number of training steps as 500 (epochs), we can increase steps to 1000,2000 but colab memory is not supporting to do it. if steps increases performance increases.
4. FP16 is True since it offers significant speedup and memory savings, making it suitable for inference tasks like this, making precision training with 16 bit instead of 32 bit.
5. save_steps (int, optional, defaults to 500) – Number of updates steps before two checkpoint saves. set to default value.
6. eval_steps (int, optional, defaults to 500) – Number of update steps between two evaluations. set to default value.
7. logging_steps is set to 25 to make Number of update steps between two logs as 25 for better performance.
8. report_to is set tensor board to save training logs to tensor board itself.
9. metric_for_best_model is choosen as 'wer' to evaluate performance of model, since wer is a good parameter in case of ASR model performance evaluation, here ASR for Marathi.

10. The initial learning rate for Adam is set to 1e-5 to get better wer rate after trying multiple learning rates.
11. warmup_steps – Number of steps used for a linear warmup from 0 to learning_rate is set to 50.
12. push_to_hub=True since i wantto pushthe trained model data and save them in my hugging face repo jagadeeshjagat/whisper-small mr

Passed above all arguments collator, compute metrics,training args etc to Trainer along with our model, dataset as below.

```
from transformers import Seq2SeqTrainer

trainer = Seq2SeqTrainer(
    args=training_args,
    model=model,
    train_dataset=common_voice["train"],
    eval_dataset=common_voice["test"],
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    tokenizer=processor,
)
```

Launched the training for 2-3 hours with below code.

```
trainer.train()
```

Acheived below results with wer as 18

Training loss- 0.192000
Validation LosS - 0.267298
WER ortho - 47.420808
WER - 18.864092

Pushing all results in to my repo:

```
kwargs = {
    "dataset_tags": "mozilla-foundation/common_voice_14_0",
    "dataset": "Common Voice 14",  # a 'pretty' name for the training dataset
    "language": "mr",
```

```
    "model_name": "Whisper Small Mr - Jagadeesh Jagat",  # a 'pretty' name for your model
    "finetuned_from": "openai/whisper-small",
    "tasks": "automatic-speech-recognition",
}

trainer.push_to_hub(**kwargs)
```

Validation on few audio samples provided from Akaike Dataset:

Loaded pretrained model "jagadeeshjagat/whisper-small-mr" by loading it from huggingface into pipeline and evaluated few marathiaudio samples for transcription as below.

```
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


from google.colab import drive
drive.mount('/content/drive')


model_id = "jagadeeshjagat/whisper-small-mr"

model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, 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,)
pipe.model.config.forced_decoder_ids = pipe.tokenizer.get_decoder_prompt_ids(language="mr", task="tra
```

```
def speech_to_text(Filename):
  path='/content/drive/MyDrive/samples/'+str(Filename)
  prediction=pipe(path)['text']

  print(prediction)
  return prediction
```

```
# This is formatted as code
import os

test_dir = '/content/drive/MyDrive/samples/'

predictions=[speech_to_text(file) for file in os.listdir(test_dir)]
```

Transcriptions are:

अनेक रचना अभंग गवळ आणि असे स्फूट लेखन त्यांनी केले

संत ज्ञानेश्वरांच्या सातव्या जन्म शताप दिनेमेत्त त्यांनी सर्व प्रथम मौगरा फुलल्या या कादंबरीचे अभिवाचन केले होते

दुर्देवाने काही लोक रावणाच्या प्रतिमेचे दहन करतात

सत्त्यवती अंबालिकेला व्यासाना बगितल्यावर डोये मिटू नकोस असे सांगते

पंजाब एक कुरुषीप्रधान राज्य आहे

ह्या पानास लेखाचे स्वरू पियायला अहे

पंजाब एक कुरुषीप्रधान राज्य आहे दुर्देवाने काही लोक रावणाच्या प्रतिमेचे दहन करतात.

references=['अनेक रचना अभंग गवळणी असे स्फुट लेखन त्यांनी केले','संत ज्ञानेश्वरांच्या सातव्या जन्मशताब्दीनिमित्त त्यांनी सर्वप्रथम मोगरा फुलला या कादंबरीचे अभिवाचन केले होते','दुर्दैवाने कांही लोक रावणाच्या प्रतिमेचे दहन करतात','सत्यवती अंबालिकेला व्यासांना बघितल्यावर डोळे मिटू नकोस असे सांगते' ,'पंजाब एक कृषि प्रधान राज्य आहे','या पानास लेखाचे स्वरूप यायला हवे' ,'पंजाब एक कृषि प्रधान राज्य आहे','दुर्दैवाने कांही लोक रावणाच्या प्रतिमेचे दहन करतात']

```
from evaluate import load
wer = load("wer")
wer_score = wer.compute(predictions=predictions, references=references)
print(wer_score)
```

0.4126984126984127

Finally, developed a model which transcripts marathi audio with wer as 18 using pretrained open ai whisper small model by fine tuning with various hyper parameters and tested the ASR model(**jagadeeshjagat/whisper-small-mr**)developed on few samples provided by alkaike as part of evaluation too as above.