Speech Recognition Using Federated Learning

Group 2:

Anvith(20mcme17) ,Sai jeevan(20mcme30) ,
Sai Rishik(20mcme05),Vishnuvardhan(20mcme13).

DATA COLLECTION AND PREPROCESSING:

Data Collection: It involves collection of diverse speech samples in languages like Hindi, Bengali, Telugu, Tamil to train accurate and robust speech recognition models.Capturing various accents and speaking styles in those languages to enhance model's comprehension of natural speech and taking care of the data should be well generalised over different speakers and contexts.

We can get this type of data from many open source datasets providers like common voice,openslr.org,hugging face,etc. I have collected the data from the dataset "google/fleurs" available on Hugging Face Datasets.We have chosen the data from the Indian languages which are part of South-Asia geographical areas of the fleurs.

Now we get datasets of Hindi,Bengali,Tamil and Telugu in which we have the audio files and tsv files with 9 attributes.And total audio samples size and number of audio samples for each language as follow,
Hindi :total audio samples size -132 MB ,no.of audio samples -239
Bengali : total audio samples size -279 MB ,no.of audio samples -402
Tamil : total audio samples size -239 MB ,no.of audio samples -377
Telugu : total audio samples size -167 MB ,no.of audio samples -311

Attributes of the tsv file:

- id (int): ID of audio sample
- num_samples (int): Number of float values
- path (str): Path to the audio file
- audio (dict): Audio object including loaded audio array, sampling rate and path ot audio
- raw_transcription (str): The non-normalized transcription of the audio file
- transcription (str): Transcription of the audio file
- gender (int): Class id of gender
- lang_id (int): Class id of language
- lang_group_id (int): Class id of language group

In this we need only some attributes and the remaining should be removed and the file should in csv format.with some requirements like the transcripted text with no comma,full stops,etc.,and we need to divide our data collected into training and test data as per the required percentage of splits.so to achieve that the preprocessing of data is to be done.

MODEL SPECIFIC TASKS:

TASKS:

A. Given an audio file model should predict the transcription for the audio and also print the respective WER for each audio file.

B. Train the openai whisper model on 5 different languages

a. Tamil

b. Telugu

c. English(Indian Accent)

d. Bengali

e. Hindi

ENVIRONMENT SETUP:

a. pip install accelerate -U (Then restart the session)

b. pip install datasets>=2.6.1

c. pip install jiwer

d. pip install gradio

e. pip install librosa

f. pip install evaluate>=0.30

g. pip install git+https://github.com/huggingface/transformers (For the datasets from huggingface and also to import transformers)

ABOUT FILES:

A. ModelTest.py

1. **Import libraries**
2. **Read csv file**
3. **Set audio directory and initialize openai whisper small model**
4. **Resample the audio and transcribe**
5. **Print information about each file, its transcription and WER**

B) CSVFileTrain.ipynb

1. **Import libraries**
2. **Data Loading from a csv file**
3. **Whisper FeatureExtractor, Tokenizer and Processor Initialization (With the corresponding language we want to train)**
4. **Data preprocessing**
5. **Data Preparation function**
6. **Data Collation**
7. **Evaluation Metric**
8. **Model Training**
9. **Save the Trained Model**

C) HuggingFaceTrain.ipynb

1. **Import libraries**
2. **Data loading from a huggingface dataset**
3. **From the step everything is same as CSVFileTrain.ipynb**

D) TrainedModelTest.py

1. **Import Libraries**
2. **Read CSV file of audios**
3. **Set audio directory and initialize trained model (which is an output from above files)**
4. **Resample the audio and transcribe**
5. **Print information about each file, its transcription and WER**

The Trained Model produced from file HuggingFaceTrain.ipynb or CSVFileTrain.ipynb contains 11 files which are:

a. **added_token.json**
b. **config.json**
c. **generation_config.json**
d. **merges.txt**
e. **model.safetensors**
f. **normalizer.json**
g. **preprocessor_config.json**
h. **special_tokens_map.json**
i. **tokenizer_config.json**
j. **training_args.bin**
k. **Vocab.json**

Parameter efficient fine-tuning(PEFT)

with LoRA and Prefix Tuning:

TASK:

Fine-tuning the model with parameter-efficient ways such as LoRA and Prefix tuning:

LoRa Tuning:

PRE-REQUISITES:

Pretrained model,

Processed dataset(suitable for finetuning the model)

Implementation:

- Installing the packages,
- Importing packages,
- Loading the pre-trained model,
- Loading the dataset,
- Setting LoRA-PEFT configurations,
- Preparing the dataset,

- Setting the training arguments,
- Evaluation with ‘wer’,
- Training and saving the model.

Result:

Generates a fine-tuned model from a pre-trained model while reducing the number of trainable parameters by learning pairs of rank decomposition matrices while freezing the original weights.

Output:

trainable params: 884,736 || all params: 242,619,648 || trainable%: 0.36465966680489126

Prefix Tuning:

PRE-REQUISITES:

Pretrained model,

Processed dataset(suitable for finetuning the model)

Implementation:

- Installing the packages
- Importing libraries
- Setting up the device
- Loading the pre-trained model
- Loading the dataset
- Processing the dataset
- Setting up optimizer and scheduler
- Fine-tuning the model in a loop
- Saving the model.

Result:

Generates a fine-tuned model from a pretrained model.

FEDERATED SETUP:

The simulation process involves a Federated Learning (FL) client designed for a flower classification task using the Whisper model. To setup the desired federated environment, we have to import the following libraries: flwr, datasets, transformers, multiprocessing etc.

We first load the trained model and the pre-processed data as follows:

```
model = WhisperForConditionalGeneration.from_pretrained("./tamilnew2")
tokenizer = WhisperTokenizer.from_pretrained("./tamilnew2")
```

```
        train_data = self.local_dataset["train"].select(range(100))
        train_data = train_data.map(prepare_dataset,
remove_columns=train_data.column_names["train"])
```

There are three main functions to simulate a Client:

- Get Parameters:
  The client initiates by retrieving the current global model parameters from the server in each round. (model.get_weights())

- Fit function:
  In the fit function of the Flower Client Class, the model is trained on the pre-processes dataset. Since the task type is Sequence to Sequence, we train the transformer model as follows:

  ```
        trainer = transformers.Seq2SeqTrainer(
        model=self.model,
        args=self.training_args,
        train_dataset=train_data,
        data_collator=data_collator
        )
        trainer.train()
        return self.model.get_weights(), len(train_data), {}
  ```

  Finally, the model weights are returned back to the server.

- Evaluate:
  In the Evaluate function, the client evaluates the locally trained model's performance on its private test dataset and returns the performance metrics.

```
        self.model.set_weights(parameters)
        metrics = self.trainer.evaluate(eval_data)
        return metrics["eval_loss"], len(eval_data), metrics
```

- We finally start the client and establish a connection with the server as following:

```
fl.client.start_numpy_client(
    server_address="127.0.0.1:5019",
    client = FlowerClient(),
)
```

Finally, the server aggregates the received model parameters, incorporating knowledge from all clients. The updated global model parameters are then sent back to each client. The entire process iterates over several rounds, allowing the model to progressively learn from different datasets across different clients while respecting the data privacy of each client.