AI TODO Personal Assistant

Basma Ashour , Saloni Vora

Abstract
In this paper we are gonna discuss how we build a speech recognition system with the wake word to detect what the user is saying and based on that we can add a new To-Do or remove a To-Do, the input is a record voice and the output is add/remove or todo or do nothing if it couldn't detect the speech or the user gave a command which is not add/remove

Keywords
Torch — Wav2Vec — NLP — WakeWord

[1] *MIU, Computer Science*

Contents

Introduction

We First implemented the wake word using: Bidirectional LSTM(Long Short term Memory)
Then after the model is active due to the wake word we implemented the speech recognition using Wav2Vec2 model
Then build the Api to process the data that has been recognized from the wav2vec
Then we showed the result in the UI

1. Wake Word Detection

This problem is to wake up a system when a specific word or specific sequence of words are used. This is to wake up a Voice Assistant system to listen to the rest of the commands and take action on them.
The approach used to solve this problem is classification. We classify the data in 0 and 1 to identify if the wake word was spoken or not. 0 indicates the use of non-wake word and 1 indicates the use of wake word.

1. Data Collection:
a. Wake word data collection : To collect the wake word data we have recorded the wake word 100 times using a script.
b. Noise data collection : To collect the non wake word data/ noise data we recorded a 10 min audio to collect the noisy data around.

2. Data Preprocessing:
a. As the wake words sample is less in proportion than the non-wake word samples, we have upsampled teh data (multiplied teh occurance by 10) to increase teh proportion to increase teh accuracy of the system.
b. Now since the noisy recording is just one sample we split it into chunks so that we get more recordings for training the system.
c. Finally after performing both the above actions, we use teh combination of these files and create training and testing datasets which contain the .wav files and their key (0 or 1).

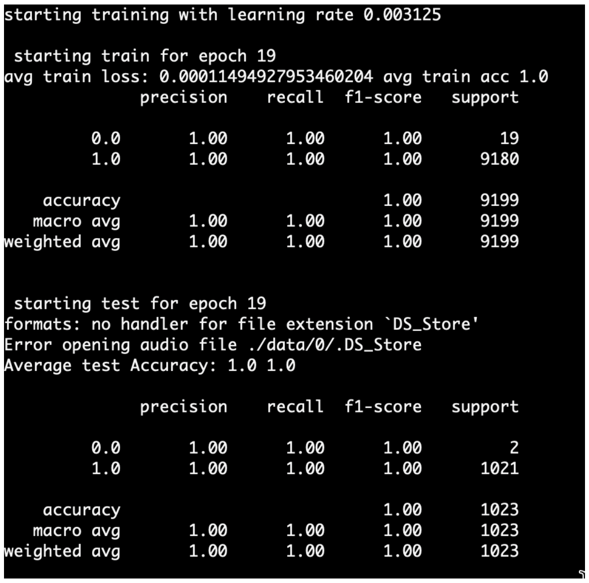
4. Model:
We have used a sequence to sequence model to resolve this problem. And the type used is a bidirectional LSTM(Long Short term Memory) model for our audio samples, with Normalization used as Layer Normalization and since our problem is a classification problem, we have used a liner classification to finally classify the result that we are predicting,

5. Training:
To train the model we need to get the sequence of numbers against the audio samples that we have used and to do the same we have used the MFCC(Mel scale is a scale that relates the perceived frequency of a tone to the actual measured frequency.) technique. This technique transforms the audio data in the sequence of its frequency(numbers), now this data set can be further trained.

7. Final Prediction:
We have created threads in the main loop, where each thread runs for two seconds to listen to the audio and identify if the

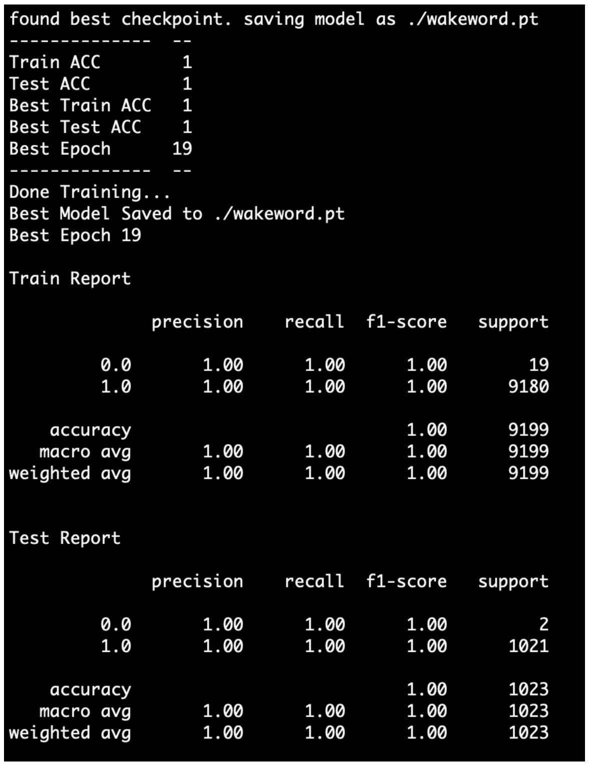
wake word was spoken or not.

Currently due to lack of resources we have taken minimal data set(10 mins of noisy data and 200 wake word samples of 2 seconds each), and trained the system for 20 epochs only(due to which the model seems to be overfitting) and based on that following are the results :



```
starting training with learning rate 0.003125

 starting train for epoch 19
avg train loss: 0.00011494927953460204 avg train acc 1.0
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        19
         1.0       1.00      1.00      1.00      9180

    accuracy                           1.00      9199
   macro avg       1.00      1.00      1.00      9199
weighted avg       1.00      1.00      1.00      9199


 starting test for epoch 19
formats: no handler for file extension `DS_Store'
Error opening audio file ./data/0/.DS_Store
Average test Accuracy: 1.0 1.0

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00         2
         1.0       1.00      1.00      1.00      1021

    accuracy                           1.00      1023
   macro avg       1.00      1.00      1.00      1023
weighted avg       1.00      1.00      1.00      1023
```

(a) Accuracy1.



```
found best checkpoint. saving model as ./wakeword.pt
--------------  --
Train ACC       1
Test ACC        1
Best Train ACC  1
Best Test ACC   1
Best Epoch      19
--------------  --
Done Training...
Best Model Saved to ./wakeword.pt
Best Epoch 19

Train Report

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        19
         1.0       1.00      1.00      1.00      9180

    accuracy                           1.00      9199
   macro avg       1.00      1.00      1.00      9199
weighted avg       1.00      1.00      1.00      9199


Test Report

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00         2
         1.0       1.00      1.00      1.00      1021

    accuracy                           1.00      1023
   macro avg       1.00      1.00      1.00      1023
weighted avg       1.00      1.00      1.00      1023
```

(b) Accuracy2

Figure 1. Showing the accuracy of 20 epochs

2. Speech Recognition

1. The model imports. We import the Wav2Vec2Tokenizer and Wav2Vec2ForCTC. The tokenizer is used for tokenization: converting the raw waveform into tokens that can be fed to the model; Wav2Vec2ForCTC represents the CTC-loss based model class.

2. Initializing the tokenizer. We use the facebook/wav2vec2-base-960h model for this. This model was pretrained on the LibriSpeech corpus and then finetuned on the 960 hours of data; hence the name.

3. Initializing the model. We use the same model for this purpose.

4. Read the sound file. Using librosa, we read the .wav file, with a sampling rate of 16000 Hz.

5. Tokenize the waveform. Using the tokenizer, we tokenize the waveform, and retrieve the input values.

6. Retrieve logits from the model. We retrieve logits from the model, reflecting the whole probability distribution over all possible output tokens.

7. Take the argmax value and decode into transcription. As with any logits, we can take argmax to find the most probable value(s) for the logits. We can batch decode these to find the text corresponding to the speech. Finally, we print this text.

3. Restful apis

1st api: to detect the record [speech recognition] based on Wave2Vec Bert
2nd api: to make string processing to process the text from the recording
3rd api: to show the result add/remove a To-Do based on the text or do nothing

4. UI

We used the Flask template to show the record a voice and then based on that voice add/remove a todo

5. Docker

We used docker to ship the system, we generated the dockers files to be compatible with Flask.

References

[1] The A.I. Hacker - Michael Phi, `https://www.youtube.com/watch?v=ob0p7G2QoHA&list`

[2] Patrick von Platen, `https://huggingface.co/blog/fine-tune-wav2vec2-e`