Text Localization in Audio - NLP project - Fall 2022

Abootorabi,
Mahdi
mahdi.abootorabi@sharif.edu
& *Ahadiniya*,
Aryan
aryan.ahadinia@sharif.edu
& *Dalili*,
Arshan
arshandalili@gmail.com
& *Foroutan*,
Saeed
Foroutan@sharif.edu
& *Haghighi*,
Parsa
Haghighi@sharif.edu
& *Rashidi*,
Sina
sina_rashidi@outlook.com

Abstract

Text Localization in audio involves the identification and localization of relevant text segments within an audio stream. This task is crucial in efficiently identifying speech segments that correspond to the words in a query text, thereby enhancing the search process. Text localization finds application in several domains, including retrieving old voice messages stored on social platforms and searching for content in audio such as tutorials or music. In this work, we develop a multimodal model using a Contrastive Learning paradigm to encode texts and audio into a shared embedding space for the Text Localization task.

1. Introduction

Text localization in audio involves the identification and localization of relevant text segments within an audio stream. This task is crucial in efficiently identifying speech segments that correspond to the words in a query text, thereby enhancing the search process. Text localization finds application in several domains, including retrieving old voice messages stored on social platforms and searching for content in audio such as tutorials or music.

Moreover, text localization can enhance the accessibility and discoverability of multimedia content, allowing users to browse and search for specific topics

or keywords within audio streams. Overall, text localization in audio is a critical problem that has diverse applications. Developing effective and efficient algorithms for this task can improve the accuracy and usability of ASR systems, generate more effective and engaging multimedia content, and augment the overall accessibility and discoverability of audio-based information.

2. Related works

The Cross-Modal BERT (CM-BERT) proposed by Yang et al. [1] is a notable example of a multimodal sentiment analysis model that uses the interaction between text and audio modalities to fine-tune the pre-trained BERT model. The model leverages a masked multimodal attention mechanism that dynamically adjusts the weight of words based on the information from both text and audio modalities. The CM-BERT model was evaluated on the CMU-MOSI and CMU-MOSEI multimodal sentiment analysis datasets and demonstrated significantly improved performance over previous baselines and text-only fine-tuning of BERT.

Siriwardhana et al. [2] proposed jointly fine-tuning "BERT-like" self-supervised models to improve multimodal speech emotion recognition. The authors explored the use of modality-specific pretrained SSL architectures to represent both speech and text modalities for the task of multimodal speech emotion recognition. The experiments conducted on three publicly available datasets showed that jointly fine-tuning "BERT-like" SSL architectures achieved state-of-the-art (SOTA) results. The authors also evaluated two methods of fusing speech and text modalities and showed that a simple fusion mechanism can outperform more complex ones when using SSL models that have similar architectural properties to BERT.

Various end-to-end models for spoken language understanding tasks have been explored recently. However, the task of end-to-end spoken question answering (SQA) has been a challenge until the proposal of SpeechBERT [3], which is an audio-and-text jointly learned language model. SpeechBERT outperformed the conventional approach of cascading ASR with the following text question answering (TQA) model on datasets including ASR errors in answer spans. This is because the end-to-end model was shown to be able to extract information out of audio data before ASR produced errors. When ensembling the proposed end-to-end model with the cascade architecture, even better performance was achieved. In addition to the potential of end-to-end SQA, the SpeechBERT can also be considered for many other spoken language understanding tasks just as BERT for many text processing tasks.

More recently, Wu et al. [4] proposed a large-scale contrastive language-audio pretraining pipeline that combines audio data with natural language descriptions to develop an audio representation. They released a large collection of audio-text pairs and designed a contrastive language-audio pretraining model with feature fusion and keyword-to-caption augmentation mechanisms to enable the model to process audio inputs of variable lengths. Their model achieved

state-of-the-art performance in text-to-audio retrieval and zero-shot audio classification tasks, and competitive performance in supervised audio classification.

3. Data processing pipeline

In order to perform the text localization task, it is necessary to have an appropriate dataset. As no such dataset is currently available, it is necessary to create one. This dataset should contain a set of speech chunks along with their keywords. To ensure meaningful segmentation, training data chunks should be larger than 10 seconds. Extracting keywords from shorter chunks would be futile, whereas longer chunks are unsuitable for the purpose of retrieving the relevant fragments of the input audio file.

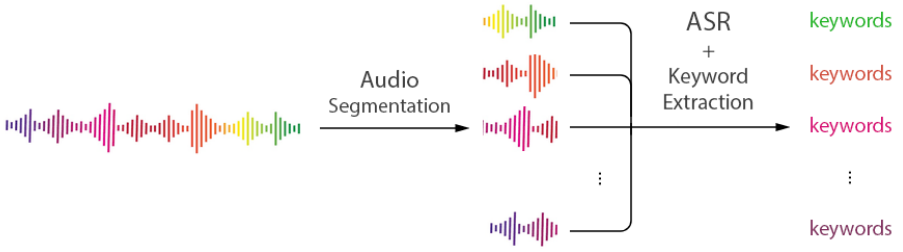

Figure 1: a pipeline that is used for generating the desired dataset in the form of chunk-keyword pairs

3.1. Collecting audio files

For the English language, We used a portion of the LibriSpeech dataset. Fortunately, the audio chunks in this dataset have the aforementioned desirable feature, and in addition, a transcript of each chunk is available in this dataset. We can create our desired dataset using a keyword extraction model.

In the case of the Persian language, existing datasets are comprised of very short chunks, which necessitated the creation of a new dataset. To achieve this, a Farsi podcast in the form of an interview with multiple speakers was selected, spanning a total duration of 70 hours.

3.2. Audio segmentation

Given the large variance in the duration of audio files, ranging from a few minutes to hours, it is essential to segment them into smaller chunks for effective training of acoustic models. To this end, we applied a segmentation technique that detects silences and generates segments of at least 10 seconds in length.

Furthermore, the audio segmentation module is also indispensable during inference time, when we need to segment each audio file and retrieve the segments that are most relevant to a given query text. This step is crucial for achieving accurate and efficient retrieval of the relevant audio segments and is therefore a critical component of our proposed system.

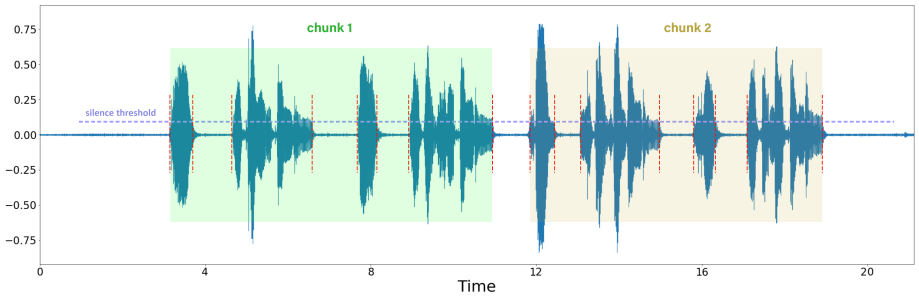

Figure 2: The audio segmentation module amalgamates non-silent segments to generate chunks longer than 10 seconds

3.3. Persian ASR

This section provides an overview of a Persian Automatic Speech Recognition (ASR) code. The code uses several libraries and tools such as Hugging Face's Transformers and Datasets, Sentence Transformers, Torchaudio, Librosa, JiWER, Hazm, YAKE, Multi-Rake, and FastText.

The code includes two main sections: ASR and keyword extraction, and query and dataset comparison. In the ASR and keyword extraction section, the code transcribes a dataset of audio files using the Wav2Vec2 model and extracts keywords from the transcription using YAKE and Multi-Rake. In the query and dataset comparison section, the code computes the cosine similarity between the average embedding of query keywords and the embeddings of dataset keywords, and returns the top N results.

The code is organized into several functions. The `transcribe_dataset` function transcribes a given dataset using the Wav2Vec2 model. The `extract_keywords_yake` and `extract_keywords_rake` functions extract keywords from a transcription using YAKE and Multi-Rake, respectively. The `generate_dataset_keywords` function transcribes a given dataset and extracts keywords from the transcription using YAKE. Finally, the `cosine_similarity` calculates the cosine similarity between two vectors.

The function takes two vectors as input and returns their cosine similarity., and the `load_st_model` function loads a Sentence Transformer model. The `get_similar_parts` function loads a dataset from a JSON file and encodes a given query using the `SentenceTransformer` model.

It then calculates the cosine similarity between the query embedding and the embedding of each chunk in the dataset, where a chunk is a portion of text with a start and end time. The cosine similarity scores are stored in a list along with the start and end time of the chunk, the transcription, and the utterance ID.

The list is then sorted in descending order based on the cosine similarity scores, and the top N most similar chunks are returned.

The `get_results` function takes a dataset file, a query, and a value for N as inputs. It loads the dataset, computes the average embedding of the query

keywords using a Sentence Transformer model, computes the embeddings of the dataset keywords, computes the cosine similarity between the query embedding and each dataset embedding, and returns the top N results.

In conclusion, the Persian ASR code provided in this report is a useful tool for transcribing Persian audio files and extracting keywords from the transcription. The code also provides a way to compare a given query to a dataset of transcribed audio files using cosine similarity. This code can be further improved by adding more advanced ASR and keyword extraction models, as well as implementing more advanced methods for dataset comparison.

As mentioned before, for English data we have transcripts so we just need to find transcripts for our Persian data. Subsequently, transcripts for each chunk were extracted using two ASR models, one of which was pre-prepared, while the other was developed by ourselves.

3.3.1. keyword Extraction

We faced several challenges during the keyword extraction section, and unfortunately, our initial attempts were unsuccessful. Our first approach was to use the `pke` package, but it wasn't suitable for the Persian language. Despite our efforts to manipulate its repository codes by applying Farsi preprocessing functions and using `hazm` posttagger, the code kept crashing. We also tried using a library called `perke`, which was a clone of `pke` but specifically designed for Farsi, but it also failed to work correctly. We even reached out to the developer of this package, but unfortunately, we didn't receive any assistance.

at the next, we opted to use the `yake` library and the `rake` method to extract keywords. In the following, we will explain these two methods in detail and provide a report on our script.

YAKE [1] is a text keyword extraction algorithm s to extract relevant keywords from a given text. Unlike other methods, it focuses on statistical significance and relevance rather than complex statistical models. It is particularly useful for short texts or documents, where traditional keyword extraction methods may not be as effective. It was developed by Campos et al. in 2020 [5] as a simple, yet effective, method for keyword extraction.

The algorithm works by first splitting the input text into individual words and filtering out any stop words. Then, it calculates a score for each word based on its statistical significance within the text, including frequency, distribution, and co-occurrence with other words. Finally, it selects the top-scoring words as the keywords for the text.

YAKE is easy to use and can be customized to fit different applications and use cases, allowing users to adjust the number of keywords returned and set additional parameters such as the maximum length of each keyword and the minimum frequency required for a word to be considered a keyword. Overall, it is a simple yet effective tool for extracting keywords from short texts or documents.

[1] Yet Another Keyword Extractor

The provided code is a Python script that performs keyword extraction on a Persian text using two methods: Yake and Rake. The script first imports several modules from the `hazm` and `yake` packages, including a normalizer, tokenizer, stop words list, and stemmer. It then defines a `preprocess` function that takes a text as input and applies several preprocessing steps to it, including normalization, tokenization, and removal of stop words. The function returns the preprocessed text as a string.

Next, the script defines a `text` variable containing a Persian text to extract keywords from. It then creates two keyword extractor objects, one using the Yake method and the other using the Rake method. The Yake extractor is configured to use a `max-ngram-size` of 3 and a `max-kw-words` of 3, while the Rake extractor is initialized with its default parameters. The script then extracts keywords from the text using each method and prints the top 10 keywords and their scores for each method. Overall, the script provides a simple and straightforward way to perform keyword extraction on Persian text using two popular methods.

4. Methodology

4.1. Baseline

The baseline architecture depicted in Figure **??** is composed of three key elements. The first component, illustrated in Figure 2, performs audio segmentation, dividing the input audio stream into smaller, more manageable segments using two different criteria. The first criterion is a fixed time interval (for example, 10 seconds), and the second criterion is a pause in speech. The minimum value between these two criteria is used to split the audio stream into chunks.

The second component of the architecture applies Automatic Speech Recognition (ASR) to each audio chunk, as described in the Persian ASR section. The third component performs keyword extraction on the audio chunks using the method discussed in the Keyphrase Extraction section. The extracted keywords from the raw audio corpus are also used to generate a common embedding for the audio and text.

This baseline architecture provides a robust foundation for processing audio data and extracting useful information from it. By segmenting the audio into smaller chunks, the ASR and keyword extraction processes can be applied more effectively, improving accuracy and efficiency. The common embedding generated from the extracted keywords enables easy integration of audio and text data, facilitating further analysis and insights.

4.2. Main model

Our main approach for tackling the Text Localization challenge is to use shared embedding space between the two modalities of Text and Audio. We look for methods to encode texts and audios to that embedding space such that relevant pairs of text and audio are close to each other (i.e., have high cosine similarity).

As shown in Figure 3, we use Contrastive Learning as our learning paradigm, in which, pairs of text and audio are given to the model at each batch, and the model tries to maximize the similarity between the corresponding pairs and minimize the similarity between different pairs (i.e., in the matrix of cross-modal pair-wise dot product, the diagonal elements, which corresponds to the related pairs, are maximized, while other elements are minimized).

Due to the resource limit, we used two pre-trained encoders for Text (BERT) and Audio (Wav2Vec2) to encode these modalities. We then implement some layers at the top of these encoders to encode them to the shared embedding space.

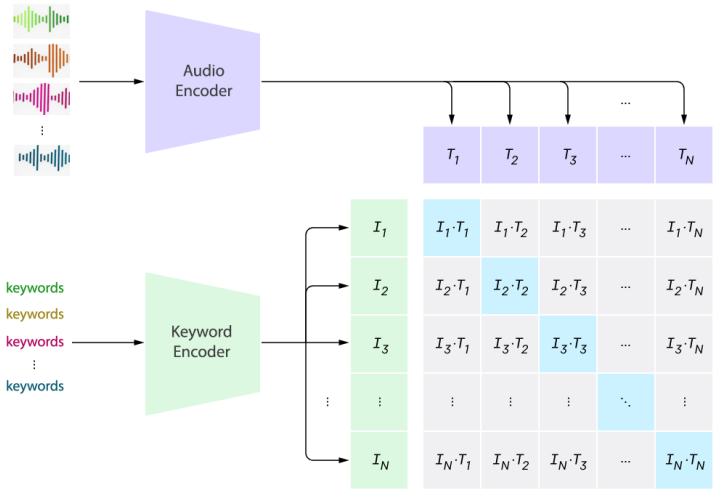

Figure 3: a contrastive learning objective is used for training the model

5. Experiments

Regarding the main model, we trained our model using SimCLR Loss (which is a variation of Contrastive Loss for unsupervised method) and trained the top linear layers of our model on our Persian and English dataset. We then performed evaluations on our self-gathered test dataset in the setting that each instance has a piece of text and six audio candidates. The candidates are shuffled to avoid possible index learning. We calculated Hits@1, Mean Reciprocal Rank (MRR), Accuracy, Precision, Recall, and F1 Macro for each model. The results are shown in the following table.

Model	Hits@1	MRR	Accuracy	Precision	Recall	F1 Macro
Proposed Model	0.163	0.406	0.1	0.5	0.05	0.09

6. Conclusion & Future Works

In this project, we worked on developing approaches and different methods to tackle the Text Localization task. We created datasets for this task in the English and Persian languages. We then designed models to encode texts and audio to the shared embedding space using Unsupervised Contrastive Learning paradigms. Due to resources limit, we could not make our model complex and we could not train the encoders from scratch. Future works can include working on ways to more efficiently localize texts and models that work properly in different settings (e.g., multilingual and tone-agnostic).

References

[1] K. Yang, H. Xu, K. Gao, Cm-bert: Cross-modal bert for text-audio sentiment analysis, in: Proceedings of the 28th ACM international conference on multimedia, 2020, pp. 521–528.

[2] S. Siriwardhana, A. Reis, R. Weerasekera, S. Nanayakkara, Jointly fine-tuning" bert-like" self supervised models to improve multimodal speech emotion recognition, arXiv preprint arXiv:2008.06682.

[3] Y.-S. Chuang, C.-L. Liu, H.-Y. Lee, L.-s. Lee, Speechbert: An audio-and-text jointly learned language model for end-to-end spoken question answering, arXiv preprint arXiv:1910.11559.

[4] Y. Wu, K. Chen, T. Zhang, Y. Hui, T. Berg-Kirkpatrick, S. Dubnov, Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation, arXiv preprint arXiv:2211.06687.

[5] R. Campos, V. Mangaravite, A. Pasquali, A. Jorge, C. Nunes, A. Jatowt, Yake! keyword extraction from single documents using multiple local features, Information Sciences 509 (2020) 257–289. doi:https://doi.org/10.1016/j.ins.2019.09.013.
URL https://www.sciencedirect.com/science/article/pii/S0020025519308588