Podcast Audio Transcript Summarization

Shuaichen Wu

1 Introduction

My project aims to generate short text summaries containing the most salient information for given podcast episodes and their automatically-transcribed transcripts.

The value of my project can be viewed from two perspectives. From a research perspective, while there is a great deal of work on summarizing text in the written text corpora, there is much less existing work on summarization of spoken content. From a podcast user perspective, a nice summary of each episode will help improve user engagement and consumption of podcast content.

In this project, I used both extractive and abstractive summarization models. For extractive models, I build a TFIDF vectorizer and a count vectorizer based summarization model. For abstractive summarization models, I build from scratch seq2seq models primarily comprised of LSTM layers. I also try a state-of-the-art BART model pretrained on a CNN news corpus dataset.

I provide an example summary in Figure 1, showcasing the LSTM Seq2Seq model, which was the model most studied and experimented with in this paper.

2 Background

2.1 Previous research

The podcast dataset was built for the Text Retrieval Conference in 2020 by Spotify engineers. There were 8 participants and 22 experiments for the summarization task. All experiments used some form of a pre-trained BART model, and all were based on abstractive techniques.

2.2 Seq2Seq, LSTM, and Attention Mechanism

The LSTM is one of the most widely adopted architectures for seq2seq models. LSTM is short for Long Short Term Memory, an improvement on the simple RNN to address vanishing and exploding gradients. It introduces a cell state to store long-term information of previous steps, whereas the simple RNN takes only the previous step naively without operations to preserve longer term information.

A Seq2Seq framework for abstractive text summarization is composed of an encoder and a decoder. The encoder reads a source article and transforms it to hidden states, while the decoder takes these hidden states and generates a summary. During the model training phase, a ground-truth summary of the source article feeds into the decoder word by word, and the decoder predicts each next word as part of a generated summary. Figure 2 shows a simple depiction of this architecture.

In an Attention mechanism based encoder-decoder architecture, the decoder not only takes the encoder's final hidden and cell states of the source article as input, but also selectively focuses on parts of the article at each decoding step.

Another Seq2Seq model architecture is BART, a large transformer model. Transformer models are not based off of recurrent neural networks, forgo recurrent hidden states, and instead uses a self-attention operation, which empirically performs well at scale for very large datasets and models. One core idea behind the self-attention operation is that its representations take into account similarities between features in the network.

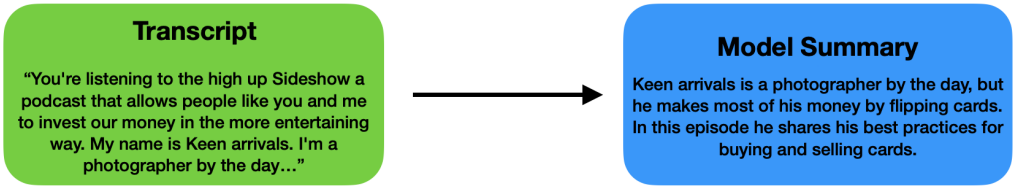

Figure 1: A transcript, summary output pair for the LSTM Seq2Seq model.

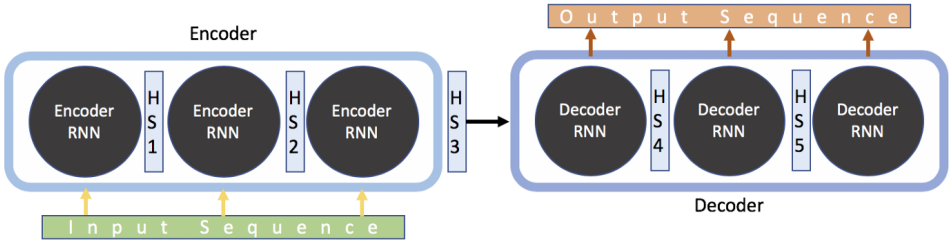

Figure 2: A figure depicting the Seq2Seq encoder-decoder architecture. HS denotes hidden state. In our case, input sequence corresponds the the podcast transcript and output sequence corresponds to the summary.

I make use of all of these operations and architectures in my experiments.

3 Dataset Information

The dataset was generated by Spotify podcast engineers and aims to facilitate research on language technologies applied to podcasts. I acquired the dataset through email contact with conference members via the following URL (https://podcastsdataset.byspotify.com). The dataset consists of just over 100,000 episodes of podcasts. Each episode comes with full audio, a transcript which was automatically generated using Google's speech to text API as of early 2020, and a description provided by the podcast creator.

The dataset is updated every year. In consideration of limited computation and memory, I only used the episodes that were collected in 2020.

In this dataset, no information about user reviews and popularity are included for shows and episodes. A natural next step for future work is to collect user data and popularity data, and build a recommendation system based on summarization of transcripts.

4 Data Preprocessing

Considering runtime and computation power, I aim to find transcripts with the highest quality to decrease the number of files my models need to deal with and produce overall better quality learned models.

The dataset has 105,360 episodes coming from 18,376 shows. I used the following steps to pick out transcripts:

1. Remove all episodes for which the language is not English, which leaves 105,025 episodes.
2. Remove episodes with duration higher than 105 minutes, which are the outliers that are beyond the 95th percentile duration. Remove episodes with duration lower than 3 minutes, which have higher probability to be non-professional. Together, this leaves me with 100,245 episodes.
3. Remove episodes with descriptions that are too long (>750 words) or too short (<20 words). This leaves me with 82,045 episodes.
4. Remove episodes with descriptions that have high similarity (50%) with other episode descriptions, which leaves 32,354 episodes.

After preprocessing, the dataset has 32,354 episodes left, which is a pretty good start to build a model.

5 Exploratory Data Analysis

During the EDA process, there are several interesting results that appeared. In transcripts, the most frequent words are 'like', 'know', and 'yeah'. Whereas in summaries, the most frequent words are 'episode', 'us', and 'podcast'. This subtly shows the clear difference between written text and vocal text.

I also found the average length of transcripts is 7000 words, whereas the average of length summaries is fewer than 100 words. I put some attention on rare words. Rare words in my project are words that show up less than 6 times in the whole corpus. For transcripts, words that shows up less than 6 times take up 80% of the transcripts' vocabulary, and the rare words only contribute to 0.23% of word counts across all transcripts. Words that shows up less than 6 times takes up 80% of the summary vocabulary, and the rare words only contributes 8% of word counts across all summaries. This shows some common words are repeated many times, which makes sense for conversational text.

Model		ROUGE-1			ROUGE-2			ROUGE-L		
		R	P	F	R	P	F	R	P	F
Baseline: Top-K Words	k=100	27	22	22	5	**4**	4	23	**19**	**20**
	k=200	37	18	**23**	**9**	**4**	**5**	**33**	16	**20**
Extractive 1: TF-IDF		23	18	18	3	2	2	19	15	16
Extractive 2: Count Vectorizer		**38**	12	17	6	1	2	31	10	14
Extractive 3: Text Rank		21	**26**	22	4	3	3	20	16	17
Abstractive: LSTM Seq2Seq	1-layer, BatchNorm	30	15	19	5	2	3	24	12	16
Abstractive: BartCNN		22	26	22	5	6	5	19	23	**20**

Table 1: ROUGE scores (%) for the tested models.

6 Modeling

I build both extractive and abstractive summarization models. Extractive summarization means identifying important sections of the text and generating a subset of the sentences from the original text. For extractive summarization, the first model I build is a TFIDF based model. In this TFIDF based model, each sentence in a transcript is treated as a document. Every token gets its score through the TFIDF Vectorizer. Then, every sentence gets its score as the sum of the tokens in it. Lastly, it ranks all the sentences by their score descending and chooses the first 5 sentences as the summarization.

A similar algorithm is used for my Count vectorizer based summarization model: Word frequency is used as the score of a token and the sentence score is the sum of scores of the tokens that belongs to it. Additionally, I extract the first 100 and 200 words from all transcripts and use them as baselines against the two models I built. Also, I apply the text rank summarization model that is written in the `genism` package.

For abstractive summarization models, I built LSTM based encoder-decoder models, using cross entropy as the loss metric. I also tested variants with Batch Normalization included and an attention layer included. For the BART transformer model, I make no modifications to its original architecture.

7 Evaluation

I split the processed podcast dataset into training, validation, and testing sets by 70%, 10% and 20% respectively. For extractive models, there is no need for a validation and test dataset since important sentences are chosen by the sum of word tokens scores alone. I build five extractive models. Baseline-100 and Baseline-200 denote two models that extract the first K words form the transcripts. The results can be found in Table 1.

Given the difficulty of the task, the baseline models actually already do a fairly good job. For k = 100, the ROUGE-L F1 score is 20%. For k = 200, the ROUGE-L recall score is 33%. The Count Vectorizer based extractive model achieved the highest ROUGE-1 Recall, and also performs well for the ROUGE-L scores. The Text Rank model gets the highest score for ROUGE-1 precision. Compared to the other extractive models, TFIDF seems to perform the worst. It may be due to, for vocal text, the most frequent topics people talk about are also the most important and salient topics for summarization. However, TFIDF may give less score to tokens that appear too frequently. Similarly, this would work in favor the top-k words baseline, which achieves strong scores.

For the abstractive models, I give results for two types of models: the LSTM Seq2Seq trained from scratch and BartCNN, a BART model pretrained on the CNN news corpus. Various hyperparameters were experimented with for the LSTM Seq2Seq model, including different number of layers, usage of batch normalization, and usage of an attention layer; the best result of these is given in the table. Overall, for the abstractive models tested, they tend to perform well across the board for ROUGE-L metrics.

8 Discussion and Future Work

Overall, I produced reasonable results across a variety of extractive and abstractive models for text summarization. Future work would be to bring a larger variety of neural architectures to apply to the transcripts. Evaluation could be improved via human evaluation, e.g. using Amazon MTurk. Data-wise, it could be beneficial to make use of audio files containing different information than the raw text, such as emotion and speech speed. This extra information may play an important role in podcast summarization. Another natural next step would be to collect user data and popularity data, and build a recommendation system based on the summarization methods built here.