CS 6301 : FINAL PROJECT REPORT

https://github.com/ravi-raj-97/CS6301_Team_08

Ravi Raj Chakravadhanula *RXC190056*

Radheshyam Kumar *RXK210024*

Kameshwari Soundararajan *KXS210013*

1 INTRODUCTION

Audio podcasts are a popular source of entertainment and information, but their conversational nature makes them overlooked for text mining. However, analyzing podcast content can provide benefits such as identifying topics and creating recommendation systems based on interests. Consolidating podcast content on a particular subject can also create a comprehensive resource for educational purposes. So, summarizing and performing topic modeling on the audio podcast content available on Spotify, will comprehensively generate a pipeline that can provide additional information about a Spotify podcast. To demonstrate and evaluate the pipeline, we use the summarization test set of the Spotify data set[1].

To show the proof of this concept, BERTSum[2] and BERTopic[3] models are being used. BERTSum is an extractive summarizer, that picks important lines from a given transcript to generate a summary of a specific size. BERTopic is used to perform topic modeling on the summary of a given podcast transcript to identify points of interest for listeners. Both these models use BERT as an underlying layer. Both the models being used are BERT based since it is extremely robust and has a wide array of applications. BERT also has extensive documentation and provides an array of packages that can easily be installed and implemented.

Using the given data set, we first run BERTSum to generate a summary. There is no preprocessing done since BERTSum needs to identify the conversational tones as well. We test the generated summaries against a few gold values. We also compare the information contained in the summary against the information contained in the entire transcript by abstraction. After a few rounds of evaluation using the ROGUE score, the best proportion for summarization is selected. The highest ROUGE score achieved is 0.394. After this, we run BERTopic to identify the top five available clusters. Here we achieved an accuracy of 57% when 10% of the summary data is manually evaluated for contextual relevance.

The test dataset contains 917 valid transcripts which were used for the summary generation and evaluation. Of these 917, 100 summaries were used for topic modeling evaluation.

2 TASK

Our aim is to create a pipeline that can be used to generate a new data set from a pre-existing dataset of transcripts. For this purpose, we use the 2020 summarization test set of the Spotify data.

2.1 INPUTS TO BERTSUM MODEL

The test set consists of 1027 transcripts, selected randomly across the entire data set. Out of these, 110 transcripts do not have a summary gold value to compare the model results against. Hence, we dropped these records and ran inference and evaluation on 917 transcripts. These 917 were put into an excel sheet that can be processed as a dataframe. Each cell in the column contains the entire transcript of an episode, which is passed to the model.

2.2 OUTPUTS FROM BERTSUM MODEL

After running BERTSUM on the input string, we get a single string of a specific length that is proportional to the length of the original string. This ratio is evaluated by us by trying various lengths and the better performing one is selected. The summaries are populated into a column of the dataframe that is then written to file for further processing.

2.3 INPUTS TO BERTOPIC MODEL

Post the evaluation of the summaries, the output from the previous phase is fed as inputs to this mode in the form of an excel sheet. All the processing in this phase happens on a single column from the excel sheet. A single string that is the summary of a given transcript is split into multiple sentences. This collection of sentences is passed as a document to the model for processing. If the summary is too small, then the entire transcript is used by the model to produce the required topics.

2.4 OUTPUTS FROM BERTOPIC MODEL

Model returns a collection of topical clusters. These clusters are stored in a column of the dataframe. These results are then manually evaluated for ~10% of the input data set. Post evaluation, the results are written to another excel sheet that is linked later in the report in section '6 Experiments and Results'.

3 DATA

The Spotify data set that we use in this project is the result of another project: "100,000 Podcasts: A Spoken English Document Corpus".

In this data set there are audio podcasts and corresponding transcripts for these audio podcasts. The compressed audio podcasts are of the size 2TB and the compressed audio transcripts are of the size 13 GB. On decompressing, we find that the transcript data is made of 8 subsections and each of these subsections upon decompressing are approximately 12 GB. There is also a test set data that is present in the repository that consists of randomly sampled transcripts. There are 1027 transcripts present in this which we can use as proof of concept to demonstrate the pipeline that we proposed. Due to the lack of computational resources, we restrict our evaluation and data set generation to this test set. All further references to the data set imply the test data set and not the complete data set. A record implies a single transcript.

A few shallow statistics associated with this test set are described below:

- Number of records that are present: 1027
- Number of records that are useful: 917
- Mean podcast length (in minutes): 36.305
- Mean number of sentences per transcript: 274.61
- Unique vocabulary size: 51906 tokens

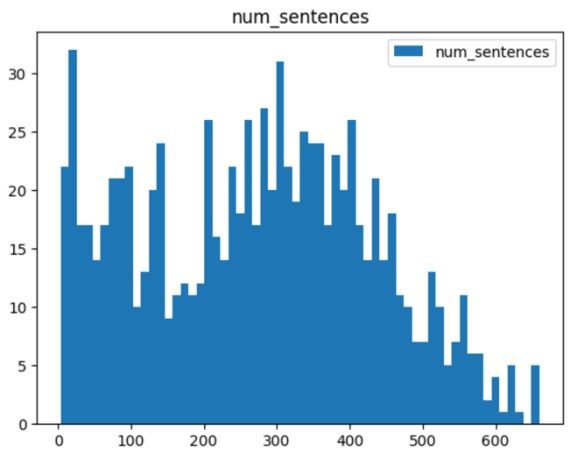

Figure 1: Distribution of the transcripts based on number of sentences.

Since we are not training any model in specific and are just using the models to generate new data we do not have any test or validation set. All 917 records are used to generate evaluation metrics for the summarization part. A random sample of 100 generated summaries is used to validate the topic modeling results. We use a subset in this case because topic modeling evaluation needs to be done manually by the members of the team.

4 METHODOLOGIES

Our project proposes a method for generating a new dataset. Hence, this section covers the logic and explanation of the models being used in this method, along with the way the models are being used.

4.1 EXPLANATION OF THE MODELS

BERTSum is a transformer-based summarization model that uses the BERT (Bidirectional Encoder Representations from Transformers) architecture. It has achieved state-of-the-art performance on several benchmark datasets, including the CNN/Daily Mail dataset. Efficiency of NLP models for text summarization can be evaluated based on various factors, including speed, memory usage, and computational resources required. BERTSum is relatively efficient compared to some of the other

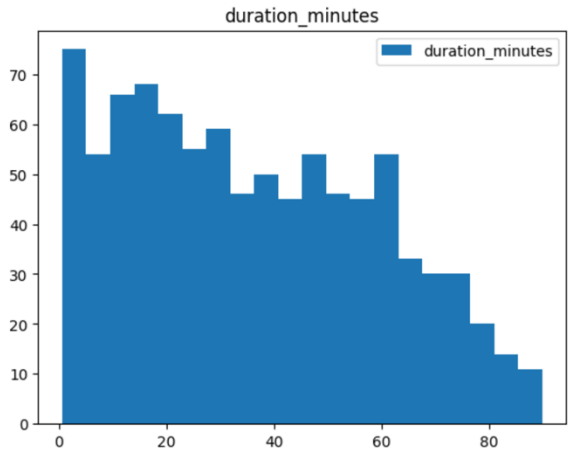

Figure 2: Distribution of the transcripts based on audio length.

transformer-based models. It has a relatively small number of parameters compared to some of the other models and can be fine- tuned for summarization relatively quickly. The model uses a two-stage approach to generate summaries, where the first stage is an encoder-decoder architecture that encodes the input document into a fixed-length vector representation, and the second stage is a transformer-based decoder that generates the summary from the encoded document representation.

BERTopic is a topic modeling technique that utilizes the power of the pre-trained language model BERT (Bidirectional Encoder Representations from Transformers) to identify latent topics in text data. It uses the contextual embeddings generated by BERT and then applies the UMAP algorithm for dimensionality reduction and clusters to group similar documents into topics.

BERTopic is considered more accurate and efficient, especially for datasets with large amounts of text data. It also provides a more interpretable output, as the resulting topics can be easily understood by analyzing the most representative words for each topic. This describes the implementation of two natural language processing techniques, namely summarization and topic modeling, on podcast data.

The first step in summarization using the BERTSum algorithm is to preprocess the text data by removing irrelevant information and segmenting the text into individual sentences. The sentences are scored based on their importance, and a summary is generated by selecting the most important sentences. The quality of the summary is evaluated using ROUGE metrics, and the summary can be refined by adjusting the length or modifying the scoring criteria.

For topic modeling using the BERTopic algorithm, the data is first cleaned by removing irrelevant information and generating document embeddings using pre-trained BERT models. The BERTopic algorithm is applied on the document embeddings to extract topics, and the topics are interpreted by examining the top words associated with each topic. The performance of the BERTopic algorithm is evaluated using metrics like coherence, silhouette score, and topic uniqueness, and the extracted topics are visualized using scatterplots or word clouds.

4.2 USAGE OF THE MODELS

In our implementation BERTSum algorithm uses a pre-trained model to generate summaries of a given transcript using different ratios. The generated summaries are evaluated using ROUGE metrics and an abstractive summarizer. While ROUGE may not be completely reliable due to the shorter length of gold summaries, abstractive summarization provides a better evaluation of the generated summaries.

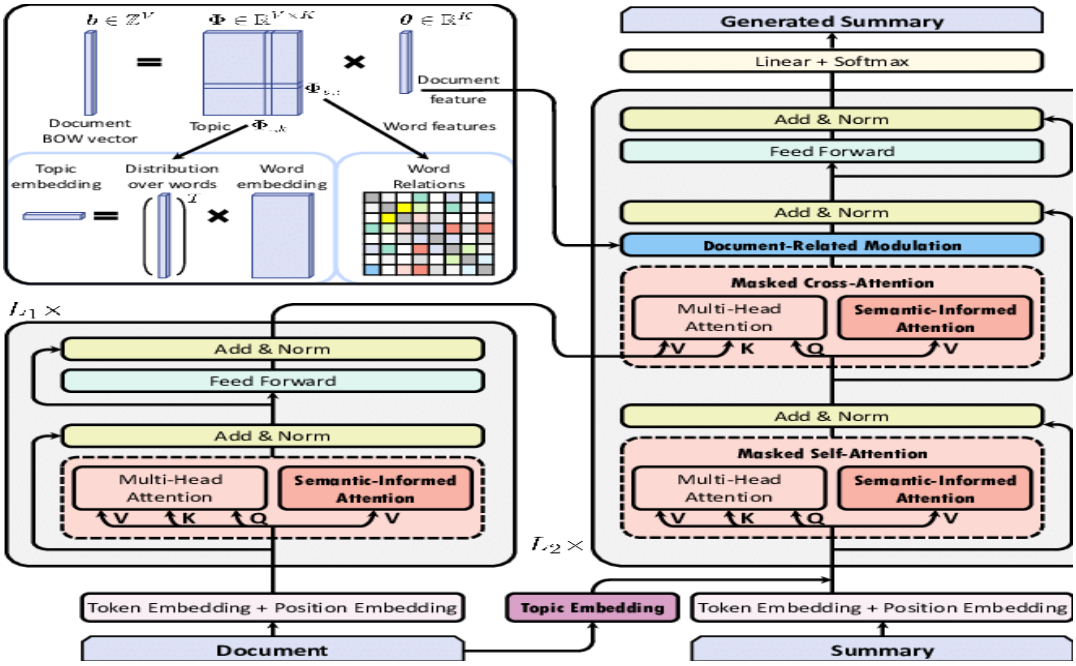

Figure 3: BERTSum Architecture

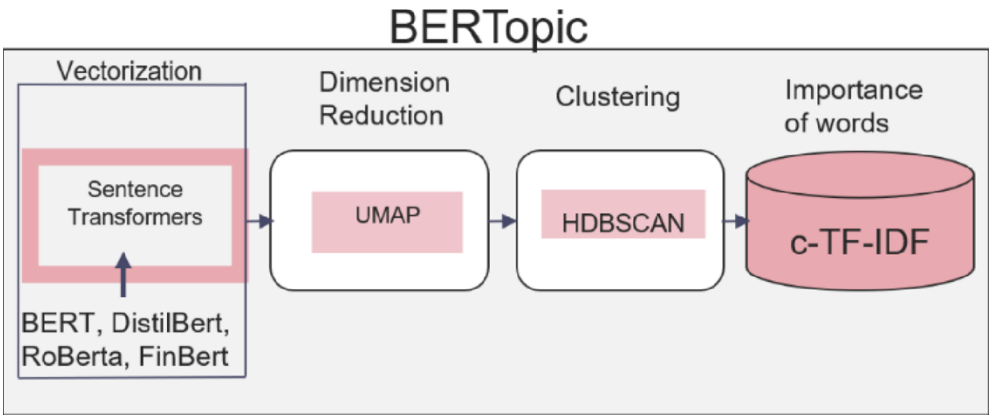

Figure 4: BERTopic Architecture

Extractive summarization picks relevant sentences while abstractive summarization generates a concise summary by understanding the content. The Facebook model[4] is used with Transformers library to generate summaries for both the entire transcript and the extractive summaries.

To ensure better and relevant results on the Topics, we perform three steps of Data Preprocessing before passing the summaries to the BERTopic model. In the first step, we remove all additional spaces and tabs that may affect the modeling. In the second step, we remove all non-English non-Alphanumeric words and characters, as the dataset is derived from Speech to Text, which may contain filler words and Stopwords. In the third step, we remove all stopwords to improve the relevancy of the Topics. To evaluate the accuracy of the modelled Topics, we randomly select two sets of Summary/Topic data and manually evaluate the relevance/non-relevance of the Topics. We calculate the percentage of relevant topics generated from the summaries constructed using a single Excel sheet as input and reference.

4.3 EVALUATION GUIDELINES

To evaluate the accuracy of summaries, it is imperative to ensure that there is no important information lost from the transcript while generating the summary. Important information is classified as information that can change the meaning or interpretations of the given text. Hence, one of the most important guidelines to follow while evaluation of similarity between test, we have to ensure that the information that is eliminated should not impact the interpretation.

So, to tackle this issue, abstractive summarization is additionally implemented. Similarity scores are not calculated. Instead, ROUGE scores are used. Further description of this part is present in section '6-Experiments and Results'.

Similarly, for evaluation of topic modeling, we need to consider the frequency of word occurrence. The words used frequently can be a deviation from the actual content of the podcast. Due to this, an automated system cannot be used to evaluate the topic model efficiency. Manually, the members of this team have evaluated the topics by comparing them against the summaries. This allowed the evaluation to happen on the interpretation of the summary and episode title, rather than the word frequencies.

5. IMPLEMENTATION

5.1 SUMMARIZATION PIPELINE

To implement the summarization pipeline, we used the BERTSum algorithm. In Python, this algorithm is implemented in the package ***summarizer***[5]. The pre-trained BERTSum model is downloaded when we initialize the package class. This model size is 1.34 GB and can be used with a GPU. To extract a summary, we pass the entire transcript to the summary along with the floating-point value called ratio. This variable represents the proportion between the size of the summary and the size of the entire transcript.

The pipeline takes an excel sheet as input and generates another excel sheet as output. The input excel sheet contains a column where each row of the column corresponds to 1 transcript. The output excel sheet will contain all the contents of the input with one

```
# Function to generate summary of a transcript
# summary size/transcript size is the ratio_val
def get_sum(transcript, ratio_val):
  small_transcript = model(transcript, ratio=ratio_val)
  return small_transcript
```

Figure 5: Function used to generate summary

```
# Code to get the rogue scores
def get_rogue_scores(row):
  # change indices depending on the columns to compare
  # rouge_scores = ROUGE.get_scores(row[14], row[13])
  # rouge_scores = ROUGE.get_scores(row[15], row[13])
  # rouge_scores = ROUGE.get_scores(row[16], row[13])
  # rouge_scores = ROUGE.get_scores(row[10], row[2])
  # rouge_scores = ROUGE.get_scores(row[11], row[2])
  rouge_scores = ROUGE.get_scores(row[12], row[2])
  r1 = list(rouge_scores[0].values())[0]['r']
  rl = list(rouge_scores[0].values())[2]['r']
  return [r1, rl]
```

Figure 6: Function used to generate ROUGE scores

additional column. This additional column contains the summary of the transcript.

5.2 TOPIC MODELING PIPELINE

For the Topic Modelling Pipeline, we've used BERTopic Topic Modelling Technique, and we've leveraged ***bertopic***[6] Python library from Maarten Grootendorst. The Model takes the summarized document as the input and returns list of generated topics with probabilities. Out of the list, we extract the 5 most probable Topic clusters and perform manual evaluation on them.

We process the summarized excel sheet and extract the summaries and pass it to the model individually. If the length of the summary is less than the stipulated requirement of bertopic (10 lines or more), we pass along the full transcript to get the top 5 generated Topics.

6. EXPERIMENTS AND RESULTS

All the data that was processed is present in this link. A description of what the columns means is present here. Since the original owners of the dataset keep it restricted, we cannot publish components of the data on a public forum. Hence, it is required that we request you to not share the results elsewhere.

6.1 SUMMARIZATION

6.1.1 DEVELOPMENT AND TEST RESULTS

Data Preprocessing for summarization:

The provided data set is such that the transcripts are stored as JSON files in a separate file structure along

```
#Function to return topics genereated when passed a Text
def model_topic(doc):
  topic_model = BERTopic()
  lines=remove_stopwords(doc)
  if(len(lines)<11):
    return None
  topics, probs = topic_model.fit_transform(lines)
  #print(str(topic_model.get_topic_info()))
  result=get_topics(topic_model)
  return result
```

Figure 6: Function used to generate topics

with the show key and the episode key. On the other hand, there is only 1 XML file for one show. So the preprocessing that was necessary was to map the summary of a specific episode extracted from the XML to the transcript of the episode. Since the episode key was not a part of the XML the mapping process had to be completed by using the episode title instead.

The JSON file structure also had additional information which was not useful for text processing and hence is discarded. Due to this the size of the useful information from the JSON was small. Hence, we aggregated all the separate JSON files into one single excel sheet. All experiments and evaluations are constructed using this single sheet as an input and reference.

Summarization pipeline:

Once the preprocessing is completed the summarization pipeline is used to generate summaries corresponding to the following three ratios:

0.25, 0.3 and 0.35

The time taken for generating a single summary using a 4GB GPU was approximately 7.5 seconds. The above three values are chosen arbitrarily. However, this is a reasonable estimate for how big a summary should be if it is detailed enough to pick up all the required content. Value is less than 0.25 would generate much vaguer summaries and values greater than 0.35 would generate very large summaries.

6.1.2 ERROR ANALYSIS

Evaluation of the Summarization pipeline:

The gold values for the summaries (which were extracted from the XML files) are used for one level of evaluation. The quality of the generated summary is evaluated using ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metrics[7].

There are three types of ROUGE scores: R1, R2 and RL. R1 compares unigrams between the summary and

Summary Ratio	Mean R1	Mean RL
0.25	0.365320	0.337778
0.30	0.380252	0.354347
0.35	**0.394339**	**0.369475**

Table 1: ROUGE scores when we compare the summary against the gold summary.

the transcript. R2 compares bigrams between the summary and transcript. RL on the other hand, compares the longest common subsequences between the two strings.

However, we believe that this might not be a foolproof method of checking if the highlighted summary ratio is the best one. This is because the gold summaries that are provided in the XML files are just episode descriptions, usually between the length of two to four sentences. Since the generated summary is a lot larger than the gold value, we believe another metric of evaluation is also necessary.

Therefore, we also implemented another method of evaluation where we used an abstractive summarizer (facebook/bart-large-xsum) to check if the information contained in the summary is like the information contained in the entire transcript.

To elaborate on the difference, extractive summarization picks sentences from the transcript without altering them. Extractive summarizers identify sentences that carry valid information that is relevant to the content in the transcript to form a summary. On the other hand, Abstractive Summarizers, understand the transcript and generate a small size summary which contains the required information. Abstractive summarization is more human like where the content is understood and interpreted in a concise format. Therefore, if the extractive summary does contain the required relevant information, it's abstractive summary must be similar to the abstractive summary of the entire transcript.

For the abstractive summarization we use the summarization pipeline present in the Transformers library. Here we use the Facebook model to generate an abstractive summary of the entire transcript and an abstractive summary for each of the rationed extractive summaries. Then, we rerun the ROGUE scoring metric comparing the summary of the transcript against the summary of the summary.

While the scores are still low, we can see that for this method as well the ratio of 0.35 is performing the best

Summary Ratio	Mean R1	Mean RL
0.25	0.286904	0.263419
0.30	0.314593	0.289201
0.35	**0.339918**	**0.312618**

Table 2: ROUGE scores when we compare the abstractive summary of transcript against the abstractive summary of extracted summary.

since it contains the most amount of information. Hence the pipeline generates summaries of size 0.35 times the size of the entire transcript.

6.1.3 SPEED ANALYSIS

With a GPU of 4GB, the Extractive summarization using BERTSUM took approximately 7.5 seconds per transcript.

The Abstractive summarization for validation using the facebook/bart-large-xsum model took approximately 7 seconds per short text (extractive summary) and 10 seconds per long text (entire transcript).

6.2 TOPIC MODELING

6.2.1 DEVELOPMENT AND TEST RESULTS

Data Preprocessing for topic modeling:

Before we pass the summaries to the bertopic model, we perform 3 steps of Data Preprocessing to ensure we get better and relevant results on the Topics. Step 1, we removed all the additional spaces and tabs that might affect the modelling. Step 2, We remove all the non-English non-Alphanumeric words and characters. As our dataset is derived from Speech to Text, Transcripts and Summaries contained a lot of filler words and Stopwords. So, for Step 3, we removed all the stopwords to get a better relevancy on the Topics.

Topic modeling pipeline:

After preprocessing, the pipeline starts generating the topics for the summaries. Initially, we perform topic modeling using the summaries. We record the summaries that are returning errors due to the small length of the summary.

Around 74 summaries were too short (where summary length is less than 10 sentences). So in the situation where this error was noticed, the pipeline is automated to use the whole transcript to model the topics. This ensures better coverage of topics and lesser empty records in the newly generated dataset.

Set	Relevancy (%)
Set 1	60
Set 2	54

Table 3: Manual evaluation of the modeled topics to check if the generated results are relevant to the summary.

6.2.2 ERROR ANALYSIS

Evaluation of the Topic Modeling pipeline:

To evaluate the accuracy of the modelled Topics, we randomly chose 2 sets of Summary/Topic data each with the size of 5% of the Total Data set. We manually evaluated the Relevance/Non-Relevance of the Topics and calculated the percentage of relevant topics generated from the Summaries.

6.2.3 SPEED ANALYSIS

With a GPU of 4GB, the entire dataset was processed in approximately 45 minutes which translates to an average of 2.94 seconds per summary.

7 CONCLUSION

Overall, both the pipelines together successfully provide a proof of concept to generate a new data set. The highest ROUGE score is 0.39, which is not very high. This is because of the obvious reduction in size of text from transcript to summary. Therefore, the need for additional abstractive summarization.

With respect to the topic modeling, there is a lot of random garbage present in every transcript and its summary due to the conversational nature of the podcasts. This explains the mediocre relevance of 57% in the test set.

For further work, with the availability of higher computational resources, both models can be fine-tuned to achieve higher accuracies.

8 REFERENCES AND CITATIONS

[1] https://arxiv.org/pdf/2004.04270.pdf - Spotfiy Podcast Dataset

[2] https://arxiv.org/pdf/1903.10318.pdf - Fine-tune BERT for Extractive Summarization

[3] https://arxiv.org/pdf/2203.05794.pdf - BERTopic: Neural topic modeling with a class-based TF-IDF procedure

[4] https://huggingface.co/facebook/bart-large-cnn - Abstractive summarization model by Facebook

[5] https://pypi.org/project/bert-extractive-summarizer/#large-example - BERTSUM python package

[6] https://github.com/MaartenGr/BERTopic - BERTopic code base

[7] https://pypi.org/project/py-rouge/ - ROUGE score python package