EXPLORATION OF PODCAST CORPORA, SUMMARIZATION, AND SEARCH

by

Mathew Perez

APPROVED BY SUPERVISORY COMMITTEE:

Latifur Khan, Chair

Weili Wu

Jessica Ouyang

This thesis is dedicated to my

grandparents, parents, brother,

aunts, uncles, and cousins. Their love and

support continually gives me strength and motivation.

EXPLORATION OF PODCAST CORPORA, SUMMARIZATION, AND SEARCH

by

MATHEW PEREZ, BBA

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2020

ACKNOWLEDGMENTS

I would like to thank my advising professor, Dr. Latifur Khan for advice and support on my work. Also, I would like to thank Dr. Jessica Ouyang for guidance in the automatic text summarization field. Additionally, this work would not be possible if not for Sayeed Salam, Erick Parolin, and Yifan Li for working with me and helping troubleshoot hardware issues.

October 2020

EXPLORATION OF PODCAST CORPORA, SUMMARIZATION, AND SEARCH

Mathew Perez, MS
The University of Texas at Dallas, 2020

Supervising Professor: Latifur Khan, Chair

Podcasts have emerged as an increasingly ubiquitous form of media. This new medium carries several idiosyncrasies, such as multiple speakers, varying audio quality, oscillating topics, (etc.). As podcast consumption grows, so too does the need for knowledge and algorithms to apply to this burgeoning data space. We focus on two useful data tasks: summarization and search, developing methods to tackle both problems and discuss how existing methods in both areas can be tailored to podcast data. Specifically, we use Spotify's podcast dataset, comprising episodes from their ever-growing database of podcasts, as a case study in the data space. Also, we explore this novel dataset, drawing several judgements and patterns regarding the nature of podcast data. Then, we conclude by considering future work and improvements as podcast data continues to grow and its analysis matures.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Podcasts are highly variant and hence understanding them as a distinct type of data corpus requires grappling with this heterogeneity. Two prescient problems in this area are summarization and search. Clearly, helpful episode summaries and search results can facilitate a more satisfying podcast listening experience. Thus, we develop extensions of existing methods and tools to solve these two problems. We use the Spotify Podcast Dataset (Clifton et al., 2020) to develop our approaches and as a foundation to comment on podcast data in general. Below, we discuss the specific definition of our search and summarization problems.

We define the summarization task as: given a podcast episode's audio or transcription text (See Figure 1.1, further details in Chapter 2 - Section 2.1.2), return a piece of text which communicates the most relevant information for a listener to decide whether to listen to the episode or not. Ideal summaries will communicate the most important topics of discussion as well as the relevant speakers. The summaries must be human-readable; however, style and format are not particularly imperative.

We define the search task as: given a query of a certain form (See Figure 1.2), return a ranked list of two-minute segments from podcast episodes which are most relevant to the user's query. Queries consist of: 1) a basic query field, which is a set of keywords, 2) a type, which is a general category the query field falls under (e.g. topical, known-item, etc.), and 3) a description, which is a longer utterance, sentence, or question giving more context to the user's intent in the search.

Currently, there is a great dearth of available podcast corpora. Taking advantage of Spotify's new dataset, we explore several general facets of podcast data, especially those relevant to the summarization and search problems. We observe that podcast data is very noisy. Often, the speech-to-text transcription of podcast audio is not accurate, resulting in possibly incoherent or inconsistent grammar. These transcription irregularities present even more

challenges on top of the multiple speakers, varied topics of conversation, varied length, and presence of advertisements that we see in podcasts. We aim to understand these obstacles and develop techniques to subvert them. Namely, we investigate the introductory section of podcasts that may contain highly-condensed yet helpful information about a podcast's content for summarization. Also, we study the prevalence and timing of advertisements in podcasts as they are usually irrelevant to podcast content and may hinder our summarization efforts. Moreover, we consider the transcription errors in podcast text and weigh the ramifications of these inaccuracies regarding the search task.

We synthesize these gathered characteristics of podcast corpora into detailed filtering schemes and similarity metrics to abet summarization and search respectively. Furthermore, we survey several recent deep-learning based approaches to summarization to evaluate the fitness of current summarization methods in the new podcast data space. Specifically, we forgo older recurrent neural network approaches and instead focus on transformers (Vaswani et al., 2017) as a means of summarization. Moreover, we utilize the process of transfer learning (Howard and Ruder, 2018; Peters et al., 2018) to further probe how transformers fair in podcast summarization. Transfer learning has emerged as a valuable process whereby models are trained on highly-general corpora and tasks. This pre-training produces a well-rounded model capable of being adapted to downstream tasks.

Using pre-trained transformers, we establish baseline summarization models. Then, we extend these transformer-based approaches with Dialogue Act Tokenization. In Dialogue Act Tokenization, we embed corpus-level understanding of podcasts by converting dialogue act predictions (See Summarization – Section 3.6 for details) into special tokens. By passing these special tokens into the transformers, we add a further level of context for our transformer to learn during training. Our process produces a viable transformer summarization approach customized for podcasts.

We also survey several established similarity measures for search, evaluating the degree to which existing information retrieval (IR) similarity scores can be applied to idiosyncratic

podcast data. More precisely, we combine current IR methods with deep-learning models to improve search results. Our deep-learning extension also includes pre-trained transformers, specifically one that has been fine-tuned for the Question-Answer task. With the compounded similarity score spanning existing IR measures and new deep-learning-based approaches, we realize improved search performance over existing, canonical IR approaches.

Now, we discuss the structure of the remainder of our work. In Chapter 2, we will describe Spotify's podcast dataset. Then we will report key insights drawn from the data to inform our approach to summarization and to search. In Chapter 3, we explore several transformer-based summarization techniques to establish baseline models. Then, we discuss Dialogue Act Tokenization to extend our baselines and compare performance using the ROUGE score. In Chapter 4, we investigate the current unsupervised methods in search and establish their performance as baselines. Then, we discuss our approach and how it extends several current approaches, using specific knowledge of our data as well as cutting-edge transformer models. Concluding our search discussion is a comparison of our approach's results with the several baselines, using the Average NDCG metric. We conclude our work by specifying different models and techniques to possibly improve our summarization and search methods.

```
 {"results":
 [{"alternatives":  // always only one alternative in these transcripts
   [{"transcript": "Hello, y'all, ... <30 s worth of text> ... ",
     "confidence": 0.8640950322151184,
     "words":  // list of words
     [{"startTime": "3s", "endTime": "3.300s", "word": "Hello,"},
...
]}]},
  {"alternatives": [
      {"transcript": "Aaron ... ",
       "confidence": 0.7733442187309265,
       "words": [
  {"startTime": "30s", "endTime": "30.200s", "word": "Aaron"}, ... ]}]},
  {"alternatives":  // last item in "results": a straight list of words
with "speakerTag"
   [{"words":
     [{"startTime": "3s", "endTime": "3.300s", "word": "Hello,",
"speakerTag": 1},
      ...
{"startTime": "30s", "endTime": "30.200s", "word": "Aaron", "speakerTag":
1},
      ...

      {"startTime": "39.900s", "endTime": "40.500s", "word": "salon.",
"speakerTag": 2} ] }] }]
}
```

Figure 1.1. Format of podcast episode text transcription files

```
<topic>
  <num>1</num>
  <query>coronavirus spread</query>
  <type>topical</type>
  <description>What were people saying about the spread of the novel coronavirus NCOV-19 in Wuhan at the end of 2019?
  </description>
</topic>
```

Figure 1.2. Basic format of queries in the search task

CHAPTER 2

PODCAST DATA

In April of 2020, Spotify released an unprecedented dataset of approximately 100,000 podcast episodes' audio files and respective audio transcription and RSS feed metadata (Clifton et al., 2020) . The dataset consists of almost exclusively English-language podcast episodes, with very few non-English episodes having slipped through Spotify's filters. Podcast transcriptions have been generated by Google Cloud Platform's Speech-to-Text API [1]. Moreover, the dataset spans a myriad of production quality, from professional to self-produced. Hence, the transcriptions generated by the speech recognition software are not necessarily correct. The dataset also represents a variety of podcast topics and structures. The subject matter ranges from specific sports or news topics to general lifestyle advice. Consequently, the structure of the dataset episodes is also highly heterogeneous, covering formal shows and informal conversations. Moreover, the average episode length is 31.6 minutes, spanning extremely short to very lengthy episodes. The details of the exact data format are discussed below.

2.1 Data Format

Spotify has delivered three different file formats for each podcast episode: audio file, text transcription file, and RSS metadata file.

2.1.1 Audio File

The audio files provided by Spotify comprise nearly 2TB of data. Due to the sheer data size and computational power needed, and given our available hardware resources, we will not use the audio files provided. Thus, the approaches and strategies we will discuss do not require any of these audio files.

[1] `https://cloud.google.com/speech-to-text/docs/video-model`

2.1.2 Transcription File

The transcription files provided constitute approximately 12GB of data. The average text length of a podcast episode in the dataset is approximately 6000 words. The transcripts are given in JSON file format. More specifically, the file maps 30-second time segments to a specific snippet of transcribed text, along with a confidence measure of the transcription and predicted speaker tag (See Figure 1.1). The 30-second time segments are marked by an offset in seconds of the start and end time of the segment. Then, the same offset time segment demarcation is used for each word spoken in that 30-second time chunk.

2.1.3 Metadata File

The RSS feed metadata comes in a tsv file format. The file has 12 columns: show uri, show name, show description, publisher, language, rss link, episode uri, episode name, episode description, duration, show filename prefix, episode filename prefix. The show filename prefix and episode file name prefix refer to Spotify's internal show and episode identifying primary key respectively.

2.2 Data Analysis

Given the two tasks of interest, we analyze the relevant and potentially helpful features of the dataset. Below, we discuss the specific insights gleaned relevant to each task. Any modifications discussed for each task are only for that task and not applied to any data used in the other.

2.2.1 Summarization Task Insights

Relevant Content Filtering

Typically, podcasts have a logical introductory period at the beginning of an episode. Many times, the host welcomes the listening audience and gives a brief explanation of topics or

introduces any guests. We recognize that this introduction is ripe with useful summarization information, and that the discussion can meander in and out of the overarching topics, as the episode progresses. Hosts and guests often joke around, go down tangential material, (etc.) the deeper into the episode we go. To investigate this phenomenon, we define a set of words indicative of the helpful introduction segment to search for: "we", "episode", "talk", "discuss", "guest", "interview", "topic", "topics". Certainly, this set of words is not exhaustive nor can we be sure it is guaranteed to indicate helpful summarization information. However, these words do provide the basic groundwork for us to check how often a helpful segment occurs as well as where these possibly helpful segments arise in a podcast. Initially, we check whether this phenomenon is abundant among our data. Using the defined set of helpful words, we see that almost all podcast episodes contain at least one of these words (See Figure 2.1).

Now with the knowledge that the occurrence of a helpful word is highly prevalent, we probe for when exactly these words most occur. We find that helpful words occur the most in the first 30-second time segments and decreasingly occur as the podcast continues (See Figure 2.2). Consequently, we decide that rather than having to track multiple speakers or sift through highly fluctuating topics for the entire podcast duration, perhaps we can find the most essential elements of an episode in its introductory 30-second segments. If we can reduce each podcast down to just its introduction, we greatly reduce storage, memory, and computational costs. Moreover, the summarization task of such a smaller text segment is a much simpler task to solve. Thus, for the summarization task, we only use the first 4 minutes (8 thirty-second chunks) of transcription text for each podcast episode. All summarization approaches discussed later use this reduced dataset.

Advertisement Filtering

Early in the process, it became clear that a significant portion of the dataset contained advertisements. Clearly, advertisements are not indicative of episode content needed for

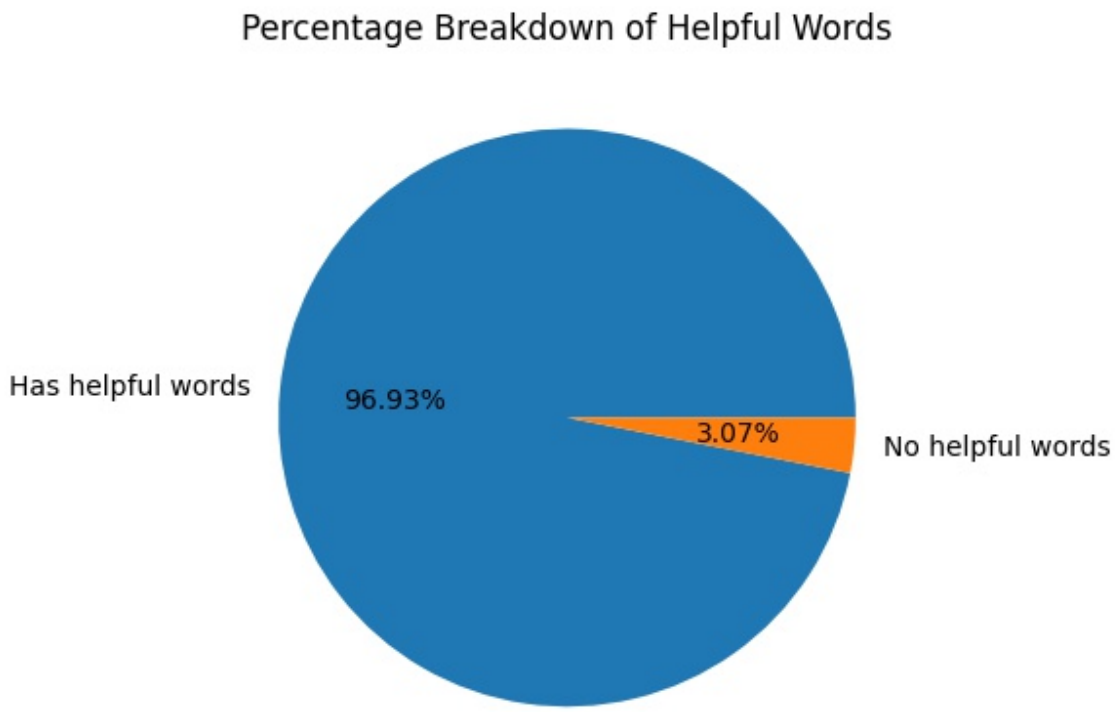

Figure 2.1. Percentages of podcast episodes containing helpful words

a user to decide whether to listen or not. Thus, we attempt to filter out the advertising content. Hence, we investigate how often advertisements occur and at what points in a podcast episode do the advertisements usually occur. To check if a text segment is an advertisement, we define a set of indicative words to search for. Our "ad-words" set is 'sponsor', 'promo', 'promotion', 'ad', 'advertiser', 'download', 'limited', 'offer', and 'anchor' (Anchor is a popular podcast hosting service that is promoted heavily in the dataset). This set of words is not exhaustive and certainly a podcast segment can contain these words as well as potentially helpful information. However, this word set provides a simple basis from which we can gather useful insights and filtering strategies. The data is broken into 30-second

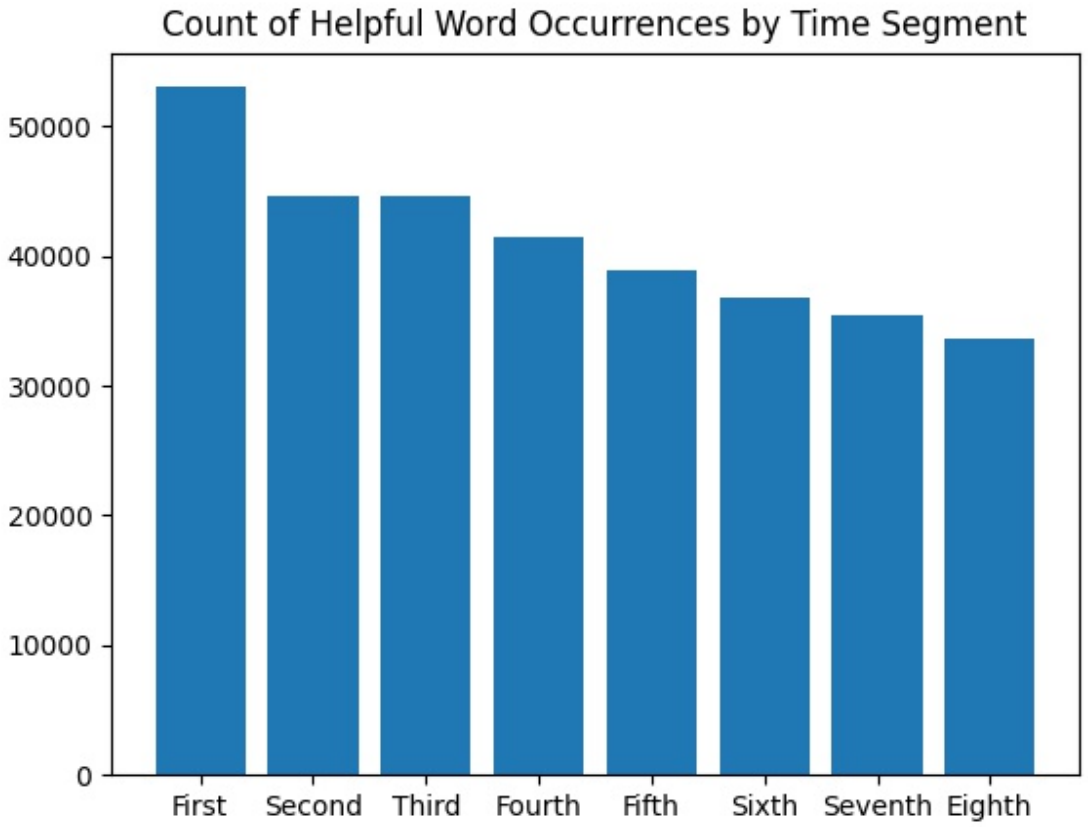

Figure 2.2. Count of helpful word occurrences per 30-sec time segment

chunks, so we look at how often any chunk in a podcast contains an advertisement (using the defined ad-words set), and we notice that much of our data contains advertisements (See Figure 2.3).

Also, we learn that most advertisements are packed into the beginning of a podcast (See Figure 2.4). Thus, we arrive at the problem that much of our previously discussed helpful time segments often occur in the same time segments as advertisements. There is no ideal 30-second time segment where we are certain most ads will not co-occur with the helpful text. Hence, we proceed by attempting to filter out ad content from our reduced first-four-minutes dataset (discussed in the previous section). We simply remove a sentence from the first four

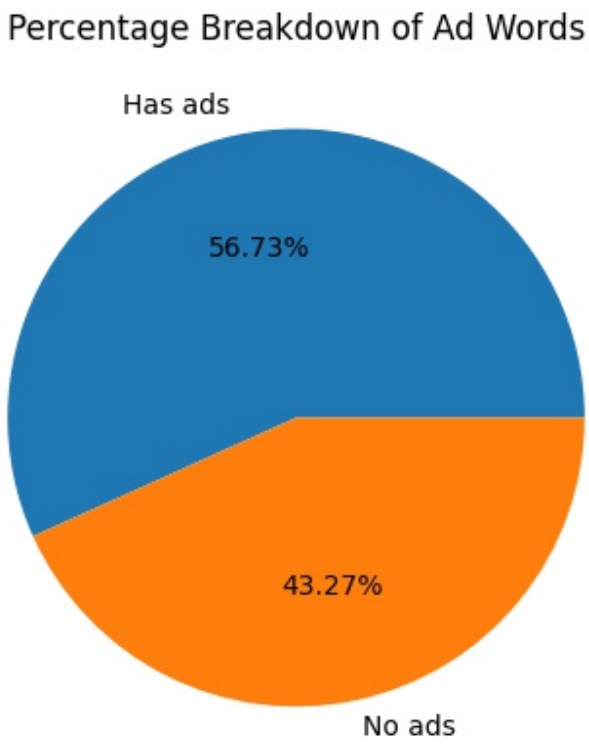

Figure 2.3. Percentages of podcast episodes containing ad words

minutes if it contains one of the defined ad words. Clearly, some helpful information may be lost if it lies in the same sentence as an ad word, but we found this filtering scheme to be intuitive and suffice in practice. This ad filtering finalizes our reduced podcast text dataset and is what we will use as input in the summarization task.

Metadata Summary Filtering

The RSS feed metadata provided contains a column named 'episode description'. This is a creator-provided summary of a given podcast episode. Clearly, this would be ideal as a ground-truth summary for supervised learning summarization methods. However, these pro-

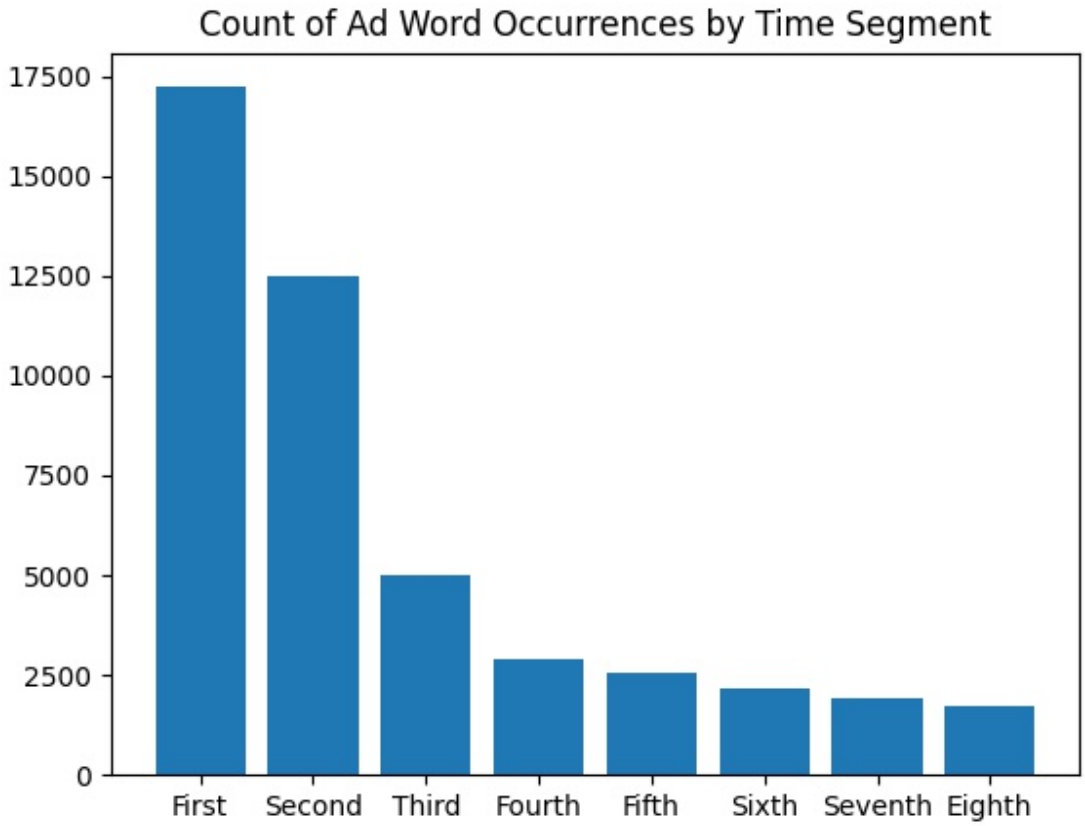

Figure 2.4. Count of ad word occurrences per 30-second time segment

vided summaries are not of the highest quality. Often this column contains advertisements, contains only social media links, or simply contains a poor summary. These creator-provided summaries vary greatly; thus, we must take a more handcrafted approach to notice patterns for filtering. Many of these descriptions have a sponsor list which is either placed after three consecutive dashes or after a Unicode non-breaking space (i.e. '\xa0'). Another pattern we see is that many episode descriptions contain hyper-links to the creator's website or social media pages as well as listing any relevant social media handles. Using these observed patterns, we filter out content coming after three consecutive dashes or after a non-breaking space and filter out any hyper-links and social media handles. To check whether these ob-

served patterns made any difference, we evaluate how many filtered episode descriptions differ by more than 5 characters from the original episode description. We find that 65.89% of filtered episode descriptions met this criterion. Once again, we cannot be certain that exclusively irrelevant information was removed by this filtering; however, it appears to have at least removed links, handles, (etc.) from nearly two-thirds of the dataset's ground-truth summaries.

Spotify Provided Filter

Spotify released a filtered list of episodes, referred to as the brass set, for better supervised training. Ideally, these episodes have higher quality episode descriptions in the metadata file. Using three heuristics, Spotify filtered out roughly one-third of the data set. The first heuristic was length. All episode descriptions greater than 750 characters and less than 20 characters were excluded. All episode descriptions which had a similarity with another episode description of at least 0.5 were excluded (similarity measure used was not specified). All episode descriptions which had a similarity with its show description of at least 0.4 were excluded (similarity measure used was not specified). This filtering scheme resulted in approximately 66-thousand episodes to use in supervised learning methods.

Final Summarization Task Data

We describe the final filtered dataset that we will use in the summarization task. We extract the first-four minutes of podcast text as described in the Relevant Content Filtering section. We filter out advertisements for these episodes in accordance with the Advertisement Filtering section. We also filter the metadata summaries as described in the Metadata Summary Filtering section. Then, we use only those episodes listed in the brass set. From now on, any summarization method we discuss will use this filtered dataset.

2.2.2 Search Task Insights

Data Parsing

The search task requires two-minute segment retrieval. Hence, we collect all two-minute segments from the JSON files across the podcasts into a repository of text files. Parsing and centralizing text for the desired two-minute time constraint will greatly simplify the segment retrieval process. These two-minute segment documents are the data to be used for the search task from now on.

Limited Training Queries

Spotify provided 8 queries, and for each query a list of approximately 75 relevant search results. We conclude that with so few training examples, using a Learning-to-Rank model will not be very effective. Thus, we will only use unsupervised methods for the search task.

Dataset Size

We tried naïve approaches to search with simple looping; however, it became immediately clear that the dataset was far too large for these simple approaches to be effective. Since search is a highly user interactive task, we must use approaches that have a relatively fast turnaround time.

Transcription Errors

Ultimately, any search task relies on somehow reconciling the text within each source document with the text of a query. Thus, with the podcast dataset, search quality is limited by the accuracy of the speech recognition software used to generate the text for each episode from its audio. We know that the text transcription of episodes is not always accurate and depends on the audio quality of the podcast. Thus, we'll have to account for potential errors in the source documents in our search strategies.

Final Search Task Observations

We elect to only use unsupervised methods considering the limited training data. Also, we will only use search methods specialized for massive datasets to keep search times reasonable. To account for transcription inaccuracy, we must also give some room for error when comparing queries with podcast text. Search methods discussed later will reflect these initial observations.

CHAPTER 3

PODCAST SUMMARIZATION

There are two general approaches to automatic text summarization: extractive and abstractive. Extractive summarization is a matter of finding the most important pieces of input text and returning those important pieces verbatim as a summary. Abstractive summarization involves learning or constructing a vocabulary and arranging a coherent and novel summary using that vocab. Our podcast text may contain offensive language or incorrect grammar from inaccurate text transcription. Thus, extracting text directly from this input text may produce undesirable summaries, so we elect to focus on abstractive approaches. However, we will still use the performance of extractive approaches as baselines. For all our methods, we will use the Average ROUGE-N and Average ROUGE-L F-score (Lin, 2004) as performance measures. ROUGE-N measures the overlap between n-grams in the predicted summary and n-grams in the corresponding ground-truth summary. Specifically, we use ROUGE-N as:

$$\frac{\sum_{S \in \{ReferenceSummaries\}} \sum_{ngram \in S} Count_{match}(ngram)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{ngram \in S} Count(ngram)} \tag{3.1}$$

Where n is the n-gram size, $Count_{match}(ngram)$ is the maximum number of co-occurring n-grams between a given predicted summary and reference summary, and $Count(ngram)$ is simply the count of n-grams in the reference summary. We use Average ROUGE-1 and Average ROUGE-2 as evaluation metrics. Also, we use the longest-common-subsequence measure ROUGE-L F-score as follows:

$$R_{lcs} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_i, C)}{m} \tag{3.2}$$

$$P_{lcs} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_i, C)}{n} \tag{3.3}$$

$$F_{lcs} = \frac{R_{lcs} P_{lcs}}{\alpha R_{lcs} + (1 - \alpha) P_{lcs}} \tag{3.4}$$

Where u is the number of sentences in the reference summary, m is the number of words in the reference summary, n is the number of words in the predicted summary, and α is a hyper-parameter that we set to 0.5. Moreover, $LCS_{\cup}(r_i, C)$ is the longest common subsequence between r_i (the set of words in the ith sentence in the reference summary) and C (the union of all words in all sentences in the predicted summary), divided by the number of words in r_i. Our third and final evaluation measure is Average ROUGE-L.

Further ROUGE implementation details can be found in its respective repository [1]. However, it is known that high ROUGE does not necessarily correspond to high quality summaries (Paulus et al., 2017). Nevertheless, this measure provides a customary basis of comparison for our methods.

3.1 Current Abstractive Approaches

Typically, abstractive summarization techniques are well-suited for supervised learning methods (i.e. neural approaches). Moreover, summarization is especially suited for Sequence-to-Sequence models. Traditionally, Sequence-to-Sequence models have been implemented through a Recurrent Neural Network with an Encoder-Decoder Framework and hidden units, such as Long Short Term Memory (Hochreiter and Schmidhuber, 1997) or Gated Recurrent Units (Cho et al., 2014). However, the recently developed Transformer framework (Vaswani et al., 2017), has offered up a new Sequence-to-Sequence implementation to use in abstractive summarization. Transformers have eschewed a recurrent nature in favor of a self-attention mechanism to track or memorize dependencies between pieces of input text. This aspect of Transformers has allowed for higher degrees of parallelization in model training than Recurrent Neural Networks have, and can consequently cut down training time by significant margins.

[1] `https://pypi.org/project/py-rouge/`

3.2 Transformer Abstractive Summarization

We briefly describe how transformers are used in abstractive text summarization; however, a more worthy, in-depth discussion of the general process can be found in the original paper (Vaswani et al., 2017) and other helpful articles [2]. Transformers still retain an encoder-decoder architecture. Thus, after input text has first gone through tokenization and positional encoding, all its words' vector embeddings are fed (in parallel) to an encoder stack and then decoder stack, each consisting of one or many pairs of self-attention mechanisms and feed forward neural networks. During training, the ground-truth summary also goes through tokenization and positional encoding, and is only passed into the decoder stack (shifted right by one token). The decoder outputs a vector of floating-point numbers, which is then fed through a fully-connected neural network, producing a logits-vector whose size corresponds to the size of the vocabulary. Moreover, each element of the logits-vector corresponds to the score of a given word in the vocabulary. The pass through the transformer is concluded by passing the logits-vector through a softmax layer, turning the logits into a probability distribution. Then, we can choose the vocab word with the highest probability to include in the summary. By using cross-entropy loss of the selected token against the corresponding token in the ground-truth summary, one can train the weights of the transformer via back-propagation as is normally done in conventional neural networks. For inference, the pass is the same except the input to the decoder is initially a start-of-sequence token (e.g. "<s >") and we iteratively add the predicted word of the previous pass to the decoder input. This process continues until an end-of-sequence token (e.g. "<\s >") is predicted, and the set of predicted words we have accumulated is returned as the summary.

[2] `http://jalammar.github.io/illustrated-transformer/`

3.2.1 Pre-Trained Transformers

As the popularity of Transformers has grown, so have their applications in many text processing tasks. Namely, using transfer learning, that is pre-training transformers on general, massive amounts of text data and then further training on the specific language task and dataset at hand, has arose (Radford, 2018; Raffel et al., 2019). Pre-Trained transformers have proved to be quite capable on a wide variety of NLP tasks, including summarization (Raffel et al., 2019; Radford et al., 2019; Liu and Lapata, 2019).

3.3 Model Building Process

Given limited and shared hardware resources, we choose to pursue only pre-trained, transformer-based neural approaches to abstractive summarization to keep training time and computational requirements reasonable. Thus, our transfer learning will involve fine-tuning several of the latest pre-trained transformers over our dataset. We will evaluate the performance of fine-tuned transformers, take the best performing models and successively extend and modify them to our task. This procedure may not account for higher-order interactions between initial models and our later modifications, but this process gives an intuitive path to develop a model that will perform well in practice.

3.3.1 Initial Models

We begin by selecting a set of cutting-edge transformers to fine-tune over our podcast dataset. We use T5-Small (~60M parameters), T5-Base (~220M parameters) (Raffel et al., 2019), GPT2 (~117M parameters) (Radford et al., 2019), and BART-Large-CNN (~406M parameters) (Lewis et al., 2019) from their most current implementations (Wolf et al., 2019). BART is pre-trained by denoising and reconstructing input text, and BART-Large-CNN is further fine-tuned with a summarization objective over the CNN/DM dataset (Hermann et al.,

2015). GPT2 is a general language model pre-trained over the massive WebText dataset. The T5 models use a general pre-training framework by converting many NLP tasks into a text-to-text problem, over the large-scale "Colossal Clean Crawled Corpus" dataset. Each transformer also has its own tokenizer. We reserve further discussion of specific experiment and implementation details of these transformers to their respective papers.

3.4 Podcast Summarization Fine-Tuning

We use two Nvidia GeForce RTX 2080 Ti GPUs and PyTorch's DataParallel module for parallelized training. Also, we use an 85%-5%-10% train-validation-test split for model training. We use the model which has the lowest validation loss (using cross-entropy loss), between 1 and 10 epochs of training. Additionally, only the first 128 tokens of input podcast text are passed into the encoder. We also use PyTorch's implementation of Adam optimization (Kingma and Ba, 2014), setting the learning rate to 2e−5, epsilon to 1e−8, and betas left to their defaults. Moreover, we generate test-set summaries of up to 60 tokens, via top-k sampling (Fan et al., 2018) with k=60 and top-p (nucleus) sampling (Holtzman et al., 2019) with p=0.92. We also set a repeating 4-gram limit (as described in (Paulus et al., 2017)), so that no 4-gram appears twice in our summary. In Table 3.1, we can see a summary of ROUGE performance for our initial set of fine-tuned transformers and some extractive baselines.

The last five methods in Table 3.1 are extractive methods. Tf-Idf Filter is a simple implementation of the classical Tf-Idf method for summarization. Tf-Idf Score Max uses the linear programming extraction method, as discussed in (Rudra et al., 2016), to summarize text. TextRank is an implementation of the canonical TextRank algorithm (Mihalcea and Tarau, 2004) and Lead4 simply returns the first four minutes of podcast text as a summary. Oracle is a greedy oracle summarizer as described in (Liu and Lapata, 2019). We observe that

Table 3.1. Summary of initial ROUGE results

Method	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
T5-Small	18.81	3.15	18.08
T5-Base	**20.93**	**4.32**	**19.44**
GPT2	13.97	3.17	12.36
BART-Large-CNN	18.79	1.17	18.62
Tf-Idf Filter	15.20	1.91	13.73
Tf-Idf Score Max	15.55	2.73	13.71
TextRank	15.50	2.92	13.90
Lead4	12.98	3.38	12.57
Oracle	17.31	3.04	16.18

the oracle's performance is still below that of most of the transformer methods, supporting the notion that extractive methods are not auspicious for our podcast data.

We observe that T5-Base performs the best out of all the initial methods and baselines. Noticeably, GPT2 underperforms when compared with the other transformer methods. The original GPT2 paper notes that the model produces summaries that closely mirror the input text over the CNN/DM dataset (Hermann et al., 2015). We investigate this phenomenon by simply evaluating ROUGE scores of the generated summaries against the input podcast text and find ROUGE-1 of 99.33, ROUGE-2 of 99.32, and ROUGE-L of 99.42. Thus, we concur with the original paper that GPT2 attends too closely to input text and has difficulty creating truly abstractive summaries. In comparison with the highest-performing model, T5-Base, we run a similar ROUGE experiment against its input text and find ROUGE-1 of 26.59, ROUGE-2 of 7.79, and ROUGE-L of 22.37. We conclude that T5 is not only the highest performing model but also produces abstractive summaries which do not recite input text. Hence, we select T5-Base as our model to extend and modify in the following discussion.

3.5 Self-Attention Observations

Transformer models employ self-attention to learn the interplay and dependencies among input tokens. This process allows transformers to learn which input tokens to attend to given a certain task. To take advantage of this powerful mechanism, we consider possible extensions of the input text beyond just the words themselves, especially considering the inconsistent podcast audio transcriptions. By examining the podcast text, we notice that it is highly conversational. However, the dialogue flows in one direction, where the show host is talking directly to the audience. Thus, we consider the task of Dialogue Act Prediction to modify the input text.

3.6 Dialogue Act Prediction

Dialogue act prediction is the problem of classifying a given speaker's utterance as one of many possible dialogue acts (e.g. statement, question, agreement, etc.). The canonical dataset in this task is the Switchboard Dialogue Act Corpus (Jurafsky and Shriberg, 1997), comprising approximately 2400 text transcriptions of telephone conversations, involving about 543 speakers and spanning roughly 70 conversation topics. A given utterance can be classified as one of 305 specific dialogue acts, represented as an integer in the range $[0, 304]$. Once again, neural approaches dominate the state-of-the-art approaches to the task. In the interest of training time and conserving hardware resources, we also choose to fine-tune transformers for the dialogue act prediction task. We try both a BERT model (Devlin et al., 2018) and a BART model (Lewis et al., 2019) for sequence classification. For both models, we use a 90%-10% train-test split, use 2 epochs, and have the same hardware and optimization configuration as was used in the first summarization task. We find that BERT achieves accuracy of 68% and BART achieves accuracy of 70%. We choose to use BART since it approaches state-of-the-art performance while also requiring very little training time.

Table 3.2. Summary of final ROUGE results

Method	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
T5-Base	20.93	4.32	19.44
DAT-T5-Base	**26.31**	**7.41**	**22.08**

3.7 Dialogue Act Tokenization

We extend the input podcast text by the following procedure. First, we divide our input podcast into utterances by splitting the text on punctuation marks. Then we feed each utterance through the Dialogue Act Fine-tuned BART model, giving us an integer k, corresponding to the predicted dialogue act. We then construct a token of the form "[k]" and prepend it to the beginning of each utterance of the input podcast text. Next, we add a list of all 305 possible Dialogue Act Tokens as special tokens in the transformer's tokenizer. Only then do we feed the new Dialogue Act Tokenized (DAT) input text to a proper tokenizer and transformer. The diagram in Figure 3.1 summarizes this process. By adding these dialogue act tokens, information about what dialogue act each utterance in an episode's text belongs to can persist as special tokens before being passed to positional encoding and then the encoder stack. Then, the transformer's self-attention mechanisms can use this extended information to learn which dialogue acts to attend to when generating abstractive summaries. In this way, we can give a richer context to the transformer to learn from as it attempts to summarize inconsistent and idiosyncratic podcast episode text.

We conduct training exactly as described in the initial summarization fine-tuning, the only difference being the Dialogue Act Tokenization process. ROUGE performance of our new model can be seen in Table 3.2. Also in the table, is the ROUGE performance results from the bare T5-Base model.

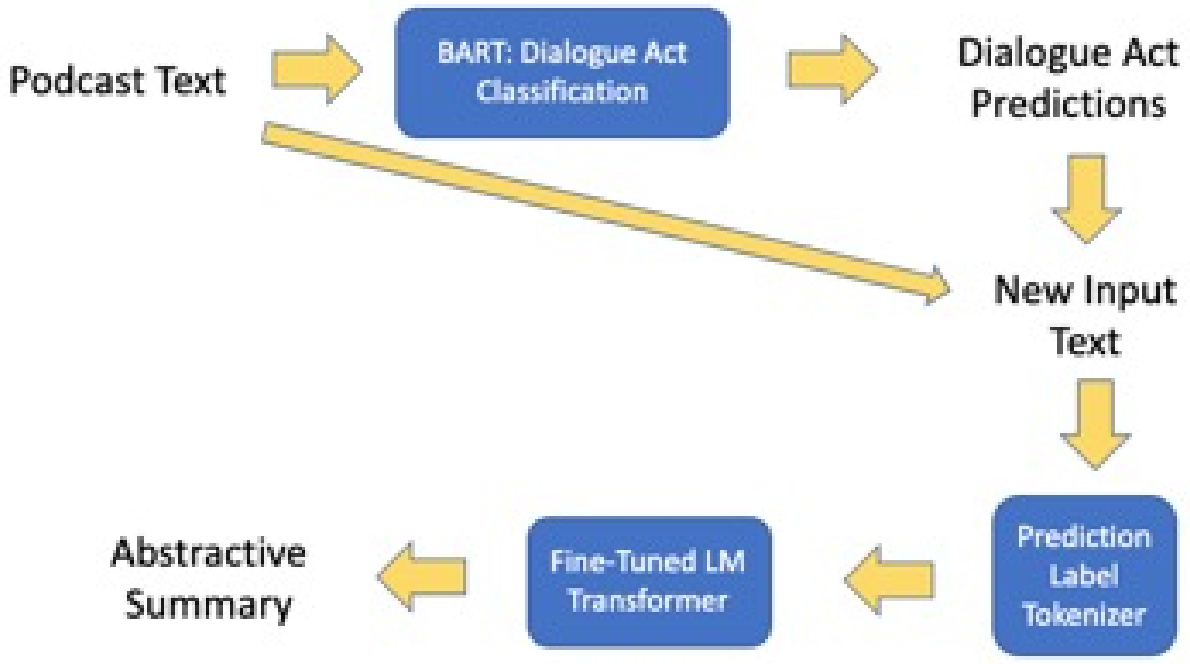

Figure 3.1. Summary of Dialogue Act Tokenization process

We find the best ROUGE performance with Dialogue Act Tokenization on top of a pre-trained T5-Base transformer. Example summaries produced by the DAT-T5-Base model can be found in Figure 3.2 with abridged versions of corresponding podcast text. We observe that our methods is capable of recognizing the relevant topics and can identify relevant names or speakers.

1. **Podcast Text**: Hi everyone, Welcome back to skincare Somali A's your favorite bi-weekly skincare podcast. I'm Natalie and I'm Jessica this week. We are talking about what we I like to call the great exfoliation debate because we've got two different points of view and we're going to Duke it out. Mano a mano this week. ... So what is exfoliation you might be asking yourself. Well it is this step in your skincare routine that is taking off all the dead skin cells on your face. And the point of exfoliation is to reveal brighter healthier skin while reducing the size of your pores and that's really how I got into exfoliation a few years ago. ...
   Predicted Summary: In this episode we discuss the importance of exfoliating and the importance of using a more organic formula in the beginning of your skincare routine. We also dive into how we consciously choose the way we exfoliate, which means what you can really do to avoid causing too much irritation

2. **Podcast Text**: Hello and welcome back to talking points. It's Whitney ... and I have a special guest here today. Say hello. Hi. You want to introduce yourself over there. My name is Katherine. I'm excited to be Whitney's first guest. I'm honored. Yay! ... So not only is Catherine our first guest we have an episode that we had talked about a while ago and we're going to talk about transitioning into company life from Student Life. Yes, it's a hard transition. I feel like no matter what your training has been you're never totally prepared for that jump from student to professional just because there's nothing like actually joining a company. Yeah for sure and it's kind of special for Whitney and I to talk about it because we met on our First day as professional dancers we did. Yeah, we started the same year at our company and so six years ago. Yeah, six years crazy six seasons with dance together. ...
   Predicted Summary: In this episode of Talking Points, Katherine and Whitney talk about their experiences of transitioning from being students to professional dancers. they share some of their own personal struggles, as well as a couple of their experiences with a new partner. this episode

Figure 3.2. Example DAT-T5-Base Summaries

CHAPTER 4

PODCAST SEGMENT SEARCH

Information retrieval involves a user query, describing their information needs, and subsequently searching source documents for the relevant information. There are many techniques in information retrieval. Namely, there are four general approaches: Set-theory models, algebraic models, probabilistic models, and feature-based models. Set-theory models employ set operations (e.g. intersection, union, etc.). Algebraic models use linear algebra representations (e.g. vectors, matrices) of queries and documents and calculate a numeric similarity between the representations. Probabilistic models evaluate probabilities across source documents to decide which are most relevant to the query. Feature-based models represent documents as a set of features and either define or learn the best features to use to calculate a relevance score for the document with respect to the query. We will use implementations from all four approaches in our discussion, comparing and combing several along the way.

4.1 Search Data Structures

Many information retrieval models can be easily implemented on small amounts of data using several simple data structures (e.g. dictionaries, arrays). On the other hand, for massive amounts of data, more specialized data structures are required. Namely, an inverted index structure (Zobel and Moffat, 2006) is typically used for search on incredibly large amounts data. At a high-level, an inverted index structure maintains a mapping of a given word to a list of documents which contain the word (usually a list of document ids). This structure means there is more overhead required when adding a document to the index and far less overhead is needed when searching for terms in the documents. As we discussed earlier in the Section 2.2.2 - Dataset Size, our podcast dataset is far too massive for simple processing

and requires specialized data structures. Thus, we use Apache Lucene [1], an inverted index implementation library.

4.2 Unsupervised Search

As noted in the Search Task Insights section, we will only use unsupervised methods. Most of the previously discussed approaches to information retrieval involve unsupervised methods. Specifically, we focus on four document scoring functions to establish baseline results: 1) Boolean, 2) BM25, 3) Lucene Similarity, 4) Language Model – Dirichlet. Boolean similarity is a set-theory model and simply scores documents by counting how many query terms appear in the document. BM25 is a probabilistic model and uses a specific formula involving the frequency of query terms in the document, the inverse-document-frequency of query terms, certain hyper-parameters, (etc.) (Robertson et al., 1995) The specific formula and details of BM25 scoring can be found in its respective paper. The Lucene Similarity is a combination of the Boolean model and Algebraic model. Documents found to have at least one query term (via Boolean Similarity) are then scored by an algebraic model. More specifically, the algebraic model uses a formula involving term-frequency-inverse-document-frequency, vector norms, (etc.). We also leave the exact formula and details to the derivation in the documentation [2]. The Language Model – Dirichlet (LMD) is a probabilistic model. It is a simple unigram model coupled with Bayesian smoothing with a Dirichlet prior as described in (Zhai and Lafferty, 2001).

---

[1] `https://lucene.apache.org/core/`

[2] `https://lucene.apache.org/core/7_4_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html`

4.3 Search Method Building

We implement the four previously described similarities and evaluate their performance via Normalized Discounted Cumulative Gain or NDCG (Järvelin and Kekäläinen, 2002) . NDCG is a measure of quality for a given set of ranked search results with respect to the ideal set of search results (i.e. ground truth) for some query. Under NDCG, we have a query, a set of ideal or ground-truth results for that query, and a set of returned results for the query, which we want to evaluate. Specifically, given n retrieved and ranked search results, we define NDCG as:

$$DCG = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)} \tag{4.1}$$

$$IDCG = \sum_{i=1}^{|IV_n|} \frac{idrel_i}{\log_2(i+1)} \tag{4.2}$$

$$NDCG = \frac{DCG}{IDCG} \tag{4.3}$$

Where rel_i is the relevance score of the ith ranked result in a given set of returned results, $idrel_i$ is the relevance score of the ith ranked result in a given set of ground-truth resutls, (typically $rel_i, idrel_i \in \mathbb{Z}^{0+}$), and IV_n is the ideal vector of ranked ground-truth search results up to position n. NDCG $\in (0.0, 1.0)$ and mainly penalizes highly relevant documents not appearing at the top of returned search results. We use the Average NDCG, across the provided training queries, as our standard performance metric. Further implementation and theoretic details of NDCG evaluation can be found in its documentation [3] and in helpful readings (Croft et al., 2009, Chapter 8) respectively. We successively extend and modify the best performing similarity score. This may not consider higher order interactions between initial scoring functions and later modifications. However, much like our model building for summarization, it does provide an intuitive process to form a scoring function which performs well in practice.

[3] `https://github.com/usnistgov/trec_eval`

Table 4.1. Summary of initial NDCG scores

Method	**Avg. NDCG**
Boolean Similarity	0.0479
Lucene Similarity	0.0605
BM25 Similarity ($b = 0.0, k_1 = 0.25$)	0.1686
LMD Similarity (μ=2750.0)	**0.1771**

4.4 Initial Scoring Functions

We use Lucene implementations of Boolean, BM25, Lucene, and LMD Similarity. Then, we return 100 ranked results for each of the eight training queries provided by Spotify. In Table 4.1, we see the initial Average NDCG results of the four baseline similarity scores across the training queries. Also, for methods requiring hyper-parameters, tuned hyper-parameters values are listed in Table 4.1. We observe that LMD Similarity performs best across the training queries. Thus, we choose to modify and extend LMD Similarity in our following discussion.

4.5 Feature-Based Combination Scoring

We extend the LMD Similarity by combining several scoring methods. The scoring can be thought of as several "rounds", where each round employs a certain similarity score to collect a certain number of documents, improving the overall results. The first similarity score is the LMD similarity, and collects 500 initial documents. We discuss the other similarity methods used, below.

4.5.1 Fuzzy Retrieval

As mentioned in the Search Task Insights section, we must account for the errors in podcast episode transcriptions. Since the transcripts may have errors, we will allow a certain amount "slack" when checking terms in the query against terms in the document. Specifically, we use a combination of Boolean Similarity and Edit Distance (using Damerau-Levenshtein algorithm (Damerau, 1964; Levenshtein, 1966)). Terms in a document having an edit distance (with respect to any query terms) less than or equal to two are considered matching. Documents are scored by the sum of edit distances of the matching terms in the document.

4.5.2 Question Answer Task

We will also extend our search scoring through a Question-Answer model, specifically a transformer. The Question-Answer task considers a given input text and a question, and tries to identify an excerpt of the input text that answers the given question.

Question Answer Transformers

The input text is the question and the reference text paired together. Special separator tokens are also passed so further encoding can recognize which part of the input tokens is the question and which is the reference text. A pass through a transformer for question answering works much like the process described in the Transformer Abstractive Summarization section. However, the decoder outputted floats vector is instead passed to a span-classification head. A span-classification head is a fully-connected feed forward neural network, which computes a span-start logits vector and a span-end logits vector. These vectors assign a score to each token in the input text. The token with the largest span-start logit is predicted as the beginning token of the answer span, and the token with the largest span-end logit is predicted as the end token of the answer span. Hence, the computed answer to the question is the inclusive interval of the input text from the start token to the end token. The transformer

may predict that the answer does not lie in the reference text, and in that case its span is empty; however, its max logits will have a value as an artifact of the neural network calculations. We use a BERT model (Devlin et al., 2018), fine-tuned over the SQUAD dataset (Rajpurkar et al., 2018), as our pre-trained transformer.

Question Answer Score

To extend our search score we modify the output of our BERT-QA model. The description component of the queries (See Figure 1.2) is passed in as the question component and a given document is passed in as the reference text. We take the max score of both the start and end span logit vectors. We sum the absolute values of these scores. If the predicted span is of non-zero length, we boost this sum by 100 (exact boost value can be fine-tuned over experiments). Then, we pass these sums through a softmax function, producing a probability distribution over the documents. We then can use these probabilities in our final document score.

4.5.3 Final Scoring Function

We now describe the final combination of previously discussed scoring methods. To ensure reasonable turn-around time, we collect an increasingly smaller number of documents as we progress through more scoring methods. Recall from the Search Task description section, that search queries have the form seen in Figure 1.2. The first round of scoring collects 500 initial documents using the simple LMD similarity and the "query" component. Then we rescore the initial documents by adding a Fuzzy Retrieval score over the "query" component, accounting for minor transcription errors. We take the top 400 documents after this rescoring. The third round adds a LMD similarity, using the "description" component, to the total score; the top 100 documents are kept. The total score is finished by adding in the Question Answer Score probability and we reorder the top 100 documents accordingly. Moreover, we

Table 4.2. Summary of final NDCG scores

Method	**Avg. NDCG**
LMD Similarity on Query	0.1771
LMD Similarity on Query + LMD Similarity on Description	0.2798
LMD Similarity on Query + LMD Similarity on Description + Fuzzy Query Rescore	0.3021
LMD Similarity on Query + LMD Similarity on Description + Fuzzy Query Rescore + BERT-QA Score	**0.3022**

For all methods above, the following hyperparameter values (where appropriate) are: $\mu = 2750.0$, $\lambda_1 = 375.0$, $\lambda_2 = 3.0$, $\lambda_3 = 20.0, \omega = 100.0$

add scaling factors between rounds as hyper-parameters to be tuned. We present the final score and hyper-parameters for a document as:

$$DocScore = LMD_{query} + (\lambda_1 * Fuzzy_{query}) + (\lambda_2 * LMD_{description}) + (\lambda_3 * softmax((\omega * isQASpanNonEmpty) + (|startLogit| + |endLogit|))) \quad (4.4)$$

The three λ values are the inter-round hyper-parameters. $isQASpanNonEmpty$ is a boolean variable indicating non-empty spans and ω is the (non-empty-span) boost hyper-parameter, described in the Question Answer Score section. In Table 4.2, we can see the performance of the successive scoring rounds and our final scoring method.

We observe that our final scoring method improves upon all the standard baseline methods presented in Table 4.1. However, we also want to know if the hyper-parameters tuned on training data caused any significant overfitting. In the absence of enough data for a training-test split, we do a sanity check over approximately fifty test queries provided by Spotify, which have no available ground-truth results. We simply check the top few results returned by our final scoring method and ensure that they are relevant to the query. Some examples from this check can be found in Figure 4.1, where we present queries and the corresponding search results returned by our custom scoring method. We conclude from the test

queries provided, that no significant overfitting has occurred. For far more abstract queries, where the user provides a very vague idea of what to search for, the results returned are not very relevant. However, in aggregate, the final scoring method suffices for much of the test queries.

- **Query**: slow travel
  Description: What is "slow travel"? I heard about this approach to traveling the world recently when reading about sustainable living, and I'd like to hear more about it. What does it have to do with sustainability?
  Results:

  1. ... However, if we change our mindset traveling can become so much more than I'm simply going from A to B. ... Slow cannot only be taken literally but also stands for sustainable local organic whole again sustainable local organic whole slow travel drawers on the ideas from the slow food movement with a concern for locality ecology and quality of life it aims to reduce its high environmental impact and major contributions to climate change of traveling in essence. ...
  2. ... Shorter distances and enriching the travel experience both on route to and at the destination your script off was an advocate of the concept even before the concept was named switch off the time machine take off the watch get rid of time pressure. ... Culture through language and engagement with local people makes for better holiday seeking out the unexpected giving back to local communities. ...

- **Query**: social media marketing
  Description: I'm looking for tips and suggestions for creating a social media marketing strategy. What kinds of things do I need to think about? How can I reach the most people online? Are there companies that can help?
  Results:

  1. ... They can do paid social content development Community Management and analytics and ROI analysis the most common solutions that agencies provide are with social media management and paid social but truly whatever you can dream up for social media and agency can help make it happen. We mentioned the budget question of my mirror go. Do you have extra budget for social media and aren't sure how to use it. ... So now that you have a sense for whether or not you need an an agency, let's talk about the ways to find a fantastic one. ...
  2. Social media for business a marketer'ss guide your small business needs to be on social media. If you aren't you're missing out on cheap and effective marketing tools with almost 70 percent of Americans on social media. Your social media campaigns can reach a lot of people social media provides targeting capability as well as reach and scale at a lower cost ... Facebook is still the most popular social media Network and every business it should have a Facebook page when used correctly a Facebook page can be invaluable to small businesses. ...

Figure 4.1. Example search queries and top two results

CHAPTER 5

CONCLUSION

Podcast data has posed several problems and idiosyncrasies which diverge from many existing corpora types. Podcast episodes may have multiple speakers, inconsistent transcription quality, (etc.). We have provided viable methods in text summarization and information retrieval specifically for podcast data. In summarization, we extend the recent transformer-based transfer learning approach through a dialogue-act-prediction tokenization scheme. In information retrieval (i.e. search), we combine the power of general language representation transformers like BERT, pre-trained on the Question-Answer task, with traditional information retrieval similarity measures.

We observe, from the positive results of our methods, that even though podcast corpora are quite novel and irregular, existing methods can still provide a viable basis to expound on. More specifically, embedding corpora-level understanding via preliminary filtering, special tokens, and custom scoring functions emerge as working extensions. Furthermore, we observe that transformers serve as a powerful tool in this unexplored data space, with perceptive self-attention mechanisms and many helpful pre-trained varieties (e.g. Question-Answer and Summarization tasks).

5.1 Further Work

There are several areas of improvement that can be fulfilled further from the current work presented. Concerning the summarization task, we use dialogue act tokenization with a BART-Sequence Classification model. We elected to use a BART-based model since it had especially quick training time, simplicity of implementation, and near state-of-the-art performance. Clearly, more complex models (such as those in (Ravi and Kozareva, 2018; Li et al., 2018)) could also serve the dialogue act prediction purpose with higher performance. Perhaps, with better dialogue act prediction performance, better summarization performance

may follow under our process. Moreover, as pre-trained transformers improve and become more numerous, there will be more candidate transformers to pair with dialogue act tokenization and potentially improve summarization. More exhaustive surveying of transformer methods to pair with dialogue act tokenization could also prove helpful in understanding the generalizability of DAT. Certainly, as podcast data itself becomes more available and of higher quality, models trained on that data can also have improved performance for many tasks, including summarization. For the search task, a more extensive review of information retrieval methods may find a better similarity measure to pair with the Question Answer Score and improve search results. Also, the inclusion of transformers in information retrieval can also grow as transformers become more ubiquitous, perhaps improving QA task performance and hence search performance. Moreover, Spotify may also provide more and more query training data over time, allowing for QA-Transformer fine-tuning over podcast queries, and hence better search performance in the data space.

REFERENCES

Cho, K., B. van Merrienboer, D. Bahdanau, and Y. Bengio (2014). On the properties of neural machine translation: Encoder-decoder approaches.

Clifton, A., A. Pappu, S. Reddy, Y. Yu, J. Karlgren, B. Carterette, and R. Jones (2020). The spotify podcasts dataset.

Croft, B., D. Metzler, and T. Strohman (2009). *Search Engines: Information Retrieval in Practice* (1st ed.)., Chapter 8, pp. 297–333. USA: Addison-Wesley Publishing Company.

Damerau, F. J. (1964, March). A technique for computer detection and correction of spelling errors. *Commun. ACM 7*(3), 171–176.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

Fan, A., M. Lewis, and Y. Dauphin (2018). Hierarchical neural story generation.

Hermann, K. M., T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom (2015). Teaching machines to read and comprehend.

Hochreiter, S. and J. Schmidhuber (1997, November). Long short-term memory. *Neural Comput. 9*(8), 1735–1780.

Holtzman, A., J. Buys, L. Du, M. Forbes, and Y. Choi (2019). The curious case of neural text degeneration.

Howard, J. and S. Ruder (2018). Universal language model fine-tuning for text classification.

Järvelin, K. and J. Kekäläinen (2002, October). Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst. 20*(4), 422–446.

Jurafsky, D. and E. Shriberg (1997). Switchboard swbd-damsl shallow-discourse-function annotation coders manual.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet physics. Doklady 10*, 707–710.

Lewis, M., Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

Li, R., C. Lin, M. Collinson, X. Li, and G. Chen (2018). A dual-attention hierarchical recurrent neural network for dialogue act classification.

Lin, C.-Y. (2004, July). ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, Barcelona, Spain, pp. 74–81. Association for Computational Linguistics.

Liu, Y. and M. Lapata (2019). Text summarization with pretrained encoders.

Mihalcea, R. and P. Tarau (2004, July). TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, pp. 404–411. Association for Computational Linguistics.

Paulus, R., C. Xiong, and R. Socher (2017). A deep reinforced model for abstractive summarization.

Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer (2018). Deep contextualized word representations.

Radford, A. (2018). Improving language understanding by generative pre-training.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever (2019). Language models are unsupervised multitask learners.

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu (2019). Exploring the limits of transfer learning with a unified text-to-text transformer.

Rajpurkar, P., R. Jia, and P. Liang (2018). Know what you don't know: Unanswerable questions for squad.

Ravi, S. and Z. Kozareva (2018, October-November). Self-governing neural networks for on-device short text classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, pp. 887–893. Association for Computational Linguistics.

Robertson, S., S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford (1995, January). Okapi at trec-3. In *Overview of the Third Text REtrieval Conference (TREC-3)* (Overview of the Third Text REtrieval Conference (TREC–3) ed.)., pp. 109–126. Gaithersburg, MD: NIST.

Rudra, K., S. Banerjee, N. Ganguly, P. Goyal, M. Imran, and P. Mitra (2016). Summarizing situational tweets in crisis scenario. In *Proceedings of the 27th ACM Conference on Hypertext and Social Media*, HT '16, New York, NY, USA, pp. 137–147. Association for Computing Machinery.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). Attention is all you need.

Wolf, T., L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush (2019). Huggingface's transformers: State-of-the-art natural language processing.

Zhai, C. and J. Lafferty (2001). A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, New York, NY, USA, pp. 334–342. Association for Computing Machinery.

Zobel, J. and A. Moffat (2006, July). Inverted files for text search engines. *ACM Comput. Surv. 38*(2), 6–es.

BIOGRAPHICAL SKETCH

Mathew Perez began studying Computer Science in the Fall of 2017, while earning his BBA in Management Information Systems and certificate in Computer Science at The University of Texas at Austin. During that time, Mathew also worked as an Undergraduate Research Assistant for Dr. Rajiv Garg, processing image data for deep learning applications and creating social data science survey software. In the summers of 2018 and 2019, Mathew interned as a Data Engineer at Facebook in Menlo Park, CA. There he worked on writing big data software and designing efficient ETL processes while collaborating with Mobile-Platform Software Engineers and Machine Learning Engineers. In the Fall of 2019, Mathew began his MS in Computer Science at The University of Texas at Dallas, where he has continued pursuing research in the Computer Science field.

CURRICULUM VITAE

Mathew Perez

September 24, 2020

Contact Information:

Email: mjp160130@utdallas.edu
Website: http://www.mathewperez.net

Educational History:

B.B.A., Management Information Systems, The University of Texas at Austin, 2019

Employment History:

Data Engineer Intern, Facebook, May 2019 – August 2019
Undergaduate Research Assistant, The University of Texas at Austin, February 2019 – May 2019
Data Analytics Intern, Facebook, June 2018 – August 2018

Recognitions and Honors:

Graduated with High Honors, The University of Texas at Austin, 2019