Podcast Topic Segmentation

Natural Language Processing

Group 9

Matthew McMurry, Lizabeth Singh, Eunji Kim, Hongcen Tu, Yijiao Zuo & Yao Chen

Abstract

Podcasts are one of the most popular mediums worldwide for information consumption and entertainment. However, as podcasts have grown in popularity, the ability to locate topics within and across podcasts has remained crude at best. This study aims to address this issue using Natural Language Processing (NLP) techniques so that podcasts can be easily sorted and segmented to target specific keywords and output meaningful sections of audio that correspond to what a user wants to hear. Using a speech-to-text API to obtain text from audio and segmenting the audio through custom code, we implement latent dirichlet allocation (LDA), cosine similarity, and sentiment scoring methods to facilitate comparison of excerpts of audio segments that can be defined by keywords input by a user.

Introduction

Currently, long-form audio and podcast discovery is rudimentary and specific content within podcast episodes is difficult to find. To solve this problem, we wanted to use NLP driven podcast analysis to facilitate search and discovery of relevant discussions. We wanted to contextualize our issue and ensure that this study wasn't duplicating previous attempts, so we looked into what types of methodologies were pursued previously in prior literature and research. However, while we found that much research has been done with regards to topic segmentation, no studies contained a solution to our specific problem as it pertains to long-form audio. One relevant article describes a "scalable video search engine based on audio content indexing and topic segmentation," which produced a discovery tool that yielded desired results, but with videos rather than strictly audio content *(Lawto et al., 2011)*. The methodology was also not

explained in depth but only briefly mentioned the use of GATE and UIMA open source frameworks.

Another article demonstrated usage of audio embedding generators using multi-class classification tasks on radio shows *(Berlage et al., 2020)*, which would be helpful for more nuanced segmenting between podcast sections within an episode, but not particularly useful in our task of only segmenting the topics. It pertained to intermittent and transitory sound effects. An article from 2017 portrayed successful implementation of bidirectional Recurrent Neural Networks which showed better accuracy and precision than other segmentation methods *(Sehikh et al., 2018)*. However, due to time constraints, we did not pursue this option. The most recent article that we found regarding podcasts specifically was an article titled "Identifying Introductions in Podcast Episodes from Automatically Generated Transcripts," in which a team set out to identify introductions of podcasts, but not topics within them *(Jing et al., 2021)*.

Hence, we decided to use the methodology we were most familiar with such as Latent Dirichlet Allocation (LDA) and cosine similarity. We also incorporated CountVectorizer, tf-idf, and sentiment scoring techniques into our analysis. As these methods are some of the most popular in natural language processing, we thought these would be sufficient for our purposes of segmentation and topic comparison.

Data Collection

The data collection process entailed using iTunes Search to locate podcast episodes that had the term "2022 Oscars" either in their description or title. Our reasons for settling on the Oscars as a subject was due to its relevance during the time of our data collection. It was widely discussed because of its prominence as a pop culture event and also because of an incident that

occurred during the ceremony. The incident entailed Will Smith assaulting Chris Rock, the host of the Oscars this year, after he made a comment about Mr. Smith's wife. Overall, this became a topic of debate hence this event became controversial and many people had a wide range of opinions on it. Therefore our primary objective became to segment discussions of the Will Smith and Chris Rock altercation from within the full podcast episodes and study the variations and the similarities in how these discussions unfolded in different podcasts. The process of collecting specific podcast episodes for our study again highlighted the issue at hand which is that it is currently difficult to search within podcasts to find particular subjects being discussed.

Overall, we downloaded 20 podcast episodes in total, with 15 that were relevant recap episodes of the Oscars, and 5 that contained no discussion of the Oscars and which were included for control and comparison purposes. Of these 5 non-Oscar podcasts, three related to the Grammy Awards, one related to Bitcoin, and one related to AirBnB. After downloading the podcast episodes in .mp3 format, we used the IBM Watson Speech-to-Text API in order to generate transcripts with which to perform our analysis. This produced 20 corpora for us to use in our analysis, equating to one full transcript per podcast episode.

Although 20 episodes may seem like a small amount, for our purposes this was a sufficient amount of data, as each episode was about an hour long and contained a substantial amount of text data. Ultimately this was plenty of data in order to test our methods, theories, and run our comparative analysis. In general, this study should be seen in the light of a prototype in order to use what we found to solve the main objective.

Methodology

Before running our analysis, we needed to account for the problem of how to intelligently segment portions of the full transcript based on topic. Topic identification and segmentation is a complicated task in theory and there are many ways to approach this, including using advanced machine learning methods to extract topics from within a large body of text. However, one rather simple approach which worked fairly well for us was to use keywords to locate and define segments. Specifically, we wrote a function in Python that took in a keyword or phrase, and if the keyword or phrase appeared in the transcript, it would return a sorted list of timestamp objects of every instance where that word or phrase occurred along with the index of each occurence. While the timestamps themselves aren't necessary for extracting excerpts, they are tremendously useful for directing users to relevant portions within the podcasts, which is our stated end goal. To produce our segments we combined keyword searches for "Will Smith", "Chris Rock" and "slap", since any text that contained these keywords almost certainly featured a discussion of the incident in question. The result of this combined search was a list of sorted index locations and timestamps of where each word was uttered, and once equipped with this information, we extracted the relevant excerpts from the larger corpus by setting boundaries of 15 words prior to the first keyword and 15 words after the appearance of the last keyword. For the non-Oscars podcasts, since they contained no discussion of the incident in question, we arbitrarily extracted 650 word segments from each corpus for comparison with the Oscars-related excerpts.

Once excerpts were generated, we needed to preprocess our data to ensure that we would be able to perform our analysis efficiently and correctly. We gathered all the data into a single dataframe, and then used helper functions to remove irrelevant symbols, stop words such as

“the,” “an,” “a,” etc., as well as word stemming which allows for better performance of word tokenization.

The methods of topic segmentation we used for this project were LDA, Cosine Similarity, CountVectorizer and term frequency-inverse document frequency (tf-idf). We chose to incorporate LDA because LDA yields better disambiguation of words and a more precise assignment of documents to topics, which was a primary goal of this analysis, to be able to tag specific podcasts to a category of words that are frequently used. We also used Cosine Similarity as another means of comparing how closely related two segments are to one another.

Results

LDA

After preprocessing, we first ran the data on an LDA model and found that the optimal number of topics is 9. We can see on the elbow curve that the relationship between the coherence score and the number of topics is sporadic(Figure 1). The final score was 0.28; although this is a fairly low number, we must take into account that this is only a prototype analysis, and with more data, this score could increase. Taking a look at the equations below(Figure2) as well as the world cloud displayed below(Figure 3) we can see the most common words found within the different podcasts such as “chris,” “rock,” “think,” “talk,” “smith,” and “like.” These commonalities were not surprising, as they were words that were relevant to the topics that we were aware of being within the podcast episode content.

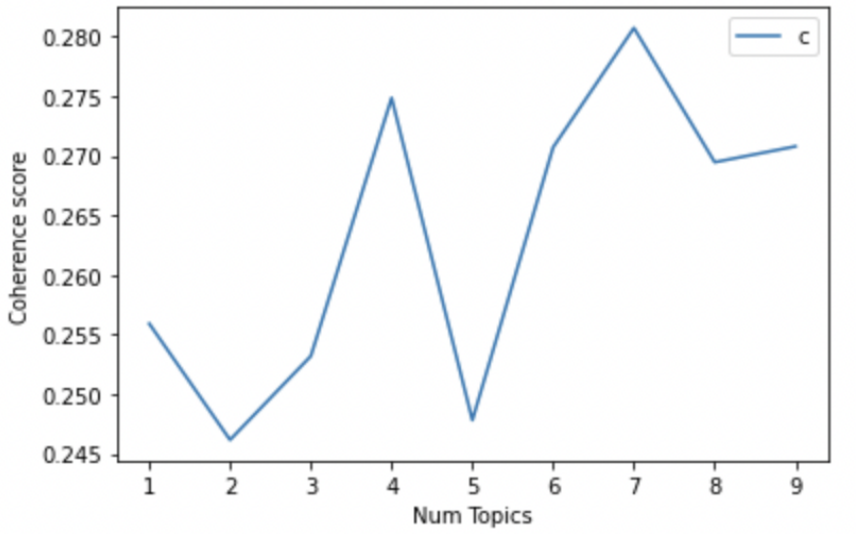


Figure 1: Elbow curve for LDA model

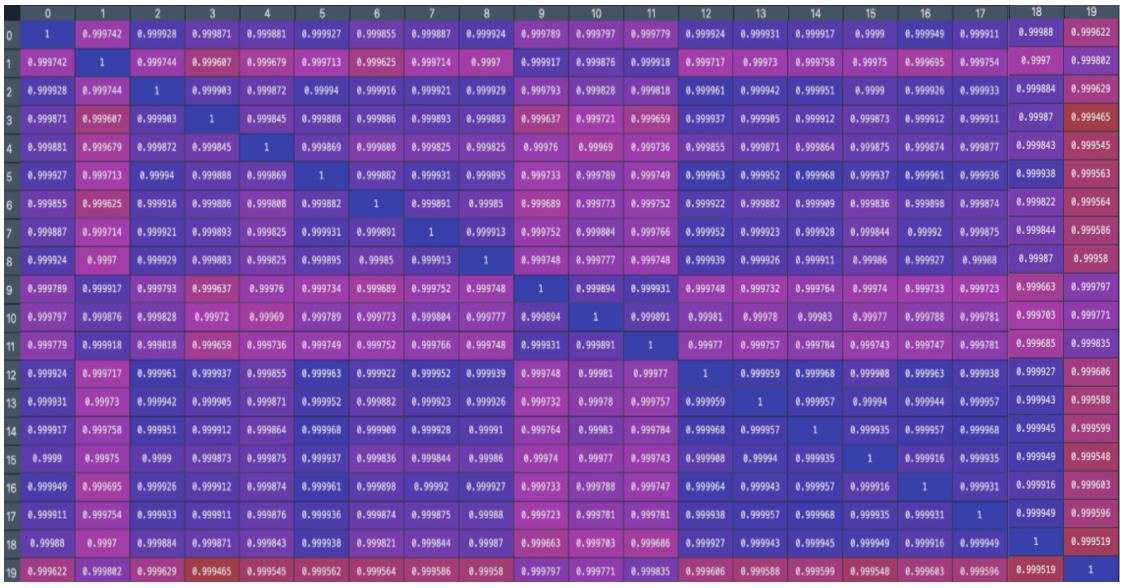
```
(0, '0.013*"know" + 0.013*"go" + 0.013*"get" + 0.013*"chri"')
(1, '0.001*"know" + 0.001*"like" + 0.001*"chri" + 0.001*"got"')
(2, '0.024*"know" + 0.020*"like" + 0.015*"think" + 0.014*"go"')
(3, '0.063*"n" + 0.011*"energi" + 0.011*"twenti" + 0.011*"climat"')
(4, '0.074*"like" + 0.028*"know" + 0.023*"yeah" + 0.021*"think"')
(5, '0.019*"like" + 0.016*"right" + 0.014*"know" + 0.013*"rock"')
(6, '0.014*"know" + 0.013*"yeah" + 0.013*"got" + 0.011*"go"')
(7, '0.001*"n" + 0.001*"know" + 0.001*"like" + 0.001*"get"')
(8, '0.001*"n" + 0.001*"like" + 0.001*"know" + 0.001*"go"')
(9, '0.142*"n" + 0.015*"like" + 0.010*"one" + 0.008*"talk"')
```

Figure 2: Topic Extraction for LDA(Main topics and keywords)

Figure 3: Word Cloud for LDA model

A common word that was originally in this group was the word “hesitation” which is an API-generated default word used for when speakers take a break or pause from talking. Therefore, it was removed after running the LDA Model the first time and is no longer in the analysis. After looking at these results, something to consider is the difference between written text opposed to spoken word that is transformed into text. Hence the word “like” coming up multiple times in this discriminative analysis. Humans use the word “like” frequently when speaking, and a way to avoid this would be to create a new dictionary of stop words that removes common conversational filler words, in order to get a more accurate depiction of the importance of non-filler words.

Cosine Similarity



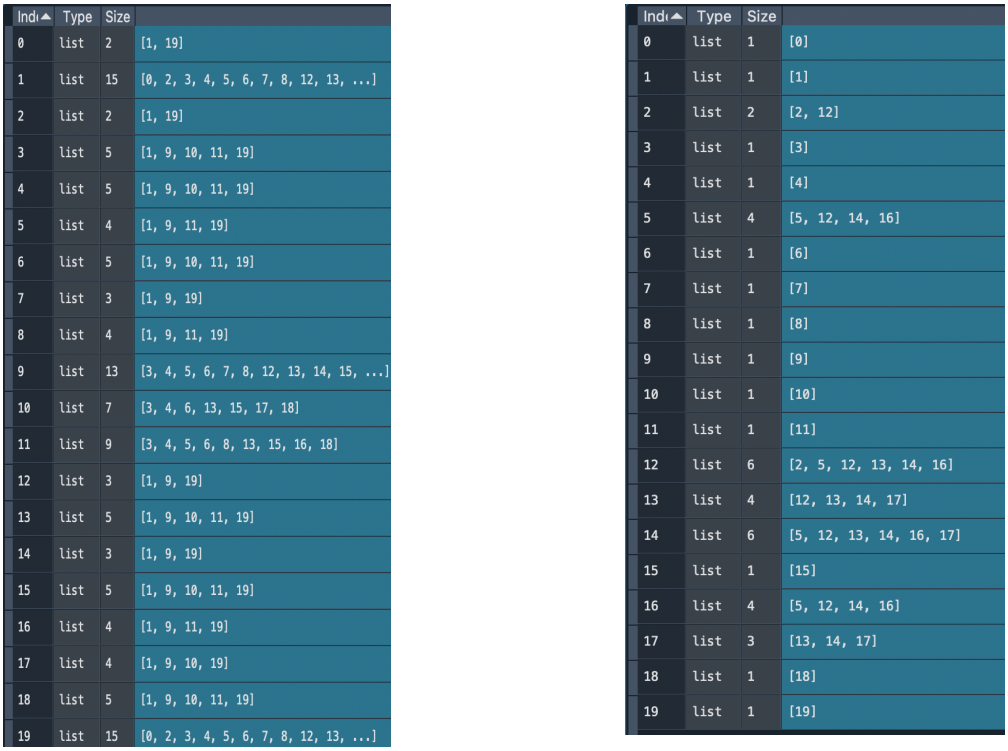
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	0.999742	0.999928	0.999871	0.999881	0.999927	0.999855	0.999887	0.999924	0.999789	0.999797	0.999779	0.999924	0.999931	0.999917	0.9999	0.999949	0.999911	0.99988	0.999622
1	0.999742	1	0.999744	0.999607	0.999679	0.999713	0.999625	0.999714	0.9997	0.999917	0.999876	0.999918	0.999717	0.99973	0.999758	0.99975	0.999695	0.999754	0.9997	0.999802
2	0.999928	0.999744	1	0.999903	0.999872	0.99994	0.999916	0.999921	0.999929	0.999793	0.999828	0.999818	0.999961	0.999942	0.999951	0.9999	0.999926	0.999933	0.999884	0.999629
3	0.999871	0.999607	0.999903	1	0.999845	0.999888	0.999886	0.999893	0.999883	0.999637	0.999721	0.999659	0.999937	0.999905	0.999912	0.999873	0.999912	0.999911	0.99987	0.999465
4	0.999881	0.999679	0.999872	0.999845	1	0.999869	0.999888	0.999825	0.999825	0.99976	0.99969	0.999736	0.999855	0.999871	0.999864	0.999875	0.999874	0.999877	0.999843	0.999545
5	0.999927	0.999713	0.99994	0.999888	0.999869	1	0.999882	0.999931	0.999895	0.999733	0.999789	0.999749	0.999963	0.999952	0.999968	0.999937	0.999961	0.999936	0.999938	0.999563
6	0.999855	0.999625	0.999916	0.999886	0.999888	0.999882	1	0.999891	0.99985	0.999689	0.999773	0.999752	0.999922	0.999882	0.999909	0.999836	0.999898	0.999874	0.999822	0.999564
7	0.999887	0.999714	0.999921	0.999893	0.999825	0.999931	0.999891	1	0.999913	0.999752	0.999884	0.999766	0.999952	0.999923	0.999928	0.999844	0.99992	0.999875	0.999844	0.999586
8	0.999924	0.9997	0.999929	0.999883	0.999825	0.999895	0.99985	0.999913	1	0.999748	0.999777	0.999748	0.999939	0.999926	0.999911	0.99986	0.999927	0.99988	0.99987	0.99958
9	0.999789	0.999917	0.999793	0.999637	0.99976	0.999734	0.999689	0.999752	0.999748	1	0.999894	0.999931	0.999748	0.999732	0.999764	0.99974	0.999733	0.999723	0.999663	0.999797
10	0.999797	0.999876	0.999828	0.99972	0.99969	0.999789	0.999773	0.999804	0.999777	0.999894	1	0.999891	0.99981	0.99978	0.99983	0.99977	0.999788	0.999781	0.999703	0.999771
11	0.999779	0.999818	0.999818	0.999659	0.999736	0.999749	0.999752	0.999766	0.999748	0.999931	0.999891	1	0.99977	0.999757	0.999784	0.999743	0.999747	0.999781	0.999685	0.999835
12	0.999924	0.999717	0.999961	0.999937	0.999855	0.999963	0.999922	0.999952	0.999939	0.999748	0.99981	0.99977	1	0.999959	0.999968	0.999908	0.999963	0.999938	0.999927	0.999686
13	0.999931	0.99973	0.999942	0.999905	0.999871	0.999952	0.999882	0.999923	0.999926	0.999732	0.99978	0.999757	0.999959	1	0.999957	0.99994	0.999944	0.999957	0.999943	0.999588
14	0.999917	0.999758	0.999951	0.999912	0.999864	0.999968	0.999909	0.999928	0.99991	0.999764	0.99983	0.999784	0.999968	0.999957	1	0.999935	0.999957	0.999968	0.999945	0.999599
15	0.9999	0.99975	0.9999	0.999873	0.999875	0.999937	0.999836	0.999844	0.99986	0.99974	0.99977	0.999743	0.999908	0.99994	0.999935	1	0.999916	0.999935	0.999949	0.999548
16	0.999949	0.999695	0.999926	0.999912	0.999874	0.999961	0.999898	0.99992	0.999927	0.999733	0.999788	0.999747	0.999964	0.999943	0.999957	0.999916	1	0.999931	0.999916	0.999683
17	0.999911	0.999754	0.999933	0.999911	0.999876	0.999936	0.999874	0.999875	0.99988	0.999723	0.999781	0.999781	0.999938	0.999957	0.999968	0.999935	0.999931	1	0.999949	0.999596
18	0.99988	0.9997	0.999884	0.999871	0.999843	0.999938	0.999821	0.999844	0.99987	0.999663	0.999703	0.999686	0.999927	0.999943	0.999945	0.999949	0.999916	0.999949	1	0.999519
19	0.999622	0.999802	0.999629	0.999465	0.999545	0.999562	0.999564	0.999586	0.99958	0.999797	0.999771	0.999835	0.999606	0.999588	0.999599	0.999548	0.999683	0.999596	0.999519	1

Figure 4: cosine similarity result for segmented podcasts;

For this result table (Figure 4), pinker indicates lower score, and bluer indicates higher score. By observing the result table, we can say that documents 1, 9, 10, 11, 19 have lower similarity scores compared to almost all other podcasts.

We first deployed word embedding, specifically Word2Vec, to vectorize each corpus, where it assigns each unique word to a corresponding vector in vector space representation. As cosine similarity compares every document with all other documents, and since we had 20 podcast episodes, the result table is a 20x20 matrix, with 1 as the diagonal (Figure 4).

The five documents 1, 9, 10, 11, and 19, are the ones that pertained to the Grammys, Airbnb, and Bitcoin, which are essentially unrelated to the rest of the podcasts regarding the Oscars. This aligns with the results we expected. However, one limitation we found was that almost all of the scores were high, greater than 0.99, which may lead to confusion about similarity.



Index	Type	Size	
0	list	2	[1, 19]
1	list	15	[0, 2, 3, 4, 5, 6, 7, 8, 12, 13, ...]
2	list	2	[1, 19]
3	list	5	[1, 9, 10, 11, 19]
4	list	5	[1, 9, 10, 11, 19]
5	list	4	[1, 9, 11, 19]
6	list	5	[1, 9, 10, 11, 19]
7	list	3	[1, 9, 19]
8	list	4	[1, 9, 11, 19]
9	list	13	[3, 4, 5, 6, 7, 8, 12, 13, 14, 15, ...]
10	list	7	[3, 4, 6, 13, 15, 17, 18]
11	list	9	[3, 4, 5, 6, 8, 13, 15, 16, 18]
12	list	3	[1, 9, 19]
13	list	5	[1, 9, 10, 11, 19]
14	list	3	[1, 9, 19]
15	list	5	[1, 9, 10, 11, 19]
16	list	4	[1, 9, 11, 19]
17	list	4	[1, 9, 10, 19]
18	list	5	[1, 9, 10, 11, 19]
19	list	15	[0, 2, 3, 4, 5, 6, 7, 8, 12, 13, ...]

Index	Type	Size	
0	list	1	[0]
1	list	1	[1]
2	list	2	[2, 12]
3	list	1	[3]
4	list	1	[4]
5	list	4	[5, 12, 14, 16]
6	list	1	[6]
7	list	1	[7]
8	list	1	[8]
9	list	1	[9]
10	list	1	[10]
11	list	1	[11]
12	list	6	[2, 5, 12, 13, 14, 16]
13	list	4	[12, 13, 14, 17]
14	list	6	[5, 12, 13, 14, 16, 17]
15	list	1	[15]
16	list	4	[5, 12, 14, 16]
17	list	3	[13, 14, 17]
18	list	1	[18]
19	list	1	[19]

Figure 5: dissimilar podcasts on left, similar podcasts on right

To clearly observe how each document is similar or dissimilar with each other, we set a threshold to compare each podcast episode with the rest of the podcast episodes (Figure 5). Using the threshold of a similarity score less than 0.99975, we got 19 lists of podcasts indicating which have lower similarity scores compared to each of the 19 segmented podcasts we have. It turned out that documents 1, 9, 10, 11, 19 have comparatively lower similarity scores when compared to the rest of the podcast episodes. Moreover, using the threshold of a similarity score greater than 0.99995, we observed lists of podcast episodes that were very similar, such as documents 2, 5, 12, 13, 14, 16.

Figure 6: word cloud for similar podcasts found using cosine similarity

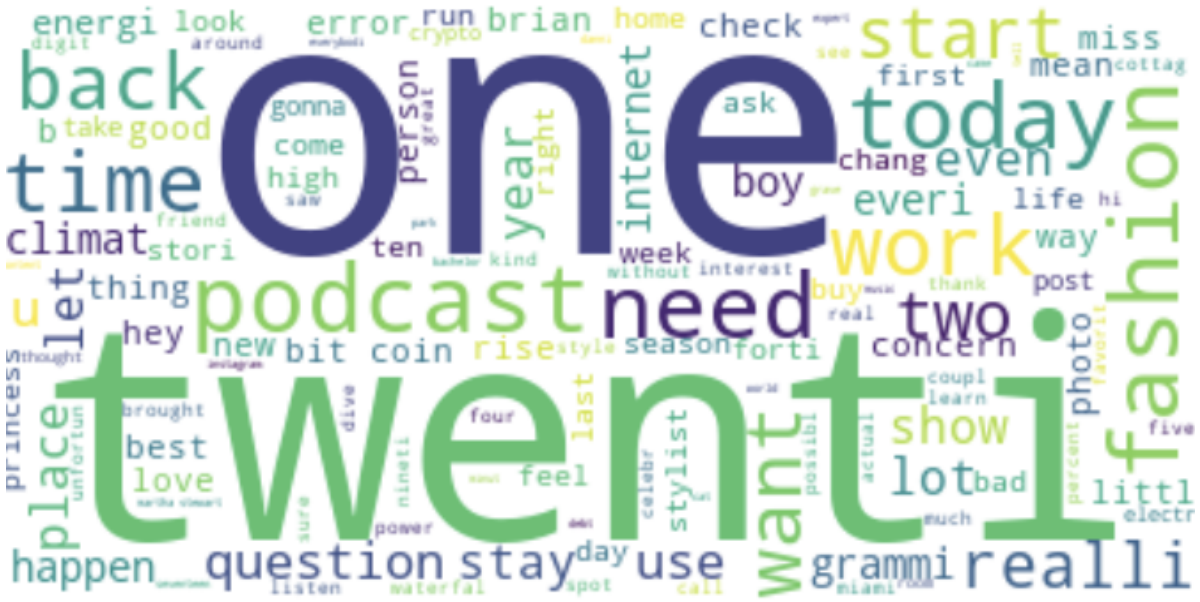
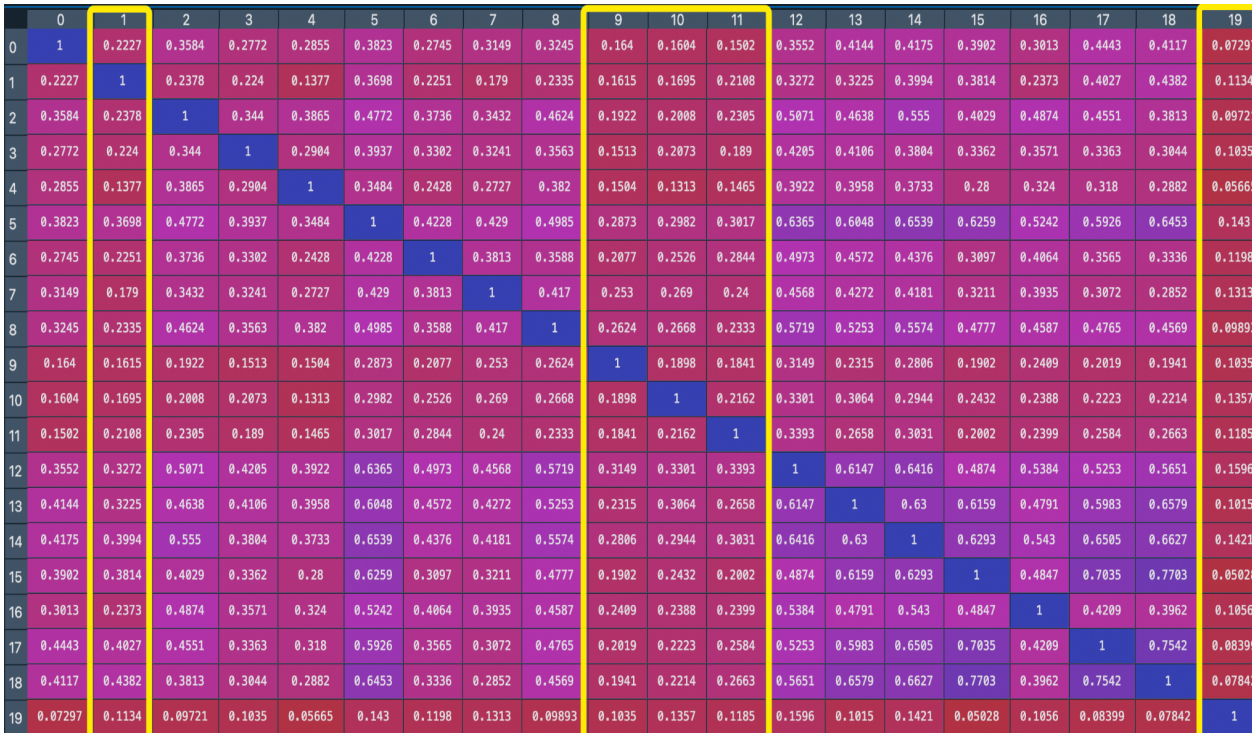


Figure 7: word cloud for dissimilar podcasts found using cosine similarity

In order to investigate and visualize the difference between similar and dissimilar podcasts, we applied Word Cloud to visualize what the most frequent words used were in similar and dissimilar podcast episodes. Bigger words implies a higher frequency within those lists of podcast episodes. Figures 6 and 7 clearly demonstrate that, putting all similar podcasts together, words like “people”, “realli”, “right”, “jock”, “smith”, “chris”, “rock”, “slap”, and “oscar” are frequently used. In the group of dissimilar podcasts, the words “one”, “twenti”, “grammi”, “bitcoin”, “place”, “climate”, and “stay” are demonstrated as the most frequently used, which is again an expected result.

CountVectorizer



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	0.2227	0.3584	0.2772	0.2855	0.3823	0.2745	0.3149	0.3245	0.164	0.1604	0.1502	0.3552	0.4144	0.4175	0.3902	0.3013	0.4443	0.4117	0.0729
1	0.2227	1	0.2378	0.224	0.1377	0.3698	0.2251	0.179	0.2335	0.1615	0.1695	0.2108	0.3272	0.3225	0.3994	0.3814	0.2373	0.4027	0.4382	0.1134
2	0.3584	0.2378	1	0.344	0.3865	0.4772	0.3736	0.3432	0.4624	0.1922	0.2008	0.2305	0.5071	0.4638	0.555	0.4029	0.4874	0.4551	0.3813	0.0972
3	0.2772	0.224	0.344	1	0.2904	0.3937	0.3302	0.3241	0.3563	0.1513	0.2073	0.189	0.4205	0.4106	0.3804	0.3362	0.3571	0.3363	0.3044	0.1035
4	0.2855	0.1377	0.3865	0.2904	1	0.3484	0.2428	0.2727	0.382	0.1504	0.1313	0.1465	0.3922	0.3958	0.3733	0.28	0.324	0.318	0.2882	0.0566
5	0.3823	0.3698	0.4772	0.3937	0.3484	1	0.4228	0.429	0.4985	0.2873	0.2982	0.3017	0.6365	0.6048	0.6539	0.6259	0.5242	0.5926	0.6453	0.143
6	0.2745	0.2251	0.3736	0.3302	0.2428	0.4228	1	0.3813	0.3588	0.2077	0.2526	0.2844	0.4973	0.4572	0.4376	0.3097	0.4064	0.3565	0.3336	0.1198
7	0.3149	0.179	0.3432	0.3241	0.2727	0.429	0.3813	1	0.417	0.253	0.269	0.24	0.4568	0.4272	0.4181	0.3211	0.3935	0.3072	0.2852	0.1313
8	0.3245	0.2335	0.4624	0.3563	0.382	0.4985	0.3588	0.417	1	0.2624	0.2668	0.2333	0.5719	0.5253	0.5574	0.4777	0.4587	0.4765	0.4569	0.0989
9	0.164	0.1615	0.1922	0.1513	0.1504	0.2873	0.2077	0.253	0.2624	1	0.1898	0.1841	0.3149	0.2315	0.2806	0.1902	0.2409	0.2019	0.1941	0.1035
10	0.1604	0.1695	0.2008	0.2073	0.1313	0.2982	0.2526	0.269	0.2668	0.1898	1	0.2162	0.3301	0.3064	0.2944	0.2432	0.2388	0.2223	0.2214	0.1357
11	0.1502	0.2108	0.2305	0.189	0.1465	0.3017	0.2844	0.24	0.2333	0.1841	0.2162	1	0.3393	0.2658	0.3031	0.2002	0.2399	0.2584	0.2663	0.1185
12	0.3552	0.3272	0.5071	0.4205	0.3922	0.6365	0.4973	0.4568	0.5719	0.3149	0.3301	0.3393	1	0.6147	0.6416	0.4874	0.5384	0.5253	0.5651	0.1596
13	0.4144	0.3225	0.4638	0.4106	0.3958	0.6048	0.4572	0.4272	0.5253	0.2315	0.3064	0.2658	0.6147	1	0.63	0.6159	0.4791	0.5983	0.6579	0.1015
14	0.4175	0.3994	0.555	0.3804	0.3733	0.6539	0.4376	0.4181	0.5574	0.2806	0.2944	0.3031	0.6416	0.63	1	0.6293	0.543	0.6505	0.6627	0.1421
15	0.3902	0.3814	0.4029	0.3362	0.28	0.6259	0.3097	0.3211	0.4777	0.1902	0.2432	0.2002	0.4874	0.6159	0.6293	1	0.4847	0.7035	0.7703	0.0502
16	0.3013	0.2373	0.4874	0.3571	0.324	0.5242	0.4064	0.3935	0.4587	0.2409	0.2388	0.2399	0.5384	0.4791	0.543	0.4847	1	0.4209	0.3962	0.1056
17	0.4443	0.4027	0.4551	0.3363	0.318	0.5926	0.3565	0.3072	0.4765	0.2019	0.2223	0.2584	0.5253	0.5983	0.6505	0.7035	0.4209	1	0.7542	0.0839
18	0.4117	0.4382	0.3813	0.3044	0.2882	0.6453	0.3336	0.2852	0.4569	0.1941	0.2214	0.2663	0.5651	0.6579	0.6627	0.7703	0.3962	0.7542	1	0.0784
19	0.07297	0.1134	0.09721	0.1035	0.05665	0.143	0.1198	0.1313	0.09893	0.1035	0.1357	0.1185	0.1596	0.1015	0.1421	0.05028	0.1056	0.08399	0.07842	1

Figure 8: cosine similarity results after implementing CountVectorizer; yellow parts are podcasts with lower similarity scores

Since the cosine similarity results for the embeddings method all turned out to be greater than 0.999, although we can still sort the most similar and dissimilar podcasts based on that result, we wanted to explore whether we could implement other methods to produce more convincing results. Thus, we utilized CountVectorizer, which transforms text into vectors on the basis of frequency of each word that occurs in the entire text. We used both unigrams and bigrams and then applied cosine similarity to the vectors we calculated.

From the result (Figure 8), we can see that the cosine similarity model performed better and more clearly demonstrates that documents 1, 9, 10, 11, and 19 are dissimilar with respect to the other podcast episodes.

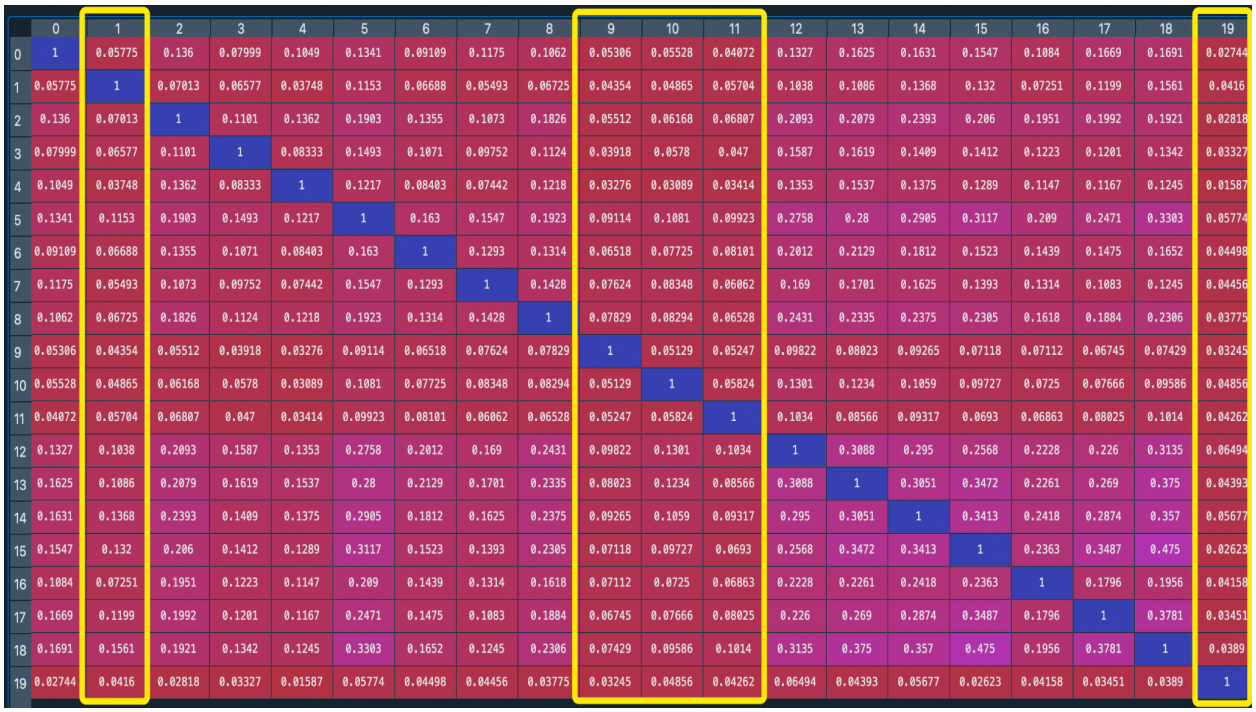
Ke	Type	Size	
0	Array of int64	(5,)	[19 11 10 9 1]
1	Array of int64	(5,)	[19 4 9 10 7]
2	Array of int64	(5,)	[19 9 10 11 1]
3	Array of int64	(5,)	[19 9 11 10 1]
4	Array of int64	(5,)	[19 10 1 11 9]
5	Array of int64	(5,)	[19 9 10 11 4]
6	Array of int64	(5,)	[19 9 1 4 10]
7	Array of int64	(5,)	[19 1 11 9 10]
8	Array of int64	(5,)	[19 11 1 9 10]
9	Array of int64	(5,)	[19 4 3 1 0]
10	Array of int64	(5,)	[4 19 0 1 9]
11	Array of int64	(5,)	[19 4 0 9 3]
12	Array of int64	(5,)	[19 9 1 10 11]
13	Array of int64	(5,)	[19 9 11 10 1]
14	Array of int64	(5,)	[19 9 10 11 4]
15	Array of int64	(5,)	[19 9 11 10 4]
16	Array of int64	(5,)	[19 1 10 11 9]
17	Array of int64	(5,)	[19 9 10 11 7]
18	Array of int64	(5,)	[19 9 10 11 7]
19	Array of int64	(5,)	[15 4 0 18 17]

Ke	Type	Size	
0	Array of int64	(5,)	[0 17 14 13 18]
1	Array of int64	(5,)	[1 18 17 14 15]
2	Array of int64	(5,)	[2 14 12 16 5]
3	Array of int64	(5,)	[3 12 13 5 14]
4	Array of int64	(5,)	[4 13 12 2 8]
5	Array of int64	(5,)	[5 14 18 12 15]
6	Array of int64	(5,)	[6 12 13 14 5]
7	Array of int64	(5,)	[7 12 5 13 14]
8	Array of int64	(5,)	[8 12 14 13 5]
9	Array of int64	(5,)	[9 12 5 14 8]
10	Array of int64	(5,)	[10 12 13 5 14]
11	Array of int64	(5,)	[11 12 14 5 6]
12	Array of int64	(5,)	[12 14 5 13 8]
13	Array of int64	(5,)	[13 18 14 15 12]
14	Array of int64	(5,)	[14 18 5 17 12]
15	Array of int64	(5,)	[15 18 17 14 5]
16	Array of int64	(5,)	[16 14 12 5 2]
17	Array of int64	(5,)	[17 18 15 14 13]
18	Array of int64	(5,)	[18 15 17 14 13]
19	Array of int64	(5,)	[19 12 5 14 10]

Figure 9: cosine similarity results for dissimilar and similar podcasts after using CountVectorizer, left to right

We sorted the five most dissimilar and most similar podcasts for each list based on the results (Figure 9), and were able to show that the five most dissimilar podcasts correspond to Airbnb, Grammys, and Bitcoin, and that all other podcasts are Oscars related.

Tf-idf



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	0.05775	0.136	0.07999	0.1049	0.1341	0.09109	0.1175	0.1062	0.05306	0.05528	0.04072	0.1327	0.1625	0.1631	0.1547	0.1084	0.1669	0.1691	0.02744
1	0.05775	1	0.07013	0.06577	0.03748	0.1153	0.06688	0.05493	0.06725	0.04354	0.04865	0.05704	0.1038	0.1086	0.1368	0.132	0.07251	0.1199	0.1561	0.0416
2	0.136	0.07013	1	0.1101	0.1362	0.1903	0.1355	0.1073	0.1826	0.05512	0.06168	0.06807	0.2093	0.2079	0.2393	0.206	0.1951	0.1992	0.1921	0.02818
3	0.07999	0.06577	0.1101	1	0.08333	0.1493	0.1071	0.09752	0.1124	0.03918	0.0578	0.047	0.1587	0.1619	0.1409	0.1412	0.1223	0.1201	0.1342	0.03327
4	0.1049	0.03748	0.1362	0.08333	1	0.1217	0.08403	0.07442	0.1218	0.03276	0.03089	0.03414	0.1353	0.1537	0.1375	0.1289	0.1147	0.1167	0.1245	0.01587
5	0.1341	0.1153	0.1903	0.1493	0.1217	1	0.163	0.1547	0.1923	0.09114	0.1081	0.09923	0.2758	0.28	0.2905	0.3117	0.209	0.2471	0.3303	0.05774
6	0.09109	0.06688	0.1355	0.1071	0.08403	0.163	1	0.1293	0.1314	0.06518	0.07725	0.08101	0.2012	0.2129	0.1812	0.1523	0.1439	0.1475	0.1652	0.04498
7	0.1175	0.05493	0.1073	0.09752	0.07442	0.1547	0.1293	1	0.1428	0.07624	0.08348	0.06062	0.169	0.1701	0.1625	0.1393	0.1314	0.1083	0.1245	0.04456
8	0.1062	0.06725	0.1826	0.1124	0.1218	0.1923	0.1314	0.1428	1	0.07829	0.08294	0.06528	0.2431	0.2335	0.2375	0.2305	0.1618	0.1884	0.2306	0.03775
9	0.05306	0.04354	0.05512	0.03918	0.03276	0.09114	0.06518	0.07624	0.07829	1	0.05129	0.05247	0.09822	0.08023	0.09265	0.07118	0.07112	0.06745	0.07429	0.03245
10	0.05528	0.04865	0.06168	0.0578	0.03089	0.1081	0.07725	0.08348	0.08294	0.05129	1	0.05824	0.1301	0.1234	0.1059	0.09727	0.0725	0.07666	0.09586	0.04856
11	0.04072	0.05704	0.06807	0.047	0.03414	0.09923	0.08101	0.06062	0.06528	0.05247	0.05824	1	0.1034	0.08566	0.09317	0.0693	0.06863	0.08025	0.1014	0.04262
12	0.1327	0.1038	0.2093	0.1587	0.1353	0.2758	0.2012	0.169	0.2431	0.09822	0.1301	0.1034	1	0.3088	0.295	0.2568	0.2228	0.226	0.3135	0.06494
13	0.1625	0.1086	0.2079	0.1619	0.1537	0.28	0.2129	0.1701	0.2335	0.08023	0.1234	0.08566	0.3088	1	0.3051	0.3472	0.2261	0.269	0.375	0.04393
14	0.1631	0.1368	0.2393	0.1409	0.1375	0.2905	0.1812	0.1625	0.2375	0.09265	0.1059	0.09317	0.295	0.3051	1	0.3413	0.2418	0.2874	0.357	0.05677
15	0.1547	0.132	0.206	0.1412	0.1289	0.3117	0.1523	0.1393	0.2305	0.07118	0.09727	0.0693	0.2568	0.3472	0.3413	1	0.2363	0.3487	0.475	0.02623
16	0.1084	0.07251	0.1951	0.1223	0.1147	0.209	0.1439	0.1314	0.1618	0.07112	0.0725	0.06863	0.2228	0.2261	0.2418	0.2363	1	0.1796	0.1956	0.04158
17	0.1669	0.1199	0.1992	0.1201	0.1167	0.2471	0.1475	0.1083	0.1884	0.06745	0.07666	0.08025	0.226	0.269	0.2874	0.3487	0.1796	1	0.3781	0.03451
18	0.1691	0.1561	0.1921	0.1342	0.1245	0.3303	0.1652	0.1245	0.2306	0.07429	0.09586	0.1014	0.3135	0.375	0.357	0.475	0.1956	0.3781	1	0.0389
19	0.02744	0.0416	0.02818	0.03327	0.01587	0.05774	0.04498	0.04456	0.03775	0.03245	0.04856	0.04262	0.06494	0.04393	0.05677	0.02623	0.04158	0.03451	0.0389	1

Figure 10: cosine similarity result table after implementing tf-idf

Next, we experimented with the Term frequency-inverse document frequency (tf-idf) method, which is intended to reflect how important a word is to a document. In contrast with CountVectorizer, tf-idf not only calculates how many times a word appears in a document, but also calculates the inverse document frequency of the word across a set of documents, which will penalize the common words. Similar to the CountVectorizer method, we can see documents 1, 9, 10, 11, and 19 showed lower similarity compared to the rest of podcasts (Figure 10).

TF-IDF Formula:

tf(t,d) = count of t in d / number of words in d

idf(t) = log(N/(df + 1))

*tf-idf(t, d) = tf(t, d) * idf(t) = (count of t in d / number of words in d) * log(N/(df + 1))*

Ke	Type	Size	
0	Array of int64	(5,)	[19 11 9 10 1]
1	Array of int64	(5,)	[4 19 9 10 7]
2	Array of int64	(5,)	[19 9 10 11 1]
3	Array of int64	(5,)	[19 9 11 10 1]
4	Array of int64	(5,)	[19 10 9 11 1]
5	Array of int64	(5,)	[19 9 11 10 1]
6	Array of int64	(5,)	[19 9 1 10 11]
7	Array of int64	(5,)	[19 1 11 4 9]
8	Array of int64	(5,)	[19 11 1 9 10]
9	Array of int64	(5,)	[19 4 3 1 10]
10	Array of int64	(5,)	[4 19 1 9 0]
11	Array of int64	(5,)	[4 0 19 3 9]
12	Array of int64	(5,)	[19 9 11 1 10]
13	Array of int64	(5,)	[19 9 11 1 10]
14	Array of int64	(5,)	[19 9 11 10 1]
15	Array of int64	(5,)	[19 11 9 10 4]
16	Array of int64	(5,)	[19 11 9 10 1]
17	Array of int64	(5,)	[19 9 10 11 7]
18	Array of int64	(5,)	[19 9 10 11 4]
19	Array of int64	(5,)	[4 15 0 2 9]

Ke	Type	Size	
0	Array of int64	(5,)	[0 18 17 14 13]
1	Array of int64	(5,)	[1 18 14 15 17]
2	Array of int64	(5,)	[2 14 12 13 15]
3	Array of int64	(5,)	[3 13 12 5 15]
4	Array of int64	(5,)	[4 13 14 2 12]
5	Array of int64	(5,)	[5 18 15 14 13]
6	Array of int64	(5,)	[6 13 12 14 18]
7	Array of int64	(5,)	[7 13 12 14 5]
8	Array of int64	(5,)	[8 12 14 13 18]
9	Array of int64	(5,)	[9 12 14 5 13]
10	Array of int64	(5,)	[10 12 13 5 14]
11	Array of int64	(5,)	[11 12 18 5 14]
12	Array of int64	(5,)	[12 18 13 14 5]
13	Array of int64	(5,)	[13 18 15 12 14]
14	Array of int64	(5,)	[14 18 15 13 12]
15	Array of int64	(5,)	[15 18 17 13 14]
16	Array of int64	(5,)	[16 14 15 13 12]
17	Array of int64	(5,)	[17 18 15 14 13]
18	Array of int64	(5,)	[18 15 17 13 14]
19	Array of int64	(5,)	[19 12 5 14 10]

Figure 11: cosine similarity score results for dissimilar and similar podcasts after implementing tf-idf, left and right

Next, we sorted 5 most dissimilar and most similar podcasts based on the result from tf-idf and got a similar result (Figure 11).

Sentiment scoring

titles	simple_sentiment
2022 Oscars Recap Excerpt	0.322034
AirBnB Podcast Excerpt	0.5
Between (Chris) Rock And A Hard Place Excerpt	-0.258065
Chris Rock Allowed Will Smith to Stay at Oscars Excerpt	0.153846
Chris Rock vs Will Smith Excerpt	-0.0909091
Chris Rock_s Response, Comics React and Josh Wolf Excerpt	0.521739
Ebro In The Morning - Talking Smack_ Conspiracy Theories Excerpt	0.322034
Ebro In The Morning - The Smack Heard Around The World Excerpt	0.47619
Episode 156_ The Will Smith and Chris Rock Fight Excerpt	0.142857
Grammys1 Excerpt	0.586207
Grammys2 Excerpt	0.0967742
Grammys3 Excerpt	-0.12
I AM ATHLETE _ Will Smith Slaps Chris Rock_ Was He Right or Wrong_ Excerpt	0.0172414
Jocko Evaluates Celebrities Slapping Each Other. Will Smith _ Chris Rock. Oscars. 1 Excerpt	0.261538
Live From Inside the Oscars_ The Will Smith-Chris Rock Slap Excerpt	0.464286
Oscars (2022) Excerpt	0.639098
Oscars Reactions, Apple Subscriptions, and The Team Behind Super Pumped_ The Battle for Uber Excerpt	0.285714
Podcast5 Excerpt	0.446809
Will Smith vs. Chris Rock NFL OT changes Excerpt	0.606557
Bitcoin Podcast Excerpt	-0.538462

Figure 12: sentiment scores for each podcast

We lastly calculated the sentiment scores for each document. Our initial expectation was that all documents except for the one regarding Airbnb would have a low score, due to the scandalous event at the Oscar, and because Bitcoin's reputation has not been positive in recent times. We used negative and positive words to calculate sentiment scores, to add one point of the audio if a word appeared in a positive word dictionary, and subtract one point if a word appeared in a negative word dictionary. If the word did not exist in either dictionaries, they were ignored. Finally, we used the total word score divided by total word count, to get our final sentiment scores.

Functions used:

sum(scores) = +1(total words appeared in positive dictionaries) + (-1)*((total words appeared in negative dictionaries)*

sum(scores)/(1(total words appeared in positive dictionaries and total words appeared in negative dictionaries))*

From the results, we found that the highest scoring topic goes to the podcast episode "Oscars (2022)" while the lowest scoring episode was the Bitcoin podcast episode. The results were not perfectly aligned with our expectations, but still made sense; other podcasts related to Oscars still had fairly low scores, which could perhaps mean that specific podcast episode does not describe the scandal in great detail. If we divide all podcasts into different categories, we can see the mean score for each category, and the score for the Oscars group on the bottom.

Conclusion

Using LDA, cosine similarity, CountVectorizer, and tf-idf, we were able to successfully compare our podcast episode excerpts generated from keyword topics, giving us valuable insight into relevant discussions within larger podcast episodes. We also looked at sentiment scores, which could be useful as criteria in delivering relevant content to users based on whether they want to hear positive or negative discussions of different subjects. Overall, we were able to show that current techniques in NLP could prove useful if incorporated into a discovery tool where users seek to listen to relevant segments and discussions based on keywords or phrases without having to manually search for them. The caveat is that this would require podcasts to be

transcribed into written text prior to analysis for such a search engine, which could be expensive to implement.

References

Berlage, O., Lux, K.-M., & Graus, D. (2020, May 14). *Improving Automated Segmentation of Radio Shows with Audio Embeddings.* https://ieeexplore.ieee.org/abstract/document/9054315

IBM Watson - speech to text. IBM. (n.d.). Retrieved May 7, 2022, from https://www.ibm.com/cloud/watson-speech-to-text

Jing, E., Schneck, K., Egan, D., & Waterman, S. A. (2021). *Identifying Introductions in Podcast Episodes from Automatically Generated Transcripts.* https://doi.org/10.48550/arXiv.2110.07096

Lawto, J., Gauvain, J.-L., Lamel, L., Grefenstete, G., Gravier, G., Despres, J., Guinaudeau, C., & Sébillot, P. (2011). A Scalable Video Search Engine Based on Audio Content Indexing and Topic Segmentation. *ArXiv:1111.6265 [Cs]*. http://arxiv.org/abs/1111.6265

Sehikh, I., Fohr, D., & Illina, I. (2018, January 25). *Topic segmentation in ASR transcripts using bidirectional RNNS for change detection.* https://ieeexplore.ieee.org/document/8268979

Team, S. (n.d.). *Home - spyder ide*. Home - Spyder IDE. Retrieved May 7, 2022, from https://www.spyder-ide.org/