Description-based Podcast Reccomender System

Willis Allstead
Computer Science & Engineering
University of Nevada, Reno

I. Introduction

Podcasting is a relatively new and prosperous form of media that increasingly more people consume daily. It is a mainly long-form conversationally-based form of media which shares characteristics with long radio programs. The main difference between podcasting and radio shows is that a podcast is split into episodes, and those episodes are accessible by anyone at any time. A podcast "feed", similar in structure to an RSS feed, is generated by a podcast creator and shared amongst podcast distributors/players like Apple iTunes, Overcast, Stitcher, etc. Those players let their users know when a feed they are subscribed to has a new episode available. They also provide functionality for recommending podcasts not in their users' libraries. This recommendation functionality is usually based on searching for podcasts in similar categories, sharing similar titles, and probably similar audiences. The currently-implemented podcast recommender systems are not documented publicly but these assumptions can be made about them given the recommendations they provide.

To provide better recommendations than those provided by popular players, it is important to discuss some information stored in a certain podcast feed. A podcast feed first holds a description of the show, then holds a description of each episode's content. These descriptions are limited to around 250 words and are meant to describe the content of a podcast or episode, but these guidelines are not enforced by any one body. Since podcast creators pay to host their own shows, unlike on popular video platforms like YouTube, they maintain control over their feed and what appears in it. They only need to adhere to basic feed requirements in order to submit their feed link to players like iTunes. While this is beneficial for podcast creators in some ways, allowing them more creative freedom, ensuring that creators own their content, etc. it also makes it more difficult for players to better recommend podcasts based on the information provided by the podcast creators. Other fields in a podcast feed file such as category, subcategory, donation links, and more are sometimes omitted completely by podcast creators. Keeping this in mind, using only certain data that most podcasts do provide, a better podcast recommender system can be created based on not only the factors currently used by players in the space, but also the description of each podcast and episode. Basing recommendations on descriptions of episodes and podcasts should help create better results because there are more possible words to compare between podcasts. Finding a more robust method for podcast recommendation based on description similarity may aid in the growth of podcasts as a medium.

Since most podcasts are long-form in nature, it is more difficult to gain initial interest in burgeoning podcasts. Since a listener doesn't have a rapport with a new podcast host, they are less likely to have an initial interest in any of the episodes created for that podcast. If a user had a better idea of what they were getting into by knowing they have more reliable recommendations, they might be more likely to listen for more than one episode or even become a subscriber of the podcast. Listener retention could be made easier by simply giving users better podcast recommendations.

II. Background

Since podcasting only recently started gaining popularity, there is a lack of research in many areas of the field. Searching for Podcast recommender systems came up short. Instead, we must rely on similar papers regarding song recommendation based on lyrics. This is the most similar well-researched topic, as it is set up in a similar structure, with one caveat. Unlike how these songs will be compared by the similarity of their actual lyrics, the podcasts will be compared by the similarity of their descriptions and their episodes' descriptions. While it is recommended that podcast creators detail the content of each episode in its description, they might include very little information or somewhat unrelated information such as external links, etc. If unrelated information is filtered out correctly, the descriptions of podcasts and their episodes should more accurately represent their contents. In a paper titled *Retrieving Similar Lyrics for Music Recommendation System*, they detailed comparing Indian songs' lyrics to each other to recommend similar songs. In retrieving the lyrics, they had to preprocess their raw data in a number of ways including removing html code, fuzzy matching certain different spellings of words, and more. For descriptions of a podcasts and its episodes, we will use a similar process. We will use a database with 10,000 pre-crawled podcast feed URLs to begin our data processing. We will need to automate the requesting of those feeds, the cleaning of the HTML response to receive plain text descriptions, the removing of any words that seem to match to links as well as any dates included in the description, and

finally use tf-idf to find the most important words to use for comparisons between podcasts.

In the paper *Retrieving Similar Lyrics for Music Recommendation System*, they made use of multiple document comparison systems such as FCMs, Doc2Vec, and SOFMs. They put the different results using each system in a table and found that the Doc2Vec system they used was not as effective as the FCMs, and that neither were as effective as the SOFMs. SOFMs use cosine similarity to calculate similarities between to documents. A similar approach to SOFMs can be taken using term frequency–inverse document frequency or tf-idf. This method would also be better than the methods described in the similar lyrics paper for podcast recommendation due to the amount of over-used words, such as "and", "the", "a", etc. because it will give less weight to the words with higher frequencies like that. Using information about word-based recommendation systems like those created in *Retrieving Similar Lyrics for Music Recommendation System,* we can more easily select a single method for comparing descriptions using tf-idf.

In *Ranking lyrics for online search,* Macrae and Dixon implement different methods for ranking lyrics based on a search terms. Using the most important words in the descriptions of a podcast and its episodes as a set of search terms in a similarly structured search tool might provide accurate recommendations. In this paper, they use a bag-of-words (BOW) format to record frequencies of words used, and they use word stemming to reduce the amount of unique words in a lyric. Stemming seems important, as a step before finding tf-idf, and could help draw relationships between different podcast/episode descriptions. Macrae and Dixon also found that using lyrics concurrence was very effective at matching search terms to songs, which is a method for ranking music lyrics based on the similarity of its lyrical content to other lyrics of the same song. This helps in calculating search results because it helps the algorithm know how similar the lyrics are within a song. This factor does not seem as applicable to podcast or episode descriptions, because there is no expected repetition in descriptions besides high frequencies of unimportant words. Therefore, this factor will not be used in calculating recommendations for similar podcasts.

Both papers used for reference make substantiated arguments for the different recommendation or search result systems they make use of. One important issue to point out with both papers is how they verified that the recommendations or search results were performing well. [1] makes the argument that manual verification of the algorithm would be too tedious, which is understandable, but [2] didn't mention the process they took to verify the quality of search results. Verification is very subjective, and we will have to rely on testing certain cases with and without description similarity comparison. A better verification method could be designed, maybe based on manual ratings of recommendations by many users on some player, or by comparing recommendations of players in a manual way to these results. Both of these verification methods would consume too much time for the scope of this paper.

III. Methodology

Obtaining podcast metadata is complicated due to the decentralized nature of podcasts. By researching podcast listening platforms, it is obvious that iTunes remains the top aggregator for podcast feeds due to having the lion share of listenership, and therefore has a wider catalogue of podcasts to choose from. Unfortunately, the iTunes API is designed only to show results for queries including a podcast or episode name. Not knowing the podcasts that will be in the dataset prior to searching through the iTunes API means that data collection using this method is impossible. This, in combination with request throttling means a different method must be implemented. Individuals have made efforts to crawl the iTunes website in order to extract information about a large number of popular podcasts. The Podcasts-Data dataset chosen for this project was compiled by Deniz Ozturk on GitHub. Ozturk used various methods to crawl iTunes and collect podcast feed URLs. He then compiled the data from those feed URLs into organized folders. The files containing data on the podcasts in the dataset are .csv files, which can be interpreted simply by most programming languages. The pertinent data on each podcast that applies to this project includes the name and the feed link of each podcast, for use when recommending podcasts based on other podcasts, and the descriptions of each podcast and its episodes, for use when comparing podcasts. This dataset contains 10,000 podcasts, which is far from the total number of podcasts available on iTunes, but should be enough to establish whether a podcast recommender system based on description similarity is effective or not. In collecting the 10,000 podcasts, Ozturk took certain steps which resulted in a dataset from a non-random sampling. He only included podcasts with over 20 episodes, and collected them starting with the top lists from iTunes for each genre and sub-genre. This means that if this podcast recommender system is successful, it would need to be tested on newer and less popular podcasts later to ensure that the system aids in content discovery and propagation.

To compare the podcasts, a bag of words will be created for each podcast description and its episode descriptions. This way podcasts can be recommended by listening behavior of other podcasts or the episodes within them and vice versa. Weights for the words in each description will be assigned using tf-idf. Before calculating the frequencies of words in the descriptions, some data preprocessing to remove noise will be required. This is similar to the method used in *Ranking lyrics for online search*. Using this method, in conjunction with a stemming algorithm such as the Porter stemming algorithm should result in more representative tf-idf vectors to compare podcasts with. After obtaining these tf-idf vectors of both the podcasts and the episodes of each podcast, similarity between pairs can be calculated using cosine similarity. Given two tf-idf vectors of

podcasts, calculating the cosine similarity between them would result in a value ranging from 0 to 1. A value near 0 would mean that the two podcasts are not similar at all while a value near 1 would mean that the two podcasts are extremely similar. Since this recommender system is based on recommending based on the similarity between podcasts and or episodes, the higher the similarity the higher up on the list of recommendations that result will appear.

To program the recommender system, the Swift programming language will be used. Using Swift to design the project is advantageous since moving from a local machine to calculate the vectors and similarities to a Linux server would take very little effort if the calculations take too long on the local machine. Also, there are standard Swift libraries which make file manipulation trivial, specifically with the file types used in the dataset. There are also Swift libraries available for popular stemming algorithms, creating TF-IDF vectors for documents, and computing cosine similarity between the descriptions of either podcasts or episodes. A web interface will be created to allow users to interact with the recommender system. Since there is no need to crawl iTunes for data since the data is already available in the dataset, there is no need for requesting data externally from the web client. The web interface will be used simply as a graphical interface for interacting the system.

The web interface will allow a user to choose either a podcast or an episode of a podcast out of the dataset that they like. The user will then be presented with a list of 10 recommended podcasts in order of most likely to be match to least likely. Since the iTunes API provides a list of 10 podcasts most listened to by listeners of a certain podcast, these two lists can be displayed together and compared for similarity. Numerically evaluating the effectiveness of a recommender system without tracking user behavior is difficult. By comparing the list of podcasts recommended by the system designed in this project with the podcasts that users actually ended up listening to, we can see if this recommender system would have been successful at recommending those podcasts without knowing previous listening behavior. The difference in percentage between the two lists will be used as a metric to evaluate the effectiveness of this recommender system. While this metric could be seen as somewhat arbitrary, it is important to address that recommender systems created without any information about user behavior are difficult to design, and that any comparisons that can be made to existing systems could provide key information in developing more robust systems in the future. While not using user behavior to suggest recommended podcasts is difficult, it is especially important for decentralized mediums like podcasts. Not relying on information from podcast players could help develop podcasting as a consumption medium, while maintaining the privacy of podcast listeners.

Designing the system to be usable through a web interface is important to allow more users to test its effectiveness later on. While out of the scope of this project, if the system is found to be effective, an API could be created for the system to more easily include podcast recommendations in any podcast listening app or client.

REFERENCES

[1] Patra, Braja & Das, Dipankar & Bandyopadhyay, Sivaji. (2017). Retrieving Similar Lyrics for Music Recommendation System.

[2] R. Macrae, S. Dixon. “Ranking lyrics for online search,” 13th International Society for Music Information Retrieval Conference (ISMIR 2012), pp. 361-366.