Deep Learning ListenBrainz Recommendation

A Preprint

Dean Cochran Music and Technology Group
Universidad de Pompeu Fabra
Barcelona, Spain
`charles.dean.cochran@outlook.com`

March 31, 2022

Abstract

Within my project I plan to create an effective retrieval model for recommendation candidate generation of songs based off what users have previously listened to. The model incorporates the embedding for artist names, and the time at which the listen occurred. The complex deep learning architecture was mainly provided by the framework of tensorflows recommender system library which trained three different variations of the retrieval model. The model was evaluated by its loss and its TopK accuracy to find that the retrieval model will capable of generation of relevant candidates, lacks the organization that a ranking model provides.

Keywords ListenBrainz · Deep Learning · Recommendation · Tensorflow

1 Introduction

Recommendation engines are powerful tools that make browsing content easier. Moreover, a great recommendation system helps users find things they would have never thought to look for on their own, like movies, retail items, recipes, and even songs. In fact, music recommendation is a large sub field in recommendation; mainly recognized by the industry's largest music streaming companies. The basic idea for a neural-network-based ranking model is to create embedding to represent each user and song by predicting how much each user would enjoy each song. To measure how much each user would enjoy each song, we must find users with similar taste and measure their enjoyment of the song. This uses the assumption that people that listen the same movies have the same taste or enjoyment of items. To achieve this result, we must create the embedding that represent the relationship between user and song. As an example, think of creating a 1-dimensional matrix, where the users are the rows, and the columns are the songs indicating which songs a user has listened to. Tensorflow within the last couple of years has now joined in with the popular recommendation libraries within the last two years. This means, the already powerful functionality of tensor flow and keras can now be modeled to preform deep learning recommendation. Due to the increase in usable, and well documented recommendation modelling libraries like tensorflow's, I plan to build my own music recommendation model from scratch utilizing creative commons ListenBrainz Data Dumps of user listening history.

2 Dataset

ListenBrainz is a sub project of the MetaBrainz which provides daily updated data dumps of their listen history databases. For the purposes of creating a deep learning algorithm model I decided to download largest full export of the ListenBrainz database to give my model enough data. I exported this file:

/listenbrainz/fullexport/listenbrainz-dump-789-20220315-040002-fulllistenbrainz-listens-dump-789-20220315-040002-full.tar

As a disclaimer, this file was 39GB and required very special care as making most queries on the data set were inefficient and needed to be optimized. The data set contains many folders which when combined contain files with a JSON object

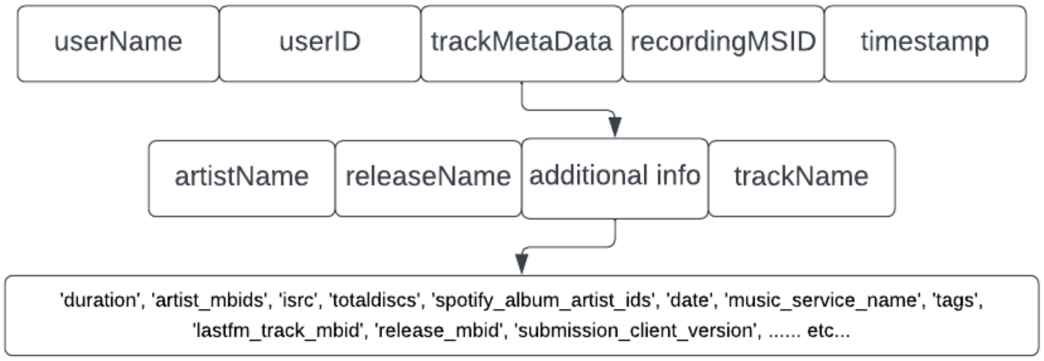
userName
userID
trackMetaData
recordingMSID
timestamp
artistName
releaseName
additional info
trackName
'duration', 'artist_mbids', 'isrc', 'totaldiscs', 'spotify_album_artist_ids', 'date', 'music_service_name', 'tags',
'lastfm_track_mbid', 'release_mbid', 'submission_client_version', ... etc...

Figure 1: JSON Listen object structure outline

for every listen created by a user. The hierarchy of a listen JSON object can be visualized in Figure 1. Once these objects are parsed, I was able to extract as much information from the objects as I wanted. However, many objects did not contain the same pieces of information. This was due to the policy of ListenBrainz which only requires a listen to have a corresponding username, userID, recordingMSID, and timestamp. Due to this limitation, I only extracted objects that had the required features and artistName, and TrackName. The result left me with over 2 million listening instances and minimal loss of data. It should be noted that within the additional information some listen objects contain Musicbrainz mbids, Spotify ids, or LastFm ids. These can be traced back to a single artist/record where more information on the artist/record can be obtained for each listen. This information includes but is not limited to the genre, lyrics, location, and audio specifically related to the song.

3 Methodology

Recommender Systems can be vaguely defined as the combination of retrieving and ranking. The retrieval stage is responsible for selecting an initial set of hundreds of candidates from all possible candidates. Its main objective is to find the best candidates to recommend amongst millions of provided options. The ranking stage takes the outputs of the retrieval model and fine-tunes the order of the candidates to select the best possible handful of recommendations from a large collection of options. Provided the limitations of my data sets provided columns and for the purposes of my project, I will not be ranking the songs, but rather implementing a recommendation retrieval model utilizing the recordMSID, timestamp, userId, and artistName to retrieve songs from the other possible recordings available in the data set.

To familiarize ourselves some necessary recommendation concepts, an embedding is a low-dimensional space that captures the relationship of vectors from a higher dimensional space. Essentially, an embedding reduces our feature's inputs such that they can be represented in a meaningful manner in a lower dimensional space. We can use an arbitrary number of dimensions to represent our features. A larger number of dimensions would allow us to capture the traits of each feature more accurately, at the cost of model complexity.

3.1 Query and Candidate Models

I made two models called the RecordingModel and UserModel that when given inputs of a particular listen instance, return an embedding that accurate defines the listen in a low dimensional space. In the query model for recommendation, which deals with learning user embeddings, I incorporated the specified user model that calculates embedding layers for each userID and timestamp passed in. For the candidate model, which deals with learning recordings, I incorporated the specified item model to calculate the embedding layer for each recordingMSID, and artistName passed in.

For simplicity's sake, both the QueryModel and CandidateModel were defined using the same deep learning architecture. Since the expressive power of deep linear models is no greater than that of shallow linear models, I use ReLU activation layers for all but the last hidden layer. The final hidden layer does not use any activation function: using an activation function would limit the output space of the final embedding and might negatively impact the performance of the model.

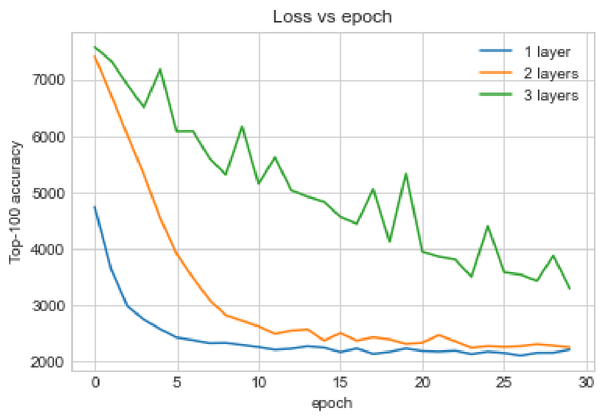

Figure 2: The loss of the trained models over time

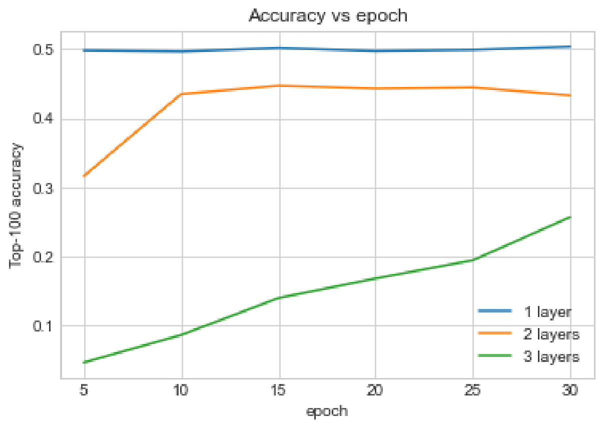

Figure 3: The Top100 Accuracy of models

3.2 Recommendation Model Architecture

The embedded query and candidate vectors are individually concatenated before passing through the mentioned series of fully connected layers, which maps the concatenated embedding into a prediction vector as output. Additionally, this final recommendation model implements the evaluation metric, TopK Accuracy, where k equals 1,5,10,100, etc.

A progressively narrower stack of layers, separated by an activation function, is a more common pattern, so I opted to extend my observations when training the model by generating 3 different models of the same recommendation model class, but different layer sizes. Additionally, I was left with a choice of selecting an optimizer, and rather than tuning this hyper parameter, I left the optimizer to be a constant Adagrad Optimizer with a value of 0.1.

4 Results

Upon training our model with an 80/20 split on our training and testing data, I received some acceptable results shown in Figures 2 and 3. Each model preformed adequately, with the single layer preforming the best with Top. However this is a good illustration of the fact that deeper and larger models, while capable of superior performance, often require very careful tuning. With appropriate tuning and sufficient data, the effort put into building larger and deeper models can lead to substantial improvements in prediction accuracy.

Upon my observations I was able to predict recommendations utilizing the single layer model and saw results that made intuitive since. I was able to predict artists and recordMSIDs that users would likely enjoy based on the model's predictions. These predictions can be found inside my notebook here at `https://github.com/deancochran/Deep-Learning-ListenBrainz-Recommendation`

Table 1: TopK ListenBrainz Recommendation Results

K	Accuracy
1	0.004
5	0.037
10	0.098
50	0.376
100	0.505

5 Conclusion

Within this report I have outlined a deep learning retrieval model capable of taking multiple sources of information to compute embedding vectors and utilizing the information provided to generate relevant predictions without ranking the candidates after retrieval. Due the fact that the model is not complete without ranking the candidates, what can be said about the results is that the retrieval model is capable of correctly identifying relevant candidates. The lack of accuracy I highly suspect is due to the ordering of which these candidates are found in. In addition to a full retrieval model, I have been successful in utilizing the ListenBrainz resources to aggregate and collect listen instances to form my own collection of user listening history. In the future, it would be useful to collect the mbids stored in some of the listens and map them to the MusicBrainz artist/record to pull more meta data and audio metadata that could be used to assist user personalized recommendation.

Finally, recommendation is a very intuitive process that can be complicated quite rapidly with deep learning. For those who are interested, I found that the tutorials on Recommendation Systems provided by Developers at Google is very well documented, and quite a fun read.

`https://developers.google.com/machine-learning/recommendation`

References

[1] Tensorflow Recommenders TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems `https://www.tensorflow.org/`

[2] MetaBrainz ListenBrainz: Visualize and share your music listening history `https://listenbrainz.org`