Music Recommendation with Autoencoder Model

Jamison Campbell[1,*]

[1]Central Michigan University, Computer Science, Mount Pleasant, 48858, United States
[*]campb2jt@cmich.edu

ABSTRACT

Music discovery is not what it used to be. No longer is one constrained by what the local disc-jockey curates for the afternoon radio session. Today, there exist a plethora of resources and platforms to discover new music with seemingly unlimited libraries to browse. Modern recommendation algorithms often fall into two categories which are either content based, meaning recommendations are made from similar features or collaborative filtering which is made from similar users' tastes. The latter being the preferred method for large streaming services with millions of users. While the technology has come a long way, oftentimes users are still left scouring for more content they enjoy. Machine learning can aim to mitigate this problem, and has certainly made great strides to appease unsatisfied users in the last decade. Many companies have used song data for music curation, specifically the attributes of genre, release year, similar users' libraries. However, music is a medium that is difficult to categorize. An artist's discography can range across genres, and genres themselves are fluid and constantly evolving. Some artist's music can be ahead of its time or pay homage to prior work made decades earlier. In addition, a user's library can consist of a wide range, from Lil Uzi Vert to Van Morrison, so it cannot necessarily be used as the greatest metric. Few have ventured into the utilization of song features to create an embedded space for mapping similarities between songs. This research will aim to create a content-based system that can curate songs for users to rediscover that joy of finding a new song.

Introduction

Music recommendation algorithms have been in great interest to the subject of machine learning for quite some time. Most recently, deep learning networks have been utilized for algorithms based on millions of users ascribed to subscription services. Training models via user information can be akin to users themselves curating a playlist for a friend. These systems present issues, however, since they require large-scale databases of user information.

Content-based filtering systems present the best opportunity for research in this area. There are hundreds of ways of going about genre classification and returning similar sounds. One can craft algorithms to map spectrographs of songs and utilize similarities or even the audio itself [1]. Attributes such as mood of a song, or even social tags applied by users on popular sites can also serve to further this aim. While there are many avenues to solve this issue, problems remain, as stated by researchers at Ghent University, "Although recommender systems have been studied extensively, the problem of music recommendation is complicated by the sheer variety of different styles and genres, as well as social and geographic factors that influence listener preferences". [2] Generally, recommendation systems fall into two broad categories: content-based systems and collaborative filtering-based systems [3]. The former deals with attributes of a song such as the beats-per-minute, genre, or year released, while the latter is based on user data which recommends items that similar users have interacted with. Content-based systems will then group songs with similar attributes to the one in question. Often called personality-based systems, the other approach looks to a user's past behavior like what songs they have saved and attempts to predict new content from other users with synonymous interests. These types of algorithms are based on matrix factorization and work almost exclusively for large-scale projects with thousands of users on the platform [4]. Usually, these algorithms will be implemented in tandem to one another. The great fault in collaborative-filtering systems is best stated by researchers from the National University of Singapore, "CF is powerless when confronted with the new-song problem - it cannot recommend songs without prior usage history" [5].

Pandora has been in the process of a large-scale assignment they dub the 'Music Genome Project' which aims to break music down by assigning over 400 attributes that describe the song in depth and classify it into a in a specific subset that is then used for curation of their exclusive stations [6]. Unfortunately, this dataset is not available to the public but there are many datasets accessible for building content-based systems. The largest scale project, Echo Nest, which was purchased by Spotify last year, has data on millions of songs including tags for similarity and mood. The public subset of this dataset is called the Million Song Dataset (MSD) which is free and split for both testing and training data. The library of perhaps the most interest is the MSD since it can also be accompanied by the official lyric dataset, MusixMatch. HookTheory is another dataset that will be extremely helpful for chord progression, there are many repositories that have scraped their website and the company has released an API. In addition, with Spotify's web API I could use a dataset of pop artists and use their "related artists" method to exponentially increase my dataset and then use other methods to query the top songs of the artists. [7]

Word2vec is a model used in numerous applications for mapping words around the context of the words surrounding it. As stated by Ramzi Karam in his article on word2vec for music recommendation, "The neural network takes in a large corpus of text, analyzes it, and for each word in the vocabulary, generates a vector of numbers that represent that word. [8] These vectors of numbers allow the algorithm to assess similar context and make scarily accurate predictions. Researchers from Queen Mary University sought to apply this idea by utilizing music slices as 'words' and inputting it into a Skip-Gram model that mapped similar tones [9]. These models made by researchers sought to apply an embedded vector space to the idea of music curation. However, they did not explore the notion of using other song features including the duration, beats per minute, or release year. Utilizing features like this, I plan to create a model that utilizes autoencoders to create the embedded space. These models work by creating a multilayered model that's layers increase in shape until an encoded layer, and then are 'decoded'. The model is forced to create compressed representations of the data in the smallest, encoded layer, which can be utilized to find similar sample regions. This is akin to word2vec by utilizing these vectors made from compressed sample data representations.

Methods

Data

The Million Song Dataset is a free and open-source collaboration between researchers from the Laboratory for the Recognition and Organization of Speech and Audio (LabROSA) and the company The Echo Nest, which was recently acquired by Spotify [10]. They provide their entire metadata with multiple features for each song, and a subset of 10,000 songs available for public, research intensive purposes. Using this subset and the script they provided, another research group called Collection of Really Great, Interesting, Situated Datasets (CORGIS), uploaded the unpacked data in comma-seperated value format [11]. Using this dataset, I was able to get 10,000 songs and roughly 35 features per song. This is ample data for splitting, and the data will be an 80-20 split specified in the model.fit method not before training. This provided enough information to work with and give ample inputs for hyperparameter tweaking. The dataset is relatively clean for every feature including information on artist 'hotness', tempo, and time signatures.

For some feature columns in the file there existed many problems when trying to feed the data into the model, initially. This required me to scrape the cells for missing values and fill them with zeros when appropriate. Since this was never that many values, it should not have much an effect on the overall outcome. Some features had to be scrapped entirely such as the year release for each sample, which lacked a severe amount of values, and zero-filling would skew the model progress. When loaded into the model two tensors were filled with 10,000 samples, one with features and another with the song title and artist for future reference. The former tensor was then normalized with the utilization of standard deviation and some simple Numpy classes. Utilizing this subset of the Million Song Dataset, it no longer seems inconceivable to create a music recommendation model.

Results

Development Environment

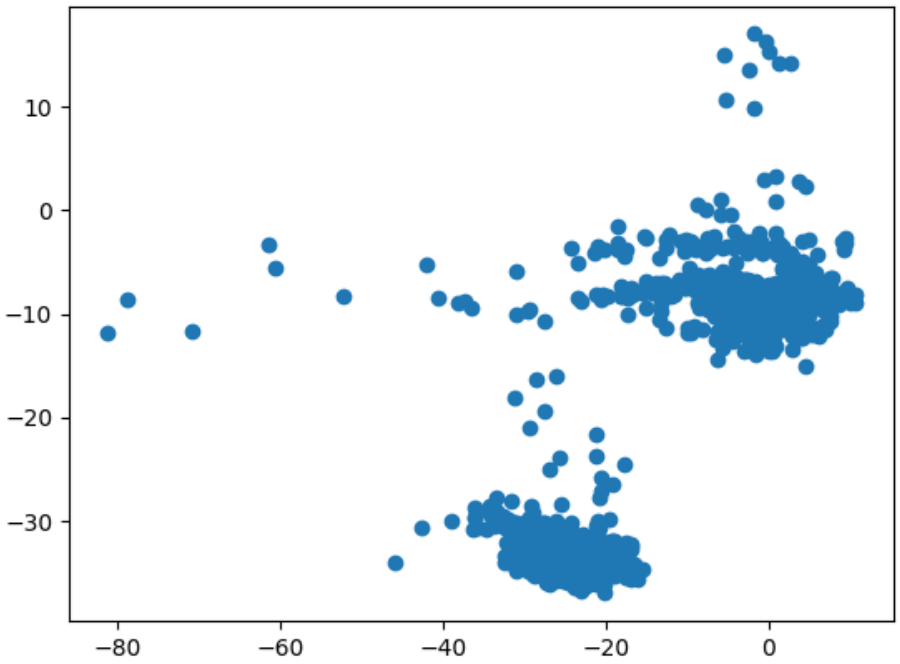
This project was made on a Linux server which every student was provided access to by the instructor. The Keras library was used for implementation of models coupled with the Theano backend.[12][13] Computations were performed using a GeForce Quadro graphical processing units each utilizing 8GB RAM. In order to perform calculations and plot progress, the dependencies NumPy and Matplotlib were used, respectively.

Training and Evaluation Protocol

After the training data had been loaded and normalized, the model was constructed. Since my plan was to create a recommendation system, the best course of action was to pursue a method of crafting an embedded layer that showed the vector space representation of similar samples. Then utilizing this to compute similarity scores and return a similar song. This was best done by creating an autoencoder from the Keras Functional API. The model consists of 5 hidden layers, an Input layer with input shape equaling the number of features per sample from my data, and a Dense layer with the same output shape. An autoencoder's purpose is to create low-dimensional representations of the input data. The model is forced to do this when each successive layer lowers the dimension by a great margin, and then proceeds to decode the data, resulting in input x and output x'.

architecture	regularizer	num epochs	loss	val loss
128-64-3-64-128	L2(.01)	20	0.4372	0.2256
128-64-3-64-128	L1(.01)	20	0.5011	0.2584
128-64-2-64-128	L2(.1)	200	0.1443	0.1971
256-128-64-3-64-128-256	L2(.01)	150	0.1547	0.1867
128-64-3-64-128	L2(.01)	100	0.1669	0.1829

Figure 1. Plot of the vector space from encoded layer



The first Dense hidden layer had a shape of 128 using an activation function of rectified linear unit. The following layer was entirely the same except it's shape was half, so 64. Following this layer, was the embedded layer with 3 units and an l1 activity regularizer set to .01. After this, the model is 'decoded' meaning the process is then reversed, thus creating a 128-64-3-64-128 architecture, as outlined in Figure 1. These offered a great deal of hyperparameter tuning that I was able to augment for the best results. For example, using the loss metric as a baseline, I proceeded to play around with the regularizer (i.e. .01 or even l1 or l2 regularization), the number of hidden layers, and the number of epochs. I was also able to use only some or all of the features I was given from the Million Song Dataset. The loss function used is mean squared error, and by having a validation split of .2, I could make comparisons of both the training and validation loss and see overfitting. This resulted in my current model, boasting a loss value of .1669 at around 100 epochs, after this there are clear signs of overfitting. Examples of hyperparameter tuning can be seen in the table on the last page, and shows a few examples of how I got to my current model on the bottom.

Shown in Figure 2 is a plotted representation of what the middle layer looks like. This embedding can give a great deal of insight into the relationships among songs from the features, very similar to that of a word embedding for countries and their capitols like the researcher Lenny Khazan [14]. Using a skip-gram model based on word2vec [15] he was able to plot these relationships and go in to detail on how this layer can be utilized. By calculating the distance of these vectors one can seek to find similar pairs and use this to make predictions on similar samples when processed with an input sample. In addition, my cleaned data includes a feature for tags for every song that act as an additional metric of comparison for me to judge how well the model performs at its intended task. The encoded layer of the model is then used to calculate distances of vectors. This was done using SciPy, a scientific calculating Python library, and calculating the distance between the starting song index (i.e. what a user enjoys) and then every other value in the layer. If this distance was lower than .025 and greater than 0, it returns those songs and the artist they are by. For example, the model will say 'If you like Sunny Came Home by Shawn Colvin' (which is the input index chosen), then it will return 'You would also like 'Louise by Willie DeVille'. By combing through differing inputs it seems the model is okay at its task. Something along the lines of Queens of the Stone Age will return other metal bands like Helmet but sometimes it will also return Ice Cube.

The results were not the greatest, but better than expected. Content based systems suffer from songs like rap that may have fast tempos like metal songs, causing it to recommend something that may seem off. Often researchers and companies alike pursuing a similar issue would likely seek to implement a hybrid recommender system, meaning it utilizes both content-based filtering and collaborative filtering methods. However, while one should pursue this endeavor for the best results, this research

was hindered at the availability of user data to be able to accomplish a dual use approach. The results I came to were satisfying given the context and scale of what I was able to work with.

Discussion

This project aimed to better my understanding of machine learning models, and it certainly has served that purpose. The advantage to an approach such as this is primarily the fact that it is content based and not based off user data which large companies stockpile for purposes such as hybrid recommender systems. Most of this data can be accessed for free and each sample can be scraped for even more features from other online repositories or APIs of companies like Spotify and MusixMatch. Each sample included song tags for access in APIs for this exact purpose. This also means that I retain the ability to alleviate or append features from my data as I see fit. Some data columns, such as year release may give the impression that they would help a model, when actually this particular feature hurt the model significantly due to the absence of many samples. In addition, I had never seen prior work using a similar idea with he use of an autoencoder.

The disadvantage to this approach is that it would not rival a hybrid recommender system in comparison. Almost every system that performs recommendations (i.e. YouTube, Amazon, Spotify) benefit from the structure of their platform, getting better with each user interaction on their application. The approach I chose was much less complex than these other models. In comparison to a naïve predictor, however, the model did perform a better. Before I modified hperparameters and excluded features the model was very bare and mirrored other autoencoding models seen online and from the textbook.

There are many aspects, however, that would have been more fulfilling if I had a greater understanding of the libraries used throughout the scope of the project. For instance, knowing how to utilize Numpy's extensive library to calculate such things as Leviathan distance across every vector in my embedded space may have served as a better recommendation system. Another thing was the ambitions of the original promise in comparison to my knowledgebase on the programming environment and similar research projects before mine. I had the initial ambition to include a feature for lyrical content from another website, however, this proved too difficult to encode as a additonal feature to the data. I wish I would have included this feature for at least the fact that I had not seen it been done before when conducting research. I think it would have improved the model further and had yet another aspect of refinement.

I feel as if it would have been a better outcome had I explored more of what has been done by other students in a similar position and built off of it. The use of autoencoders was fresh and exciting but also came with a lack of prior research to learn from. A new direction that could have been explored is utilizing an entirely new dataset that had features for tones, and then creating an embedded space from that data. I had seen similar work like this in the research portion. Researchers were able to make a 'Bag of Tones' model based of the Bag-of-Words scheme to classify music genres [16]. However, this did not make recommendations, rather, it was built to classify the samples by genre. In the end, I put practical skills to the test and achieved the goal of applying Machine Learning to a problem and coming to a resolve that is satisfying, so I can say I am happy with how it turned out.

Conclusion

Machine learning is a constantly evolving field with applications beyond the scope of what most people would consider. Music curation is just one aspect of our lives that has been greatly improved by techniques employed from this field of data science. Coupled with the plethora of publicly available data on millions of songs from researchers like LabROSA, models can be crafted that enable people to access entirely new avenues for music recommendation. Using the Keras API, this project included a model that created an encoded vector space that, when plotted, showed the representations of similar samples of songs. These tuples made by the compressed representation could, in turn, be used for calculation of distances and returning what the model deems a similar sample. While the results of the model were not as initially, ambitiously expected, the encoded layer was able to be utilized to make recommendations that could help to people find new music, and hopefully, a new favorite song.

References

1. Murray, M. Building a music recommender with deep learning. (2017).
2. van den Oord, A., Dieleman, S. & Schrauwen, B. Deep content-based music recommendation. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q. (eds.) *Advances in Neural Information Processing Systems 26*, 2643–2651 (Curran Associates, Inc., 2013).
3. Popper, B. How spotify's discover weekly cracked human curation at internet scale. (2015).
4. Griesmeyer, R. Music recommendation and classification utilizing machine learning and clustering methods. (2011).

5. Wang, X. & Wang, Y. Improving content-based and hybrid music recommendation using deep learning. In *ACM Multimedia* (2014).

6. Walker, R. The song decoders. (2010).

7. Wilson, L. Dj random forest: Song recommendations through machine learning. (2018).

8. Karam, R. Using word2vec for music recommendations. (2017).

9. Herremans, D. & Chuan, C. Modeling musical context with word2vec. *CoRR* **abs/1706.09088** (2017). 1706.09088.

10. Bertin-Mahieux, T., Ellis, D. P., Whitman, B. & Lamere, P. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)* (2011).

11. Bart, A. C., Whitcomb, R., Kafura, D., Shaffer, C. A. & Tilevich, E. Computing with corgis: Diverse, real-world datasets for introductory computing. *ACM Inroads* **8**, 66–72, DOI: 10.1145/3095781.3017708 (2017).

12. Chollet, F. *et al.* Keras (2015).

13. Bergstra, J. *et al.* Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, vol. 3, 1–48 (Citeseer, 2011).

14. Khazan, L. Autoencoders and word embeddings. (2016).

15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q. (eds.) *Advances in Neural Information Processing Systems 26*, 3111–3119 (Curran Associates, Inc., 2013).

16. Qin, Z., Liu, W. & Wan, T. A bag-of-tones model with mfcc features for musical genre classification. **8346**, 564–575 (2013).