Improving Hit Song Prediction with Music Industry Co-Collaboration Network Data

Trevor Hubbard
UMass Amherst
tmhubbard@umass.edu

Phillip Sifferlen
UMass Amherst
psifferlen@umass.edu

Abstract

In this work, we show how music industry co-collaboration network data significantly improves the accuracy of "hit song" prediction methods. These methods predict whether or not a song will enter into top sales charts. After identifying approximately 180,000 songs across 1990-2010, we constructed a "co-collaboration" network using collaborator information from Genius.com. This network was auto-encoded into a vector embedding, and then used as training data for a neural network. This Hit Predictor neural network was able to predict whether or not a song ended up on the Billboard Hot 100 with 86% accuracy, which outperforms prior techniques by roughly 10%.

1. Introduction

Each day, the music streaming service Spotify says that it adds roughly 20,000 new tracks to its platform[1]. With such an influx of new music, the question of which small fraction of these songs will become popular arises. This problem has been called the “Hit Song Science Problem”. While there is no universally accepted definition of a “hit”, most research on the topic uses presence on some form of sales chart to determine song success. Typically, various high-level acoustic features are used to make predictions. Our approach combines these acoustic features with social network information regarding the musicians and industry professionals involved with making the song. More specifically: we associate a vector-space encoding to each song, where each encoding is a combination of auto-encoded collaborator embeddings we calculate from a co-collaborator graph we’ve built. The graph is a representation of connections between collaborators - each node in the network represents a collaborator, and edges between nodes indicate collaboration on a song. Our approach was motivated by the idea that a song’s success not only relies on the song itself, but also on those who have worked on it: songs that are performed by popular artists are more likely to be popular.

2. Background/Related Work

There have been a number of attempts to solve the Hit Song Science Problem. Most of these attempts make predictions using varying representations of the song’s raw audio files. A “low-level” acoustic representation could be a time-series of fine-grained features describing the audio signal (like frequency, rhythm, etc.); “high-level” acoustic representations are usually predictions about more abstract qualities, like how “danceable” a song might be[2]. These methods generally can predict a song’s presence on charts with 75% accuracy[3].

A handful of the Hit Song Science papers mention including some notion of “artist popularity” as a basis for future work. One paper included a “superstar” variable for songs - this variable indicated whether a song’s artist had appeared in the sales charts in years past [4]. This paper was able to achieve up to 85% accuracy by including the “superstar” variable.

The high predictability associated with this superstar variable served as a driving motivation for our project. Instead of explicitly assigning the superstar label, though, we wanted to design a method that could infer this "superstar status" in a more implicit way. We took inspiration from the field of Social Network Analysis (SNA), and decided that we'd try and create a music-industry co-collaborator network as a basis for making this inference.

3. Approach

Our approach for this project was broken down into three major tasks. The first concerned data collection: we'd need to gather and create a dataset of songs, where each song would be associated with a set of high-level acoustic features, collaboration information, and whether or not it charted on the Billboard Hot 100 Chart[5] at some point. The next task involved converting the songs' collaboration info into a co-collaborator social network, which we then auto-encoded. The third and final task involved creating and training a neural network on this dataset!

3.1 Dataset Creation

The first major task in the project was to create a dataset of songs with each song's audio features, collaborators, and hit status. To get started, we looked at the Million Song Dataset (MSD)[6]. The original dataset included 1,000,000 tracks, each of which were associated with both low- and high-level audio features, as well as some metadata. Unfortunately, the full MSD is no longer being hosted online. A summary of the dataset that *only* includes the metadata (title, artist, release, etc) is still being hosted, though. From this summary, we were able to extract the titles and artist names of all of the MSD songs released between 1990 to 2010; in total, this list was 432,005 songs long.

We then moved to determine which of these songs were hits, and which weren't. As was previously mentioned, we decided that a song would be considered a "hit" if it, at some point, appeared in Billboard's Hot 100 Chart. We utilized the Python Billboard API[7] to scrape all songs from the Hot 100 chart from the years 1990 to 2010. Any hits that weren't already included in our dataset were added, and labeled with a 1 to represent their "hit" status. All other songs in our dataset were labeled with a 0, which indicated that they lacked the hit status. Overall, we were able to find 7,530 hits with this approach.

For the audio features for each of these songs, we decided to use the high-level audio features offered by Spotify's API[8]. These features include:

- Duration
- Key
- Mode
- Time Signature
- Acousticness
- Danceability
- Energy
- Instrumentalness
- Liveness
- Loudness
- Speechiness
- Valence
- Tempo

Some of these features are essentially confidence values of Spotify's own prediction systems - "danceability", for instance, is some prediction about whether or not a song would be danced to. The predictions themselves are based on analyses of the spectrograms of the audio's waveforms. Other features, like duration, are easily calculated by looking at metadata about the audio itself[9]. We acquired the features for each of the songs in our list through use of the

Spotipy Python wrapper for this API[10]. The title and artist for each track in our list was sent in a search request to the API; in return, the search response contained titles, artists, and identifiers (URIs) for each search result. To ensure that a selected result matched the title and artist of the song we'd been searching for, we compared the two using Levenshtein distance[11]. Those that matched then had their audio features scraped via the fetched URI.

Finally, we gathered the collaborator information associated with each song. For this, we used the Genius API [12]. Genius is a website that's well known for its collection of music lyrics; the website also holds valuable data about song collaborators. (An example of this collaborator information is shown below, in Figure 1.)

"WEIRD FISHES / ARPEGGI" TRACK INFO

Written By	Philip Selway, Ed O'Brien, Colin Greenwood & 2 more
Mixing	Nigel Godrich
Music Video Directed by	Tobias Stretch
Mastered by	Bob Ludwig
Engineer	Richard Woodcraft, Nigel Godrich, Hugo Nicolson & 1 more
Release Date	October 10, 2007
Interpolated By	Radiohead Meets The Police - Live Looping Mashup by Elise Trouw
Cover By	Weird Fishes / Arpeggi by Alexa Melo 4 more
Performed Live As	Weird Fishes/Arpeggi (Live From The Basement) by Radiohead

Figure 1: Genius collaborator information for the song *Weird Fishes / Arpeggi* (by Radiohead) [13]

The Genius scraping strategy mimicked the technique we used while scraping Spotify: we'd query Genius's API with the title / artist pairs from our dataset, and verify the results using Levenshtein string comparison. From our original list of 432,005 MSD songs, we were able to find the required data (acoustic features, collaborator information, and hit status) for 183,095 songs; of these, 5,043 were hits.

3.2 Social Network Creation

Once all of the collaboration data had been collected, we moved into crafting the co-collaboration network. In Genius, each collaborator is associated with some unique artist ID; we figured that these IDs could serve as a basis for disambiguating collaborators. For the most part, this was a fair assumption.

For each Genius artist ID in our collaboration data, we created a single node in the network. Nodes had an (undirected) edge drawn between them if the collaborators associated with the nodes had worked on a song together; edges between nodes were weighted according to the frequency of collaboration. We also attributed a "type" list and a "maxType" to each node. The type list was a list of the various labels ("Written By", "Mixing", "Mastered By", etc.) Genius associated with each collaborator, and the maxType was the most common label associated with that collaborator across all of their songs. The network is shown below, in Figure 2:

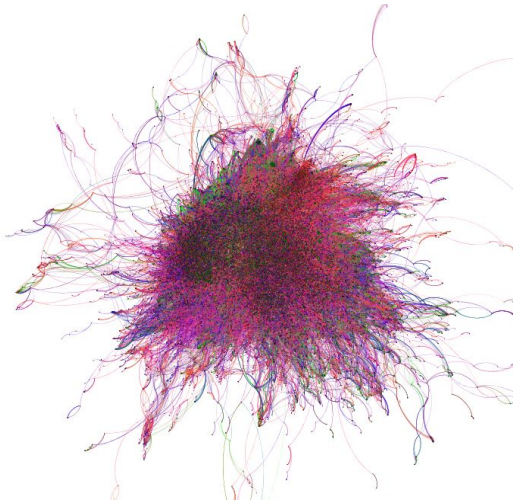

Figure 2: The largest connected component in the music industry co-collaboration network

In total, the network consisted of 69,385 nodes, and 704,872 edges between these nodes. 58,119 of the nodes are connected in a single connected component; the remaining 11,266 are distributed amongst a number of *much smaller* connected components.

3.3 Auto-Encoding the Network

Once we'd created the network, we aimed to embed each of the nodes into a vector space. That way, we'd be able to use the network as training data for a neural network. In order to achieve this, we would need to use a network auto-encoder. Auto-encoding is an area of machine learning concerned with learning feature representations for data that minimize some loss function. Generally, this loss function is related to some "reconstruction" of the original data from the feature representation. There are a number of proposed techniques for network auto-encoding; for our project, we chose node2vec [14]. node2vec encodings aim to preserve a node's "neighborhood"; this neighborhood is determined by a random walk of the network, starting from the node we're interested in.

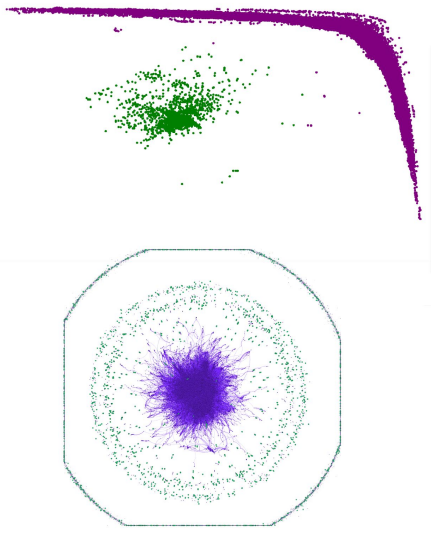

Figure 3: A 2-dimensional embedding of our co-collaboration network (shown above), and the network itself (shown below), colored according to the coloring of the node2vec embedding.

Figure 3 is an example of how node2vec is able to preserve some level of the network's structure, even in just two dimensions. We used node2vec to produce multiple encodings of the network at different dimensionalities, which we'd later use as training data for a neural network.

Each embedding represents an artist's position in the network vector space; since we wanted to make predictions over a set of songs, though, we needed some way of encoding a song as a combination of the embeddings of its collaborators. For this, we chose to simply average the embeddings associated with a particular song's collaborators:

$$song\ embedding = \frac{\sum collaborator\ embedding}{\#\ of\ collaborators}$$

4. Experiment

Our experiment stage involved building and training a variety of neural networks. Each of these networks would be trained on some combination of the song features, and output a prediction as to whether or not that song was a hit. We first created the training and test sets from our overall dataset. We took all 5,043 songs that were hits, and then randomly subsampled from the non-hits to create a 50/50 representation of the two classes.

4.1 Audio Features Network

The first network solely uses the thirteen high level audio features from Spotify as input, with binary classification of hit or non-hit. This network consists of four fully connected layers, with hidden layers of 256, 128, and 32 nodes. The output layer is run through a sigmoid function and then rounded to get the predicted class. For training, we used the RMSprop[15] optimizer with binary cross entropy loss with logits for a loss function.

The net gives an accuracy of 71.39% with a small amount of overfitting. The final cross-entropy loss after 10 epochs was 0.382.

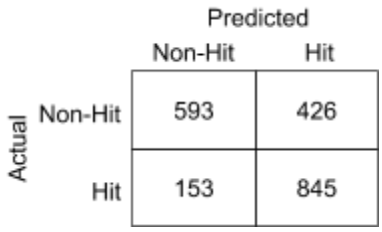

Figure 4: Audio Features Only Confusion Matrix

The precision and recall on the validation set were 0.665 and 0.847, respectively. The F1 score for this model was 0.745. The confusion matrix shows a decent amount of false positives, with around two thirds less false negatives.

4.2 Co-Collaborator Embeddings Network

The second network uses only the co-collaborator network embeddings; it's the counterpart to the Audio Features network. This network consists of five fully connected layers, with hidden layer sizes of 1,024, 512, 256, and 128. Similarly to the Audio Features network, the output layer is run through a sigmoid function, and then rounded to get the predicted class. For training, we used the Adam [16] optimizer with binary cross entropy loss with logits for a loss function. This net is able to perform significantly better than the Audio Features network.

After 10 epochs of training, we have an accuracy of 84.38% on the validation data. This has led us to believe that the network embedding is largely more indicative of a song's chart success than the high-level acoustic features!

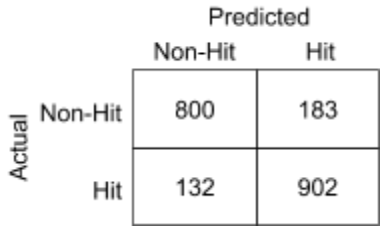

Figure 5: Network Embedding Only Confusion Matrix

The precision and recall on the validation set were 0.831 and 0.872, respectively. The F1 score for *this* model was 0.851; clearly, we can see that the Network Embedding model is outperforming the Audio Features one.

We also ran some experiments using different versions of the network embeddings, each with different dimensionalities. (Figure 7 demonstrates the impact of choosing different sizes.) After experimenting with hyperparameter tuning for the network, we eventually decided on using 10-dimensional node2vec embeddings.

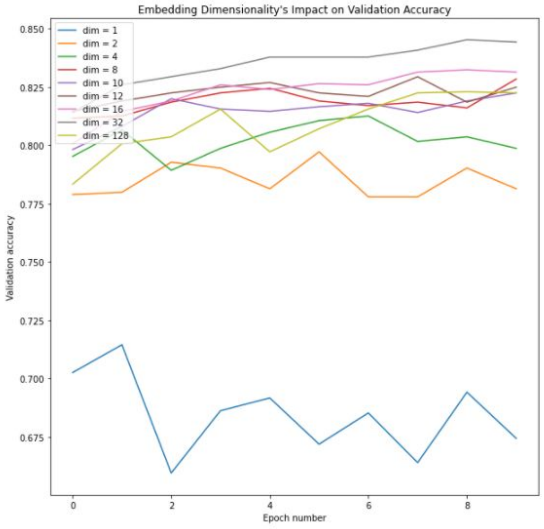

Figure 6: Impact of embedding dimensionality on Embedding Network validation accuracy

4.3 Ensemble Networks

With models for both acoustic and collaborator data created, we wanted to figure out a way to combine the data into a single, stronger model. We came up with four different strategies for making this combination. The first of these was our Bagging strategy. With this, we took the output scores of our audio and embedded models, passed them through a sigmoid to keep them between 0-1, and then averaged them together. The result was then rounded to get the class score of either 0 or 1. We found that this strategy performed as well as the sole embedding model, without improving the accuracy in any significant way.

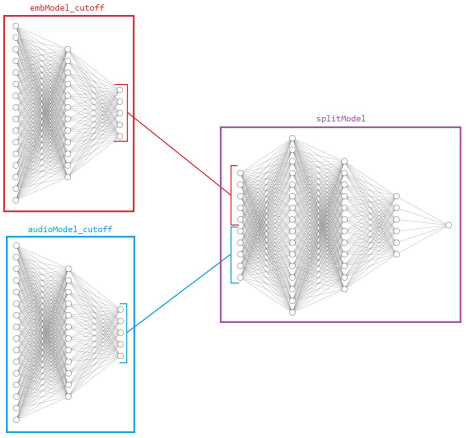

Figure 7: Split Network Architecture

The second combination strategy was our "Split" strategy. With this, we have two separate models that have the same architecture as the normal audio and embedding ones, except here, the final layer is removed. The outputs of either model are concatenated together, and then used as the input to a third, five-layer fully-connected "splitModel" network. The output of the "splitModel" is used to calculate loss, which is then backpropagated through all three networks. We found that this mixture does perform better than the individual networks. This arrangement regularly achieves an accuracy of 85%, commonly reaching 86%, and topping out at 87% occasionally.

The third strategy was a return to simplicity. We concatenated both the audio and network features together, then used this as the input to a five layer neural network. This model tends to underperform, reaching accuracies of 84% - about similar to the sole embedding model.

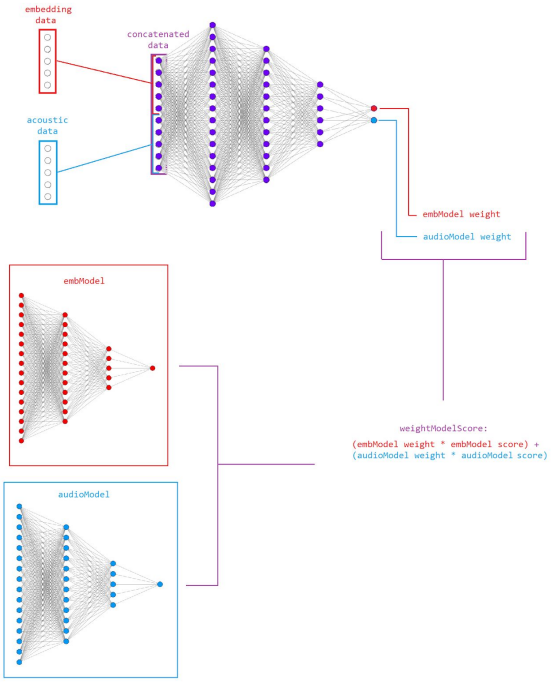

Figure 8: WeightNet Architecture

Our final mixing approach was what we called "WeightNet". Here, we were trying to use some of the intuition behind the Bagging model. The main difference is that the combination of scores from each model will not simply be an average, but a learned weighting from a third network. Using two pretrained networks (theAudio Features network and the Embedding network), we trained a third "WeightNet" that takes a concatenation of these features and outputs two scores. These scores are used as weights for the linear combination of the scores given by the two individual pre-trained models. This strategy performed about as well as the "Split" strategy, reaching accuracies of 86%; due to the increased complexity, thought, it's a slightly inferior design.

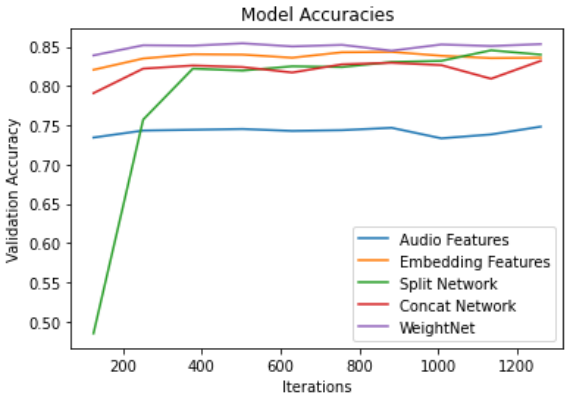

Figure 9: Model Comparisons

Model	Highest F1 Score
Audio	0.74505
Embedding	0.85134
Bagging	0.85230
Split	**0.86077**
Concatenation	0.83967
WeightNet	0.85701

Figure 10: Highest F1 scores of each model

5. Conclusion

Overall, we're very happy with the results of our project. We successfully built a co-collaboration network and auto-encoded it into a vector space. We were able to utilize this vector space to create individual song encodings; these encodings have resulted in substantial increases in hit song prediction accuracy. Finally, through a variety of ensemble methods, we have built a network that leverages both these new collaborator encodings with traditional song audio features, to further increase the hit classification accuracy.

In the future, we'd want to look at expanding the Hit Song Science Problem from its current form (simple binary classification). By considering the time period that a song is on the charts, we might be able to build a more detailed predictor. When building our dataset, we used a single appearance on the Billboard Hot 100 chart as determining the boolean "hit" value. In future work, we hope to include information about the dates and durations of songs on these charts. With a little more work, we could build a recurrent neural network that attempts to predict the popularity of a song in a given time period.

Additionally, there are a number of ways we could have expanded the creation and use of the industry network embeddings. Research into "attributed network auto-encoders" is burgeoning; in addition to encoding the topological structure of a network, these attributed network auto-encoders will *also* try to encode any nodal attributes [17]. Encoding the "type" attributes of each node in the co-collaboration network (i.e., "Primary Artist", "Guitarist", "Producer", etc.) might have led to better prediction accuracies!

Furthermore, there are a number of additional potential uses for the auto-encoded co-collaboration network. Music recommendation systems that use "artist proximity" could have interesting results. It also could be interesting to try and track artist influence in the network - this might give you a better understanding of how the landscape of popular music evolves with time!

6. Resources

As follows is the code + dataset we created:

6.1: Project Github Repo

https://github.com/tmhubbard/CS-682-Hit-Song-Science-Project

6.2: Project Datastore

https://mega.nz/folder/0LoxhagD#Ngs65bJk2_mtWzSCFdIpWg

References

[1] Withey, Conrad. “In A&R, 'Gut vs. Data' Isn't a Binary Choice.” *Music Business Worldwide*, 8 May 2018, www.musicbusinessworldwide.com/in-ar-gut-vs-data-isnt-actually-a-binary-choice/.

[2] Eva Zangerle, Ramona Huber, Michael Votter, and Yi-Hsuan Yang. “Hit Song Prediction: Leveraging Low- and High-Level Audio Features.” In Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR 2019), 2019

[3] Elena Georgieva, Marcella Suta, and Nicholas Burton. “HitPredict: Predicting Hit Songs Using Spotify Data” Stanford Computer Science 229: Machine Learning (2018)

[4] Myra, Interiano, et al. “Musical Trends and Predictability of Success in Contemporary Songs in and out of the Top Charts.” *Royal Society Open Science*, 16 May 2018, royalsocietypublishing.org/doi/full/10.1098/rsos.171274.

[5] “The Hot 100 Chart.” Billboard, www.billboard.com/charts/hot-100.

[6] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. “The Million Song Dataset.” In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.

[7] Guo, A. Python API for Billboard Data. Github.com. Retrieved from: https://pypi.org/project/billboard.py/

[8] “Web API.” Spotify for Developers, developer.spotify.com/documentation/web-api/.

[9] “Get Audio Features for a Track.” Spotify for Developers, developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/.

[10] Hilden, F. A Python package for using Spotify API. https://pypi.org/project/spotipy/

[11] Algorithms and Theory of Computation Handbook, CRC Press LLC, 1999, "Levenshtein distance", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 15 May 2019. (accessed TODAY) Available from: https://www.nist.gov/dads/HTML/Levenshtein.html

[12] “Genius API Documentation.” Genius API, docs.genius.com/.

[13] Tasker, Reuben. “Radiohead – Weird Fishes/Arpeggi.” Genius, 10 Oct. 2007, genius.com/Radiohead-weird-fishes-arpeggi-lyrics.

[14] Grover, Aditya, and Jure Leskovec. “node2vec: Scalable Feature Learning for Networks.” *ArXiv.org*, 3 July 2016, arxiv.org/abs/1607.00653.

[15] Hinton, Geoffrey. Neural Networks for Machine Learning Lecture 6e. 2012, www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[16] Kingma, Diederik P., and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” *ArXiv.org*, 30 Jan. 2017, arxiv.org/abs/1412.6980.

[17] Dave, Vachik S., et al. “Neural-Brane: Neural Bayesian Personalized Ranking for Attributed Network Embedding.” *ArXiv.org*, 20 Aug. 2018, arxiv.org/abs/1804.08774.