Live Musi(que)?

Rahul Khanna, Zerui Xie
rahulkha@usc.edu, zeruixie@usc.edu

1 Introduction

As Covid-19 has kept us all indoors, the prospect of live music seems a distant possibility. However, in preparation for the probable deluge of live shows (both from pent-up energy and financial reasons), one can spend this time discovering new artists/bands[1] to see live. Some common problems a person might face in discovering new artists might be lack of awareness, ability to judge live performance, and information pertinent to an artist being spread across the internet. We attempt to use knowledge graphs and machine learning to help solve these problems.

Live Musi(que)? is a prototype service that employs the power of knowledge-graphs, graph embeddings, natural language processing (NLP) and neural networks to help users find artists to follow. This service puts together 1) Live Music Data (Songkick.com), 2) Award Data (Wikipedia.com), 3) Discography Data (Musicbrainz.com) 4) Popularity Data (Billboard.com) and 5) General Artist Information (Wikipedia.com) to power an artist recommendation service.

Our recommendations are not just backed by a knowledge graph and a neural encoding process, but are also practical; making sure that recommended artists actually visit a particular city (when a city is provided). We also attempt to provide a summary of all reviews that an artist has received on Songkick by utilizing NLP to measure overall sentiment, extract top adjectives used and generate an abstractive summary of all reviews. Finally we provide several rankings based on location, sentiment-score, songkick-rank, genre and record label membership, as well as some basic information about an artist. In all we hope our service will enable users to find live performances to see in the (hopefully) not so distant future.

2 Challenges

Crawling: Our main challenge stemmed from there not being a directory of artists' with pages on Wikipedia. While one can't expect such a directory to exist, it does prohibit automatic link collection. To solve this issue we took advantage of Wikipedia's search API to collect Wikipedia urls and utilized Songkick's artist directory as a source of artist names to look up. This actually made our entity linking strategy a lot easier. We also ran into standard challenges of being rate limited.

Scraping: While the apparent structure of tables make them seem easy to scrape, the actual css and html used to style tables differ greatly. This made scraping award data information quite challenging, and we had to develop several cleaning functions that would standardize a table's format. We were actually planning on using Wikipedia for discography information on artists, but the tables were very unwieldy and so instead used MusicBrainz's API.

Entity Linking and Resolution: As mentioned, we used Wikipedia's Search API to retrieve Wikipedia urls for artists that we had found on Songkick. By using this API, we didn't have to worry about linking Songkick Records with Wikipedia Records, as per Songkick record we retrieved a single Wikipedia url to use for scraping. To ensure we selected the correct Wikipedia url from the search results we gathered the title of the page and the first few sentences of the page (both provided by the API) and looked for either, the artist's name, mention of a musical genre (rap, hip-hop, jazz, etc.) or a mention of a musical profession (singer, drummer, musician, etc.). By relying on a combination of Wikiepdia's search-intelligence (their results are ordered by relevancy) as well as our rule based matching we saw very high accuracy in matching the correct Wikipedia url to an artist's name. This is supported by both a spot check, and also by the fact that for over 5K artist names, only 10 map to a non-unique Wikipedia url.

Once we linked Wikipedia and Songkick, we linked to MusicBrainz by utilizing MusicBrainz's search API and ensuring the entity type was a *Person* or *Band*. As all entities on MusicBrainz are already musically inclined, we didn't have to employ our rule matching strategy.

[1] will use artists from now on to reference both

The final notable linking that had to occur was between Billboard Top 100 and 200 weekly lists to the artists in our graph. We did this by first checking if two artists' names matched and then checked if that song/album was in candidate artist's discography (collected from MusicBrainz).

Ontology Mapping: In developing our Ontology we made sure each class was an extension of a class in a larger ontology. We utilized common ontologies such as DBPedia and Schema.Org for this purpose. In total our ontology consists of 10 and 16 custom classes and relations, and 3 and 21 classes and relations from larger ontologies.

Summarizing Live Performances: In an attempt to summarize an artist's live performance reviews, we first concatenated all reviews into one large review. Next, we utilized Vader Sentiment (Hutto and Gilbert, 2014) to calculate a sentiment score. We noticed that our sentiment scores were incredibly biased towards being positive (expected), so we calculated sentiment z-scores to showcase relatively how enthusiastically a fan base is about an artist. We also used a combination of spaCy[2] and sklearn[3] to extract the top 10 adjectives by tf-idf weighting used to describe an artist. A document in our tf-idf process was a collection of all the adjectives used to describe that artist. Finally, we used a pre-trained PEGASUS (Zhang et al., 2020) model to output a short summary of all reviews. Admittedly this last step has flaws, but it was a chance to work with a state-of-the-art model.

Graph Embedding: To power our recommendation service we needed to develop vector representations for each of our artists. To do this we sliced our graph in various ways and created a list of relations we thought would be useful in identifying a particular artists. We converted certain relations in our graph to better link artists together, for example we created a relation for each *rank* possible in a billboard weekly top 100/200 chart (rank_of_i), where the object of that relation is weekly listing the artist appeared on (billboard-top-100-weekDate). In this way artists who were on the same weekly list are connected by a common object, and artists who appear in similar positions on the top 100/200 lists will share a common relation.

Next we ran several experiments with various encoding strategies (Complex, DistMult) and embedding sizes to encode the entities in our filtered/manipulated graph. We found that using DistMult and an embedding size of 500 worked best; holding all other parameters the same as in the Ampligraph Tutorial[4].

These embeddings then act as seed embeddings for a compression neural network, that outputs a final compressed embedding to represent an artist. Our compression network is trained using TripletMarginLoss, where the positive examples of similar artists (per artist) are a union of the following criteria: artists who tour together (Songkick), similar artists (Wikipedia), artists who have collaborated (MusicBrainz discography data). We sample one negative example per pair of positive as per (Schroff et al., 2015). Additionally we utilize an Embedding Layer, similar to NLP nets, and treat each artist as a token, in this way the gradient flows all the way back to the initial DistMult embeddings. We also use a dropout of 0.5 in our Embedding Layer, to help prevent over-fitting, as our positive labels are not exhaustive, nor are our negative labels always accurate. We find that we can compress the embedding size to R^{50} while keeping loss comparable to much higher dimensions. In the model that we use for finally encoding our artists, our dev-set loss drops from 0.7 to 0.26 points, showing that the model learns to cluster similar users together, while reducing the size of our embeddings per artist[5].

Recommendations: To produce recommendations we offline compute a batch dot-product between normalized vectors of each artist and sort by largest to smallest (cosine); recommendations are filtered if needed based on additional provided information (genre, city, record label).

Our video can be found here: `https://youtu.be/uf6pFKJTbhk`

3 Conclusion

Our project is a prototype of how one could build recommendation systems using knowledge graphs and the plethora of machine-learning tools now available. We hope to explore additional problems that our knowledge graph might be able to solve, like award show prediction, which we unfortunately did not get to. We are encouraged by our net's ability to cluster representations together, and hope to collect user feedback on our recommendations to update our strategy.

[2] https://spacy.io/

[3] https://scikit-learn.org/stable/

[4] https://docs.ampligraph.org/en/1.3.2/tutorials.html

[5] graphs will be included and linked in our video

References

C. Hutto and E. Gilbert. 2014. VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. In *ICWSM*.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A Unified Embedding for Face Recognition and Clustering. *arXiv:1503.03832 [cs]*. ArXiv: 1503.03832.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2020. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. *arXiv:1912.08777 [cs]*. ArXiv: 1912.08777.