**Spotify Project**

**Team 20**

**Ankit Bhargava, Erik Subatis, Mark McDonald**

**TA Advisor**: Rashmi ([rjain29@gmail.com](mailto:rjain29@gmail.com))

**Project Repo**: <https://github.com/subatis/CS109a_finalproject_group20>

**Project Database**: <https://drive.google.com/drive/folders/14OBw3t3gKwPgxX3tx_ogHHpXpx6iWXRC>

**Table of Contents:**

[Summary](#summary)

[Data Structure](#data_structure)

[EDA](#eda)

[Models](#models)

[Scores and Comparisons](#scores_comparisons)

[Summary](#summary)

|  |
| --- |
| **Summary** [(back to index)](#table_of_contents) |
| **Project Statement**  At its core, the project seeks to answer the question:  *“How do I generate a desirable playlist for a listener?”*  More specifically, the project seeks to answer the above question *given* a context consisting of one or many songs provided by the user. The result is a playlist of 20 songs that are similar to the song(s) provided by the user. The recommendations are primarily based on preferences from other users measured by other users’ playlists. |
|  |

|  |
| --- |
| **Data Structure** [(back to index)](#table_of_contents) |
| <explain data received and how it was organized>  **Database**  The project data has been stored in a database for uniformity, speed and simplicity of access. In addition to the playlists provided, tables were added to include additional data for tracks and artists.  **Utility Functions and Project API**  A set of API wrappers were created to streamline access to Spotify data and the Team Database. These API’s simplify Spotify authentication and avoid the need to navigate arguments that are not relevant to the project.  **Data Inconsistencies and Corrections**  Various records needed to be deleted or changed after reviewing their validity. Some artist URL’s had changed in Spotify and some tracks were no longer available. These changes were not extensive and did not have a major impact on the usefulness of the data. |
|  |
|  |

|  |
| --- |
| **EDA** [(back to index)](#table_of_contents) |
| **Playlists**  The Playlists table is extensive with 999k playlists and 66M tracks    There are some outliers with very long lengths, but the average playlist is 50 songs long and the most common length is 20. |
| **Artists**  In the playlists, 296k unique artists exist:    By examining the number of appearances in playlists, we are able to determine the popularity of Artists based on our dataset. Spotify also supplies a field called ‘artist popularity’; however, we found that a majority of Artists in our dataset had a popularity of 0, so we will not rely on the Spotify popularity data.    Spotify supplies genres by artist. After extracting this data, we determined that over 60% of artists had no genre assigned by Spotify, so we will not rely on this data. |
| **Tracks**  2.2M unique tracks can be found in the playlists.    After querying data from Spotify, various additional useful fields are available for each track. Values are assigned to a significant portion of the population making these features useful for building recommendation lists. Distributions of these features are available in the accompanying notebook. |
|  |

|  |
| --- |
| **Models** [(back to index)](#table_of_contents) |
| <summary of models and explanations> |
| **Spotify Recommender Model Types** For recommendations, 2 types of recommendation strategies:   1. **Content-Based**   Predicts based on what a user has listened to in the past. Uses features of songs to find similar songs.   1. **Collaborative**   Predicts based on what other listeners like Focuses on what songs other users liked who also liked a chosen song. |
| **KNN-Classification**  **<description of KNN classification>** |
|  |
|  |
| Word2Vec Word2Vec is a process that uses vectorized words to predict other words. It does this by ingesting a series of documents, parsing out the words, vectorizing the words and then using the vector representations to predict other words. The vectors are built in such a way that each word has a unique vector that is based on its usage in the documents. The result is a vector space filled with words where related words have vectors that are similar. This vector space is referred to an an **embedding**. This embedding is used in two common word prediction tasks: Skip-Gram and Continuous Bag of Words.  **Skip-Gram**  The Skip-Gram model asks for a single word and then predicts words surrounding the word.  **Bag-of-Words**  The bag-of-words model asks for a series of words and will return the missing word.  For the Spotify Recommender, we will use Word2Vec to assign vectors to Songs by providing the model with a series of playlists instead of documents.  **Embeddings**  The vectorized space of words is referred to as an embedding. This embedding is used to train a Skip-Gram or a Bag-of-Words model. The embedding without the models is quite useful. It represents a vectorized vocabulary of words where vectorized words can be added or subtracted from one another to find the sum or difference of their meanings. Synonyms of words are other vectors with that are nearby in the embedded space. Below, we will use this embedding to create a playlist without a model and compare it to playlists that are used with the models described above. Making a Playlist To make a playlist, we simply convert Songs to Vectors and then find new songs by finding other songs with similar vectors. To achieve this, we can use the Bag-of-Words or Skip-Gram approach as mentioned above. Provide a song, a Skip-Gram model can supply a playlist. Provide a list of songs, and Bag-of-Words model can give you the next song.  3 approaches using Word2Vec and Embeddings are explored: 1. Embeddings from Playlists - Song ID - Unsupervised Here, we will take data from Spotify that included 1M playlists and the songs in each playlist. We'll use the Word2Vec process supplying playlists as documents and each song's unique id is used as the word.   After the embedding is created, we can skip the creation of building and training a BOW or Skip-Gram model. All we need to do is find vectors that are similar to a song or a list of songs. 2. Embeddings from Playlists - Song ID - BOW We can use the same embedding to create a BOW model. 3. Embeddings from Playlists - Song ID - Skip-Gram Let's use the embedding from the playlists and use Word2Vec to create a Skip-Gram model. |
|  |
|  |

|  |
| --- |
| **Scores and Comparisons** [(back to index)](#table_of_contents) |
| <metrics for each model and comparisons> R-precision R-precision is the number of retrieved relevant tracks divided by the number of known relevant tracks (i.e., the number of withheld tracks):  R-precision=∣∣G∩R1:|G|∣∣|G|.  The metric is averaged across all playlists in the challenge set. This metric rewards total number of retrieved relevant tracks (regardless of order). Normalized discounted cumulative gain (NDCG) Discounted cumulative gain (DCG) measures the ranking quality of the recommended tracks, increasing when relevant tracks are placed higher in the list. Normalized DCG (NDCG) is determined by calculating the DCG and dividing it by the ideal DCG in which the recommended tracks are perfectly ranked:  DCG=rel1+|R|∑i=2relilog2(i+1).  The ideal DCG or IDCG is, on our case, equal to:  IDCG=1+|G|∑i=21log2(i+1).  If the size of the set intersection of G and R, is empty, then the DCG is equal to 0. The NDCG metric is now calculated as:  NDCG=DCGIDCG. |
|  |
|  |

|  |
| --- |
| **Conclusion** [(back to index)](#table_of_contents) |
| <findings> |
|  |
|  |

**NOTE**:

To convert this to Git markdown:

pandoc -f docx -t gfm Spotify\ Project-ProjectDocument.docx -o README.md