Music Recommendation System

Springboard Data Science Bootcamp 2020

Capstone Project 1

Joseph Tran

Introduction

This report summarizes the first in a series of two capstone projects during the course of the data science career track at Springboard, and is a compilation of the intermediary reports submitted over the course of the project, from initial project ideas to an in depth report on the final deliverable. After considering a few different project ideas, I chose to build a music recommendation system. This project was chosen because, while having some experience with machine learning models, I've never studied recommendation systems and their associated algorithms. In addition, my course mentor is an expert in the field.

Initial Project Ideas

Recommender system

I am interested in building a recommender system for music. Recommender systems are implemented in many fields to increase revenue for its business domain. I have studied some data science algorithms, however I have no experience or knowledge with recommender systems and therefore would like to use this opportunity to gain some knowledge in the area. I chose a music recommender because I have the most domain knowledge in that field being a musician and having studied music. I have done some cursory research and it seems there is a lot of interest in music recommenders for mood regulation, which is an area I am interested in. The data I would use for this project is the nowplaying-rs dataset which was released in 2018, and was designed for research in context-aware recommender systems.

Luxury watch company business analysis

It has been brought to my attention from a family member that a high end luxury watch-maker in my area is experiencing cost overruns and inefficiencies in its shipping and marketing departments. I would like to do an analysis of the business practices and methods implemented at the company and use its data to find areas in which the company could increase its bottom line by specifically identifying the inefficiencies. The data I would use is proprietary business data that I would have access to if I so choose to proceed with this project. My main concern is that I have not seen any of the data and do not know what condition it is in, or if it has been properly collected.

Plastics manufacturer website/ business analysis

I have a relationship with a company I formerly worked for that sells its entire catalog via its website. I am interested in exploring historic demand for certain products in order to forecast demand for upcoming seasons to help the manufacturer maximize its profits while reducing inventory. I would use proprietary business data generated from the website in addition to historic sales data.

While each project idea provides opportunity for gaining experience in the data science field, I chose to make a recommendation system for a few reasons: I've never been exposed to recommendation systems, the data for the project was readily available, my course mentor is a subject matter expert, and these types of systems are ubiquitous in many facets of our digital lives.

Project Proposal

The problem I am interested in learning more about are recommender systems, particularly in the music field. Being that I am a musician and have studied music, I feel it is an area where I have the most domain knowledge. I, like many others, have benefitted from good recommendations that have brought new music into my life that I am appreciative of. While many people benefit from recommendations generated from a good system, the target client of this product would be music streaming services. The idea is that revenue for the service could be increased by continuously introducing its users to new music and artists that they would otherwise not have been introduced to. This results in a lower churn rate for the client which, in turn, allows them to retain customers while continuously adding to its customer base. Additionally, this product could be refined for a specific genre and deployed as a website where people interested in a genre, but do not know where to start, could input an artist or song and the website would return a list of tracks or artist the user would most likely enjoy. In this case, the client would be the users of the website.

The data set I will be using for the project is the nowplaying-rs data dataset. There are other sets available for this type of project however, I chose this data set because the authors designed it to be the first data set offered for research in context-aware recommendation systems. The data comes in three files with content/listening event info with musical attribute info gathered from spotify, hashtag data related to the song, and sentiment scoring for those hashtags. The individual sets are designed such that they can be joined together to combine all the info in one dataframe. An immediate issue I see with the content data is that there are no song, artist, or track names associated with the listening event. What is provided are long strings of numbers that correspond to spotify URL's, or uniform resource locators. In order to gather this related meta-data I intend to use the Spotify API to pull the relative info to be able to join with the cleaned data set. I'm not sure that this is necessary for this project as I would be looking at the metric that scores the algorithm. However, it would be nice to have that data to

make sense of what songs and artists are being recommended, rather than the long unidentified string of a URL. However, this data would be beneficial for the second capstone project in which I plan on extending the first to a more refined and sophisticated model.

Recommender systems range in complexity from simple content based collaborative filtering methods, to deep learning hybrid systems that incorporate meta-data related to the item. For the capstone one project I intend to use content and model based collaborative filtering methods to create the recommendation system. I will implement two collaborative filtering models, one using the alternating least squares algorithm with the Implicit library, and another matrix factorization model using the lightfm library. I will then benchmark these systems against one another to investigate the performance between the two using AUC. If possible, I would like to try a couple different libraries, specifically scikit-learn and light-fm to run the models and also benchmark against the different libraries. The idea is to gain an intuitive understanding of the more standard models in order to be able to extend this project to the second capstone where I would like to implement a hybrid system.

I plan for the deliverables for this project to be the code used to implement the model, supplemented with a detailed paper that outlines the steps I used to clean the data and create the model. The paper will include a detailed reasoning of the decisions I made while creating the model, what I learned while completing the project, and the benchmark results gathered from performing the analysis.

Data Wrangling

The data wrangling report encompasses information about the dataset used for the project of building a music recommendation system, and the steps taken to prepare the data for a machine learning model. The dataset chosen was the #nowplaying-RS dataset and was published in Jan. 2020. It was constructed by scraping song and hashtag information from Twitter with the intention of creating a dataset capable of context and content aware recommendation systems, and was created by Eva Zangerle from the University of Innsbruck Austria. The #nowplaying-RS dataset is comprised of three separate files: a listening event dataset that includes user id's, track id's, hashtag, and timestamp that contains roughly 17 million listening events. A sentiment file that contains sentiment scores for the hashtags. Lastly, the dataset also includes a file containing various information about context and content. Mainly, it includes the Spotify music attributes obtained from the Spotify API.

The main object required for a non-negative matrix factorization algorithm is the user-item matrix. This can be obtained solely from the listening event file. However, the three files will be merged so all information is contained in one dataframe. Merging the files will also help to reduce observations and thus, reduce the sparsity of the user-item matrix.

The main file is the listening event file that contains user id's, track id's, hashtags, and timestamps. I chose to work with this file first as it contains the information needed to create the user/item matrix necessary for a matrix factorization model. The data was inspected by calling some of the dataframe attributes and was found to have 138,223 unique users and 344,536 unique songs, with a shape of (17560113,4). Inspection of the head reveals identical rows. This was identified by looking at the 'created_at' column and seeing matching timestamps. These duplicate rows were generated from a 'one to many' relationship with a listening event and hashtags. The duplicate rows will be preserved until the datasets are merged. It is advantageous to save all the corresponding hashtags associated with a listening event. Dropping the duplicate rows at this time would discard relevant hashtags. Next, null values were removed as there was only a single null value in the hashtag column.

The next file that was worked on was the sentiment values dataset. The file contains sentiment scores from four popular sentiment dictionaries; AFFIN, Opinion Lexicon, SentiStrength, and Sentiment Hashtag Lexicon. Scores were only assigned to unique hashtags that were scored by at least one of the sentiment dictionaries. In addition to the score, minimum, maximum, sum, and average scores for the four dictionaries are included. Of the four dictionaries used, scores were only assigned to 5290 of the 32208 unique hashtags in the main file. Upon inspecting the head of this dataset, it was found that columns were mis-aligned and nested in a hierarchical index under the column name 'hashtag.' After inspection it was found that the first four columns did not have column names and somehow got thrown into the multi index. To fix this, column names were created with the dictionary name abbreviation and score, indicating that the column is the score generated from the dictionary. The new column names are assigned to the column in the order the dictionaries appear in the dataset. The multi-index column name 'hashtag' is renamed to 'ss_score' indicating the score for the sentiment dictionary. Afterwords, the new column names are assigned to the different levels of the multi-index, and the index is reset to create a standard range index. This is verified by calling type on the index. The only information that is desired is the hashtag, sentiment score, and average. Some of the sentiment scores have null values so the average score column is retained to impute those missing values row-wise, rather than imputing the mean or some other transformation column-wise. A key-error was produced on the first attempt to drop the columns. Upon inspection, it was found that many of the field names contained leading white spaces. The columns are renamed to a standard form and then dropped from the dataset. When checking the null values of the resulting dataframe, it was found that the score columns for each dictionary contained 1423 null values. Instead of taking the mean of the column and imputing, the average score for each dictionary is imputed row-wise from the corresponding average score column. After the imputation, the null values are put into a bar chart and it is found that the ol_score resulted in the least amount of null values. This column is selected for our data and the remaining column dropped, afterwhich the null values are dropped from the data. The resulting dataframe consists only of the 'hashtag' and 'ol_score' columns, with all null values dropped from 'ol_score,' leaving 4831 unique hashtags, and their score from the Opinion Lexicon sentiment dictionary.

The last file to be worked on was the context/content file containing Spotify attributes and other contextual information. The columns: 'coordinates','id','place','geo','entities','time_zone' are dropped as they are not necessary for our model. The ‘tweet_lang’ and ‘lang’ columns will also be dropped, but first the dataset is filtered based on English. Once the filtering has been done these two columns are also dropped.

The next data wrangling process was to merge the three datasets. First, the listening event dataset is merged with the sentiment dataset with an inner join on ‘hashtag.’ This merged dataset is called ‘temp_df.’ Finally, all three sets are combined by merging the content/context dataset with temp_df with an inner join on 'track_id','created_at','user_id.’

With all the data in a single frame we can now preserve the hashtags and drop the duplicated rows. This is achieved by creating a dictionary to store the hashtags associated with a listening event. The values are added to the dataframe by mapping the dictionary keys to the user_id and track_id in the dataframe. After the new column containing the hashtags is added to the dataframe, the duplicate rows are dropped.

This is the point in the data wrangling process that we filter the track_id’s so that each track appears no less than 10 times. A temporary dataframe is built from a value_count operation that limits the track count to be at least 10. The track_id count dataframe is then joined to the main dataframe on the track_id column, returning the track_id’s with 10 or more plays. Lastly, the user_id, artist_id and track_id are assigned category codes for sequential ordering.

Overall, the length of the final dataset is reduced from 17.6 million listening events to 4.4 million, and contains columns from all three files in the dataset. The final dataframe is saved as a csv titled ‘NPRS_FINAL.csv.’

During the data wrangling process attempts we made to pull song metadata from Spotify API. The presence of the content features alerted me to the fact that these features were pulled from Spotify, and suggested I could use the track_id to pull the extra data. However, after spending several hours learning to connect to the Spotify API, I learned that the track_id and song_id were not Spotify labels, but hashed values intended to protect private information. Thus, I was not able to pull any metadata for the tracks. I reached out to the authors of the dataset, but they were unwilling to provide the hash keys. Not being able to pull the track and artist names from Spotify introduces the problem of not being able to qualitatively evaluate how the system is performing. With no track metadata, we will only be able to quantitatively assess the model. In this case the metric used to evaluate the model will be the area under the curve.

Statistical Analysis

The goal of this project is to become familiar with recommender systems, and the tools used to build them, focusing on model based collaborative filtering. This differs somewhat from a supervised model where a set of features, X, is used to predict a value for a target or class variable. Typically, statistical analysis is performed on the predictor variables to look for relationships, correlations, or test hypotheses one might believe about certain features. This project is different in the sense that to perform model based collaborative filtering only the user/item matrix is required. Being that the only input for the model is user/item interaction data, we can plot the distributions of the count of user events and track plays.

First, we create a column of ones to aggregate in a group-by operation, groupby by user id and track id separately and summing the count column. We now have two data frames representing each user id and how many plays they had, and track id's with play counts.

The initial distribution plot of users and listening events shows a highly right skewed distribution. This is due to a few users who had an unusually high amount of listening events. To reduce the skewness without eliminating these users with high listen counts we can perform a logarithmic transformation. After the transformation we get a clearer view of the data which appears to be exponentially distributed. We can also filter and keep only the users with at most 200 plays. This removes the users with the unusually high song plays, and accounts for 94% of the users in the dataset. What we see after plotting is a distribution that looks similar to the distribution generated from log-transform of the counts. Finally we can plot a cumulative distribution for the filtered data and see that users with between 10 and 15 plays account for roughly 25 percent of the filtered data.

We also plot the distribution of play per track. We can see in the first plot that again the data is right skewed. Performing the log transform yields a distribution similar to the user play counts showing a distribution that looks exponential. Again, we can filter the outliers by removing the track id's with play counts over 250. By filtering songs with 250 or less plays, we retain 94% of the data. Doing this yields a distribution similar to that of the log-transformed data. As we can see, there were about 16000 songs with between 15 and 20 plays. Lastly, the cumulative distribution shows that about 80% of the songs in the filtered dataset have 50 or less plays.

Data Story

Since the only object necessary for a basic matrix factorization algorithm is the user-item matrix, it is difficult to weave a story with only that information. However, the final dataset from the data wrangling process contained numeric content features that we can plot as time series to look at the behavior over time. In addition, there is the categorical information we can plot and look at to find descriptive characteristics.

The first plot is a tabulation of total listening events per day. A vertical marking the day with the highest song count is plotted for reference. The plot has a lot of dips in song count over the year 2014 time period. It is hypothesized that the dip in song count is due to the web scraper breaking, being fixed, and collecting information again. I apply a 7 day rolling average to the data for smoothing in the next plot.

The next eight plots in the notebook are time series plots for the daily average value of the feature plotted over the time period. The indication is that the population listens to popular music. I would expect to see features such as ‘instrumentalness’ to have low values, as a value closer to 1 indicates a track with no vocals. This would mean genres like classical or jazz, that are not popular. I expect that the mean daily value for this feature is low. The rest of the plots for the Spotify features follow this logic and suggest what one may think about the listening habits of the masses: on the whole, there is no tendency to radical listening habits.

In addition to the content features, bar plots are made for the user_id, track_id, artist_id, and hashtag combinations. For the matrix factorization model, the user_id and track_id represent the most relevant information as these are the values in the user item matrix.
I expected these charts to be boring and not reveal anything interesting, but I was mistaken. Plotting the counts for each user it was revealed that one user, user ‘303’ was not an actual user, but some time of bot. Looking more closely at the user revealed that the time series plot of LE’s generated by the ‘user’ was flat throughout the time period with around 303 LE’s recorded per day. In addition, there are no breaks in the data from when the scraper was not working. This is in contradiction to all the other time series plos. The user being assigned a code of '303' and the count per day of the user being 303 is a weird coincidence as the user code was generated from the hashed user ID. Being that a bot was discovered, this ‘user’ will be removed prior to constructing the user-item matrix.

The artist ID bar plot revealed that the artist with the most occurrences was artist ‘7582.’ This artist appeared more than 55,000 times, with a unique song count of 79 different songs. It would be interesting to see who this is, but it is not possible with the data presented.

The bar plot of the hashtag combinations show that the single hashtag ‘nowplaying’ occurs, by itself, way more than any other hashtag. The single hashtag of now playing is removed to view the next highest count of hashtag combinations. Many of these combinations also include ‘nowplaying’ with another hashtag. The hashtag combinations show that the majority of the time users are listening to some type of rock related music, whether it be punk, classic, doommetal, or rockmusic.

Lastly, some correlation plots are made for selected content features that would intuitively seem correlated: energy/loudness, energy/danceability, energy,valence, and speechiness/instrumentalness. Of the plots generated, only one showed some type of correlation which was energy/loudness. This makes sense as loudness is proportional to energy. The correlation seems to be vaguely linear as both features increase in value.

In Depth Analysis

This report summarizes the methods used to implement an implicit recommender system. In the analysis the implicit and lightfm libraries are used to implement collaborative filtering using matrix factorization. The implicit library implements alternating least squares to decompose the user/item matrix into latent user/item vectors used for predictions. The lightfm library offers the flexibility of creating a hybrid model, and offers several loss functions for the collaborative filtering algorithm. A model is implemented in the implicit library and the auc score for the predictions are compared to popular recommendations. Another model is implemented in lightfm, and the auc score is then calculated and compared with the implicit model.

Explicit v. Implicit Data

The data for recommendation systems comes in two flavors: explicit and implicit. Explicit data is user defined data. An example of explicit data would be a user supplied rating based on some type of scale. This type of data relies on user input to rate an item, and as a result much less is generated. At the same time, ratings will depend on the generosity of the user. A five star rating may mean something different to a professional critic, than it would for the average user.

Implicit data does not rely on user input, and is generated from a user interacting with the platform. As a result, the data is more widely available. The implicit data that we have for our model are listening events. The listening events can be used to infer whether a user liked or disliked a song. We can introduce a notion of confidence by looking at how many times a user interacted with an item. For example, if a user listened to a song 16 times, we can have more confidence that the user liked the song than if the user had only listened to it once.

Creating User/Item Matrix

In order to compute the latent user/item features using matrix factorization we need to create a user/item matrix where the rows are users, and columns are items. With smaller datasets, this can be achieved with the pivot function in pandas, and stored as a dataframe. However, user/item matrices are generally quite large and very sparse. This leads to memory errors when trying to perform the pivot in pandas. A better structure for the user item matrix is a compressed sparse row, or csr matrix. Not only is the csr matrix required to store the data, both libraries require that the input be in the form of a csr matrix. The data is loaded with the pandas read csv function loading only the user id and track id. A count column is created and the value is set to 1. This is so we can get a count of how many times a user listened to a song. Next, we group by user id and track id and aggregate the values in the count column by summing. This leaves us with a dataframe that contains the user id, track id, and the count of interactions per user and item as a confidence metric. Lists of unique user/item id's are created to specify the shape of the matrix, and category codes for each user/item are created for sequential indexing. The csr matrix is created by supplying the lists to the function and specifying its shape. In our case, we

generate a csr matrix with a shape of 21220 users and 81343 items, with 1,053,887 stored elements representing user/item interactions.

Creating Train/Test Split

In the standard supervised machine learning model, splitting the data into train and test sets is fairly straightforward. One can simply sample a small percentage of rows in the data for testing, and then train on the remaining data. This method is inadequate for our model because we need the interactions to complete the matrix factorization. An alternate method is to make two copies of the user/item matrix, alter a percentage of the non zero interactions and use the altered user/item matrix to perform the matrix factorization during training. The test set is then stored as a binary preference matrix with the interactions set to 1. This is achieved by making copies of the data and using the train set copy to find the indices where interactions took place. These indices are then stored and sampled from, for interactions to mask. We then re-assign the value of zero for the sampled interactions indices in the training set.

Implicit: Alternating Least Squares

The alternating least squares algorithm was chosen for its ability to introduce a confidence value to each interaction. As mentioned before, we can use the amount of times a user listened to a song to put a confidence on how well we believe a user liked an item. When the data is implicit, the goal of the model is to be able to say will the user like the item. This is represented by 0 and 1 with 1 meaning they will like it, and 0 denoting no preference.

$$p_{ui} \, \varepsilon \, (0, 1)$$

It is important to note that a value of 0 does not mean a user will not like an item, which means a user is likely to like it, rather than a user is likely to dislike it. As such, this model will not be able to tell us what a user does not like. The next component in the model is the recording. In the case of explicit data, this would be the rating a user gave to an item. In implicit data this represents the recording of how many times a user interacted with an item, or in this case, listened to a song. These are the values in our user item matrix.

$$r_{ui} \, \varepsilon \, R$$

The relationship between the preference and recording is as follows…

$$p_{ui} = \{1 \; if \; r_{ui} > 0, \; 0 \; if \; r_{ui} = 0\}$$

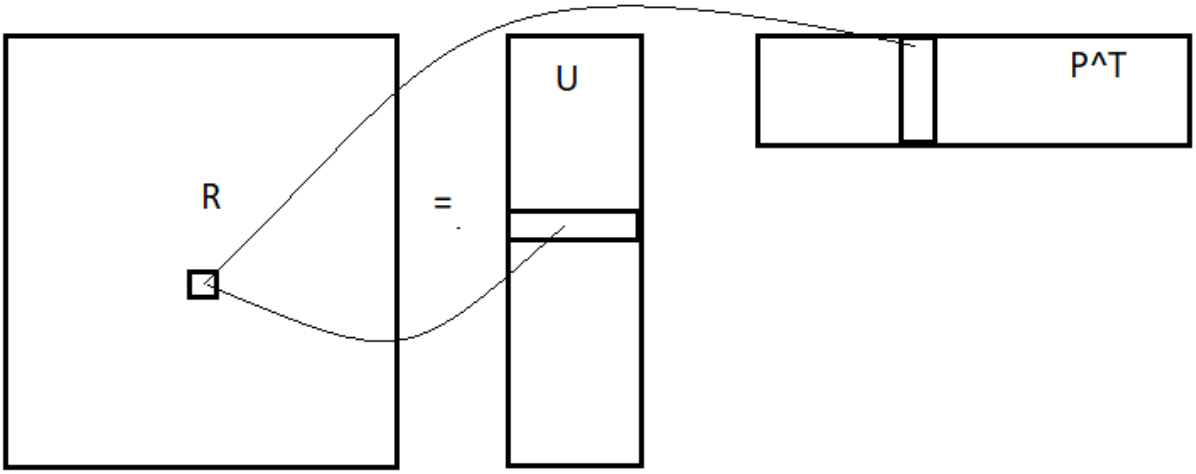
In the case of our listening events, the equation states that if a user listened to a song once, they liked the song, otherwise no preference. However, listening to a song once does not necessarily mean that the song was liked. This is where the notion of confidence is introduced.

$$C_{ui} = 1 + \alpha r_{ui}$$

This above confidence equation allows us to have some confidence in listening events that occur one or a few times, and more confidence in interactions that took place many times. At the same time we would like to have some confidence that an interaction is 0. If r_{ui} is zero, the confidence that there is no preference in the item is captured in the 1 out front. α is a tuning parameter that weights the confidence that a listening event is 0 or 1. When putting this all together to implement alternating least squares, the equation we want to minimize is...

$$C_{ui}(p_{ui} - U_u X_i^T)$$

Where U_u is the user vector, and X_i^T is the item vector. The goal is to find the user vector U, and item vector X, such that the equation is minimized.



The above image is a representation of the task at hand. We start with a sparse user/item matrix, R, and perform alternating least squares to approximate the latent vectors U and P^T. The algorithm achieves this by holding U constant,and solving for P^T, and then holding P^T constant and solving for U. The algorithm alternates between the two until convergence is found. The shape of U and P^T is determined in part by the shape of the ratings matrix. The

latent user vector will be the length of unique users, and the item vector the length of the number of unique songs. The width of each is defined by the number of components, or latent factors chosen when instantiating the model. For our model the number of components was chosen to be 20. The method of determining the number of components by examining explained variance was attempted, however the local system did not have enough memory to run the routine.

The model is created by instantiating the alternating least squares algorithm in the implicit library with the factors set to 20. The alpha parameter is set to 40, as it is mentioned in the literature that 40 is a good value to start with. The output of the model are the latent user and item feature vectors that are used to make predictions. The shape of each is checked to ensure the shapes are as expected.

In order to evaluate the model, we will use the predictions at the indices where interactions were masked, and compare these values to the binarized test set. We want to examine how the order of recommendations for each user with a masked interaction compares to the songs they actually listened to. For this, we will use the ROC, or receiver operating curve. A greater area under the curve means the user listened to songs near the top of the recommendation list. In order to do this a function that calculates area under the curve is implemented for all users that had interactions masked in the training set. An AUC score is also calculated for popular items to see how our recommendations compare to just recommending the most popular items. After running the function, we are left with AUC scores of 0.916 and 0.875 for the ALS algorithm and popular recommendations, respectively. The ALS algorithm beat the popularity measure by about 4%, this is encouraging as it is difficult to beat popularity recommendation models. At this point, we could try to nudge the parameters, or increase the number of components, but a better way to increase the score would be to add more data. This is where the lighfm library comes in handy.

Lightfm Model

The lightfm library was chosen for its capability to construct hybrid models. It has the ability to add user and item features to enhance the quality of the recommendations by taking into account user and item metadata such as, hashtags, artist name, song features, and any other user or item metadata. The hybrid model is beyond the scope of this project, but the lightfm library was chosen in the event that the project is extended. In our case, if no user or item features are supplied to the model, the algorithm reduces to the standard matrix factorization. This standard model is used to compare the performance of the Implicit model. To use the lightfm model, an instance needs to be instantiated with the number of components and loss function. For fair comparison, we choose 20 components to be in line with the ALS model. There are several options to choose for the loss function. The loss functions that are relevant to the data we have are the ‘BPR,’ Bayesian Personalized Ranking and ‘WARP,’ weighted approximate rank pairwise. The ‘BPR’ method maximizes between a positive example and a randomly chosen negative example, and is useful when only positive interactions are present

and optimization for ROC AUC is desired. The 'WARP' method maximizes the rank of positive examples by repeatedly sampling negative examples until a rank violating one is found, and is useful when only positive interactions are found and optimising the top of the recommendation list is desired.

Next, use lightfm's dataset class to fit the data. This is done by instantiating the dataset class and supplying the unique users and items. The model instance is then fit with the train data. Afterwards, we score the model using lightfm's AUC function using the test data.

Both the 'BPR' and 'WARP' loss functions are used to see which produces a better score. The AUC using WARP was found to be 93.7%, which beats both the ALS model and the popularity. The model using the 'BPR' loss function was scored and found to be 80.6%, losing out to the ALS, WARP, and popularity models. This was somewhat confusing as the BPR loss function is used when trying to optimize AUC.

Conclusion

At this point, we could try and improve the results by creating a hybrid model within lightfm. This would allow for the use of the item data contained in the nowplaying_rs data set. However, this is beyond the scope of this work and will be looked at in the future. Both the Implicit and lightfm models scored higher than popular recommendations, with the lightfm model beating ALS by a few percentage points, this is determined to be sufficient for the purposes of building a basic recommendation system with matrix factorization.

Not having any experience with recommendation systems presented many obstacles during the course of the project, and made me aware of some of the differences between a standard model and recommendation systems. The data wrangling aspect of the project was the most difficult, and while I struggled at many turns I tried to appreciate the fact that I was learning during the process. Using such a large dataset presented problems I've never dealt with such as handling memory errors and learning how to use sparse matrices. Furthermore, evaluating the model posed challenges as the data splitting process is different to that of your standard model, which presented further needs for data wrangling as the train and test sets are created differently. In the end, I am glad I was exposed to these obstacles because it provided me the opportunity to resolve these issues and understand some of the nuances and complexities of building recommendation systems.