Crate Digger

Final Report, CS 121

Jack Bauman
UC Santa Cruz
Santa Cruz, CA
jcbauman@ucsc.edu

Young Choe
UC Santa Cruz
Santa Cruz, CA
ydchoe@ucsc.edu

Ismael Chavez
UC Santa Cruz
Santa Cruz, CA
ischavez@ucsc.edu

ABSTRACT

In this final report, we will demonstrate the objectives behind our development of our music recommendation Android application, Crate Digger, as well as go into the specifics of its components, development process, and its future potential. We will also discuss the contributions each of our group members made towards its development in this class.

OBJECTIVE

Music is a powerful force that speaks to people across disciplines, and musical taste is uniquely personal and relates to a person's biology and background. There is also a wealth of music in the world that we would thoroughly enjoy, yet unfortunately never get the opportunity to hear. This is the issue we worked to target in our mobile application, Crate Digger.

"Crate digging" is a term used by radio disk jockeys to describe searching in boxes of vinyl of records for a great new album that catches their eye. With Crate Digger, we aimed to create a user experience reflective of this, by making it incredibly easy for a user to be introduced to variety of new music, including the ability to preview what a song actually sounds like, as well as easily locate the songs they like in other contexts outside of our application, such as on the internet, or their favorite music streaming app. We also wanted to create a sophisticated algorithm that learns a user's own music preferences to better recommend new music they haven't heard, as well as aesthetically display their genre preferences and statistics.

Many music streaming applications exist that try to identify music preference and recommend music based on this data. The difference in Crate Digger is its focus on ease of use, where users can easily "swipe" through a good volume of music, mimicking popular dating apps like *Tinder*, where all the information needed for a user to decide if they like a song is provided seamlessly and easily. Additionally, many music recommendation apps' algorithms operate as black boxes, with no ability for users to see or change how it is working. Crate Digger aims to give users a clear picture of what it is recommending, as well as give users the ability to tweak their algorithm directly if they are interested in seeing more music that defies their previous genre preferences, or more music that relates to their previously-liked songs. Above all, we wanted to provide a viable recommendation service to users that would not require a paid subscription to a music service like Spotify but would integrate with the app if a user currently uses it.

COMPONENTS

When users first open the app, they are greeted with a simple and easy-to-use interface which displays the app's first recommended song, loading album artwork and track metadata (title) in a Card View. The app also downloads a portion of main part of the song's audio which begins playing automatically through the Media

Player. We obtain this track information and song links through calls we made to the Spotify Web API. The user has the ability to review the music, then use intuitive controls, swiping left or right, or pressing the thumbs-down or thumbs-up button, which communicates whether they like or dislike the song.

Displayed songs are saved into a database on the user's phone by URL along with the user's verdict, so that the songs are not shown again. When a song is liked, the genre metadata of the song is taken into account by our recommendation algorithm, which keeps active counts of user genre preferences. Our algorithm then recommends a new song, which downloads its album artwork and audio preview and loads it behind the first card, so that there is always a constant stream of recommendations presented to the user.

Figure 1: Main Activity SwipeCard feature, displaying album art and allowing users to like or dislike songs.

Users can then press the heart icon in the app's menu bar to be taken to new activity, displaying a list of all the songs they "swiped-right" on, or "liked," sorted by most recently-liked songs on top. The list displays small thumbnails of album artwork, along with song title and artist. The user can easily select a song on their list and the app will send them over to the Spotify application, where the full song will start playing, and users can save it to their own Spotify library, or share it with friends. If the users do not have a Spotify account, it will instead open the song in their phone's web browser. This feature makes it easy for users to translate their finds and preferences in the Crate Digger app to the real world.

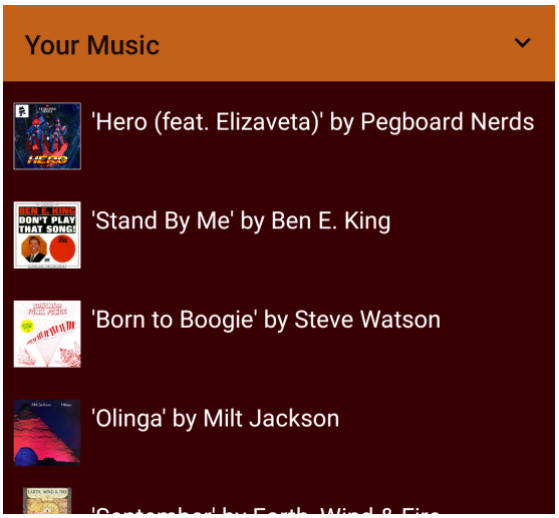

Figure 2: List View activity, displaying album art thumbnails along with track artist information, allowing users to view the song in Spotify upon selection.

Users can also press the "stats" icon on the app's menu to reveal up-to-date information about their music tastes in a new activity. The view displays an animated graph, summing the genre metadata of all of their liked songs in an interactive way. The user also sees text displaying Crate Digger's analysis of their top genre, or which type of music they "dig" the most. Below the statistics interface is a slider which allows the user to

tweak their recommendation algorithm. The slider value is automatically saved as an app preference on the phone, and users can choose whether or not they want the app to show them "more music they'd like," or "something different." This selection influences how intensely our recommendation algorithm looks for songs based on the user's top genre.

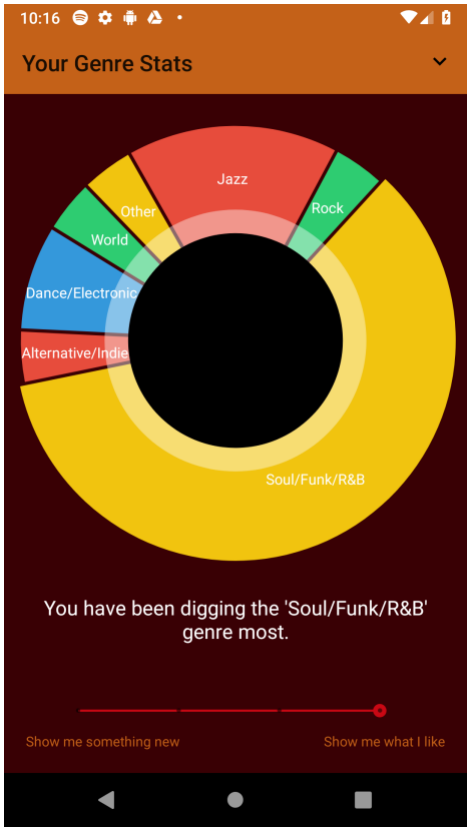

Figure 3: Stats Activity, displaying an animated genre preference chart, a message about the user's top genre, and a slider to select algorithm tweaks.

In the back end, Crate Digger's recommendation algorithm makes basic Bayesian inferences to predict a user's preferences. Each time a song is liked, the algorithm adds up the number of songs a user has liked in each particular genre, then selects the genre for the next song to be displayed. The app then tries to recommend a random song from that genre, making sure to cross-reference the database to make sure the song has not been shown before. As a user's swipe data increases, more-frequently liked genres then have a higher probability of being recommended, although songs from every genre always have a chance of showing up, regardless of a user's genre statistics. Additionally, the user's recommendation tweak preference from the stats page either increases or decreases the probability of the user's most-liked genre being recommended next.

DEVELOPMENT

The most essential component Crate Digger required was a way to play music, such as a database of songs to pull from and recommend. Our first choice was to use the Android SDK provided by Spotify, which we integrated into our application. Although it contained some of the functionalities we needed, it would require users to have the Spotify application and a Spotify premium account. It also allowed us to download and play full songs, yet it was unable to give us track metadata such as album artwork and song links.

Because we wanted Crate Digger to be first and foremost, a stand-alone product, we instead looked into using the Spotify Web API. It could return JSON metadata about music artist, albums, and 30-second preview tracks using Representational State Transfer (REST). We spent several weeks trying to integrate a deprecated Spotify Web API wrapper with Retrofit from Github that the Spotify developer community asserted to be our best and only option for connecting within the Spotify web service. We were able to fix the issues with the wrapper and use it to make call requests to the API from our app, and it returned the metadata we needed.

Soon, we discovered Spotify had changed their authentication policy for obtaining metadata and now required users of the API to have a Spotify Premium account in order to request for new songs, as well as re-authenticate with these

credentials every hour. This again conflicted with our goals to have our app be a stand-alone product. Therefore, we used our own Spotify accounts to authenticate with the API and pull a large group of songs from the Web API and stored the metadata in our own database, rather than have our app pull directly from Spotify with Crate Digger's users' Spotify accounts. With this format we were able to deliver on the core functionalities we set out to do with our recommendation service.

The other components were more straightforward to implement than the first. We implemented a card swipe feature for our main screen, then a statistics page with a pie chart graph view that represented the user's taste in music. We used the Media Player to play songs within the app from the song links we obtained, and we worked on pausing the music when the user left the app or navigated to other activities, and have the music start from where it left off when they returned. In the backend, we built an SQLite database to store song information, and whether a song had been shown to the user, and whether the user liked the song. We then created a list view to show an updated list of songs from the database that the user liked. The list view used a custom adapter to show small thumbnails of the album art, as well as link to the appropriate song in Spotify when pressed.

The main component was creating an algorithm for recommending new music to the user. We decided to make it related to genre preferences instead of related-artists as we originally imagined, due to our Spotify Android SDK issues. This involved us reducing/categorizing songs of numerous subgenres into seven main genres, such as "Rock," "Soul/Funk/R&B," "Electronic/Dance," "Jazz," etc. This both improved our algorithm's ability to recommend similar songs, as well as decluttered our statistics page. We also implemented a slider that allowed users to alter the algorithm's probability of recommending their top genre, which we saved into a preference on the phone. Our finished product met all of the main goals we had set out in our project proposal and stayed true to most of the original concept designs.

CONTRIBUTION

We were very pleased with the group's work ethic towards this project. Everyone took part in the initial design and mockups and met frequently together to discuss the development process. One issue we faced was when one of our original group members, Carly, seemed to disappear from our group slack and meetings, then said that she had withdrawn from the class mid-quarter. This set us back for some time with Spotify SDK implementation, because Carly had made the critical advancements towards integrating the Web API wrapper, and that work was essentially lost to us when she left.

However, we persevered as a group of three. As project leader, Jack took charge of editing project reports, built the swipe cards activity, the statistics activity, and much of the app's UI. David (Young) built the app's SQLite database, worked with Media Player, and designed the Bayesian recommendation algorithm. He also took charge and refactored the entire project's code when it was determined to be too messy. Ismael also worked on the database and the list view of liked songs linked to Spotify. He spent numerous hours pulling songs for our database from the Spotify Web API.

We all had a hand in helping each other out in each of our own tasks, collaborating on presentations and reports, and were always willing to stay up into the early hours of the morning to implement a specific feature for that week.

FUTURE WORK

One feature we could develop with more time would be a more complex recommendation

algorithm than what we currently have. Given the amount of songs that we were able to save in our database, the simplistic recommendation algorithm we currently have in place is sufficient enough to change its recommendations based on user's preferences, but it would be preferable to have a more complex algorithm that takes into account other aspects of songs, such as related artists, style, and more specific genre categories.

A feature still missing from the final application is the ability to automatically connect the application to the Spotify API, given the user has a Spotify Premium account, and pull songs directly from there as opposed to pulling songs from the database within the app itself. This was not implemented as we felt that it will severely limit the availability of the app to the general public since a Spotify Premium account would have been required for the app to function.

If work on Crate Digger were to continue into the future, we find a way to keep users authenticated, either using a login flow with their own Spotify accounts, or to have a separate account for those without the streaming service. This would help us automate the retrieval of JSON metadata to immensely widen the pool of songs our application can choose to display to the user. We could also connect to other music listening services besides Spotify, such as Apple Music and iTunes.