Final Report: Collaborative Spotify Playlist System

Andrew Xavier
UNI:ahx2001
ahx2001@columbia.edu

Alban Dietrich
UNI: ad4017
ad4017@columbia.edu

Kenneth Munyuza
UNI: km3829
km3829@columbia.edu

Abstract

Spotify is the dominant music streaming service in the world with 433 million active users in 2022. Additionally, its database of over 80 million of songs, and 4 billion playlists are all available through its developer platform allowing the creation of external tools and features. We build a custom playlist generator using inputs from multiple users in the form of existing playlists. We allow users to interact with our model using an online user interface where they can link their profiles and submit generation requests. They can register the output playlist on their Spotify account or listen to it directly on our online web interface.

1. Introduction

Music streaming services have grown to dominate how people listen to songs and audio and with it, a plethora of data on users, tracks, albums, and playlists [7]. This data-rich industry opens opportunities for big-data applications and analytics. Spotify is by far the largest streaming service as of 2021 with a 31% hold of the global market share and over 433 million users as of 2022 [9]. Spotify's growing library of music includes over 80 million unique tracks [2] and also provides a platform for podcasts where nearly a quarter of all American residents get their news from [13]. Additionally, Spotify has a substantial user development platform allowing third-party apps which can access Spotify's database to integrate with the platform. Our project builds upon Spotify's development platform using the Python library, Spotipy [10].

Spotify's commercial application includes song recommendation services based on the user's personal library of music, listening habits, and created playlists. Spotify recently released a new feature, Blend which allows users to sync their libraries, producing a shared playlist with music from both users' collections of liked songs [3]. However, this feature only includes songs that already exist within each of the users' collections. Songs that neither user has heard are not included in the mix. Our project looks to tackle this issue and generates playlists composed of novel songs (songs not in each user's library) based on the songs in each user's library. Using the features provided by Spotipy for each track, we use a model to select songs that the user would be most likely to include in their libraries. This is then returned as a playlist to the user. We look to provide access to this service through a web app using Flask to connect the inputs of the user to the data access and modeling in Python.

The 3Vs (volume, variety, and velocity) of this project are the following:

- Volume:
  - 80 Million Tracks.
  - 433 Million Users.
- Volume:
  - Qualitative and Quantitative.
  - User inputs and Spotify's library.
- Velocity:
  - Spotipy: 1000 Tracks/70 seconds.

This project/app would provide a novel collaborative use of Spotify, such as:

- Social value of having Spotify over other streaming services increases.
- Help fill the missing social aspect for Spotify thus increasing the use cases of Spotify.
- Increase reliance on Spotify in the streaming services for users.
- Expose users to new music and genres.

2. Related Work

Spotify, being one of the most popular music apps in the world, has a very important and interesting music database. Therefore Spotify created a Python library, called Spotipy, which allows people to access the data and its features (e.g. danceability, energy, valence, etc.) very easily. Several people have already used it to perform analyses to better understand the links between different songs [4, 5]. This library facilitates the use of machine learning algorithms for diverse applications, such as playlist creation. In fact, Spotipy itself provides functions to do so [8]. However, the obtained results are sometimes not very accurate, and can thus be improved. Some people tried to do so by using different ML algorithms such as Random Forest Regression [6], OPTICS Clustering [11], K-means [12], etc.

In this project, the aim is to use previous and new ideas to make a better AI algorithm and interface to create collaborative playlists.

3. Data

3.1. Overview and Features

Our data is composed of a holistic collection of Spotify's music library and the collection of tracks in one or more user-selected playlists. While Spotify's entire collection of tracks numbers over 80 million, we choose to compose our collection as a subset of the top 100,000 tracks (coming from a 26 million Kaggle dataset[1]). This is largely due to the limitations of Spotipy which can only retrieve 50 songs per API call. This can be improved by running multiple threads to collect non-overlapping subsets of the data which would increase the selection of songs that could be selected from and would allow this model to generalize for a more diverse set of users.

However, for the purposes of this project, 100,000 songs are a sufficient sample. Even if a user has 5,000 songs that are all in the top 100,000 songs, there are still 95,000 possible recommendations in our data set. As Spotify's collection of data is ever-growing and the tracks in the top songs change regularly, we update our library of possible songs to recommend weekly.

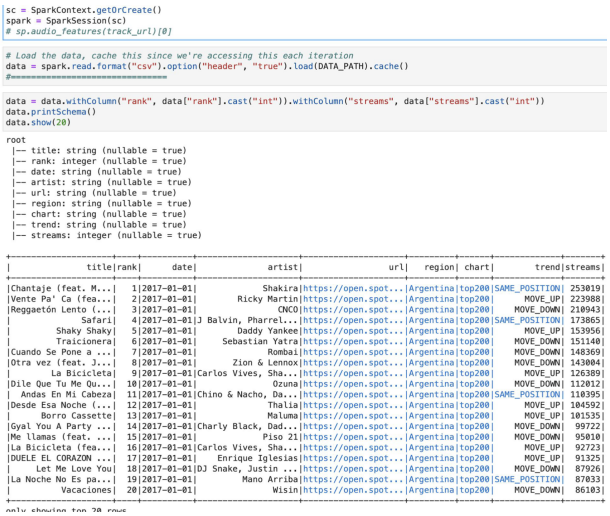
Each song or track comes with a set of attributes ranging from the label it was released under and the artist to the unique Spotify id for the song and its URI. These are contained in the features for each track [1]. While some feature types are strings and URLs, others are quantitative metrics describing the qualities of the tracks. These include danceability, tempo, and speechiness. Leveraging these features, we can understand the distribution of our data and train models to suggest songs based on that distribution. While some features are harder to interpret (i.e. valence, instrumentalness), we can still use these to train our model and generate our output playlist as they reflect the distribution and characteristics of songs of the user.

[1]Source: https://www.kaggle.com/datasets/dhruvildave/spotify-charts.

3.2. Collection and preprocessing

Our songs library uses a Kaggle dataset containing 26 million songs[1] which is saved under a CSV file. As seen in the figure below, it contains information such as title, rank, artist, URL, etc.

We upload the CSV file on the Google Cloud Platform (GCP) and run the data on a Dataproc cluster. Using Spark, we select 100,000 songs (see figure 1) from the 26 million which correspond to our songs library.



```
sc = SparkContext.getOrCreate()
spark = SparkSession(sc)
# sp.audio_features(track_url)[0]

# Load the data, cache this since we're accessing this each iteration
data = spark.read.format("csv").option("header", "true").load(DATA_PATH).cache()
#============================

data = data.withColumn("rank", data["rank"].cast("int")).withColumn("streams", data["streams"].cast("int"))
data.printSchema()
data.show(20)
```

```
root
 |-- title: string (nullable = true)
 |-- rank: integer (nullable = true)
 |-- date: string (nullable = true)
 |-- artist: string (nullable = true)
 |-- url: string (nullable = true)
 |-- region: string (nullable = true)
 |-- chart: string (nullable = true)
 |-- trend: string (nullable = true)
 |-- streams: integer (nullable = true)

+--------------------+----+----------+--------------------+--------------------+---------+-------+----------+-------+
|               title|rank|      date|              artist|                 url|   region|  chart|     trend|streams|
+--------------------+----+----------+--------------------+--------------------+---------+-------+----------+-------+
|Chantaje (feat. M...|   1|2017-01-01|              Shakira|https://open.spot...|Argentina|top200|SAME_POSITION| 253019|
|Vente Pa' Ca (fea...|   2|2017-01-01|         Ricky Martin|https://open.spot...|Argentina|top200|   MOVE_UP| 223988|
|Reggaetón Lento (...|   3|2017-01-01|                CNCO|https://open.spot...|Argentina|top200|  MOVE_DOWN| 210943|
|              Safari|   4|2017-01-01|J Balvin, Pharrel...|https://open.spot...|Argentina|top200|SAME_POSITION| 173865|
|         Shaky Shaky|   5|2017-01-01|         Daddy Yankee|https://open.spot...|Argentina|top200|   MOVE_UP| 153956|
|         Traicionera|   6|2017-01-01|      Sebastian Yatra|https://open.spot...|Argentina|top200|  MOVE_DOWN| 151140|
|Cuando Se Pone a ...|   7|2017-01-01|              Rombai|https://open.spot...|Argentina|top200|  MOVE_DOWN| 148369|
|Otra vez (feat. J...|   8|2017-01-01|        Zion & Lennox|https://open.spot...|Argentina|top200|  MOVE_DOWN| 143004|
|       La Bicicleta|   9|2017-01-01|Carlos Vives, Sha...|https://open.spot...|Argentina|top200|   MOVE_UP| 126389|
|Dile Que Tu Me Qu...|  10|2017-01-01|               Ozuna|https://open.spot...|Argentina|top200|  MOVE_DOWN| 112012|
|  Andas En Mi Cabeza|  11|2017-01-01|Chino & Nacho, Da...|https://open.spot...|Argentina|top200|SAME_POSITION| 110395|
|Desde Esa Noche (...|  12|2017-01-01|              Thalia|https://open.spot...|Argentina|top200|   MOVE_UP| 104592|
|       Borro Cassette|  13|2017-01-01|             Maluma|https://open.spot...|Argentina|top200|   MOVE_UP| 101535|
|Gyal You A Party ...|  14|2017-01-01|Charly Black, Dad...|https://open.spot...|Argentina|top200|  MOVE_DOWN|  99722|
|Me llamas (feat. ...|  15|2017-01-01|              Piso 21|https://open.spot...|Argentina|top200|  MOVE_DOWN|  95010|
|La Bicicleta (fea...|  16|2017-01-01|Carlos Vives, Sha...|https://open.spot...|Argentina|top200|   MOVE_UP|  92723|
|DUELE EL CORAZON ...|  17|2017-01-01|    Enrique Iglesias|https://open.spot...|Argentina|top200|   MOVE_UP|  91325|
|     Let Me Love You|  18|2017-01-01|DJ Snake, Justin ...|https://open.spot...|Argentina|top200|  MOVE_DOWN|  87926|
|La Noche No Es pa...|  19|2017-01-01|         Mano Arriba|https://open.spot...|Argentina|top200|SAME_POSITION|  87033|
|           Vacaciones|  20|2017-01-01|               Wisin|https://open.spot...|Argentina|top200|  MOVE_DOWN|  86103|
+--------------------+----+----------+--------------------+--------------------+---------+-------+----------+-------+
only showing top 20 rows
```

Figure 1: Data collection

Furthermore, utilizing GCP, we preprocess the data using Spark. Indeed, as the information we are looking for corresponds to the audio features (i.e. danceability, tempo, energy, etc.), we need to transform the data. This preprocessing part consists of obtaining the id of every song and getting the corresponding audio features by using the Spotify API. From these features, we select some parameters (see Figure 2) such as danceability, energy, valence, tempo, etc.

The dataset is first transformed as a DataFrame and then as a CSV File. This data is saved on a Firestore database (see next section).

In the figure below, we can see which final dataset we obtain. This dataset of 100,000 songs corresponds to the one we will use to create a new playlist based on users' preferences.

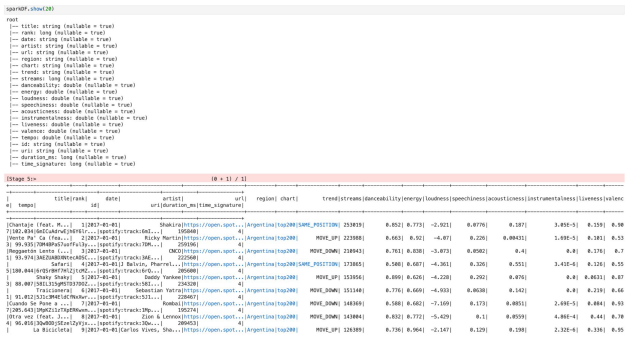


Figure 2: Data preprocessing

3.3. Data Storage and Backend Format

Our data is stored using GCP's Firebase database. We use a Firestore database to hold both the 100k set of tracks (reference dataset) and the playlist data that users send in while using our app. The data uses a NoSQL framework. An example of the reference data and user data is shown below in Figure, 3, 4, and 5.

First, we have the reference dataset of 100k which was obtained after the preprocessing part explained in section 3.

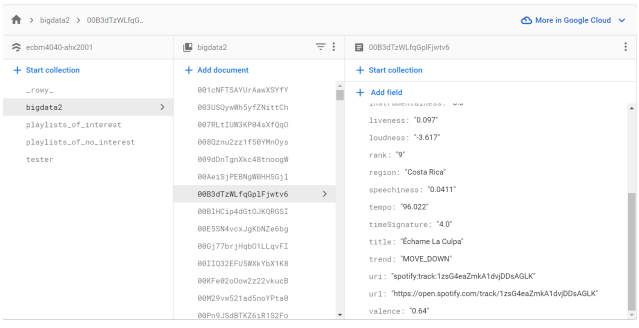


Figure 3: Reference data of 100k songs stored in Firestore

Then, we stock the playlist(s) and songs selected by the user(s) from our web interface.

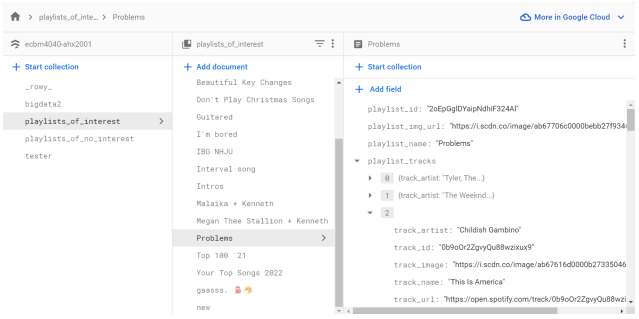


Figure 4: User data of selected playlists stored in Firestore

Finally, all the playlists from the different users which were not selected are saved as playlists of no interest and will be used to train the model.

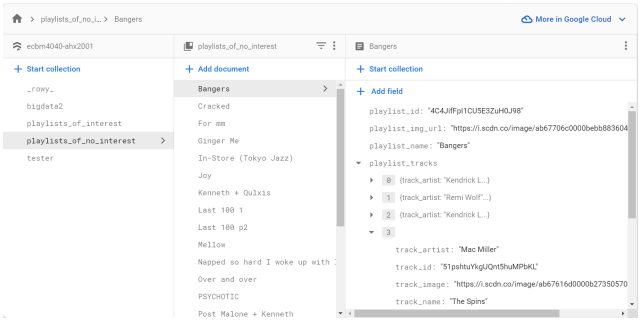


Figure 5: User data of playlists that were not selected stored in Firestore

In the 100k track set, each element is a track and contains the metadata as well as identifying features such as the song title and URI.

For the user data, each element is a playlist dictionary that contains information about the playlist, such as the title and the URL. Each dictionary also has a key "playlist_tracks" that maps to an array of track dictionaries containing information about each track in the playlist such as the track's id and name.

4. Methods

We use a set of approaches to generate track suggestions. In each, the libraries of the users serve as the guide for what qualities should be present in the songs we choose for our generated playlist.

Below, we present different models we tried, used, and studied. We finally chose the best of them, which in our case was the Random Forest Classifier.

But before trying to find a good model for our project, let us visualize the data and see if some relationships or behaviors can be understood.

4.1. Data visualization

Let us first study the distribution of song genres. By plotting a histogram depending of the genre of the songs, we can notice that most of the songs in our database correspond to pop songs.

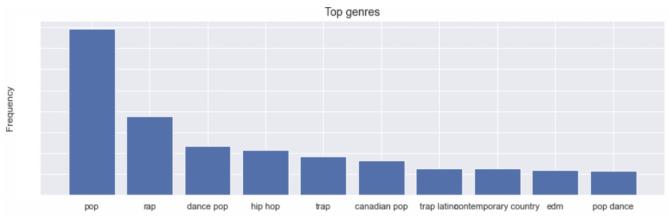


Figure 6: Histogram based on the songs genre

Another important visualization is the correlation between features. Here below we can for example remark that the energy and the loudness are positively correlated and that the energy and acousticness are negatively correlated which makes sense. Indeed, in general, loud songs tend to be energetic.

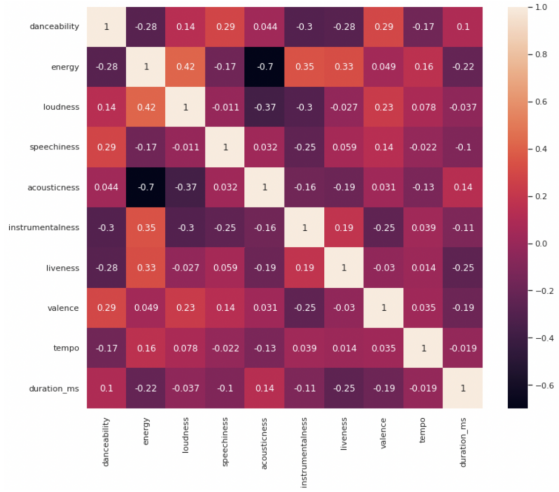


Figure 7: Correlations between different parameters

To further study the relationship between the energy and the loudness, we can plot a level curve graph. We can see that the higher the loudness, the higher the energy will be, and vice-versa. This confirmed what we said before. These features seemed to be important factors to consider in our model.

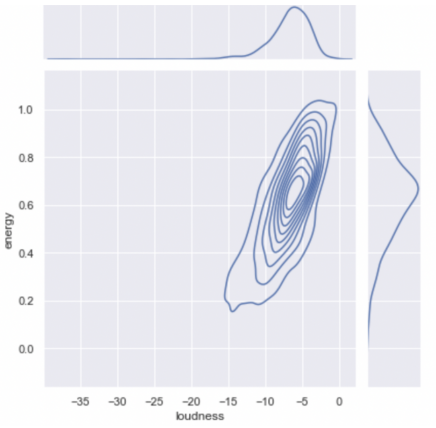


Figure 8: Level curve graph of energy in the function of the loudness

We plotted a polar graph of some characteristics to get an idea of what might make a song popular. We deduce from the figure below that the danceability seems to be the most important factor. Therefore we will include it in our model.

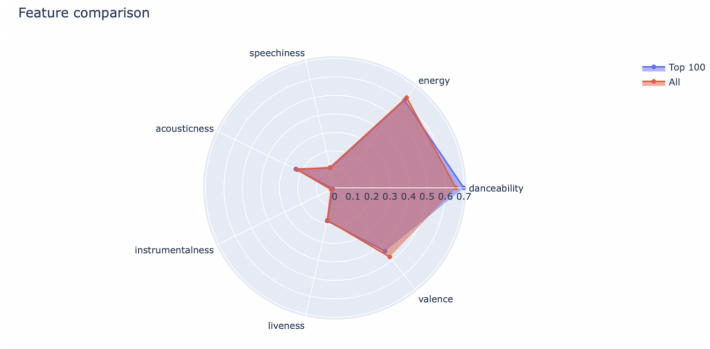


Figure 9: Polar graph different audio features

Finally, we plotted songs according to 2 features (danceability VS tempo) and their genre to see if some cluster could be formed depending on the genre.

Figure 10: Plot of the danceability in the function of the tempo

In the figure above, we can notice that on average, genres such as trap Latino have higher danceability than rap songs.

We tried clustering using K-Means to see if the clusters would correspond to a certain feature or type of song like rap. But as you can see in the figure below, the outputs did not give significant results.

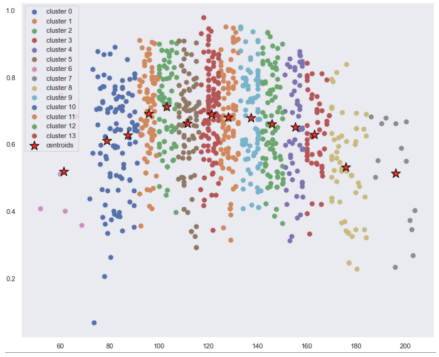


Figure 11: Clustering output by using K-means

Now that we did some visualizations and that we understood the data better, let us find our generation system to generate an output playlist.

4.2. Baseline model

First, let us start we the first model we tried. This simplest baseline model uses feature selection based on variance within the features of the track. For this approach, we take the collection of the user-selected playlist(s) as a reference point and then calculate the variance for each feature after normalization as shown below in Figure 12.

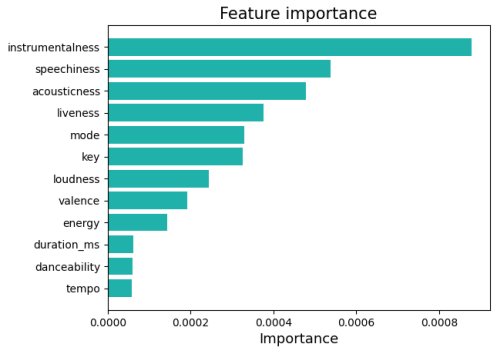


Figure 12: Features sorted by variance

From there, we select the features with the least variance as our features to give the most weight when choosing new tracks. The purpose of this variance-based feature selection is to capture what qualities are central to the user's taste. For example, if there is a lot of variance in the danceability of the tracks in the user's library of liked songs, then the value of danceability is unlikely to affect whether or not a user likes a song. In contrast, if the tempo has very little variance and the songs in the liked library all have similar tempos, it is more likely that the user is pickier about this feature when adding it to their collection. This has a two-part effect: it helps make sure the preferences of the user are retained by matching the more important features but also helps increase variety and variance in songs that are suggested. Purely choosing songs that are most similar to the existing library of song limits can limit variability along with features that aren't as selective. The pros of this approach are its simplicity and ease to understand. Its downsides are that it does not look at features together, but independently and thus will fail to capture patterns/variance in combinations of features. An improvement of this approach would be to use PCA to look at how the features function together to explain variance.

With the features selected, this initial approach takes the average of these features in the user's library to function as the center or centroid. This is the reference point used to compare potential songs from the overall Spotify music library. The more similar the songs are to the centroid based on the features previously selected, the more likely it is to be selected. Based on that distance metric, we collect the 30 most similar songs and then randomly sample 20 songs from that group to help improve variability. Finally, these songs are used to generate a Spotify playlist and are returned to the user as shown below in the example in Figure 13.

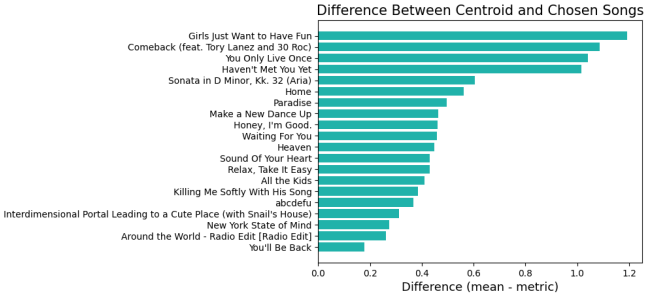


Figure 13: Chosen songs and their distance/difference from the centroid representing the user's library

The outputs obtained were good but can be improved by using other models.

4.3. K-nearest neighbors

Another model we tried is using K-nearest neighbors (KNN). In this model and also the others we will present after, we considered 2 training sets: X_train and y_train.

As explained in the previous section an improvement that could be done is by using Principal component analysis (PCA). To do so, we first created an X dataset using our Firestore database from the input playlist(s) selected by the user(s). This dataset contains the following features: acousticness, danceability, duration_ms, energy, instrumentalness, liveness, loudness, speechiness, tempo, and valence. Then we used PCA on it by using the decomposition.PCA() function from sklearn. Thus, we fitted this dataset to the optimal PCA to finally obtain the X_train dataset.

The y_train set corresponds to a list of 0 and 1 where a 0 corresponds to a bad song and 1 to a good one.

Good songs are the songs in the playlist(s) selected by the user(s) and bad songs are the other playlist(s) of the user(s) that are not selected. These correspond to the songs in the playlist of interest and no interest defined in section 3.

A challenging problem in our project is the fact that we cannot really use metrics to judge if an output playlist is good or not. Indeed, musicality is subjective and depends on the user(s) tastes. Therefore, we could not analyze the outputs by using plots or other visualizations.

However, we still could evaluate our outputs by asking different users if they were satisfied with the obtained playlist.

For the different models we used, hyper-parameters optimization has been done and will be explained in the experiments section.

For KNN, we trained the model by choosing 20 neighbors and obtained good outputs in general. But sometimes, we could have random outputs. Therefore, we decided to find another model to improve the results.

4.4. Density based methods

Another idea was to use density-based methods. These approaches correspond to the idea that a cluster is a zone of high point density, isolated from other clusters by sparse regions.

We tried density-based methods such as DBSCAN and OPTICS. Unfortunately, we did not get good results. The outputs were bad or random.

By looking at figure 14, we can see that the data is quite sparse, which explains why the density methods do not work very well.

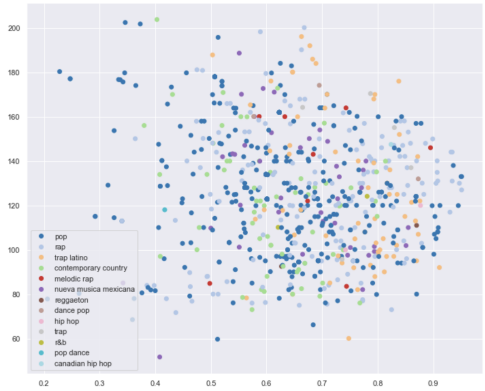


Figure 14: Sparsity of the dataset

4.5. Random Forest Classifier

Finally, the last method we studied is the Random Forest Classifier where the goal is to classify a song as good (corresponds to a 1) or bad (corresponds to a 0).

Here again, we used the same X_train and y_train datasets.

The output playlists obtained for this model were the best and gave often interesting, new, and pertinent playlists. Therefore we selected this model for our application.

5. Experiments

5.1. Data and Model Development

Concerning the data, we first decided to extract the features of each song using Spotipy library on our local computer to create our big dataset. This idea took too long and so as explained in the Data section (see section 3), we decided to use GCP and Spark.

After selecting the data, we visualized and studied it as discussed in the previous section. Once the data was better understood, we tried different models.

As mentioned before, comparing each model to each other thanks to a metric is pretty difficult as the output accuracy can only be judged by the users. Thus, the different models explained in the methods section were compared to each other by looking at the outputs and seeing if it corresponds to the user(s) tastes. We do not have a strict metric. Even though, we could study the different models and see that the Random Forest Classifier gave the best outputs.

Except for this metric problem. We optimized the K-nearest neighbors (KNN) and the Random Forest Classifier

by using grid search. Here below are the experiments we did to find the best hyperparameters for each method.

5.2. K-nearest neighbors

To find the best KNN parameters, we used grid search. The function we used is called GridSearchCV() and comes from the sklearn library.

We used Stratified K-Folds cross-validation. Here, we split the training dataset into 5 parts.

In the figure below, we plot the accuracy of the training depending on the number of neighbors for each split of the dataset and the corresponding mean. We can notice that the best number of neighbors corresponds to 20.

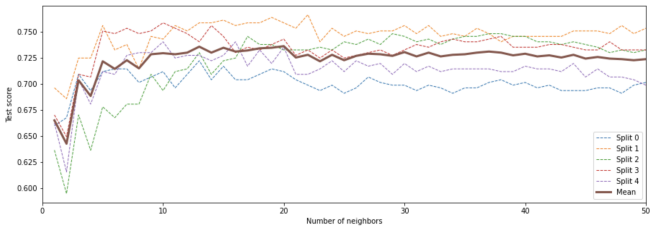


Figure 15: KNN accuracy depending on the number of neighbors

5.3. Random Forest Classifier

In addition, we used grid search to find the best parameters for the Random Forest Classifier.

We also performed cross-validation and split the dataset into 5 parts.

Here below we plot the score of the training depending of the number of parameters. We can observe that around 44 parameters we get the best accuracy.

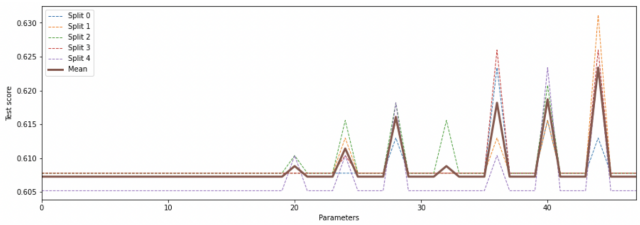


Figure 16: Random Forest Classifier score depending on the number of parameters

6. System Overview

6.1. Architecture

On Figure 17 is a schema of our project architecture.

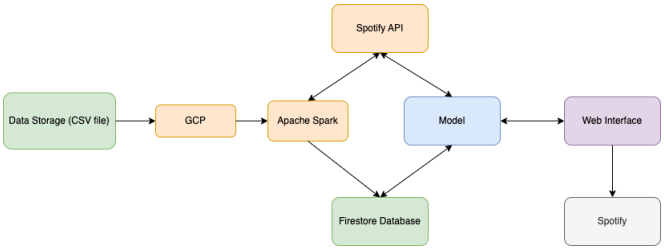


Figure 17: Project architecture

As seen above and explained in the Data section (section 3), we first collect the data from a Kaggle dataset (saved as a CSV file).

Then, we upload it on GCP and run the code on a Dataproc cluster. In the code, the preprocessing part is done by using Spark and Spotify API to gather the features of each song (e.g. danceability, energy, etc.).

Next, the preprocessed data is stocked in a Firestore database.

The model uses the data from our Firestore database containing our 100,000 songs and the user(s) input data from our web interface. The input data, once selected, is stored in our database.

After all the users selected their inputs, the model can be run on the website.

Finally, the output result corresponding to the new playlist generated is stored on the user's Spotify account.

6.2. Bottlenecks and improvements

The bottlenecks of our projects would be the fact that our model is for the moment limited to 25 users because of Spotify rules (see Figure 24). An improvement would be to extend this number of users by proposing a final commercial project to Spotify and asking for an extension.

Other improvements can be done such as adding different types of inputs (not only playlist(s)) such as specific songs, mood, genre, etc. This would give more freedom to the users.

Furthermore, we could also speed up the model by optimizing the code.

6.3. Coding languages and software packages

The project was coded in Python. The website used HTML, CSS, and Javascript.

We used a lot of different Python libraries. We will not mention them all, for example not the most common ones like NumPy, pandas, etc. The most important ones we use are sklearn, sparse, flask, requests, JSON, and Spotipy.

Finally, we used the Spotify API.

6.4. Web application

For easy and nice use of our model, we decided to develop a website interface.

On the homepage, you have the option to log in to your Spotify account.

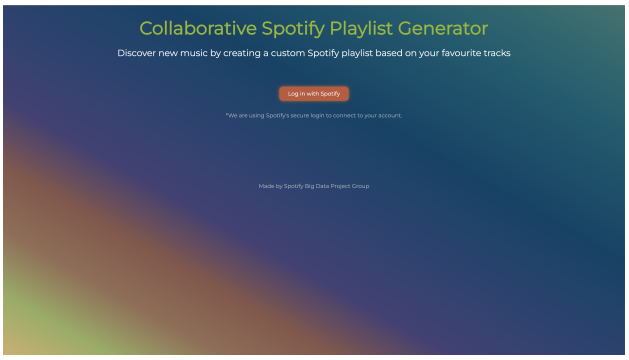


Figure 18: Home page

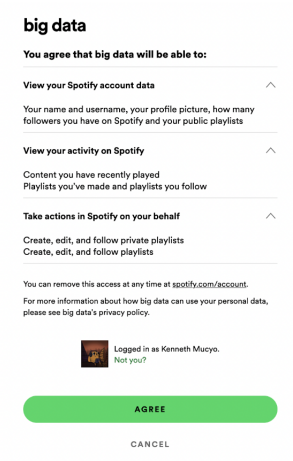


Figure 19: Spotify log in

Once logged in, you can select the playlist(s) you want to use as input(s). You can then add them to our Firestore database. At the same time, one or more users from a different device can do the same and the inputs will be added to the same database (allowing to generate of an output playlist according to the different user's tastes).

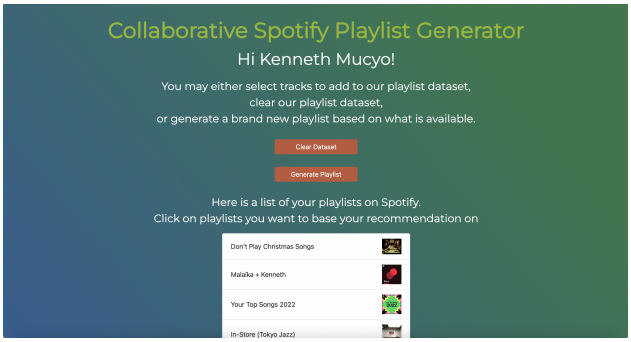


Figure 20: Playlist(s) selection

Once we are ready with the selections, we can generate the output playlist.

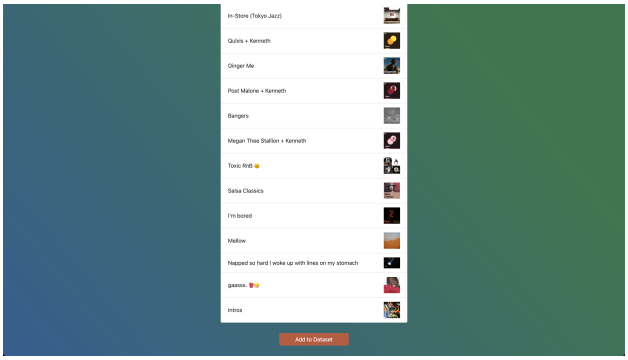


Figure 21: Generate output playlist

Afterward you can check your output playlist on the website.

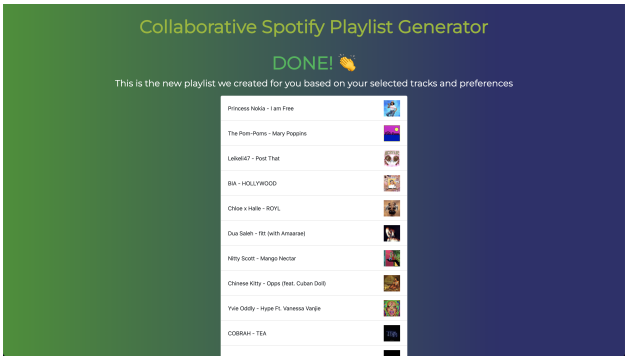


Figure 22: Output results

Finally, you can save the output playlist to your Spotify account and have also the opportunity to listen to your new playlist on our web interface.

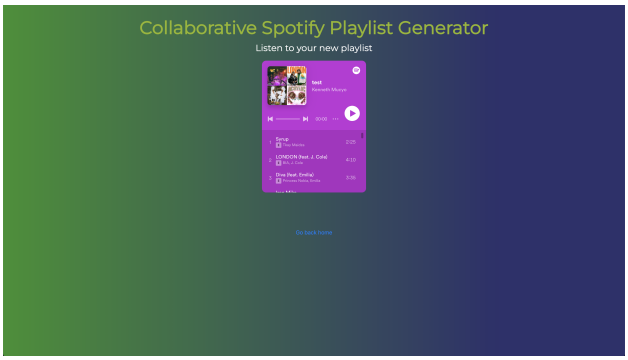


Figure 23: Save playlist and online player

For information, Spotify allows only developers to use their model up to 25 users (see Figure 24). An extension to more users is possible for a commercial goal but is prohibited for school projects as can be seen in the picture below. Therefore we decided to not host our website and use a local version.

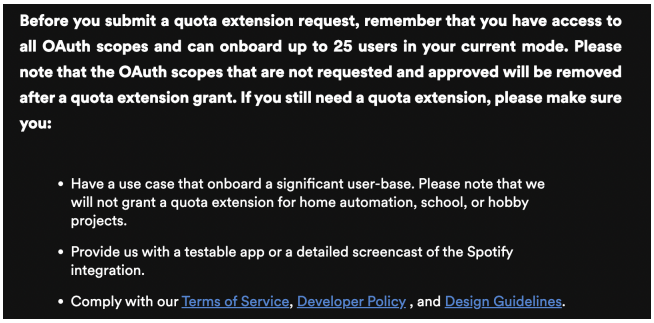


Figure 24: Spotify Developer extension

7. Conclusion

We propose a collaborative Spotify playlist generator available on our web interface (we use a local website). One or several users can give one or several playlists as input and we generate a new one from them. Unlike Spotify, we solve the problem of generating new songs according to different users' tastes.

After preprocessing our data with GCP and Spark, and visualizing and studying the data, we tried several models to get the best output possible. The one we finally chose is the Random Forest Classifier as it gave the best output playlist.

As previously said, one of the biggest challenges of our project was the fact we could not really use a metric to judge if the output playlist was good or not. Therefore we asked users and could choose our final model thanks to their feedback.

In this project, we learned to deal with a big database, process it, and use it for some AI models. We also had the opportunity to develop our skills in web design (using Javascript, HTML, and CSS).

Finally, we delivered a good final product but improvements can be done:

- Extend the use of our application to more than 25 users by submitting our app to Spotify and having a commercial plan.
- After the Spotify extension, host the website online.
- Add new types of inputs such as mood, genre, etc. to give more freedom to the user's tastes.
- Optimizing the code to speed up the application.

References

[1] Web api reference: Spotify for developers. https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features.

[2] About spotify. https://newsroom.spotify.com/company-info/, 2022.

[3] How spotify's newest personalized experience, blend, creates a playlist for you and your bestie. https://newsroom.spotify.com/2021-08-31/how-spotifys-newest-personalized-experience-blend-creates-a-playlist-for-you-and-your-bestie/, 2022.

[4] Abhisar Ahuja. The best spotify data analysis project you need to know. https://www.simplilearn.com/tutorials/data-analytics-tutorial/spotify-data-analysis-project, 2022.

[5] Tomer Chaim. Spotify playlist generator. https://github.com/AcrobaticPanicc/Spotify-Playlist-Generator1.

[6] Manik Garg. Build your spotify playlist using machine learning . https://medium.com/swlh/build-spotify-playlist-using-machine-learning-45352975d2ee, 2020.

[7] Scott Hiller and Jason Walter. The rise of streaming music and implications for music production. *Review of Network Economics*, 16:351–385, 12 2017.

[8] Anthony Li. Machine learning and recommender systems using your own spotify data. https://towardsdatascience.com/machine-learning-and-recommender-systems-using-your-own-spotify-data-4918d80632e3, 2021.

[9] Jon Porter. Streaming music report sheds light on battle between spotify, amazon, apple, and google. https://www.theverge.com/2022/1/20/22892939/music-streaming-services-market-share-q2-2021-spotify-apple-amazon-tencent-youtube, 2022.

[10] Spotipy-Dev. Spotipy-dev/spotipy: A light weight python library for the spotify web api. https://github.com/spotipy-dev/spotipy.

[11] Andrew Udell. Creating spotify playlists with unsupervised learning. https://towardsdatascience.com/creating-spotify-playlists-with-unsupervised-learning-9391835fbc7f, 2020.

[12] Cristobal Veas. Spotify-machine-learning. https://github.com/cristobalvch/Spotify-Machine-Learning, 2020.

[13] Mason Walker. Nearly a quarter of americans get news from podcasts. https://www.pewresearch.org/fact-tank/2022/02/15/nearly-a-quarter-of-americans-get-news-from-podcasts/, 2022.

The project was evenly split between Andrew, Alban, and Kenneth. Here below are the main tasks each of us worked on:

- Andrew: Database management (Firestore), web interface development, presentations, and reports.
- Alban: AI models, web interface development, presentations, and report.
- Kenneth: Data collection and preprocessing, web interface development, presentations, and report.

Here is the link to our video for this project: https://youtu.be/nn9dBC-jdiM. We know that the maximum length of the video asked was about 10 min, ours is about 13 min but, if you want, you can stop watching the video from 10 min because the last part is about the explanation of our Firestore database which is an extra.

Here is the link to our GitHub: https://github.com/Sapphirine/202212-5-Collaborative-Spotify-Playlist-System.