CMPT 353 Final project

Group: Sons of Ramesh

Members:

Georgios Giovanis, ggiovani Tanish Manku, tanishm   Dawu Liu, dla189

Playlist Song Extensions

Goal

Our search for high quality music data led us to stumble upon the Spotify Million Playlist Dataset Challenge, which asks participants to create a model that can take a seed of [playlist name, playlist songs] and predict the next few songs within that playlist. This inspired us to ask a similar question: **Given a series of playlists songs, could we make good song recommendations for that playlist?**

Data

To answer our question we had to find some data. Luckily, Spotify has a public API that can be used to retrieve song information, so finding full-featured datasets was simple. Furthermore, we were able to find a dataset with approximately 175,000 songs, complete with characteristics, name, duration, etc. for songs released from 1921 to 2020. Finding a dataset with playlists was a little more challenging, but with some searching we were able to find a similarly sized dataset full of Spotify playlists, complete with playlist names, and songs up to 2019.

Previews of these datasets can be seen below in **Appendix A**. The most important song features we used in our analysis were 'name', 'valence', 'year', 'acousticness', 'danceability, 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', 'duration', 'mode'.

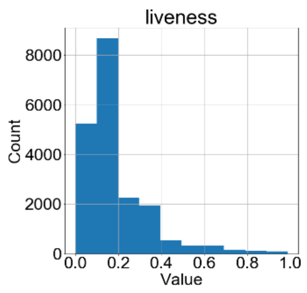
Analysis on what makes a song popular

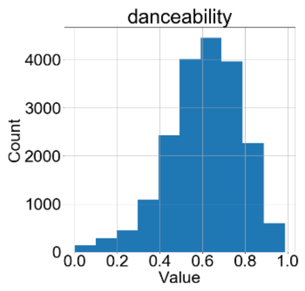
Before we make the song recommendation for the playlist. We will be analyzing the data first. The purpose of this section is to find out what makes a song popular. First, let's look at what do those terms for each song mean before we can proceed our analysis:

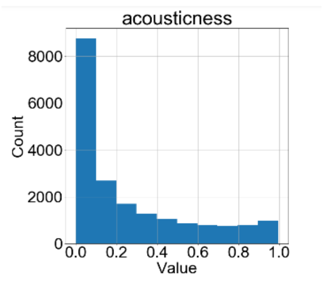
- Popularity: The rank of the top 100 tracks on the Billboard.
- Danceability: Float value from 0 to 1. A value of 0.0 is least danceable and 1.0 is most danceable.
- Energy: Float value from 0 to 1. Describe how energetic a track is.
- There are 12 unique integer value in this variable, those are music keys which are: C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab, A, A#/Bb, B
- Loudness: Measurement for relative loudness of tracks.
- Mode: Integer, 1 for major, 0 for minor.
- Speechiness: Float value from 0 to 1. Probably the proportion of speech in the music.
- Acousticness: Float value from 0 to 1. How acoustic a song is.
- Instrumentalness: Float value from 0 to 1. How likely the song is an instrument track/no vocal. Over 0.5 means the track is more likely to be an instrumental track.
- Liveness: Float value from 0 to 1. How likely the song is recorded live.

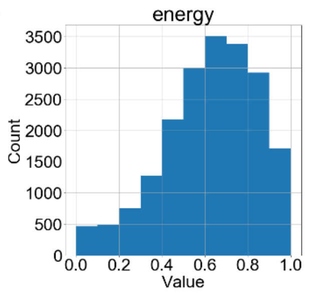
- Valence: Float value from 0 to 1. Higher means happier, lower means more depressed.
- Tempo: Beats per minute.

Since the data set contains the Billboard Hot 100 from year 1927 to 2020. It would be a good idea to look at tracks that are more recent. Let's take a subset for 2010 or later. And here are some findings with histogram on different audio features.

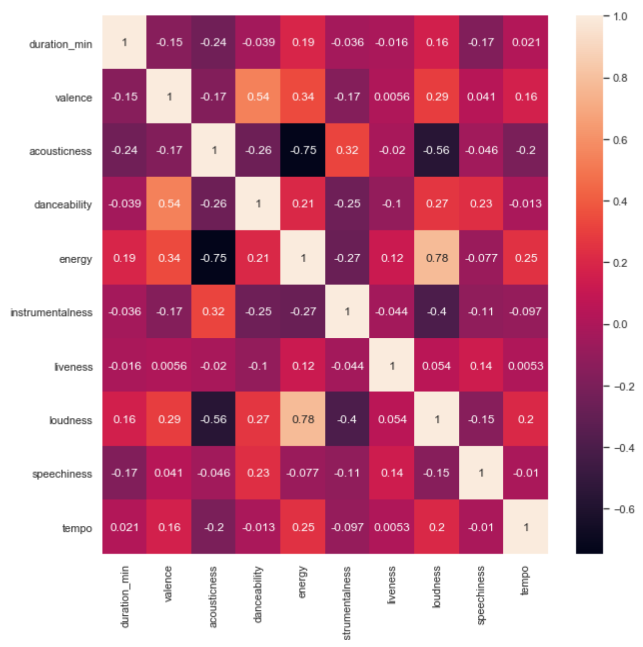








Most of the tracks on the top 100 billboard are more likely to be recorded in studio rather than live recording. Also, most of the songs are considered less acoustic. From the two histograms on the right, we can see that the tracks are more likely to be suitable for dancing, which makes sense that the tracks are more likely to be energetic as well. To have a better look at how each audio feature is correlated to each other. A correlation matrix with a heat map would be helpful.



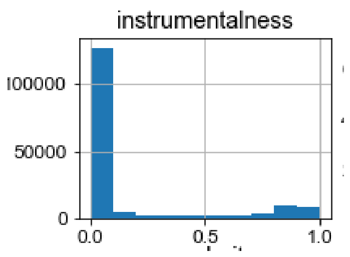
	duration_min	valence	acousticness	danceability	energy	strumentalness	liveness	loudness	speechiness	tempo
duration_min	1	-0.15	-0.24	-0.039	0.19	-0.036	-0.016	0.16	-0.17	0.021
valence	-0.15	1	-0.17	0.54	0.34	-0.17	0.0056	0.29	0.041	0.16
acousticness	-0.24	-0.17	1	-0.26	-0.75	0.32	-0.02	-0.56	-0.046	-0.2
danceability	-0.039	0.54	-0.26	1	0.21	-0.25	-0.1	0.27	0.23	-0.013
energy	0.19	0.34	-0.75	0.21	1	-0.27	0.12	0.78	-0.077	0.25
instrumentalness	-0.036	-0.17	0.32	-0.25	-0.27	1	-0.044	-0.4	-0.11	-0.097
liveness	-0.016	0.0056	-0.02	-0.1	0.12	-0.044	1	0.054	0.14	0.0053
loudness	0.16	0.29	-0.56	0.27	0.78	-0.4	0.054	1	-0.15	0.2
speechiness	-0.17	0.041	-0.046	0.23	-0.077	-0.11	0.14	-0.15	1	-0.01
tempo	0.021	0.16	-0.2	-0.013	0.25	-0.097	0.0053	0.2	-0.01	1

This heap map is a correlation matrix for audio features of track from year 2011 to year 2020. There are not many strong correlations between different audio features. We can still see that 'energy' and 'accousticness' are moderately negatively correlated, so as so as 'loudness' and 'accousticness'. At the same time, 'loudness' is moderately positively correlated with 'energy', meaning energetic tracks are relatively louder.

Data transformation and machine learning model testing:

In this section, we are trying to transform the data and fit them into machine learning models that give us predictions of the song's popularity. We've tested different methods to produce machine learning models, and one of the models that gives us better results is **LinearRegression**.

In this method, we treat the observations of predictors as multi-dimensional vectors, and the prediction 'population' would be a linear combination of them. The predictors include different audio features such as valence, acousticness, etc.



We notice that most of the 'instrumentalness' values in the data are very close to zero, with some values that are heavily right tailed. Meaning including this predictor will not provide much to our model. And the heavy tailed values (leverage point) would 'pull' the regression and affect the accuracy of the model. Therefore, we removed this predictor. Also, from the correlation matrix, there doesn't appear to be strong correlation between predictors that can cause multicollinearity issues(i.e. highly correlated predictors can affect the accuracy of the model). This means it is a safe assumption for us to use a linear regression model for prediction.

Here are the predictors for the model:

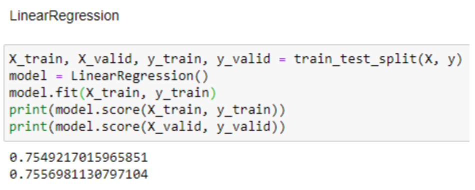
Audio features:

'key', 'mode' ,'explicit' are integer values in the data, and set as categorical variables. 'valence', 'acousticness', 'danceability', 'energy', 'liveness', 'loudness', 'speechiness', 'tempo', are float values. Most of them appear to be approximately normal distributed, but features like 'liveness' and 'speechiness'' are right skewed, therefore, log/sqrt transformation were performed on those. As a result, they became more 'normalized' which is helpful for linear regression. All of the above features then got scaled from 0 to 1 using MinMaxScaler().

Other features: 'duration_min' and 'year'. Track length('duration_min') with more than 12 minutes are considered as outlier, and those tracks got removed. Then a MinMaxScaler was used to scale this variable. For the predictor 'year', we made it a categorical variable. Intuitively speaking, the preference of people changes as the year passes by, setting 'year' as a categorical variable would make sense here.

Model testing:

A 75/25 split on train/valid is applied on the data set.

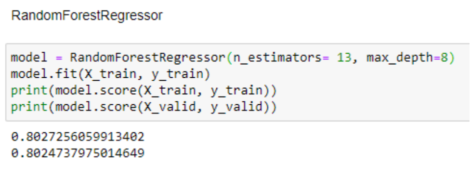

LinearRegression

```
X_train, X_valid, y_train, y_valid = train_test_split(X, y)
model = LinearRegression()
model.fit(X_train, y_train)
print(model.score(X_train, y_train))
print(model.score(X_valid, y_valid))
```

```
0.7549217015965851
0.7556981130797104
```

LinearRegression gives accurate scores around 75% on both test and valid dataset. And it doesn't appear to introduce any overfitting issue. Since the goal of this section is to give better prediction results rather than interpreting data, using **RandomForestRegressor** might give better accuracy.

Here is the result for **RandomForestRegressor** :


RandomForestRegressor

```
model = RandomForestRegressor(n_estimators= 13, max_depth=8)
model.fit(X_train, y_train)
print(model.score(X_train, y_train))
print(model.score(X_valid, y_valid))
```

```
0.8027256059913402
0.8024737975014649
```

By setting the number of estimators to 13, and max depth to 8, we were able to obtain scores above 80% accuracy for both train and valid data set, which is a 5% improvement over the linear regression method. Increasing the parameter count further does not help much but also introduces overfitting issues. And it is less cost efficient.

The root mean squared error was also calculated for the result of the prediction. The RMSE is calculated by the formula on the right. The RMSE for the RandomForestRehressor method is around 9~10. This means we can expect the prediction to be around 10 ranks off the actual rank of the track.

$$RMSE=\sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

Building song recommendations for that playlist

ETL

We were fairly lucky as both datasets we obtained were clean and well organized. To further modify the data we simply dropped rows from both datasets containing null values in important fields (i.e. in feature fields for songs, or song name fields for playlists). Given the size of the datasets being used, we decided to do as much as we could in Spark, and the rest in Pandas.

Methodology

The process we employed to answer our question is outlined below:

1. Train a model to predict the popularity of a song based on its features. Improve the model that was generated earlier to obtain an accuracy of over 80% on validation data.
2. Perform an inner join using playlists and songs to bring song features together with playlists
3. Perform a groupBy operation on each playlist to obtain the average of each song feature in [‘name’, 'valence', ‘year', 'acousticness', 'danceability, 'energy’, 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo’, 'duration’, 'mode']
4. Using the vector of averaged song features for each playlist, find the five closest songs by Euclidean Distance for each playlist
5. Using the model trained in Step 1, predict the popularity of each song
6. Multiply the predicted popularity with the inverse of the Euclidean Distance to obtain a ‘score’
7. Return a dataframe with the top 5 recommendations for each playlist, sorted by ‘score’

Implementation

All steps were implemented in Python. Steps 1, 4, 5, 6, and 7 were performed using Pandas and Steps 2 and 3 were performed using Spark. Step 8 was done subjectively using our collective understanding of music.

For Step 1, we set out to improve the model we had previously generated for predicting popularity. This time we made use of a Random Forest Regressor, which trains by creating a number of decision trees, assigning those trees a subset of data points, and having each tree predict the output. The average of these predicted outputs is then taken as the prediction. We decided to use this type of model because in our testing it outperformed all of the others (k-nearest neighbour, GaussianNB), and can run efficiently on large data sets.

Steps 2 and 3 were basic Spark commands, however implementing step 4 was tricky. The issue we faced was figuring out how to calculate the Euclidean Distance between each playlist’s averaged values and those of every song in a tractable amount of time. To do this, we created Pandas dataframes from the playlist and song Spark dataframes. We then made use of Pandas `df.apply(find_recommendations)` to do the calculations. Inside our function `find_recommendations()` we made use of `np.linalg.norm(playlist_values - song_values)`, which calculates the l2 norm (the Euclidean Distance). We then chose the 5 songs with the lowest distance to the playlist averages.

Next, Inside the same function we performed Steps 5, 6, and 7 using our previously generated model to return the final result. The formula used to rank each song was as follows:

```
score = predicted_popularity * euclidean_distance^-1
```

This was done for every playlist, and for every song.

Results

Model

Our use of a Random Forest Regressor improved our ability to predict song probability to around 80 - 82% for validation data, an increase of about 5-7% over the Linear Regressor. Multiple attempts were made to boost the performance of the model through tweaking of the parameters. Ultimately, we stuck with 50 samples per decision tree and a maximum depth of 15. This increase in accuracy betters our ability to predict popularity, and as a result provides a more accurate way to rank song recommendations from Step 6.

Song Recommendations - Performance

Our decision to do the bulk of our calculations in a Pandas `df.apply()` was not the most optimal way to go about the calculation. This decision was made for coding convenience. Namely, we were unsuccessful in manipulating Spark to perform a Euclidean Distance calculation, and doing the equivalent in Pandas was a couple lines. That being said, even on our local machines we are able to provide song recommendations for about 7 playlists / second. This amounts to about 25,200 in an hour, for just under 6 hours to create recommendations for the complete set of 170,000-ish playlists in the dataset. Considering that each playlist must be compared to around the same number of songs (around 29 billion comparisons total) we are content with that amount of time.

Note that we did not compute recommendations for the entirety of the playlists, but rather for a subset of around 10,000.

Song Recommendations - Relevance

We were unable to find a good metric for evaluating how good our song recommendations were other than by listening to a playlist and the recommendations and deciding for ourselves. We each spent about an hour doing this, and to our surprise the recommendations were actually pretty good. The recommended songs are similar in beat and feel, and they are very similar in genre, which is impressive considering we did not make use of any genre data in our calculations.

An example playlist is shown below:

Playlist: “Summer Hipster”	
Artist	Song
Midnight Magic	"Beam Me Up - Jacques Renault Remix"
Radiohead	"Bloom (Harmonic 313 RMX)"
Bon Iver	"Calgary"
Weekend	"Coma Summer"
The National	"Exile Vilify (From the Game Portal 2)"
Jamie xx	"Far Nearer"
Lykke Li	"I Follow Rivers (I Follow You Tyler The Creator Mix)"
Nicolas Jaar	"I Got a Woman"
The Miracles Club	"Light of Love (Cut Copy Re-vision)"
frank ocean	"Nature Feels (feat. MGMT)"
Metronomy	"The Look"
SBTRKT	"Wildfire (feat. Little Dragon)"
Washed Out feat. Caroline Polachek	"You and I"

And it’s corresponding recommendations:

Song	Score	Popularity Prediction	Euclidean Distance
“Smoothie Song”	338.40738704	66.99340326	0.19796673
“Pacer”	311.93839573	69.78183245	0.22370389
“Loneliness Loneliness”	305.72454333	65.97869849	0.21581093
“Miss Judy's Farm”	305.09892852	64.40495454	0.21109532
“One Night of Love”	298.43100789	67.7322196	0.22696107

Conclusion

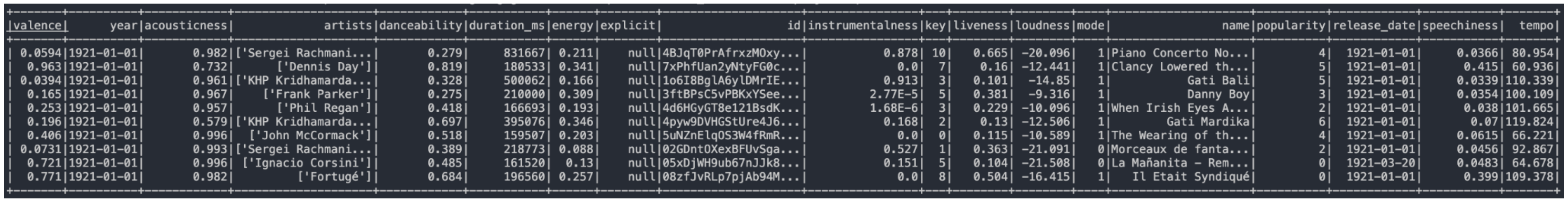
This was a very enjoyable data science experiment. We were able to generate song recommendations pretty well (it is hard to say how well. We think Spotify must measure their recommendation success by how many users actually add those recommendations to their playlists. Alas we do not have that information). We were able to use our learning from our first experiment with predicting popularity to further improve our model, and use this model to help with our recommendations. Note that we could have just used the provided ‘popularity’ data from the dataset in our calculations but wanted to see how far we could get with the model (it felt more fun that way).

To improve on our work here we think that making use of ‘genre’, ‘artists’, and other categorical data could help improve our model accuracy. 81% isn’t bad (I suppose if we could predict which stock would rise with 81% accuracy we’d be rich) but it isn’t groundbreaking either. Another important metric that is missing from the data is listen count. Many users create playlists with many songs in them but skip through a lot of them to find ones they prefer. Without this information it is harder to recommend songs that users would like to hear.

Another improvement we would aim to make is performing the entirety of the calculations in Spark. This would likely lead to massive performance increases in runtime.

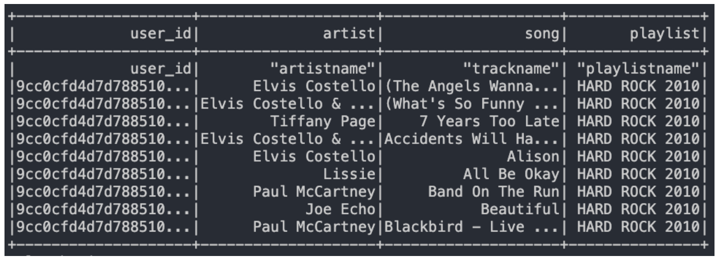
Appendix A - Data Preview

Spotify song dataset (10 rows)



```
+-------+----------+------------+--------------------+------------+-----------+------+--------+--------------------+----------------+---+--------+--------+----+--------------------+----------+------------+-----------+-------+
|valence|      year|acousticness|             artists|danceability|duration_ms|energy|explicit|                  id|instrumentalness|key|liveness|loudness|mode|                name|popularity|release_date|speechiness|  tempo|
+-------+----------+------------+--------------------+------------+-----------+------+--------+--------------------+----------------+---+--------+--------+----+--------------------+----------+------------+-----------+-------+
| 0.0594|1921-01-01|       0.982|['Sergei Rachmani...|       0.279|     831667| 0.211|    null|4BJqT0PrAfrxzMOxy...|           0.878| 10|   0.665| -20.096|   1|Piano Concerto No...|         4|  1921-01-01|     0.0366| 80.954|
|  0.963|1921-01-01|       0.732|      ['Dennis Day']|       0.819|     180533| 0.341|    null|7xPhfUan2yNtyFG0c...|             0.0|  7|    0.16| -12.441|   1|Clancy Lowered th...|         5|  1921-01-01|      0.415| 60.936|
| 0.0394|1921-01-01|       0.961|['KHP Kridhamarda...|       0.328|     500062| 0.166|    null|1o6I8BglA6ylDMrIE...|           0.913|  3|   0.101|  -14.85|   1|           Gati Bali|         5|  1921-01-01|     0.0339|110.339|
|  0.165|1921-01-01|       0.967|    ['Frank Parker']|       0.275|     210000| 0.309|    null|3ftBPsC5vPBKxYSee...|         2.77E-5|  5|   0.381|  -9.316|   1|           Danny Boy|         3|  1921-01-01|     0.0354|100.109|
|  0.253|1921-01-01|       0.957|      ['Phil Regan']|       0.418|     166693| 0.193|    null|4d6HGyGT8e121BsdK...|         1.68E-6|  3|   0.229| -10.096|   1|When Irish Eyes A...|         2|  1921-01-01|      0.038|101.665|
|  0.196|1921-01-01|       0.579|['KHP Kridhamarda...|       0.697|     395076| 0.346|    null|4pyw9DVHGStUre4J6...|           0.168|  2|    0.13| -12.506|   1|        Gati Mardika|         6|  1921-01-01|       0.07|119.824|
|  0.406|1921-01-01|       0.996|  ['John McCormack']|       0.518|     159507| 0.203|    null|5uNZnElqOS3W4fRmR...|             0.0|  0|   0.115| -10.589|   1|The Wearing of th...|         4|  1921-01-01|     0.0615| 66.221|
| 0.0731|1921-01-01|       0.993|['Sergei Rachmani...|       0.389|     218773| 0.088|    null|02GDntOXexBFUvSga...|           0.527|  1|   0.363| -21.091|   0|Morceaux de fanta...|         2|  1921-01-01|     0.0456| 92.867|
|  0.721|1921-01-01|       0.996| ['Ignacio Corsini']|       0.485|     161520|  0.13|    null|05xDjWH9ub67nJJk8...|           0.151|  5|   0.104| -21.508|   0|La Mañanita - Rem...|         0|  1921-03-20|     0.0483| 64.678|
|  0.771|1921-01-01|       0.982|         ['Fortugé']|       0.684|     196560| 0.257|    null|08zfJvRLp7pjAb94M...|             0.0|  8|   0.504| -16.415|   1|   Il Etait Syndiqué|         0|  1921-01-01|      0.399|109.378|
+-------+----------+------------+--------------------+------------+-----------+------+--------+--------------------+----------------+---+--------+--------+----+--------------------+----------+------------+-----------+-------+
```

Spotify Playlist dataset (10 rows)



```
+--------------------+--------------------+--------------------+---------------+
|             user_id|              artist|                song|       playlist|
+--------------------+--------------------+--------------------+---------------+
|             user_id|        "artistname"|         "trackname"| "playlistname"|
|9cc0cfd4d7d788510...|      Elvis Costello|(The Angels Wanna...| HARD ROCK 2010|
|9cc0cfd4d7d788510...|Elvis Costello & ...|(What's So Funny ...| HARD ROCK 2010|
|9cc0cfd4d7d788510...|        Tiffany Page|    7 Years Too Late| HARD ROCK 2010|
|9cc0cfd4d7d788510...|Elvis Costello & ...|Accidents Will Ha...| HARD ROCK 2010|
|9cc0cfd4d7d788510...|      Elvis Costello|              Alison| HARD ROCK 2010|
|9cc0cfd4d7d788510...|              Lissie|         All Be Okay| HARD ROCK 2010|
|9cc0cfd4d7d788510...|      Paul McCartney|     Band On The Run| HARD ROCK 2010|
|9cc0cfd4d7d788510...|            Joe Echo|           Beautiful| HARD ROCK 2010|
|9cc0cfd4d7d788510...|      Paul McCartney|Blackbird - Live ...| HARD ROCK 2010|
+--------------------+--------------------+--------------------+---------------+
```