Analysis of Spotify's Worldwide Daily Song Ranking

Adriana Concha
Mario Garrido
Tamara Gutiérrez

February 9, 2021

1 Goal

In order to investigate worldwide music preferences we analyise Spotify's Worldwide Daily Song Ranking dataset, using Apache Spark, Hadoop and Python. Finding the original dataset to be lacking some critical data for our analysis, we decided to extend the dataset by joining the required information, scraped from the MusicBrainz project. It was our goal to immerse ourselves into this data to paint a landscape of global music preferences, and, caveat emptor, it's mostly pop and reggaeton.

2 Data

The Spotify's Worldwide Daily Song Ranking is a 3441198 records long dataset, which encodes the following information taken from the top 200 songs in spotify (sorted by number of streams), per day: Position, track name, artist, number of streams, URL, date and Region of playback. This dataset comes from Kaggle[1], in a single *.csv* file, with a filesize of 369MB. At this size it is only operable through code, as using any text editor to inspect it is severely cumbersome. Since our original motivation was to have a better grasp of music reproduction behaviour, not only relating to world regions, but also to artist's origins, we found it necessary to complement the information of this dataset with the country of origin of each artist and, if possible, the area where they began their careers. In order to do this we used, as mentioned previously, the Musicbrainz dataset[2], but as the sheer size of the project makes it prohibitive in time to operate, we decided (perhaps not wisely so) to use the html requests server made available by Musicbrainz[3]. The downside of this approach was double: The request server had a very strict policy regarding the number of requests it would serve (but at least not as strict as to issue an ip ban after beating

at it for 8 hours straight) and it was necessary to complete our extended dataset in order to perform the queries we wanted answers for.

3 Methods

The tool selected to perform the queries, over the Spotify database, was Apache Spark[4], mainly because it seemed to us that it was the most stable service among the options installed in the cluster (not a single complaint in U-cursos), and the relative ease of use when compared to other options. Also, given the size of our Dataset (not so big), Spark was a better option than MapReduce,for example, and in this case, provides a better performance than other options. This tool, was used with Java 8 in Eclipse for build the jar file, then it was uploaded to the Hadoop cluster, and the results were stored in HDFS.

Since the original and the extended database are both contained in a single file, it wasn't hard to reference. The other widely used technology was the Python language, which was used to perform the requests to the Musicbrainz server, collect the information and compute the join of the Spotify dataset with the 2 columns of information we wanted to add. It should be noted that the integration of this information was not trivial, since the Musicbrainz database has, sometimes, several entries related to a single artist or intellectual property, and due to its size it is not uncommon to find artists with very similar names. It was of paramount importance to devise a way to properly filter and merge the information retrieved by the queries to the server. In order to do this we employed the following strategy:

- Query for artist's name and filter all results with respect to name similarity. To do this we keep a result only if the artist name used to query is a substring of the artist name retrieved.
- Query for song's name and filter all results with respect to song name similarity and artist name similarity. We again use the substring criteria defined above.
- Match the arid (artist id) of the artists found in the song name query with the arid of the artists found in the artist name query. If anything comes out of this, it will constitute our best guess.
- Select, from the list of best guesses, those which have the minimum difference in number of characters in artist name fields.
- If the best guesses list is empty, then apply the minimum difference in number of characters to the artist name query and select the best from there.
- If both lists are empty, give up and return empty list.

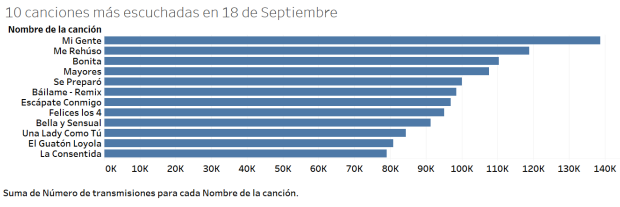
Since, performing this procedure over the complete dataset would have been a brutality, it was performed for all the unique (Artist, Song) pairs, which has a more manageable size of 19923 tuples. From these (Artist, song) tuples, now augmented to (Artist, song, Proposed Artist, Country of origin, Begin area), we constructed the Extended Spotify's Worldwide Daily Song Ranking dataset (has a nice ring to it, gotta admit it). At this point we were ready to query to our heart's content.

4 Results

The following results compile the outputs of the queries in the projectSpark file, which have pretty straightforward names with respect to the plots. At this point it was fairly obvious that our country had been completely absorbed by global culture, and everywhere you looked it was mostly reggaeton and local pop. So, basically, a small portion of the Spotify catalogue accounts for most of the platform's reproductions, but I guess that's just Zipf's law manifesting itself once again.

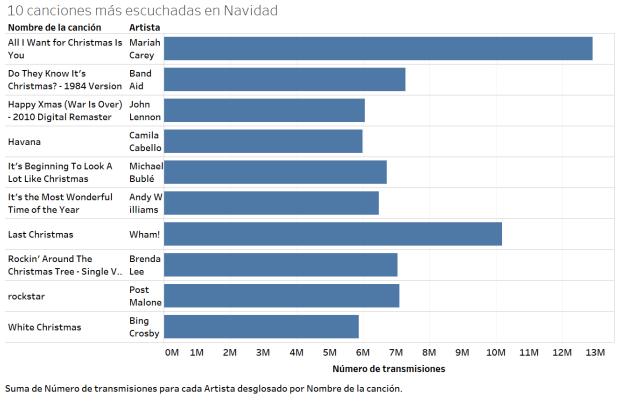
- Best songs: Get the number of times a SongsArtist was Top 1 in the Ranking of Spotify in 2017 (independent of the country)

- Top streamed songs in Chile: Get the most streamed songs in September 18, 2017 in Chile

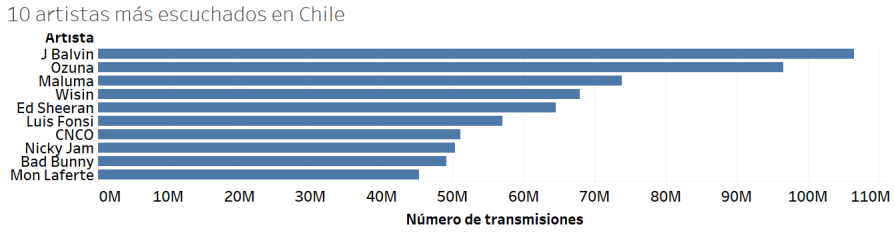


  I mean, you've got to hand it to us, there's little time for cuecas when you can have some good old báilame, from classical artist Nacho.

- Top streamed songs in Christmas, worldwide



- Top streamed artists in Chile in 2017



5 Conclusion

We learned that even if you're an impolite person, that has never even considered to use exponential backoff, there are still kind-hearted servers out there willing to tolerate your transgressions. We experienced, first hand, the woes of trying to join databases that have absolutely no consideration to the chaos that a single ',' can generate in a misplaced record, this being a major problem until we were forced to simply purge all commas from the extended dataset.

Regarding the tools used, we experienced first hand the advantages of using Apache Spark, not only in terms of performance but mainly in ease of use (and did we mention that it never faltered in the cluster?).

References

[1] `https://www.kaggle.com/edumucelli/spotifys-worldwide-daily-song-ranking`

[2] `https://musicbrainz.org/`

[3] `https://musicbrainz.org/doc/Search_Server`

[4] `https://spark.apache.org/`