Last.fm dataset: Characterizing artists and users to stop arguing with your friends about music

Olivier Lam (274550), Robin Zbinden (274236), Stanislas Jouven (260580)
EPFL, Switzerland

Machine learning for behavioral data project (CS-421)

***Abstract*—In this project, we analyze a music dataset collected on the *Last.fm* platform to find some similarities between users and artists. We propose different embeddings techniques to transform the users and artists into vectors. Then we apply several clustering algorithms to group the users and artists by similarity. Finally, we introduce different methods to find an optimal list of artists, i.e., a playlist, for a group of arbitrary users.**

I. Introduction

With the growth of online streaming music, the area of music recommendation systems have gained in importance over the past years. A large number of songs listened are now proposed to the users based on the suggestion of one these systems. Moreover, these systems are able to build playlists depending on the user's preferences and conditioned on a specific type of music. Improving the quality of these systems is then important to provide a better experience to the user.

Efficient music recommendation systems rely on estimating the similarities between music contents, but also between the users. In this project we aim to characterize the users and artists by answering to the three following research questions:

1) Can we group the artists by similarity?
2) Is it possible to find a correlation between the user interests and the age, gender and country origin?
3) Can we predict an optimal list of artists for a group of arbitrary users?

The two first questions are independent, but the methods used to answer them are similar. Furthermore, answering the two first questions is necessary to provide a good answer to the third question, since we need to understand and characterize the artists and users to produce the optimal list of artist. The term optimal in the third research question is difficult to define and we show in subsection IV-D several ways to do it.

To answer these questions, we use the *last.fm* dataset which contains demographic information about the users of a music platform as well as the number of plays of each user for each different artists. We first describe and clean this dataset to build the so-called user-artist matrix. This matrix is then fed to an embedding technique to transform each artist and user into a vector. In addition, we apply clustering algorithms on these vectors to find similarities among artists and users. Finally, we answer to the third research question by finding a list of artists satisfying a group of arbitrary users.

II. Explanatory Data Analysis

The provided *last.fm* dataset is divided into two smaller datasets. In this section, we describe and perform an explanatory data analysis on these two datasets.

A. User's Demographic Informations

The first dataset contains the demographic informations, such as the age, the gender, the origin and the sign up date on the platform of 359,347 users. In particular, these user's features are crucial to answer to the second research question. We do not use the sign up date in this project.

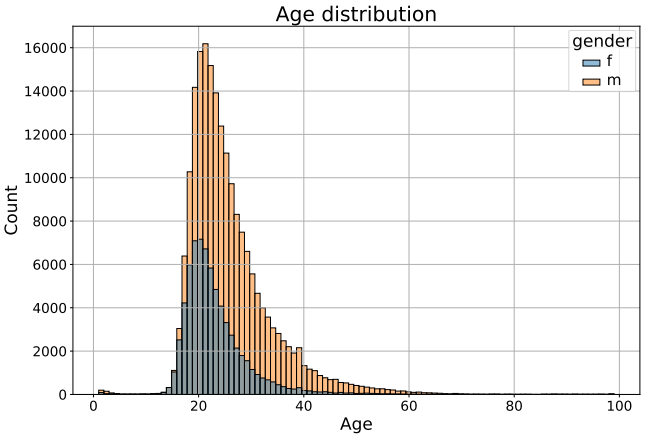

Figure 1. Histogram showing the age's distribution per gender of the 359,347 users on the *last.fm* platform.

Figure 1 shows the age distribution of the male and female users of the dataset. We clearly notice that the young population dominates and that it is biased according to the gender. Indeed, the mean age of the users is 25.1 years old and the median age is 23 years old, being way larger than these statistics on the whole world population. Moreover,

the dataset contains 241,642 male and 84,930 female users, which represent respectively 67.24% and 22.76% of the users. The age and gender are missing for 74,900 and 32,775 users, respectively.

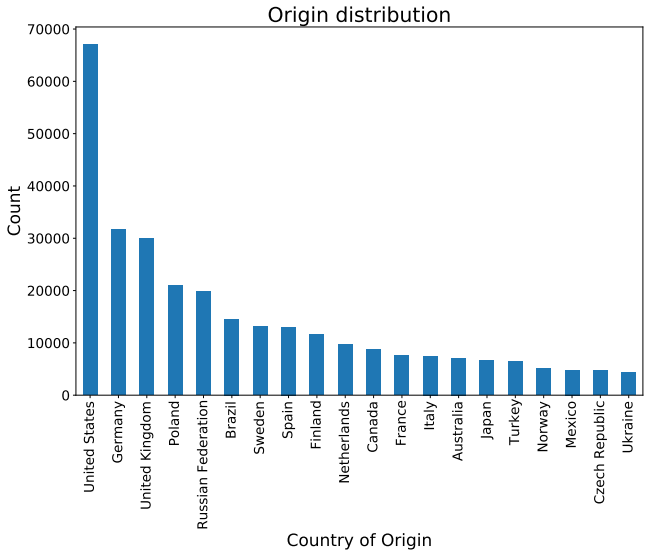

Figure 2. Bar plot showing the top 20 countries of origin's distribution of the users.

In Figure 2, we can observe the origin's distribution of the users. We again notice that this dataset is biased since it contains users from western countries in majority, e.g., almost no Asiatic or African countries are represented. Most of the users are from the US (18%).

B. User's Consumption Dataset on Last.fm

The second dataset contains information on which users listened to which artists. In other words, for each user, we can know how many times she/he listened each different artist. Artists are represented by an artist id and by their name. However there is no one-to-one mapping between this id and this name. Indeed, we observe that for the same artist id, the artist name is the same but often spelt differently, e.g., the famous rock group *guns n' roses* has only one artist id, but is also spelt *guns & roses*, *guns nroses*, or *guns and roses*. This is not really a problem since we first consider the artist id and then retrieve the name of the group with this id.

There are 160,112 unique artist id. However, the artist id is missing for 226,137 rows and the artist name for 204 rows. Since the dataset is large (17,535,655 rows) and that without the artist id it is difficult to identify uniquely a group, we decide to not consider the rows with missing artist id. For the missing artist name, we still have the artist id so this is not a problem.

Now we can compute some statistics on the number of plays by artists and by user. Figure 3 shows us an histogram

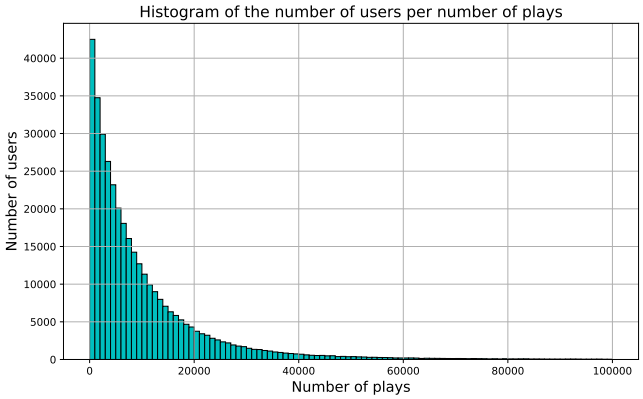

Figure 3. Histogram with the total number of plays of each user. Only the users with less than 100,000 plays are considered.

of the number of plays of each user. We first remark that the majority of the users are not frequent users of the *last.fm* platform, since the number of plays is small. In addition, we observe that this histogram looks like a heavy-tail distribution, as there is still a consequent number of users who played songs a lot of times (more than 40,000).

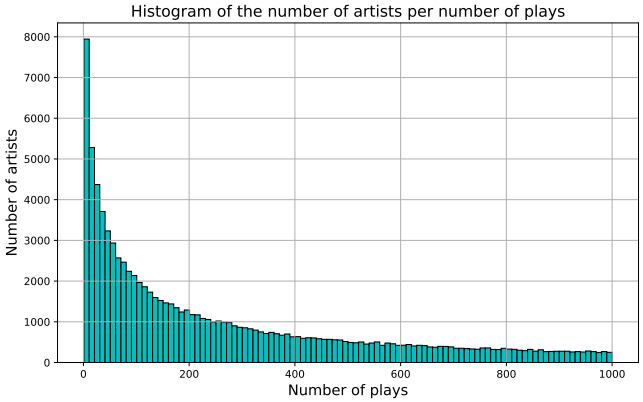

Figure 4. Histogram with the total number of plays of each artist. Only the artists with less than 1,000 plays are considered.

In Figure 4, we can see how often the artists are played by the users. We again notice that this distribution looks like a heavy-tail distribution (note that we only consider artists played less than 1,000 times, but the distribution is also heavy-tailed if we consider all the other artists). It implies that a large fraction of the artists are not played a lot, but there is still a substantial number of artists who are played a lot. These statistics are important to bear in mind when we build the user-artist matrix in the next subsection.

C. User-artist matrix

Now we would like to handle an object summarizing all the relationships between users and artists, where a relationship is simply the number of plays of a user for an

artist. We can do this by generating the user-artist matrix (or plays matrix), whose rows correspond to artists and columns to users. The entries are simply the number of plays of a user for an artist. This is of course a large sparse matrix, since as we can see in Figure 5, most of the users have played less than 100 different artists. However, considering all the users and artists for this matrix is a problem for two reasons:

- Some embedding techniques such as PCA have some difficulty to handle very large matrix without increasing considerably the need of computational and time resources.
- We want to consider only meaningful artists and users, i.e, artists or users which have enough and reasonable interactions with the *last.fm* platform.

In order to do that, we define 4 different thresholds to reduce the number of artist and especially the number of users:

- $T1$: Minimal number of plays per user.
- $T2$: Minimal number of artists per user.
- $T3$: Minimal number of plays per artist.
- $T4$: Minimal number of users per artist.

However, we note that these 4 thresholds are interdependent, e.g., if we reduce the number of users because of threshold $T1$, then threshold $T4$ could not be satisfied anymore since it is possible that there would not have enough users for some artists. Therefore, to satisfy all of them simultaneously, we need to apply an iterative algorithm to achieve convergence to a list of users and artists satisfying all the thresholds. This is done by alternatively applying $T1$ and $T2$ together and then $T3$ and $T4$ together.

We select the values $T1 = 100$, $T2 = 49$, $T3 = 50$ and $T4 = 5$, based on the distribution shown in Figure 3, Figure 4, and Figure 5 and also on the objective to reduce considerably the number of users. Finally, there are 104,897 users and 45,076 artists remaining.

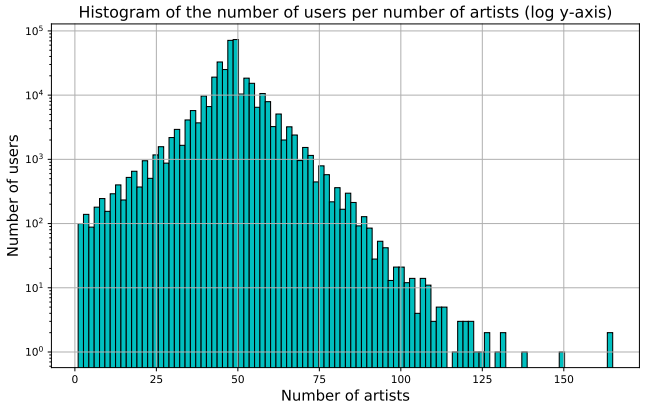

Figure 5. Histogram with the number of users per number of artists played. The y-axis is a logarithmic scale.

III. Embeddings

In this section, we propose different models to represent both the artists and the users as vectors. The objective is that the distance between two vectors corresponding to two similar artists (or users) is small. In that way, the nearest neighbors of an artist (or a user) in the vectorspace should be the most similar artists (or users). We consider only artists and users satisfying the 4 thresholds defined in the previous section. Then in a second time we propose and use several measures to assess the quality of these embeddings.

We can use different sort of matrices as input to the embeddings algorithms:

1) **User-artist matrix**: Simply the user-artist matrix as defined in subsection II-C. We use it to compute the embeddings used by word2vec.
2) **User-artist matrix normalized**: The user-artist matrix but with columns being normalized. We use it to compute the embeddings used by PCA.
3) **User-artist matrix with tf-idf**: The user-artist matrix but we apply term frequency–inverse document frequency (tf-idf) on the entries. We use it to compute the embeddings used by PCA.

A. Principal Component Analysis

Principal component analysis (PCA) is an algorithm which consists of finding the principal components of a matrix, and then to perform a change of basis based on them and eventually subsample only the most important components. Therefore it enables us to reduce the number of dimension of a matrix, but while still keeping a large fraction of the information contained in the original matrix. To do so, it uses the SVD decomposition and truncates the resulting matrices. This algorithm enables us also to choose the dimension k of the output.

Using this method, we are now able to associate a vector to each artist and each user in the user-artist matrix. Moreover, we can transform a new fresh user or artist into a vector just by knowing its list of plays and by using the matrices returned by the SVD decomposition. The size of this vector is the parameter k, and we compare and evaluate different values of k in order to choose the best one.

B. Word2vec

Word2vec is one of the most popular model to transform words into vectors. From a text corpus, it uses a neural network model to find and learn words associations. In particular, each word is represented by a vector and this vector should encapsulate the semantic meaning of the word. Fortunately, word2vec is easily adaptable to other sort of data different from words. In particular we can then define two new algorithms which we call artist2vec and user2vec. We will focus and explain in detail how artist2vec works, but user2vec has a similar functioning just by swapping the users and the artists.

As its name suggests, artist2vec has the objective to convert artists to vectors. Since now we have artists instead of words, we need to define the context of these artists. This is done by defining for each user a list of 10 artists, where 10 is then the window context of the artists. These 10 artists are sampled with replacement among the artists listened by the user. The probability to sample one artist is also weighted by the number of plays of this user for this artist. The final input of artist2vec is then the list (one element by user) of list of artists.

C. Embedding evaluation

In the previous subsections, we showed several embedding techniques. However, to be able to compare the performances of the different embeddings, we need to develop evaluation procedures. In the following explanation of the methods, we will consider the artist embedding but the same reasoning can be applied on the user embedding.

We propose two methods to assess the performance of an embedding:

1) *Jumper ratio* and *Position ratio*
2) Axis projection evaluation method

However, the second method didn't work so well and therefore the explanations and results of this method is in the Appendix . We focus then on the first method in this subsection. Let k be the number of users selected to compute the *Jumper* and *Position ratio*.

In order to introduce the *Jumper ratio*, the *User jumper* and the *Random jumper* need to be defined. To compute the *User jumper* value, we retrieve k users from the set of users and for each selected user, we select a pair of two artists which were listened by this user. We then simply compute the euclidean distance between the two based on the vectors produced by the embedding. The *Random jumper* is computed similarly, but the k pairs of artists are instead drawn at random from the whole set of artists.

From the *User* and the *Random jumper* expressions, the *Jumper ratio* is formulated as follows:

$$Jumper\ ratio = \frac{\sum_{i=1}^{k} User\_jumper_i}{\sum_{i=1}^{k} Random\_jumper_i}$$

Computing the *Jumper ratio* enables to capture how well the embedding behaves compared to random:

- $Jumper\ ratio \approx 1$ means the embedding was established in a quasi random manner.
- $Position\ ratio < 1$ means the embedding is doing better than random (lower results are better).
- $Position\ ratio > 1$ means the embedding is doing worse than random.

Our second measure to assess the quality of the embedding is the *Position ratio*. We again retrieve k users and a pair of artists for each of them. For every pair, we define the *Position index* as being the index position of the first artist in the list of the nearest neighbors of the second artist. If n is the total number of artists, then the *Position ratio* is defined as follows:

$$Position\ ratio = \frac{\sum_{i=1}^{k} Position\_index_i}{nk}$$

Similarly to the $Jumper\ ratio$ value we compare the result obtained from the random embedding:

- $Position\ ratio \approx 0.5$ means the embedding was created in a random fashion since on average two artists from a random model would be separated on average by $\frac{n}{2}$ artists (in terms of the nearest neighbor ranking).
- $Position\ ratio < 0.5$ means the embedding is doing better than random (lower results are better).
- $Position\ ratio > 0.5$ means the embedding is doing worse than random.

D. User embedding results

The *Jumper* and *Position ratio* have been applied to the user embedding with $k = 2000$ since it give us a good trade-off between time complexity and bias. The results can be found in Table I. We can clearly observe that the embedding using PCA with the tf-idf weights and with 25 dimension looks like the best user embedding. Indeed, it has the lowest *Jumper ratio* as well as the lowest *Position ratio*.

	User embedding			
	Jumper Ratio		Position Ratio	
	mean	std	mean	std
norm_pca25	0.79	$2.9e^{-4}$	0.27	$1.4e^{-4}$
norm_pca50	0.84	$2.9e^{-4}$	0.29	$1.5e^{-4}$
norm_pca100	0.86	$2.6e^{-4}$	0.3	$1.5e^{-4}$
tf-idf_pca25	0.76	$2.0e^{-4}$	0.23	$1.4e^{-4}$
tf-idf_pca50	0.82	$2.1e^{-4}$	0.26	$1.5e^{-4}$
tf-idf_pca100	0.85	$2.2e^{-4}$	0.29	$1.6e^{-4}$
word2vec25	0.98	$2.1e^{-4}$	0.5	$1.5e^{-4}$
word2vec50	0.99	$2.1e^{-4}$	0.49	$1.4e^{-4}$
word2vec100	0.98	$2.1e^{-4}$	0.5	$1.5e^{-4}$

Table I
JUMPER AND POSITION RATIO RESULTS COMPUTED WITH THE USER EMBEDDING. 25, 50 AND 100 ARE THE DIMENSIONS OF THE OUTPUT.

E. Artist embedding results

Similarly as for the user embedding, the *Jumper* and *position ratio* have been computed for $k = 2000$. The results are shown in Table II.

However, the comparison between the different embedding techniques is not as obvious as it is for the user embeddings. Indeed, the lowest *Jumper ratio* score as been achieved for *norm_pca100* and *tf-idf_pca50* whereas the lowest *Position ratio* score has been achieved by *tf-idf_pca25*. However, *norm_pca100* have a too high *Position ratio* to be kept. Thus, the embedding using PCA on tf-idf weights with 25 and 50

dimensions have been selected by these methods. Further analysis is done in the next section in order to select the best one.

Unfortunately, the two ratios obtained with word2vec are not good. However, if look at the neighbors of some famous artists in the vectorspace, we obtain similar artists, e.g., For *Keith Jarrett*, we get *Bill Evans*, *Chick Corea*, and *Duke Ellington*, all of them are masters of piano Jazz music.

	Artist embedding			
	Jumper Ratio		Position Ratio	
	mean	std	mean	std
norm_pca25	0.47	$1.1e^{-3}$	0.38	$1.6e^{-4}$
norm_pca50	0.45	$7.3e^{-4}$	0.39	$1.6e^{-4}$
norm_pca100	0.44	$6.0e^{-4}$	0.41	$1.5e^{-4}$
tf-idf_pca25	0.45	$1.1e^{-3}$	0.37	$1.6e^{-4}$
tf-idf_pca50	0.44	$7.2e^{-4}$	0.38	$1.6e^{-4}$
tf-idf_pca100	0.43	$6.0e^{-4}$	0.41	$1.5e^{-4}$
word2vec25	2.62	$5.8e^{-4}$	0.51	$2.1e^{-4}$
word2vec50	2.64	$5.8e^{-4}$	0.51	$2.1e^{-4}$
word2vec100	2.65	$5.9e^{-4}$	0.51	$2.1e^{-4}$

Table II
JUMPER AND POSITION RATIO RESULTS COMPUTED WITH THE ARTIST EMBEDDING. 25, 50 AND 100 ARE THE DIMENSIONS OF THE OUTPUT.

IV. APPROACHES

In order to group the artists and users by similarity, we apply clustering methods to the resulted embeddings and perform multiple evaluation techniques on the clusters. In this section IV, we explain our first approach by using K-Means and describe how Gaussian Mixture Modelling, combined with HDBSCAN, can improve the clustering model. Ideally, we want to choose the embedding that yields the best jumper ratio and position ratio scores. However, we notice that there is not a users embedding that performs much better than another. Thus, in order to extract the best possible model, we compare the clusters from the following two embeddings: *tf-idf_pca25* and *tf-idf_pca50* for the artists **and** the users. We note that before feeding the embedding to clustering models, we standardize it and this pre-processing step is very important as it solves the issue that the clustering model assigns the same label to more than 95% of the dataset. Finally, we present the evaluation methods to determine the best model.

A. K-Means

K-Means is one of the most simple and popular clustering methods in the recent years. This is the reason why we decide to use it in order to group similar users and artists.

In order to get the best clustering model, we compute the silhouette, the BIC and the distorsion score for a range of values for K, which is the parameter that defines the number of cluster in the K-Means algorithm.

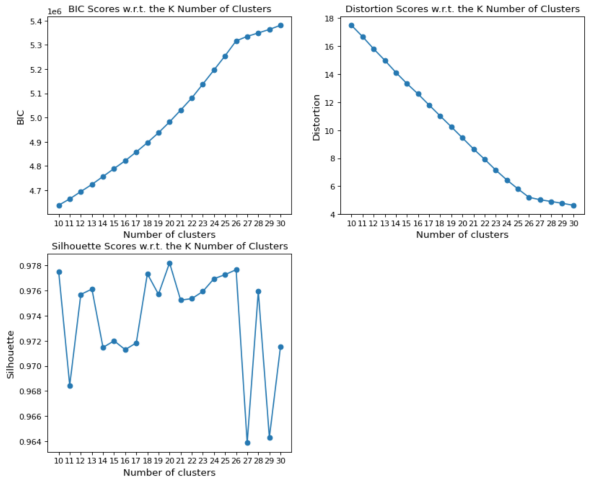

Figure 6. Silhouette, BIC and Distorsion score for multiple values of K when applying K-Means to the *tf-idf_pca25* embedding.

Figure 6 shows the score of the different cluster evaluation method. We recall that the the lower the Bayesian Information Criterion (BIC) and the distortion scores are, the better the clustering is and this is the opposite for the silhouette score, i.e., we are looking for the highest silhouette score. Combining the three plots, it is difficult to determine the optimal K without a deeper analysis. Moreover, the behavior for the distortion score is expected and imply that $K = 26$ is optimal since from that K, the decreasing rate of the distortion score is much smaller. For the silhouette score, $K = 26$ yields the third highest score and since the behavior of the BIC score is not as expected, we decide not to consider it. This bad behavior may come from the fact that we face a noisy input and thus, the clustering may not be perfect.

Finally, for all the reasons listed above, we choose $K = 26$ for the final clustering of the users where the 2D-projection can be seen on Figure 7.

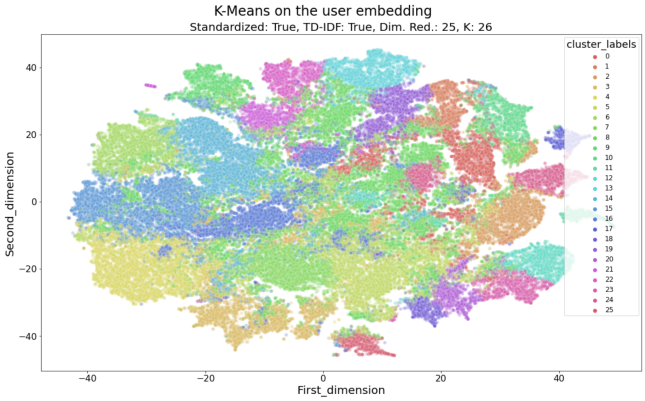

Figure 7. 2D representation of the users with the corresponding cluster label with K-Means at $K = 26$.

Figure 7 shows some encouraging results even though it

is far to be perfect. We can observe that the clusters are split quite homogeneously among the whole input space, which is a good sign. However, the dataset remains very noisy, which involves the high density of the points in the figure above. Moreover, we can observe that some visible cluster are represented by multiple labels, which should not be the case. For those reasons, we decide not to analyze the results of this clustering model and to find another approach in order to reduce the noisiness of the data first.

B. Gaussian Mixture Modelling and HDBSCAN

There are two main reasons which yield the bad results when applying K-Means to the users embedding. The first comes from the fact that the data is still noisy and the second can be explained by the limitation of the K-Means algorithm. Indeed, K-Means assumes that each cluster is a sphere around its center and performs the clustering on a fixed number of clusters. Since the input data is of dimension 25 and 50 and is also noisy, it is almost impossible that the data is represented by "spheres" through the input space, which would enable a good performance of K-Means. For that reason, we develop a new approach with the use of Gaussian Mixture Model and HDBSCAN.

Before diving into the detailed techniques, let us summarize the idea of this approach. Recall that we are interested to group the users that have a similar taste of music, i.e., listen to similar artists. Therefore, it would be interesting to first build a clustering model (from the artist embedding) such that each cluster represents a distribution of the artists (described in subsubsection IV-B1). Then, we are able to compute, for each user, her/his "artist vector" which is given by all the artists she/he listens to and feed this to the clustering model on the artists. Thus, each user is represented by a vector of probabilities where each probability tells how much the user listen to the artists of a given cluster (described in subsubsection IV-B2). Finally, this gives a more fine grained embedding of the users on which we can apply HDBSCAN in order to cluster the users (described in subsubsection IV-B3).

1) Artist model: The first task with the new approach is to build a clustering model for the artist. We decide to choose the Gaussian Mixture Model (GMM) to perform the model since its main advantage over K-Means is that it comes with different options to constrain the covariance of each class. Thus, it doesn't force each cluster to be spherical, which is an issue as seen in subsection IV-A. Again, we find the best model by tuning the parameter K, which defines the number of cluster. This time, we minimize the Davies Bouldin score, which has proven its efficiency in previous works. $K = 24$ yields the minimum score as shown in Figure 8 and we use it to build the model.

2) A better user embedding: For each user, we have her/his number of plays per artist. Moreover, each artist is represented by a vector from the embedding *tf-idf_pca25* or *tf-idf_pca50*. Thus, we create a new vector for each user where we sum all the artists vectors (element-wise) for all the artists he has listened to. Naturally, we assign a weight for each artist vector in the summation, where the corresponding weight is the normalized number of plays by the user. Then, the resulted vector is fed to the model as described in subsubsection IV-B1, which outputs a probability vector over the 24 classes. Applying this method for all the users yields a new user embedding, which should by less noisy than the initial user embedding.

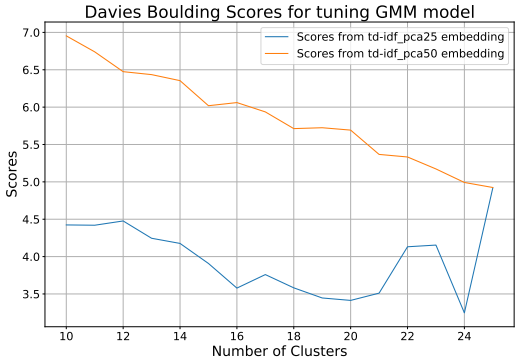

Figure 8. Davies Bouldin score on the Gaussian Mixture Model for different value of K, the number of clusters.

3) User model: As the user embedding should already group the user with their music preference, the goal now is to class the users which have the same preferences. Therefore, we choose to use the HDBSCAN clustering algorithm. This method find the optimal number of cluster by itself and is based on a density-based criterion. Therefore, it does not assume that a cluster has a convex shape. We also tune the model by running multiple values for the parameters S, which represents the minimum size of one cluster and M_s, which represents how conservative the clustering is. For a high value of M_s, the clustering will set more points as "noise" and for high value of S, the less cluster there will be. Figure 9 shows the improved version of the user clusters. We can observe that the data is really less noisy compared to the clustering with K-Means in Figure 7; the clusters are clearly formed and almost each cluster correspond to a unique class. The next sections will use this user-based clustering in order to answer to our research questions.

C. Results and Validation

Recall that the evaluation on the embeddings (in subsection III-E) did not enable us to find one that performs much better than the others. Therefore, we apply the approach as described in subsection IV-B to the following two embeddings on the artists: *tf-idf_pca25* and *tf-id_fpca50*. Choosing the best clustering model on the users from the two embeddings is not an clear and simple task. Therefore, for each user cluster, we compute the top 10 artists that have

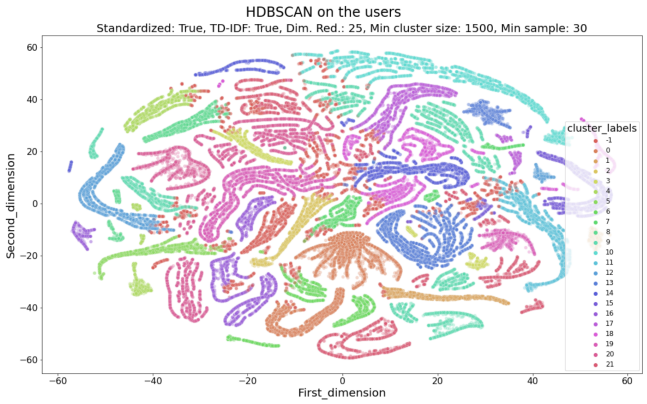

Figure 9. 2D representation of the users with the corresponding cluster label with GMM and HDBSCAN.

the highest number of average plays among the users and manually check if that correspond to the user demographic informations.

The manual validation shows that the clustering on the embedding *tf-idf_pca25* works the best. Indeed, the cluster that have the highest female proportion (30.53% vs 22.76% from the whole dataset) yields the following top artists: *Britney Spears, Kylie Minogue, Madonna, Christina Aguilera etc.* Moreover, the cluster that contains the oldest users (29.5 years old vs 25.1 years old from the whole dataset) yields the following top artists: *Rush, The Beatles, Madonna, Duran Duran etc.* All of those artists were active during the 70s and the 80s, which make sense. Furthermore, when checking the top artists per cluster, we observe that most of the top artists comes from the same era of music and/or produce the same genre of music. For example, one of the user cluster has the following top artists: *Nas, Kanye West, Jay-Z*, which are all American rappers and another user cluster yields *Iron Maiden, Metallica, Megadeth,* etc. as top artists and they are all Heavy Metal artists.

In the other hand, the results for the clustering on the embedding *tf-idf_pca50* is not as good. There is no cluster that yields a specific group of top artists, as some of the famous artists, i.e., The Beatles, Radiohead, Coldplay etc. are appearing in almost every user cluster. We can explain this bad behaviour by the fact that the dimension of the embedding is too large and with the curse of the dimensionality, it becomes very difficult to find a good clustering model.

Before performing any embedding from section III to subsubsection IV-B3, we keep 10% of the users in a test set in order to test our clustering model. We recall that all the evaluation methods (from the embeddings to the clustering model) are performed exclusively on the training data. Finally, we feed the clustering model on the users with the test data and compare the top artists for each user clusters from the training and testing set. Again, we only perform a manual check since we do not have additional information about each artist as its genre tag per example. Recall that the users cluster that contains the highest proportion of females has *Britney Spears, Kylie Minogue, Madonna, Christina Aguilera etc.* as top artists. By looking at the top artists from the test users that are classified in this cluster, we have *Britney Spears, Kylie Minogue, Linkin Park, Girls aloud etc*. Except for *Linkin Park*, we have again only female Pop stars, which is a good sign! For the cluster of users that listen to the American rappers, taking the test users from this cluster yields *Nas, KoRn, 2pac, Kanye West etc*. Again, except for *KoRn*, we have a majority of American rappers. Therefore, we observe that the testing dataset confirms the good performance of our clustering model and that the model can definitely group the users that have the same music preferences.

D. Playlist Prediction

Having developed the embedding and the clustering model enable to answer to the third research question which is to predicts an optimal list of artists for a group of arbitrary users. However, answering to this question is not trivial since it can be achieved through many ways depending on how we define the term optimal. In this subsection, we propose several methods to find this optimal list of artists.

1) Mean vector user: The first method computes simply the mean vector of the group of users. Then, we select the nearest neighbor of the user mean vector, namely *user_ref*, in order to select the artist, namely *artist_ref*, with the maximum number of plays *user_ref* listened to. The final step consists of randomly selecting artists appearing in the same cluster as $artist\_ref$.
However, this procedure could be far from optimal at predicting artists satisfying the group of user. Indeed, computing the mean vector of different users could yield a *user_ref* that has completely different preferences compared to the users from the group. Therefore, it is possible that no user in the group would be satisfied by the playlist.

2) Majority vote: By taking into consideration the problem evoked on the previous procedure, we develop another method based on the $majority\ vote$ principle. We first retrieve the set of all the artists played by the users in the group and then select the artist cluster having the most number of artist. Similarly to the first procedure, we randomly retrieve artists appearing in the same cluster as the one we selected. The benefit of this procedure relies on the fact that the music genre having the greater percentage of users will be played. Yet, selecting artists belonging to only one cluster is not always the best solution since the whole group of users could only listen to one type of music. With the next procedure, we try to address this downside.

3) Mean vector user per cluster: This method consists in mapping each user of the arbitrary group to the cluster

where he/she belongs. We then apply the $mean\ vector\ user$ method to each user cluster that contains at least one user. Indeed, for each user cluster, we average the user vectors of the users belonging to that cluster. Similarly to $mean\ vector\ user$, by using the nearest neighbor method, we retrieve the corresponding artist, namely $artist\_ref$ and then randomly select artists appearing in the same cluster as $artist\_ref$. The final list of predicted artists consists of the concatenation of the artists found for every selected user cluster. Therefore, as opposed to the other two developed procedure, this algorithm will predict different type of music liked by at least one person in our reference group of users.

V. Discussion

Throughout this project, we exclusively worked with unsupervised models which makes the validation part non trivial. Although unsupervised, we developed procedure to validate and determine the best embedding or the best clustering method. However, further analysis could have been done with additional data:

- A dataset mapping artists with their corresponding genre: Having this additional data would validate our clustering methods since artists of a same genre should be often grouped on the same cluster.
- User feedback: In order to validate our list of artists for a group of users, it is essential to have user feedback. Indeed, the quality of a playlist is only determined by the user's tastes.

Moreover, we have seen that answering to the third research question without this user feedback is not a easy task. The procedures which we propose to find an optimal list of users are simply ideas of what could make a good playlist and we are aware of all the limitations of these procedures.

Concerning the first research question, we have seen that the different embeddings techniques enable us to group the artists by similarity quite efficiently. Indeed, we have seen that artists playing a similar genre of music and during the same era were often in the same cluster.

Finally, to answer to the second research question, we have seen that the gender and age in particular play an important role into the type of music listened by the users.

VI. Conclusion

In this project, we proposed different methods to find similarities between users and artists. We showed that we were able to group artists by genre and era of music without having any prior knowledge on them by using several methods such as PCA, word2vec and GMM. Moreover, we found that there is a correlation between the gender and the age of a user and the genre of music she/he listens. Finally, we proposed three different methods to find an optimal list of artists to a group of arbitrary users.

As mentioned in the Discussion Section, integrating other datasets such as one containing the music genre tag for each artist could be a possible direction for the future steps of this project. Another possibility could be to incorporate the methods we proposed into a user interface which would give music recommendations to the users. It could also enable us to collect some precious user feedback on our methods.

References

[1] I. Waller and A. Anderson, "Community embeddings reveal large-scale cultural organization of online platforms," 2020. [Online]. Available: https://arxiv.org/pdf/2010.00590.pdf

APPENDIX A.
AXIS PROJECTION EVALUATION METHOD

We present here another procedure to evaluate the performance of the embeddings. However, we obtained unexpected results from this method. We put here the results and explanations for completeness.

Introduced in the work for the Reddit social network [1], the axis projection method quantify the positioning of artists along selected cultural dimensions. We chose to look at dimensions in our embedding corresponding to gender.

In order to compute the axes vectors that characterize the dimensions, we first identify the seed pair. The seed pair should be as close as possible but differ only in the cultural dimension. Considering the dataset, we select *Rihanna* and *Drake*, women vs men artist oriented for the gender axis.

With the aim of robustly characterizing these cultural differences we expand the seed pairs with 9 other similar pairs automatically created. Based on the same idea of very similar artists that differ only on the cultural dimension: for each channel we create 10 pairs between the artist and its nearest neighbors. All the pairs of artists are then ranked according to the cosine similarity of their vector difference with the vector difference of the seed pair. From this ranking the top 9 pairs are selected to end up with 10 pairs (the seed pair + the 9 selected pairs).

The vector difference of the selected pairs are finally averaged to produce a single vector that robustly depict the dimension.

Once a vector for an axis has been obtained, artists can be assigned a score measuring the similarity this artist has with the artists of the seed pair. Thus, a artist much more similar to one artist of the seed pair than the other will have a score at the poles (large positive or negative score) whereas neutral artists compared to both artist will have a score close to 0. This score is computed by projecting the artist vector on the axis vector. Let $\vec{c}$ be the artist vector and $\vec{d} = \frac{1}{n}\sum(B_i - A_i)$ the normalized axis vector where (A_i, B_i) represent the selected pairs. The community score is then defined as:

$$cos(\vec{c}, \vec{d}) = \frac{\vec{c} \cdot \sum(B_i - A_i)}{n\left\|\vec{d}\right\|} = \frac{1}{n\left\|\vec{d}\right\|}\sum(\vec{c} \cdot B_i - \vec{c} \cdot A_i)$$

Figures 10, 11 and 12 depicts the results obtained for the embedding using PCA on tf-idf weights with 25 dimensions. Nevertheless, by looking at Figures 11 and 12 we observe that the embedding classify correctly known artists. Indeed, artists such as *Jennifer Lopez*, *Beyoncé*, *Shakira* or *Selena Gomez*, being female oriented have a negative projection score whereas famous male oriented artists such as *Florida*, *DJ Khaled*, *Big Sean* or *Kid Cudi* have a positive projection score.

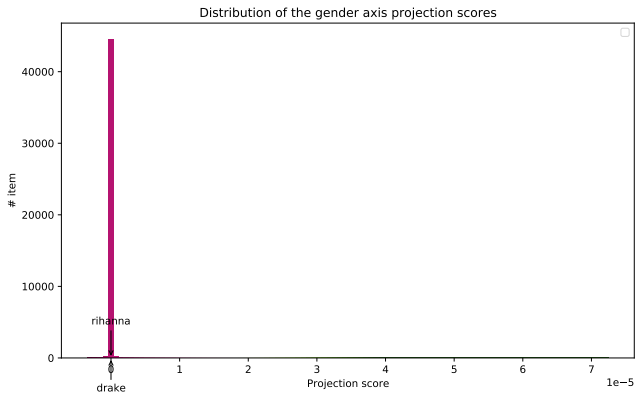

Figure 10. Distribution of the gender axis projection scores.

	name	projection
1432	dj khaled	-1.966886e-08
26085	big sean	-1.215343e-08
31830	florida	-2.002780e-08
39590	kid cudi	-2.086029e-08

Figure 11. Manually famous artists retrieved, male oriented.

	name	projection
4222	beyoncé	2.037449e-07
33653	shakira	1.914074e-07
40247	selena gomez	3.994675e-07
42287	jennifer lopez	2.086489e-07

Figure 12. Manually famous artists retrieved, female oriented.