Configuring Song Playlist based in user preferences: A Case Based Reasoning application

Alejandro Ariza-Casabona ALEJANDRO.ARIZA.CASABONA@EST.FIB.UPC.EDU

Víctor González-Navarro VICTOR.GONZALEZ.NAVARRO@EST.FIB.UPC.EDU

Jorge Sierra-Acosta JORGE.SIERRA@EST.FIB.UPC.EDU

Aram Javier Villasana-Falcón ARAM.JAVIER.VILLASANA@EST.FIB.UPC.EDU

Abstract

Configuring and planning a playlist of songs in order to meet the preferences of a user is a complex task. User preferences are related to many parameters describing a song, which make impossible to find an exact solution when trying to recommend the perfect song (or list of songs) for a user in a certain moment. In this scenario, a recommender system based in Case Based Reasoning (CBR) shows up as a perfect tool for giving an appropriate solution to the configuration and planning of a play-list of songs. This project presents the architecture of a CBR system which uses basic attributes of a user profile (age, gender), as well as attributes stating her current-state preferences (preferred music genres, required concentration, current level of happiness, current level of energy) and her future-state preferences (desired change in level of happiness, desired change in level of energy), in order to configure a play-list of songs with the recommended parameter values for: valence, tempo, danceability, energy, loudnesss, acousticness and instrumentalness. Some domain rules are used to plan the order of the songs, based in the literature about the effect of valence and tempo in the levele of happiness and energy of people (3).

Keywords: Case Based Reasoning, Recommender System, Hierarchical Memory, Playlist Recommendation

1. Introduction

Many people enjoy every day listening to music for several reasons. Some of these reasons are related to our mood. Sometimes, we choose to listen a certain song because we want to keep feeling the way we feel; but sometimes, we choose a song trying to change our mood. Actually, extensive research relating music and the mood of the listeners has been conducted. As a result of these studies, relevant effects over people mood, have been attributed to different parameters of music. Among these parameters, the valence of music seems to have effect about the happiness people feel; while the tempo of a song, seems to affect the perception of a person about her level of energy (3). However, common people are not experts evaluating the valence and tempo of a song, and besides these parameters, there are many other parameters used to describe music that are not easy for people to appreciate. Moreover, if we think about a person trying to choose the best combination of songs (playlist) that could help her to get a certain desired effect about her mood, and we consider the thousands of thousands of songs that exist, we realize that this is a very complex task for a person. This situation represents a perfect scenario for getting support of a recommender system.

The goal of this project is to architecture a Case Based Reasoning (CBR) solution, capable of considering the profile and preferences of a user, in order to recommend a music play-list that produce a certain effect, desired by the user, over her own mood. As in any typical CBR, the solution architecture must be able to store a library of cases, perform retrieval of most similar cases according to an user query, adapt a solution from existing cases, evaluate the suggested solution and learn relevant new cases. To get this, our solution handle cases as vectors of attributes, containing the attributes corresponding to the identification of the case, as well as the attributes corresponding to solution of the case. The vector of attributes is explained in detail in the section 4.1 Case Structure. The cases are stored in a hierarchical structure using a discriminant tree, in order to speed up searches and give a better structuring and interpretation of the case base (1). To perform retrieval, our solution follows a partial matching approach, which means that we retrieve the cases with most similar values using only the attributes for which there are stored cases in subsequent branches of the hierarchy, to avoid reaching empty branches. Retrieval algorithm details can be found at the section 5.1 Retrieval. In the adaptation stage, we perform a configuration of the parameters which compound the solution, as well as a planning of the order of the songs (playlist), that comply with solution parameters, To perform the configuration , we take advantage of domain knowledge to apply some rules that correct the means of the values of the parameters of the solution (all solution parameters are numerical), according to the preferences of the user in concentration, and the current and future (desired) states of happiness and energy. See section 5.2.1 Configuration for further details. To perform the planning of the playlist, the algorithm starts by defining and initial point (first song) and a direction to search for next songs. The first point is selected giving more weight to the attributes corresponding to the desired future statuses of happiness and energy. The direction of the search is determined by hand-made rules, which looks to provoke the desired future state in the mood of the user, considering the starting situation described by the user. The search continues until completing the number of songs desired in the playlist. See section 5.2.2 Planning for

further details. To perform evaluation, besides receiving the direct feedback of users, the algorithm integrates an automatic evaluation procedure described in Section 5.3 Evaluation. To perform learning, the algorithm stores only cases with satisfactory solutions (Section 5.4).

When starting to build this CBR, one of the main challenges was that we could not found an appropriate set of cases to feed our system. Because of this, we have to develop a survey in order to get enough cases for feeding the initial case library. The survey was conducted through a web service, and distributed with the support of family and friends. A total of 176 cases were collected through this survey, using 130 cases for feeding the initial case library and 46 cases for evaluating the system. The survey questionnaire can be observed in the annexes and also can be accessed at the link:

https://newqtrial2015az1.az1.qualtrics.com/jfe/form/SV_6PDJdNah1rwA8qp.

2. Analysis of Requirements

Considering the perspective of user experience, we identified the following requirements for the system:

- **Understand the user and her preferences**: Ask for basic attributes of user profile (age, gender),attributes stating her current-state preferences (preferred music genres, required concentration, current level of happiness, current level of energy) and her future-state preferences (desired change in level of happiness, desired change in level of energy)
- **Return a list of songs that contribute to reach the desired effect**: Configure the ideal parameters of a song that helps the user to reach the future state she is pursuing, and get most similar songs to this ideal as the recommendation.
- **Order the list of songs in a playlist with sense**: Deliver the list of songs ordered in a way that make sense to the user that is trying to reach a certain desired state about her mood.
- **Real-Time Response**: Deliver the recommendation to the user almost immediately after recording her preference, to keep her interest.
- **Get feedback of the user about the recommendation**: Ask to the user if the recommendation is satisfactory or not, in order to qualify the recommendation.
- **Learn success solutions and forget mistakes**: Retain successful solutions for later recommendations and forget wrong solutions

On the other hand, the technical requirements for the system are identified as follows:

- **Maximum Time Response**: Time is a Priority. The system needs to get back an answer in no more than 1 sec, since the user is expecting a real-time response. Because the database structure is based on a tree, time response is expected to be faster than

a simply linear stored database, however the playlist generation (see 5.2.2) requires to iterate over all the songs in a linear manner, so time complexity is lower bounded by $\Omega(n)$ in the best of the cases in the size of the database.

- **Maximum Memory Size**: Currently, memory size is not a priority. There is no memory limit fixed yet, since at this first stage of development, we are concerned first about generating as many new cases as possible to get more accurate solutions. Expected memory complexity for this system is $O(n)$ in the size of the database.

3. Functional Architecture

All Case Based Reasoning systems are described by a four-step process which we summarize below. The input to the Case Based Reasoning is a new case whose attributes describe the personal information of a user, such as the age, the gender and others. This new case inputs the **retrieval** system, whose objective is to find the closes instances in the Case Library. Since the proposed solutions of the closest instances may differ significantly, they are input to the **adaption** (reuse) system, which tries to find an optimal solution based on the solutions proposed by the retrieved instances, and the difference between the new input and the closest instances. Once the solution has been found for the new input pattern, the Case Based Reasoning must evaluate the performance of the proposed solution (**evaluation** or revise system) either by performing the solution in the real world, asking to a human expert or running a simulation. Finally, the **learning** (retain) system determines how the new case along its proposed solution can be use to improve the Case Based Reasoning. This task can be achieved by learning from success and learning from failure. Due to the fact that our application is used to generate a song playlist, we have added an additional system to the hole architecture whose goal is to create a playlist from the updated solution of the new case.

We illustrate in Figure 1 the architecture of our Case Based Reasoning system.

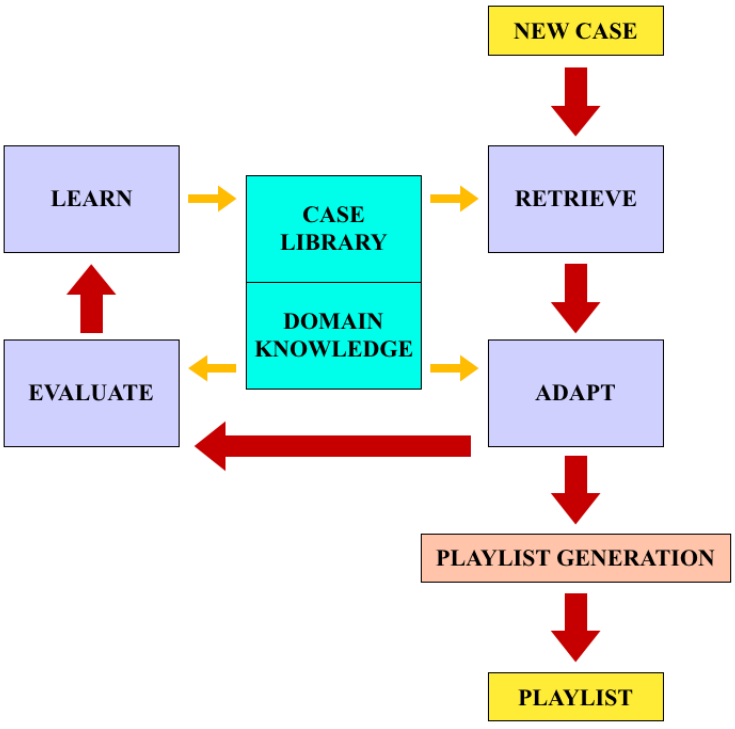

Figure 1: Case Based Reasoning architecture.

4. Proposed Solution

4.1. Case Structure

The Case Structure consists of an attribute-value vector containing 8 antecedents representing the user profile, his/her current status and wishes after listening the next song as well as 7 other features corresponding to the solution. The user profile contains the age, gender ("male" or "female") and the 3 preferred music genres out of the following options: "Instrumental-Classical", "Vocal", "Blues", "Jazz", "Rock", "Hard Rock", "Dance", "Pop", "Reggaeton", "Reggae", "Latin". The current status involves concentration requirement of the current activity (ordinal feature considering the states "not at all", "some concentration", "yes, my full concentration"), the level of happiness (ordinal feature considering the states "really unhappy", "not so happy", "neutral", "happy", "very happy") and the level of energy ("very calm", "calm", "neutral", "with some energy", "with a lot of energy"). The wishes after listening the next song refers to the new level of happiness (3 levels from "less happy" to "happier" passing through "the same") and energy (3 levels from "more relaxed" to "with more energy" passing through "the same"). An example of a new case for which our algorithm needs to find a solution could be:

Age, Gender, Preferred Genres, Concentration, Happiness, Energy, Future Happiness, Future Energy
37, 'Male', 'Blues,Instrumental-Classical,Rock', 'some concentration', 'happy', 'calm', 'the same', 'the same'

Regarding the solution features, we have considered different numerical metrics of the song. These metrics include "danceability", "energy", "loudness", "acousticness", "instrumentalness", "valence" and "tempo".

4.2. Case Library

Regarding the Case Library, there are multiple options to choose from, starting with the decision of either flat, structured(hierarchical, Object-Oriented, etc), mixed or unstructured. We have opted for the hierarchical case library structure in order to improve the search efficiency. At this point, we wanted to not only build an efficient case library but also an interpretable one. This is the reason why we decided to implement a Discriminant Tree (in which the nodes are actual attributes) rather than Shared Feature Network (in which the nodes correspond to a clustering run). Therefore, the structure will not be optimal with respect to the distribution of the training set but it becomes easier to visualize and manage. Moreover, we have implemented several heuristics in order to select the best attributes at a particular node:

1. Knowing the number of possible values an attribute can take, the first score computes a ratio of the unique values of the attribute that remains in the filtered dataset of the corresponding tree branch with respect to the total number of possible values.
2. If the number of possible values of an attribute is extremely large e.g. 65, the attribute is penalized with a correction factor.
3. In case two or more attributes have the same main score, the algorithm takes into account the counts of each attribute-value and picks the one with lowest standard deviation in its counts vector.

The first point tries to allocate cases to most of the possible next branches. For instance, if an attribute can take 3 values (red, orange, blue) and at this node, the instances no longer contain the value blue for this attribute, the branch corresponding to this attribute will not have any case and we are forced to set its following node as a leaf of the tree. Therefore, what we try to achieve using this heuristic is to delay this phenomenon from happening as much as possible so that the case library will be more distributed among its branches.

The second heuristic occurred due to an incoming feature with lots of unique attribute-value pairs that was extremely partitioning the cases at the beginning of the tree. Therefore, we opted for penalizing it.

The third heuristic is also based on a better structuring of the case base. Supposing that the same ratio of branches is going to contain cases, we want to reward those attributes for which the number of cases destined to each branch is as similar as possible. Thus, an attribute that divides the remaining cases in three equal-sized parts (e.g. 30-30-30) will be preferred rather than another that separates it irregularly (e.g. 70-10-10).

Furthermore, in case of a numerical attribute exists in the cases (e.g. age), the algorithm performs KMeans clustering on the specific attribute in order to capture the distribution of the feature in discrete classes and obtain an efficient partition. The number of classes used to discretize numerical attributes is 5.

Last but not least, the Case Base also contains a dataset that stores the available songs including the following attributes: [“Song”, “Artist”, “Genre”, “danceability”, “energy”, “key”, “loudness”, “mode”, “speechiness”, “acousticness”, “instrumentalness”, “liveness”, “valence”, “tempo”, “duration_ms”, “time_signature”, “id_Spotify”, “Link_Spotify”, “type”, “uri”, “track href”, “analysis_url”, “Link_YouTube”, “Questionnaire”]. The songs library has a flat structure for optimal song retrieval when building the playlist.

5. Case Based Reasoning cycle

In this section we describe the technical details of the implementation of the four main modules in the Case Based Reasoning system.

5.1. Retrieval

One of the most important steps consist in retrieving meaningful instances of the Case Library given a target problem. If the retrieved solutions are similar to the new input case, we are likely to predict a successful solution to the target problem. Therefore, given a new input case defined by: the age, gender, top three preferred music genres, current activity, level of happiness and energy, and finally future desire of level of happiness and energy; the system must determine the most similar cases according to these attributes.

To perform such task, we use a **partial matching approach**. This procedure consists in, at each node, follow the closest attribute-vale and the second closest attribute-value. Then, when moving towards the closest attribute-vale, we repeat the same process described before. Nonetheless, when moving towards the second closest attribute-value, we follow at this point the closest attribute-values without considering the second closest. Regarding the stopping criteria, we have decided to stop the search before arriving at a node without instances in the Case Library. Therefore, we may be retrieving cases that are similar to some attributes but not to others since they have not been evaluated. We show in Figure 2 a graphical illustration of the retrieval implemented.

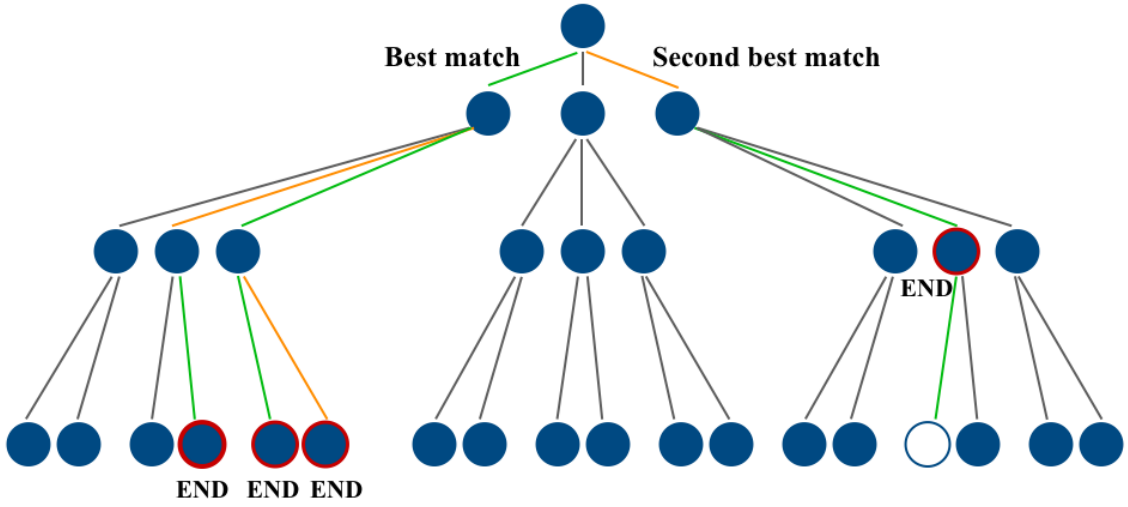

Figure 2: Retrieval system. White nodes indicate leafs covering any instance in the Case Library and blue nodes indicate leafs covering at least one instance.

Due to the fact that most of the input feature are categorical, we have defined a distance metric to deal with these cases. For example, regarding the top three music genres, we com-

pute the intersection between the top three genres of the input case and the top three genres of the feature-values. Consequently, the option "Blues, Jazz, Rock" is closest to "Blues, Jazz, Pop" than to "Vocal, Instrumental-Classical, Latin". In case of ties, we have also defined, using prior knowledge, the distance between different music genres. That is, "Blues" is closer to "Jazz" than "Blues" to "Reggaeton". Below I detail the distance scale used between the different music genres:

Classical–Vocal–Blues–Jazz–Rock–Hard Rock–Dance–Pop–Reggaeton–Reggae–Latin

Regarding the other attributes, simpler heuristics are used to determine the distance between different values. For example, in case of the happiness level attribute, it is well known that "really unhappy" is closer to "not so happy" than to "very happy".

5.2. Adaptation

5.2.1. Configuration (Parameters for Playlist)

Once the closest instances have been retrieved, we must generate a solution for the new input case. Since all the solution features are numerical (not categorical), we can compute the mean value for each feature between the retrieved solutions. Nonetheless, due to the dissimilarity between the input case and the retrieved cases, we have adapted the solution to satisfy the requirements of the user. To do so, we use prior music knowledge (3) to create some rules, which are summarized below:

- If the user is performing a task that requires more concentration than the mode concentration of the retrieved cases, the tempo of the song is reduced. That is,

$$+ \text{ Concentration} \rightarrow \text{- Tempo}$$

- If the user feels with more energy than the mode energy of the retrieved cases, the tempo of the song is increased. That is,

$$+ \text{ Energy} \rightarrow + \text{ Tempo}$$

- If the user wants to feel, after listening to the song, with more energy than the mode energy of the retrieved cases, the tempo of the song is increased. That is,

$$+ \text{ Future Energy} \rightarrow + \text{ Tempo}$$

- If the user feels happier than the mode happiness of the retrieved cases, the valence of the song is increased. According to spotify, the valence describes the musical positiveness conveyed by a track. Songs with high valence sound more positive (cheerful or euphoric), while songs with low valence sound more negative (depressed or angry). That is,

$$+ \text{ Happiness} \rightarrow + \text{ Valence}$$

- If the user wants to feel, after listening to the song, happier than the mode future happiness of the retrieved cases, the valence of the song is increased. That is,

$$+ \text{ Future Happiness} \rightarrow + \text{ Valence}$$

The increase or decrease in tempo or valence is computed as follows: we first compute the distance between the mode attribute in the retrieved cases and the input value. For example, in case of dealing with the happiness attribute, we may encounter this situation: the mode is "neutral" and the input case is "very happy", whose distance is 2 since we have the following values: "really unhappy", "not so happy", "neutral", "happy", "very happy". Afterwards, we increase/decrease the tempo/valence a 10% of the range multiplied by the distance. In our example, due to higher happiness, the increase of the valence would be $2 \times 10\% = 20\%$ of the valence's range. Note that, since the maximum distance for this attribute is 4, the maximum percentage of increase/decrease in the tempo is 40%.

In addition to the updated values of the solution, we also retrieve the minimum and maximum values for each attribute of the retrieved cases and the number of instances retrieved to improve the evaluation metric of the Case Based Reasoning as well as the playlist generation system.

5.2.2. Planning (Playlist Order)

For the playlist creation, the algorithm takes 3 arguments: the new case with the user profile, the solution obtained in the Adaptation [Configuration] step containing a minimum, mean and maximum value per each solution attribute and the length of the playlist (by default it is set to 5).

The first step to build the playlist is to select the Initial Point according to the values of the Adaptation [Configuration] solution. Therefore, what we propose is to perform a voting protocol of the case antecedents on each one of the solution attributes. Given some hand-made rules and the vector formed by [min, mean, max] for a given attribute of the solution, each antecedent selects a value in the range of [0-2] uniformly distributed. The set of rules is defined as follows:

- If the user requires more concentration, wants to feel happier or increase the energy, these three antecedents will vote for an initial point closer to the minimum for the attributes danceability, energy, loudness, valence and Tempo, and closer to the maximum for the attributes acousticness and instrumentalness. The reason behind this rule is to select a point within the admitted values from which the playlist can follow a search direction that is able to achieve the desired future states incrementally.
- If the user is currently feeling happy, the voted initial point will be closer to the minimum for the attributes acousticness and instrumentalness and closer to the maximum for the attributes danceability, energy, loudness, valence and tempo.
- If the user is currently feeling with energy, the voted initial point will be closer to the minimum for the attributes acousticness, instrumentalness and valence and closer to the maximum for the attributes danceability, energy, loudness and tempo.

Moreover, we have included some weights for each of the antecedents to influence more or less on the initial point selection depending on its importance. The most important features to consider are the future statuses (happiness and energy) with a weight of 0.3 each, followed by the concentration requirement with a weight of 0.2 and, finally, the current statuses (happiness and energy) with a weight of 0.1 each. As an example, the "concentration requirement" antecedent can take 3 possible values. In case, the user does not require concentration at all, this antecedent would vote to select the maximum value for the acousticness and instrumentalness attributes and the minimum value for the rest. On the contrary, if some concentration is required the votes would go to the mean values of every attribute and if the maximum concentration is required, the votes would be the inverse of the corresponding to no concentration required. The same example can be extrapolated to the other attributes with the particular case of the current statuses that have 5 possible values. For those cases, the vote will corresponding to a value in [0;0.5;1;1.5;2] with 0 being the minimum, 1 the mean and 2 the maximum. At the end, the algorithm computes the weighted average of the antecendents' votes for each solution attribute and the initial point can be extracted accordingly. For instance, if the voted value for the tempo attribute is 1.2, its initial value corresponds to the mean value plus a 20% of the distance between the mean and maximum values.

Once the initial point has been obtained, the algorithm selects a search direction to follow in the retrieval step to capture the most similar cases from the case library. The search direction is attribute-wise and it is initially set towards the opposite half of the solution range. That is, if the voted initial point was selected within the range mean-maximum, the direction would be towards lower values of this attribute. This heuristic makes sense because antecedents with higher weight (future states and concentration) had voted to start from the most undesired point [within the range of admitted values] and the search direction will lead the playlist towards the desired state.

The next step is to find the first song that will be added to the playlist. This step requires a normalization of all the numerical features of the songs dataset to avoid incorrect feature weights when computing the [euclidean] distances. The normalization technique selected is MinMaxScaling that normalizes numerical features to the range of 0-1 keeping the same distance ratios.

For the first song, the algorithm looks for the most similar instance in the songs library, although the search direction is not used yet. From this song onward, the algorithm selects the most similar song in an iterative process following the search directions for as long as possible. If there is no other song in the library that satisfies the search direction (the dataset does not contain a huge number of songs for each genre), then, a new search direction vector that minimizes the distance with the existing one (minimizes the number of directions that change) is extracted and the searching of songs continues with the new directions. The playlist is considered complete once the length of the playlist reaches the pre-specified playlist length or there is no other available song in the library. Finally, the console prints the playlist containing the ordered set of songs defined by the following attributes: ["Song", "Artist", "Genre", "Link_Spotify"].

5.3. Evaluation

The third step in the Case Base Reasoning methodology is the evaluation of the solution to qualify its performance. Three common methods exist for this purpose:

1. Perform the solution in the real world
2. Ask to a human expert
3. Run a simulation

The topic of this project prevents the simulation of the solution, which is more suited for other areas (e.g. planning city routes), there is no possible way of simulating how well the user is going to respond to our solution because of the uniqueness of each individual, however, other automatic methods exist (see section 5.3.1). The most straightforward evaluation procedure consists on asking the user whether the solution was satisfactory, but it's important to note that the output from the adaptation step is not directly feed into the user but it's firstly transformed into a playlist (which may affect the judgment of the user). If the software is run with the $EVALUATION\_AUTO$ flag set to $False$, after the playlist have been printed, the software will ask the user a yes or no question regarding whether the solution was satisfactory.

5.3.1. Automatic evaluation

For the scope of this project, another automatic approach aside from the simulation has been implemented. The purpose of this procedure is to evaluate the likely-hood of the solution parameters of acousticness, danceability, energy, instrumentalness, loudness, valence and tempo to have their returned values. The histograms provided by the Spotify API have been used, which show how the distribution of values for each variable is in different bins (figure 3).

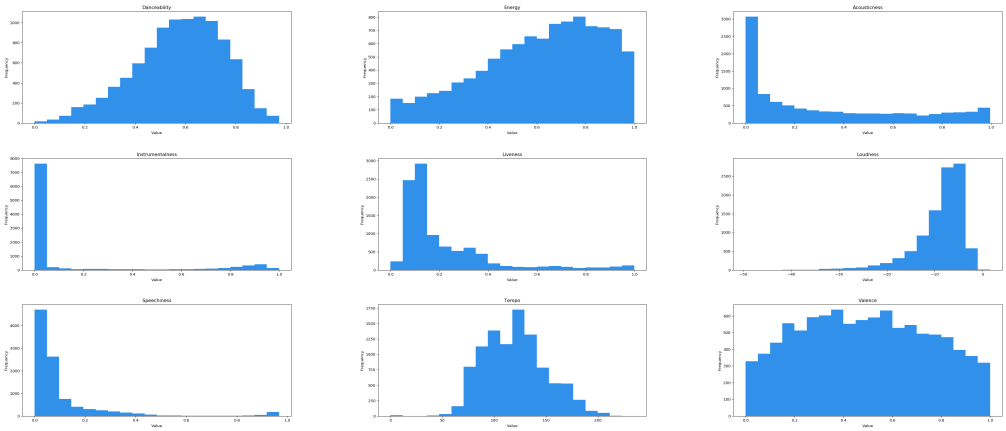

Figure 3: Histogram of the distribution of all the possible values for the parameters of a song.

Each histogram is normalized using the total number of songs and then, for each song, a likely-hood $p(song)$ can be calculated multiplying the likely-hood of each variable alone according to the histogram. In order to prevent underflow errors, the value used is $log_{10}(p(song))$. As an example, for the song "I Shot the Sheriff" from the Author ("Bob Marley"), $log_{10}(p(song))$ is calculated in the table 1. A song that has more likely parameter values is probably more appropriate to use in the generation of a playlist for the user.

Variable	Value	Bin	Log Likely-hood
danceability	0.444	9	-1.121
energy	0.739	14	-1.118
loudness	-8.554	13	-0.566
acousticness	0.515	10	-1.559
instrumentalness	0.000	0	-0.129
valence	0.808	16	-1.322
tempo	175.898	14	-1.274
$log_{10}(p(song))$			-7.088

Table 1: "I Shot the Sheriff" from Bob Marley

Moreover, songs from each different genre seem to be centered around a determined value of likely-hood as seen in figure 4 following a gaussian distribution, which allows the software to check if the value of $log_{10}(p(song))$ for the generated parameters in the previous step is close to the distributions of the genres the user has chosen. For this purpose, the software calculates for the log likely-hood of the solution the distance in relation to the center of the distribution of the songs of the genres selected using the number of standard deviations (the solution is marked with an x in figure 4).

A sigmoid function is then applied and the distance is transformed into a similarity value. The worst similarity value is discarded and the mean of the rest of the values is calculated as the "goodness" of the proposed solution. If the goodness is above a threshold which by default is 0.6, the case is considered to have a satisfactory solution.

5.4. Learning

In the learning step, if the solution was correct or satisfactory, the case and it's solution are stored again. In general two approaches exists: retain good solutions or retain bad solutions. Our implementation is based on retaining good solutions, because the retain of bad solutions would require different and additional data structures to analyze and maintain which do not compensate (in time, memory, and software complexity) for the performance gained if any. However **the software can retain bad solutions if asked to do so as if they where good solutions, this option is just implemented for testing purposes** used as a validation to check that the accuracy of the CBR system decreases while retaining bad solutions and should not be used for a real case scenario (also for testing purposes, the options to retain all the cases and not a single case are also available). **When a solution is retained the tree structure is modified to optimally incorporate it.**

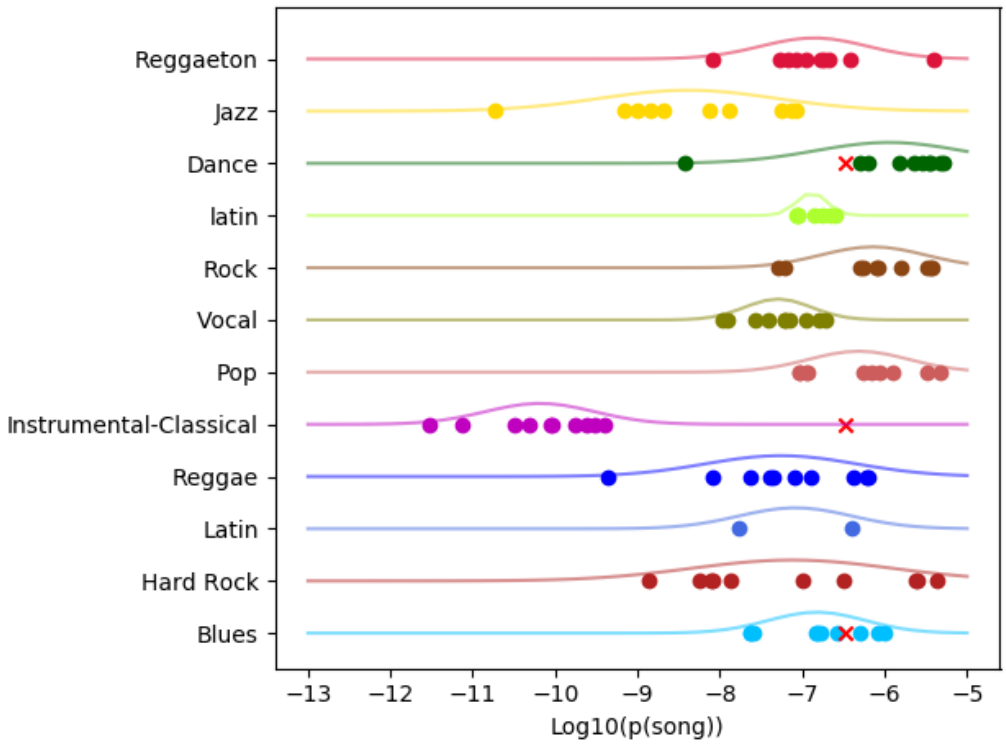

Figure 4: Log likely-hood values for different songs and genres

6. Testing and Evaluation

The training consists of a file of 130 instances while the testing file has 46 instances. Because the final output of the CBR system is not a single song but a playlist the method selected to measure the performance of the playlist generation with the data in the test set is as follows:

1. Create a CBR instance
2. For each instance in the test set
   (a) Run the CBR system for that input
   (b) The instance and the solution may be retained according to section 5.3.1
   (c) Compare the similarity of each song in the output playlist against the ground truth (unique) song
   (d) Get the mean of the similarity for the n most similar songs in the playlist (N_SIM parameter)
   (e) Store the value in a "results" array
3. The mean of the results, is defined as the mean similarity among all the instances between the output playlist and the ground truth

In order to make a good comparison, all the songs in the database where tested against each other with similarity; obtaining a mean similarity of 0.7155. Which is the expected value of mean similarity to obtain from the tests if the CBR outputs are completely random. Four tests where made: without retaining any case, retaining only good cases (expected to improve), retaining only bad cases (expected to get worse) and retaining all.

```
# PLAYLIST_LENGTH = 5
# N_SIM = 2
# RETAIN_THRESHOLD = 0.6
Tests performance: 0.8192 Avg Sim (retain: None, retained: 0)
Tests performance: 0.8211 Avg Sim (retain: good, retained: 30)
Tests performance: 0.8192 Avg Sim (retain: bad, retained: 9)
Tests performance: 0.8234 Avg Sim (retain: all, retained: 45)
```

From the results above it can be seen that with the configuration used, in all of the four cases, the CBR system is performing better from what is expected from a random playlist generator. Furthermore, unlike negative ones, retaining positive cases results in a better average similarity. However, retaining all instances helped to achieve an even higher value, not only that, but results are **very** similar among them and are very dependant on the values of the hyperparameters. The issue is that the system is already performing really good and adding new cases often only serves the purpose of changing the tree structure that in some cases by pure randomness of how the cases are distributed will generate a better or worse solution. A bigger dataset with thousands of songs, and hundred of users should be employed if the evaluation and retain steps of the CBR had to be tested in detail.

7. Conclusions and future work

With the appearance of new applications (such as Spotify and Apple Music) which allow the user to listen to millions of songs by a small amount of money, it has become a challenge to find a playlist you like. For this reason, in this project we present an efficient Case Based Reasoning application that is able to generate a song playlist suitable for the user. That is the reason why our dataset contains the wishes of the user at a particular moment in terms of happiness and energy. It is clear that in many cases, different feelings can emerge from listening to a variety set of music songs. Consequently, selecting songs in the correct order can help the user obtained the desired effect at the end of the playlist.

The main steps of a Case Based Reasoning are: retrieval, adaption, evaluation and learning. Retrieval is focused on finding the most similar songs in the Case Library. Nevertheless, due to the small amount of training instances, we perform a partial matching approach which allows to retrieve as many songs as possible that are considered to be similar as well. In the adaption, the Case Based Reasoning is designed to propose a new solution from the retrieved songs (the solution consists of the parameters of an hypothetical ideal song). Instead of generating a playlist, we reuse and adapt the solution to the specific user and input the previous result to a new module whose objective is to generate that playlist. Since playlists must be seen as a whole, that is, a random order of the songs would not produce the same effect that a proper order, we have designed this module to produce the expected feeling that the user is looking for. Afterwards, the evaluation step must determine if the new proposed song's parameters are suitable for the user, and if so, the new case is stored in the Case Library. In this case, instead of evaluating the playlist, we have decided to evaluate the proposed ideal song by the adapting phase, since the Case Library stores individual songs, not playlists.

The planning algorithm for sorting the playlist and the automated evaluation algorithm stands out by their ingenuity, trying to get advantage of the knowledge of the domain in order to fulfill the goals of the system. When testing the performance of our application, we have seen that it performs better than a random playlist generation. In addition, we have concluded by testing with individual users, that they had a positive experience when listening to the recommended songs. In future work, we would like to increase the size of the dataset to obtain a more accurate application as well as a larger testing dataset to ensure the correctness of the application. Moreover, including additional parameters such as speechiness or language can also strengthen our recommender system. In addition to the playlist generation branch of the algorithm, the configured solution could also be feed to a new system to create a brand new song from multiple music bases that best matches the requirements of the user. This new system would be presented as a new functionality of the overall application and it may be achieved by a Generative Adversarial Network.

In summary, we present a supervised machine learning algorithm that efficiently generates a playlist based on the preferences of similar users already registered in the Case Base Reasoning application. We believe and demonstrate that our new technique can be successfully with most of the users.

References

[1] Sànchez-Marrè, M. PART 2- CBR SYSTEM COMPONENTS. Material of the course "Supervised and ExperIential Learning" at the Master of Artificial Intelligence. Universitat Politècnica de Catalunya (2019)., < http://www.cs.upc.edu/ miquel/sel/ > [Online]. Seen on 29th of May, 2019.

[2] Sànchez-Marrè, M. PART 4- CBR DEVELOPMENT PROBLEMS. Material of the course "Supervised and ExperIential Learning" at the Master of Artificial Intelligence. Universitat Politècnica de Catalunya (2019)., < http://www.cs.upc.edu/ miquel/sel/ > [Online]. Seen on 29th of May, 2019.

[3] Fernández-Sotos, A., Fernández-Caballero, A., Latorre, J. M. (2016). Influence of Tempo and Rhythmic Unit in Musical Emotion Regulation. Frontiers in computational neuroscience, 10, 80. doi:10.3389

Appendix A. Annexes

A.1. Survey

This is the questionnaire used in the survey for collecting the cases...

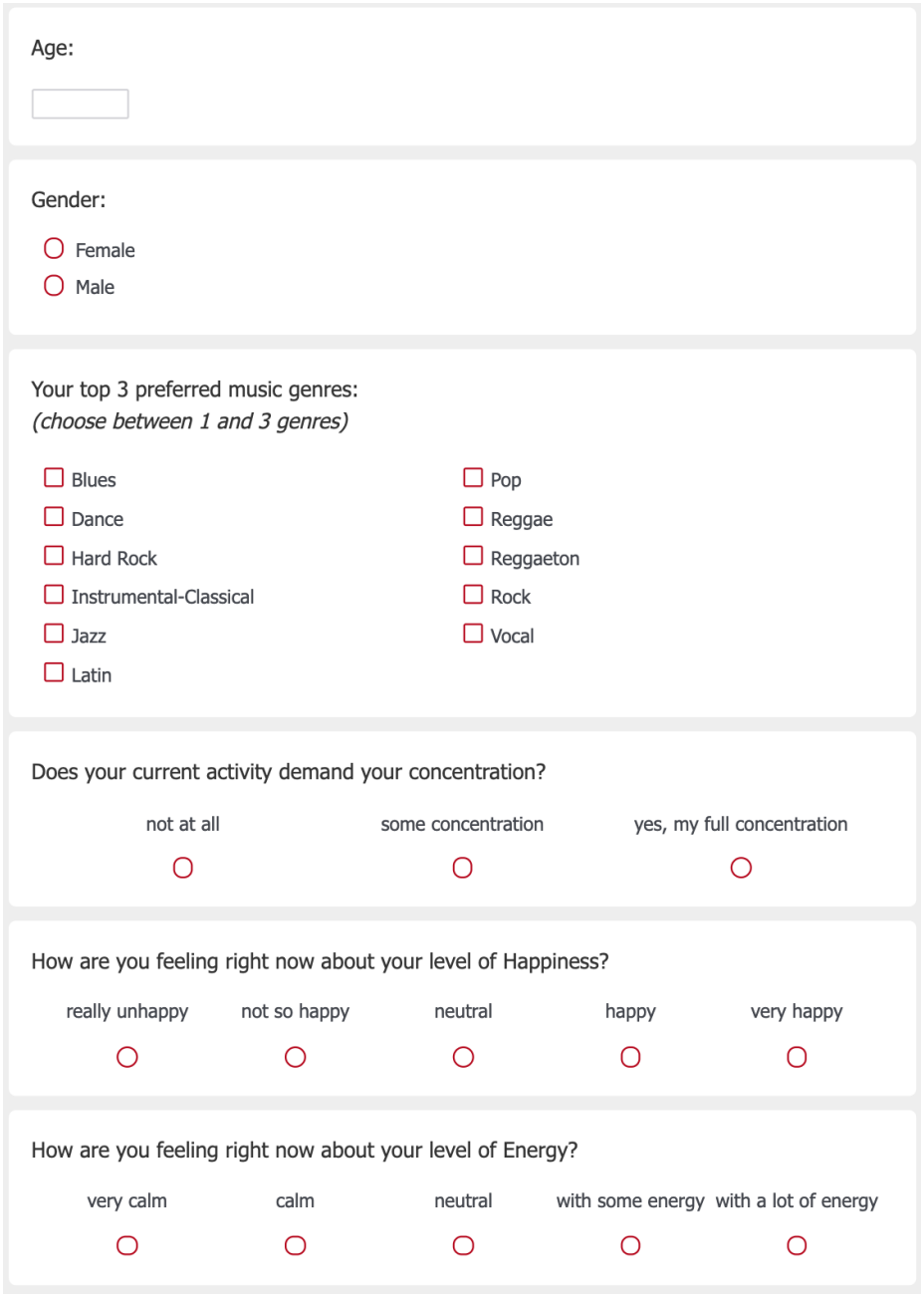

Figure 5: Survey Questionnaire Part 1

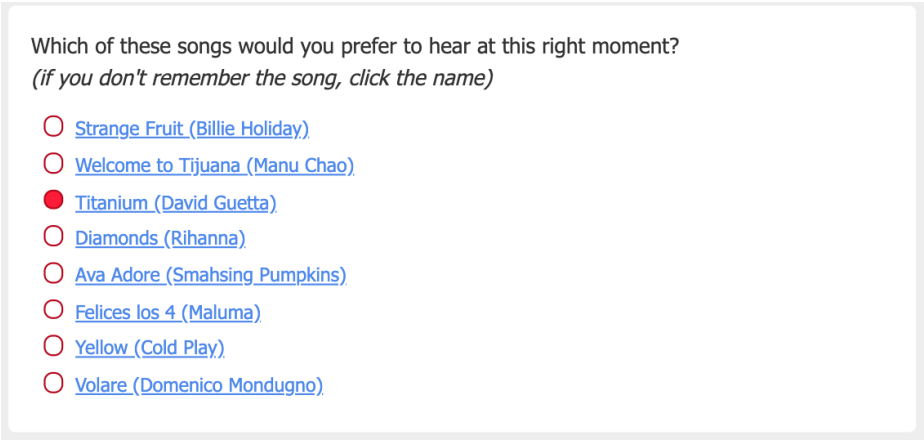

Figure 6: Survey Questionnaire Part 2

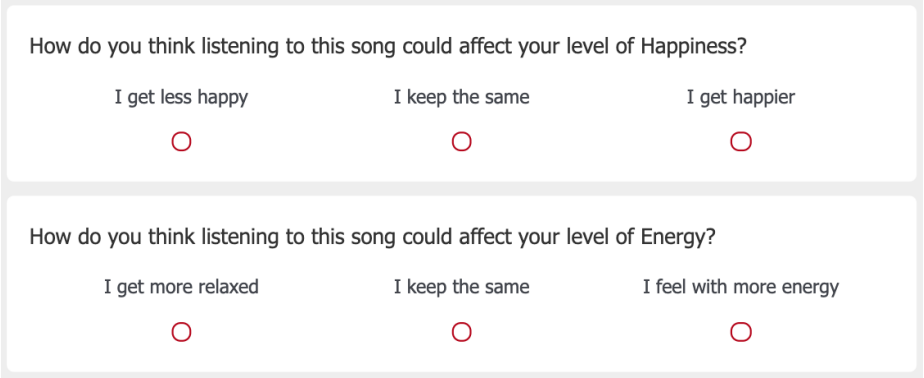

Figure 7: Survey Questionnaire Part 3

The instances collected with the survey can be observed at the file ***Song Preferences.xlsx*** located at the folder of this delivery.

Appendix B. User Manual

When running the application, the user is asked to answer the following questions described in Figure 8. Afterwards, the application returns an ordered playlist to the user, as presented in Figure 9. The number of songs in the playlist is specified by a hyperparameter in the application.

```
WELCOME TO THE PLAYLIST GENERATOR.
Please, answer to the following questions:
*Introduce you Age: 23
*Are you Male or Female?
   0 -Male
   1 -Female
   Introduce the number of your option: 0
*Choose your three favourite genres from the following list:
   0 -Blues
   1 -Dance
   2 -Hard Rock
   3 -Intrumental-Classical
   4 -Jazz
   5 -Latin
   6 -Pop
   7 -Reggae
   8 -Reggaeton
   9 -Rock
   10 -Vocal
   Introduce the numbers of your first option: 1
   Introduce the numbers of your second option: 6
   Introduce the numbers of your third option: 8
*Choose how much concentration do you need while listening to the playlist:
   0 -Not at all
   1 -Some concentration
   2 -Yes, my full concentration
   Introduce the number of your option: 1
*Choose your level of happiness:
   0 -Really unhappy
   1 -Not so happy
   2 -Neutral
   3 -Happy
   4 -Very happy
   Introduce the number of your option: 3
*Choose your level of energy:
   0 -Very calm
   1 -Calm
   2 -Neutral
   3 -With some energy
   4 -With a lot of energy
   Introduce the number of your option: 2
*Choose the level of happiness do you want to have after listening to the song:
   0 -Be less happy
   1 -Keep the same feeling
   2 -Be more happy
   Introduce the number of your option: 2
*Choose the level of energy do you want to have after listening to the song:
   0 -Be more relaxed
   1 -Keep the same feeling
   2 -Feel more energy
   Introduce the number of your option: 1
```

Figure 8: User menu of the application.

The generated playlist is:

	Song	Artist	Genre	Link_Spotify
0	Diamonds	Rihanna	Dance	https://open.spotify.com/track/7Kt59L2ZZGtOnIhvMwzG6f
1	La Bicicleta	Shakira & Carlos Vives	Reggaeton	https://open.spotify.com/track/0sXvAOmXgjR2QUqLK1MltU
2	Despacito (Remix)	Luis Fonsi & Daddy Yankee & Justin Bieber	Reggaeton	https://open.spotify.com/track/6rPO02ozF3bM7NnOV4h6s2
3	Happy	Pharrell Williams	Pop	https://open.spotify.com/track/5b88tNINg4Q4nrRbrCXUmg
4	El Perdon	Enrique Iglesias	Reggaeton	https://open.spotify.com/track/7qCAVkHWZkF44OzOUKf8Cr
5	Somebody That I Used To Know	Gotye & Kimbra	Pop	https://open.spotify.com/track/4wCmqSrbyCgxEXROQE6vtV
6	Gangnam Style	PSY	Dance	https://open.spotify.com/track/03UrZgTINDqvnUMbbIMhql
7	Sin Pijama	Becky G	Reggaeton	https://open.spotify.com/track/2ijef6ni2amuunRoKTlqww
8	Sexy And I Know It	LMFAO	Dance	https://open.spotify.com/track/5tGFmUFsBBSPvj5ZLQ686F
9	Nota de Amor	Carlos Vives & Wisin & Daddy Yankee	Reggaeton	https://open.spotify.com/track/4YW2Fr25fHwUt6qgvM1iE1

Figure 9: Playlist generated by the application.