Vrije Universiteit Amsterdam:

– Final Group Project –

Knowledge and Data

Authors:
Zoë Azra Blei,
Caspar Grevelhoerster,
Auke Hofman,
Lior Tabibi

Project Group 25

Date: 28-10-2021

The Goal of the Project

The goal of the application is to find new music through the use of filters that specify suggestions. Music platforms like Apple Music or Spotify already provide music suggestions for their users with exploring pages or suggested playlists; the "Discover weekly" and "Release radar" are two recommend-features that Spotify suggests new music with to its users. These suggestions are based on what users listen to, how often, and what they save to their library [1]. However, most of these platforms do not provide the option for users to filter suggestions to personal likings. This application makes it possible for the user to find recommendations based on a preferred decade, specific artists or artists in general, music writer, producer, song feature (e.g. 'danceability' or 'feelings'), genre or instrument. When a user inputs at least one of these preferences, the algorithm will suggest music that fits them.

The Stakeholders

As mentioned in the goal of the application most music platforms do not give the option to find suggestions based on a user's personal liking. Therefore, this application can be of use to people who, perhaps, find such platforms insufficient but do know what their preference is regarding music. One can find new music within the range of the preference filters, mentioned in the goal of the application, as well as explore a new genre or decade, for example. Additionally, it is also possible for users to go back in time and find a song or artist that was completely forgotten. Thus, the application is not for those who want to listen to music, but those who are purely interested in enriching their musical libraries with music they have never heard before as well as music they have not heard in years.

The Design and Visualizations

Once the user input its preferences the recommendation algorithm will process the user input and represent it in an attractive and informative way. The input can vary within the range of decade, artist, music writer, producer, song feature (e.g. 'danceability' or 'feelings'), genre or instrument(s) which is to be taken into consideration when aiming to output a suggestion.

RADAR CHART. One way of representing our data is with a radar chart (fig. 1). This radar chart takes 'song-feature' values that are assigned to every song in our database. The song features include: danceability, feelings, night/time, instrumentalness, valence, and energy. The values are scores between 0 and 1.

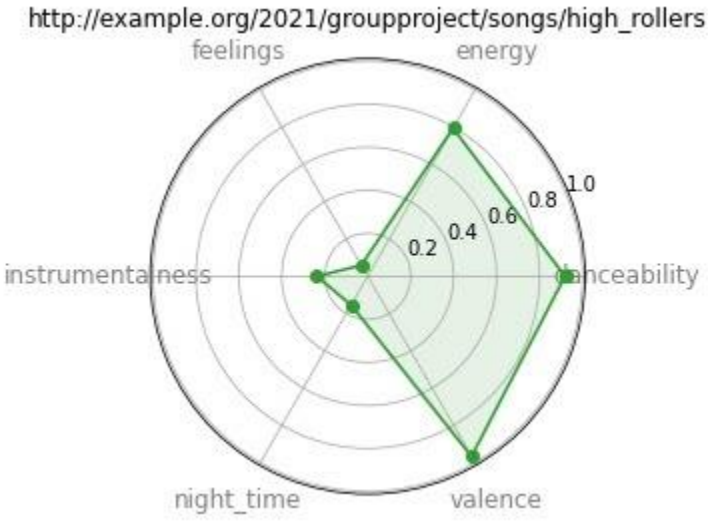


Figure 1: This is a radar chart of the song 'high rollers' that represents the song-feature values in ascending order.

TABLE. Another way of showing data is by using a table (fig. 2). In a table we can show relations between artists and recommend songs of specific artists, for example. Using a table to visualize the output of the recommendations is a clear and structured way to represent them, and easy for the user to understand and use the information. An example can be found in figure 2, where recommendations are represented based on a user input that used the genre 'country' music as a filter. The

table shows 10 recommendations of songs, with their corresponding artist, that are part of this genre.

	name	song
1	Alan Jackson	http://example.org/2021/groupproject/songs/leaning_on_the_everlasting_arms
2	Atlanta Rhythm Section	http://example.org/2021/groupproject/songs/quinella
3	Billie Jo Spears	http://example.org/2021/groupproject/songs/_hey_won_t_you_play_another_somebody_done_somebody_wrong_song
4	Bobby Bare	http://example.org/2021/groupproject/songs/lynchin_party
5	Bruce Cockburn	http://example.org/2021/groupproject/songs/no_footprints
6	Darius Rucker	http://example.org/2021/groupproject/songs/this_is_my_world
7	Eddy Arnold	http://example.org/2021/groupproject/songs/somebody_like_me
8	George Strait	http://example.org/2021/groupproject/songs/love_comes_from_the_other_side_of_town
9	John Anderson	http://example.org/2021/groupproject/songs/i_fell_in_the_water
10	Johnny Cash	http://example.org/2021/groupproject/songs/i_feel_better_all_over

Figure 2: This table represents the recommendations retrieved from the dataset (with a limit of 10 results) we created, based on the filter 'genre' as given input.

A second example including a table as a visualization tool is one where 'artist' is used as a general filter (fig. 3). This could be used when one has an interest in finding new artists, and the genre or instrumentals, for example, do not matter. In this case the recommendation system will retrieve artists and their associated artists.

	artistName	associatedArtistName
1	Cardi B	YG
2	Alan Jackson	Hank Williams Jr.
3	Ashford	Simpson
4	Big Sean	Chris Brown
5	Chris Brown	Ty Dolla $ign
6	Damizza Presents Shade Sheist	Nate Dogg
7	Nate Dogg	Kurupt
8	Nate Dogg	MC Ren
9	Darius Rucker	Luke Bryan
10	Dizzee Rascal	Calvin Harris

Figure 3: This table represents an association relation between artists, based on the general use of the 'artist' filter, with a limit of 10 results.

A more specific example with a table visualization and the use of the 'artist' filter is one that uses a preferred artist as user input. This results in a rather informative table than recommendation based, as it outputs all the information about the specific artist the user searched on (fig. 4).

	artistName	createdSong	associatedArtistName	BirthPlace
1	Donna Summer	http://dbpedia.org/resource/Heaven_Knows_(Donna_Summer_song)	Barbra Streisand	http://dbpedia.org/resource/Boston
2	Donna Summer	http://dbpedia.org/resource/Any_Way_at_All	Harold Faltermeyer	None
3	Donna Summer	http://dbpedia.org/resource/Love's_About_to_Change_My_Heart	None	None
4	Donna Summer	http://dbpedia.org/resource/Love's_Unkind	None	None
5	Donna Summer	http://dbpedia.org/resource/Love_Has_a_Mind_of_Its_Own	None	None

Figure 4: This table represents all information about a single artist including created songs, associated artists and their birthplace. In this example a limit of 5 results was used.

The table visualization tools come from [2].

The Reuse of Ontologies

We have decided to use DBpedia, FOAF (friend of a friend), and OWL. All projects together cover a wide range of data but differ from each other in such a way that we have found useful. For example, DBpedia is a structured content form of information based on and created by Wikipedia, and has an enormous amount of useful data about musical artists, albums, music genres, etc. FOAF on the other hand is a project describing people, their actions and activities, and their relations to other people and to objects. Moreover, we have found the usage of OWL, which is a family of knowledge representation languages and has several different predicates that can be useful when discussing artists and their similarities and differences (owl:sameAs, owl:differentFrom). One major ontology that is created in OWL is SKOS [3], its thesaurus embeds predicates that we have found useful in describing

music genres (skos:broader, skos:related) and can be easily combined with OWL ontologies [4]. Overall, we believe that our domain can be fully covered and described by these ontologies.

The External Sources

Generally, there are many datasets and lots of information available on the internet regarding music. We are thinking of using a combination of different sources in order to fill our database and to get meaningful recommendation results. The first dataset comes from a data conference of 2019 [5].

It includes different knowledge bases in .csv data file formats that cover network connections between different Spotify artists as well as their genre. This might be useful for recommendations. We could make all artists and genres instances and relate them using skos:broader or similar structure. When searching for a music artist, our program could traverse the graph for artists that are related or are related to a similar genre. The downside of this dataset is the limited time-scale of 2017 to 2019. The scientists only collected data from Spotify charts, not the entire database which means that older artists that were not popular at that time will likely not be contained. To control for this issue, we searched for another relevant dataset containing all top-100 billboard charts from the years 1970 to 2017 [6].

This dataset, however, is problematic since it only contains the most popular over the years. But we would like to be able to make recommendations for anyone - no matter how specific or detailed it may be. The first publication from the music data conference [5] also contains a dataset mapping a variety of different artists to their respective genre no matter their popularity. This file was scraped in order to provide recommendations in less popular genres.

Another interesting dataset was made from around 83000 songs and their lyrics [7]. Each song has different categorical values of e.g. danceability, loudness, energy, topic, instrumentalness, etc. assigned that were computed using those lyrics.

This might help us in finding similar songs. Further, the songs are all mapped to their respective genre which can also help finding other songs that might sound alike.

For the SPARQL endpoint, we will use DBPedia since that website elaborates on music artists and maps artists to their respective genre.

The Domain and Scope of the Ontology

The domain of our ontology is music. More specifically, we intend to give recommendations of music that are determined by multiple factors: a user could input the year they want the song to be from, the genre, and different types of characteristics, such as danceability, energy, feelings, instrumentalness, night time, and valence. These properties are expressed in a percentage and determine the recommendation of the song. Additionally, we output the same artists of a genre so that people can discover new artists within a genre and also associated artists.

Therefore, the ontology meant to give recommendations to all users that like to listen to music. For example, a user can be in the mood for some house music that was made in the year 2010, yet does not know any of these songs themselves to look up. The user could then access our application, fill in the year, genre and characteristics of the song they want to listen to and we will output the top 10 most relatable songs for them.

Accordingly, the scope of our domain music goes as far as all genres with their artists and songs go. Ideally, the scope could therefore be all the music that exists in the world. As it is not the idea to create an extremely extensive database in which we would lose the overview, we have limited the ontology to more or less 500 artists, so this is in fact the scope we have opted for.

The Methodology for Constructing the Ontology

To create the ontology, we utilized a middle-out approach as has been discussed in the lecture. Therefore, we combined general statements and relations that we identified as trivial for our ontology and consequently looked at the data that could be found on that in existing ontologies. So, artists should make songs and songs are part of a genre, for example. Some general classes were created with this in mind, afterwards the ontology was slowly populated in multiple steps as we will demonstrate in the Jupyter Notebook. Continuing, the graph was filled with non-RDF sources in multiple steps. The import from each database was limited to 100 .csv lines only. This was done to prove the concept works without stumbling upon issues as query-performance. Existing ontologies such as Zenodo and Data.world were useful to be integrated in our ontology, so we reused those so we would not have to insert all the factors like songs, artists and genres manually.

After adding key concepts to the ontology, we proceeded to insert more vague terms like popularity, danceability properties, etc. Thus, we first inserted the trivial items into the ontology and in 6 documented steps in the Jupyter Notebook we constructed the whole ontology from concrete to more vague items. These vague terms have been evaluated multiple times as we were not sure how it would serve our means. Finally, we found that danceability for example is useful to show a recommendation, yet a sub-genre for example only complicated our project.

We expressed these concepts in a combination of languages, for example we used SKOS for simpler mappings like the genres using SKOS:Broader and SKOS:Narrower, RDF Schema for the main items as for example the domain, range and subclass relations, and OWL for the more complex relationships, think of inverse relationships or irreflexive properties (an artist cannot collaborate with themselves).

The Conceptualization of the Domain

The 15 classes we created are:

RecordLabel. This class contains record labels that artists in the database are represented by.

Award. this class contains awards that music artists have won.

Place. this class contains cities that are the birthplaces of the artists in the database.

Person. this class contains people.

Artist. This is a subclass of Person and contains people that are artists, and in our ontology specifically are musical artists: singers, producers and songwriters.

Producer. This is a subclass of Artist and contains musical artists, singers, that are record producers as well.

Songwriter. This is a subclass of Artist and contains musical artists, singers, that write music as well.

BandMember. This is a subclass of Artist and contains members from bands in our database.

publishedMusic. This class includes music that is published on musical streaming platforms like Spotify or Apple Music.

Song. This class is a subclass of publishedMusic. This class includes songs that are published on musical streaming platforms. Additionally, they can either be part of an album or be a single.

Album. This is a subclass of publishedMusic and includes musical albums that are published on musical streaming platforms.

songMood. This class represents musical features songs can have. Examples of such features are the danceability or energy of a song, represented as a score between 0 and 1.

Genre. This class represents musical genres. Within this genre we use skos:broader and skos:narrower to reach the main genres and subgenres.

Decade. This class includes decades

Band. This class consists of music bands.

The object properties we created are:

wasBornIn. this property maps an artist to the city it was born in. It has the domain Artist and range Place.

associatedArtist. this property maps an artist A to another artist B that is associated with artis A. It has the domain and range Artist.

createdAlbum. this property maps an artist to albums that they have created. It has the domain Artist and the range Album.

albumCreatedByArtist. this is the inverse of createdAlbum and maps an album to the artist that created it. It has the domain Album and the range Artist. This property is also the inverse of createdAlbum.

createdSong. this property maps an artist to songs that they have created. It has the domain Artist and the range Song.

songCreatedByArtist. this is the inverse of createdSong and maps a song to the artist that created it. This is the inverse of createdSong. It has the domain Song and the range Artist.

collaboratedWith. this is a sub-property of associatedArtist and maps an artist A to another artist B that artist A made a song with. It has the range and domain Artist.

wonAward. this property maps an artist to an award that he or she has won. It has the domain Artist and the range Award.

isRepresentedBy. this property maps an artist to a record label that it is represented by. It has the range Record Label and the domain Artist.

Represents. this is the inverse of isRepresentedBy and maps a record label to the artist it represents. It has the domain Record Label and range Artist.

The datatype properties we created are:

isActiveToday: this data property maps an artist to a Boolean value. Artists that have a start year and end year, which represents the year they started and ended their careers are artists that are not active anymore. Those that only have a start year are active and are mapped to a True value. This property has the domain Artist and the range is a Boolean value.

makesMusicOfGenre: this maps an Artist to a Genre. It has the domain Artist and range Genre.

has_mood: this maps a song to a Boolean value. It has the domain song.

has_danceability_property: this maps a song to a Boolean value. It has the domain song.

has_energy_property: this maps a song to a Boolean value. It has the domain song.

has_insturmentalness_property: this maps a song to a Boolean value. It has the domain song.

has_feelings_property: this maps a song to a Boolean value. It has the domain song.

has_valence_property: this maps a song to a Boolean value. It has the domain song.

has_night_time_property: this maps a song to a Boolean value. It has the domain song.

publishedInYear: this maps a song to an integer, year. It has the domain song.

usesInstrument: this property maps an artist to an instrument that it uses or plays. It has the domain Artist.

wasActiveIn: this maps and Artist to an integer, Year. It has the domain artist.

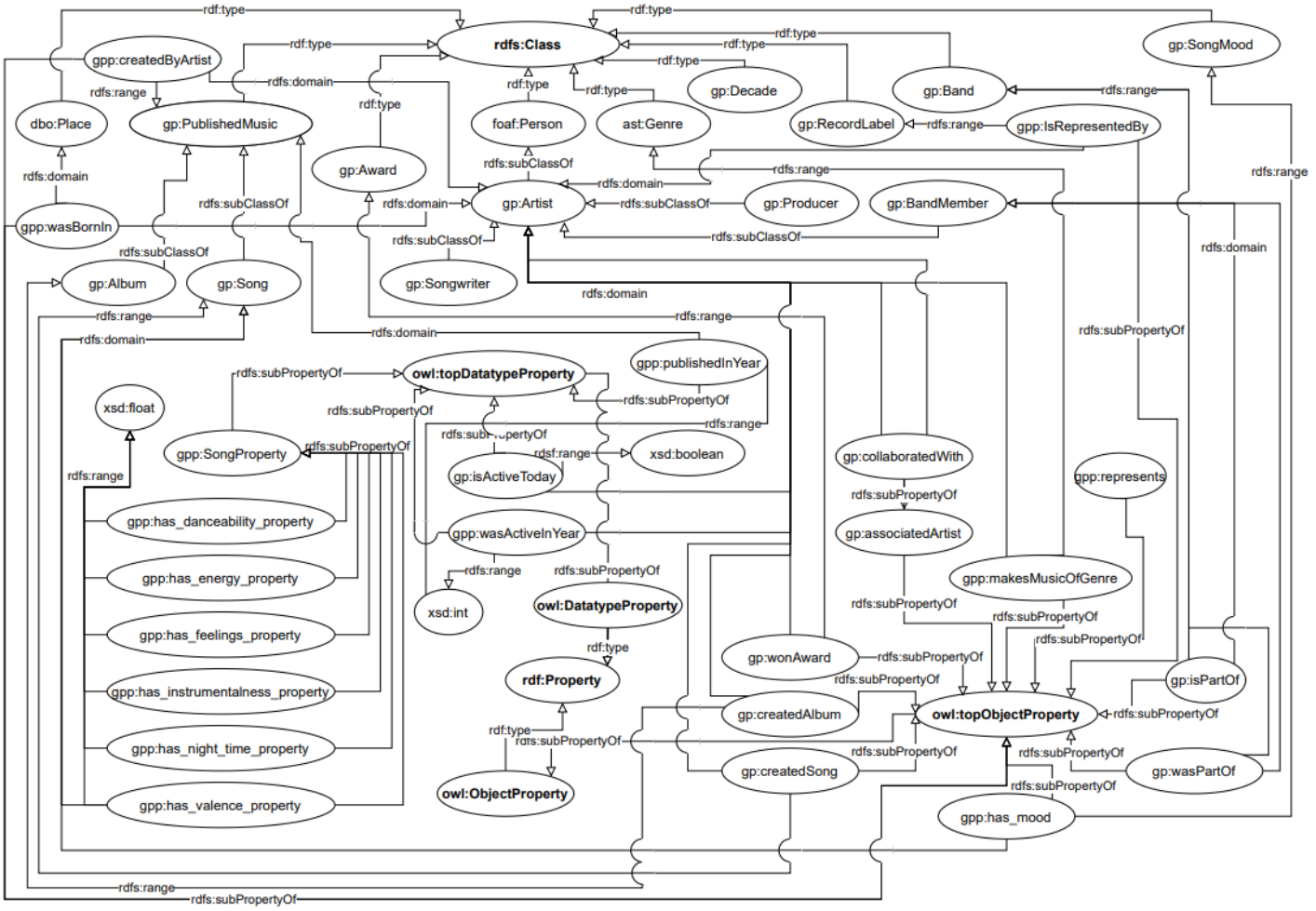


Figure 5: Graph representing our ontology structure.

Ontology description

The ontology includes the classes and properties described in the previous questions. These classes were made with the aim to eventually represent our gathered data in a way that is desired by the user. 3 class restrictions we made are:

For class Record Label. A record label should at least represent some Artist.

For class Producer. A producer should at least have created a song.

For class Artist. An artist is disjoint with class Genre.

The ontologies we reused are:

DBPedia. was used to retrieve properties from that were missing in our ontology, like dbo:associatedMusicalArtist and dbo:birthPlace.

FOAF. Is an ontology that describes people, relations among people and the interactions between them. We retrieved foaf:Person to make a class in our ontology that consists of people.

SKOS. Is a general ontology that organizes knowledge systems in a thesaurus structure. We used skos:broader and skos:narrower in combination with the class Genre. The genre gp:australian_pop for example has the skos:broader genre gp:pop. The genre gp:hip_hop has the skos:narrower genre gp:alternative_hip_hop.

Integration of Datasets

Our database consisted of Artists and Songs. We used dbpedia to retrieve missing information, like the predicate dbo:associatedMusicalArtist. This is a predicate that exists on DBpedia and we retrieved the objects that this predicate mapped from there by first retrieving all the artists in our database, and subsequently finding the same exact artists on DBpedia. After finding the equivalent ones on DBpedia,

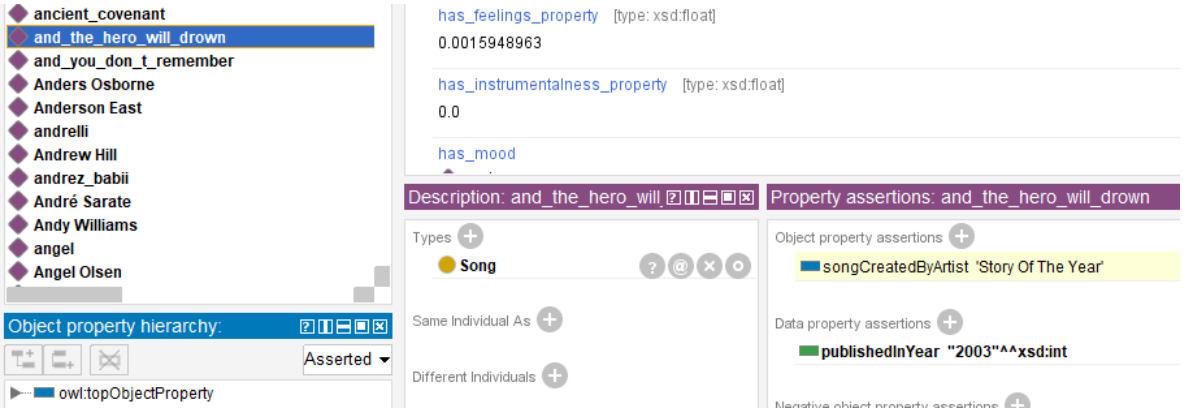
we retrieved the objects connected to the dbo:associatedMusicalArtist predicate, and added them to our ontology with an insert or construct query.

Another example of data integration is after first retrieving the artists from our dataset and finding the matching one in DBpedia, we retrieved the predicates dbo:activeYearsStarYear and dbo:activeYearsEndYear and checked for all the matched artists if they have the predicate dbo:activeYearsEndYear. If this was the case we created our own predicate gp:isActiveToday and assigned that value to be "false" for all of them. For the artists that only had the predicate dbo:activeYearsStarYear we assigned the value of the gp:isActiveToday predicate to be "true." These integrations made it possible to map artists to a value for the predicate gp:isActiveToday, so that we can show users information on which artists are currently active or not.

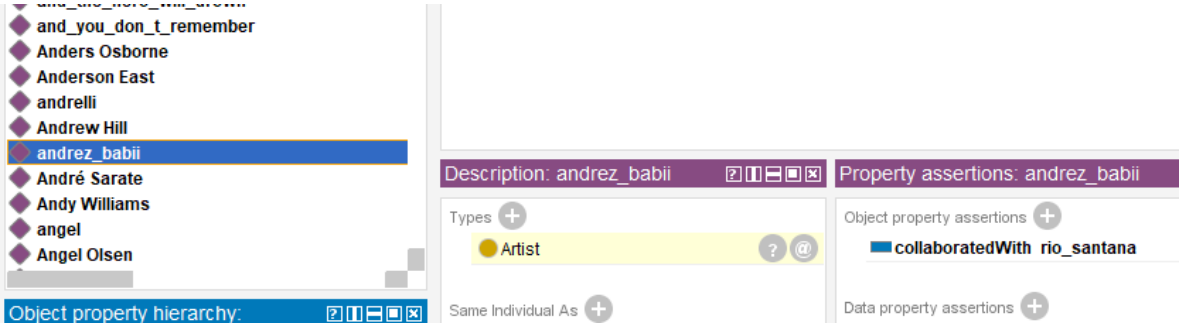
Use of our Ontology: Inferences

In our music application, users receive recommendations for songs according to genre, decade/year and feature preferences. Before implementing the infrastructure that allows users to filter songs to their liking, we are required to form a net of songs, albums, artists that are all interrelated.

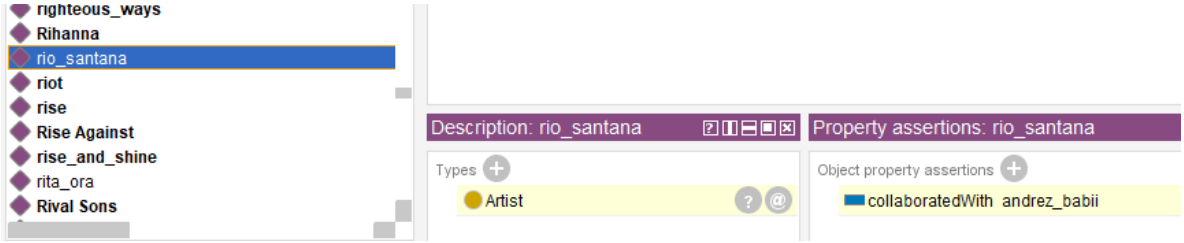
In example number 1, a song "and the hero will drown" is being inferred to the artist "Story of the year" on the basis of the "inverse" characteristic. This inference is emerging out of the prior condition that the artist "Story of the year" created the song "and the hero will drown".



In the second example, an instance of a person is being inferred to class “Artist”. This inference is occurring on the basis that the artist “andrex_babii” is collaborating with another person, in this case, Rio Santana. “collaboratedWith'' is under the domain of “Artists”, therefore if an instance is assigned to this predicate, the instance is immediately assigned to be of class “Artist”.



Lastly, as a result of example 2, example number 3 emerges. When Andrex Babii is assigned to be collaborating with Rio Santana, two inferences are formed:
Rio Santana is now collaborating with Andrex Babii.
Rio Santana is inferred to be an Artist, due to the fact that if a person is collaborating with someone else, he is immediately considered as an artist.



Use of our Ontology: SPARQL Queries

In order to create our ontology and graph, we have formed multiple insert sparql queries for each one of our classes, for example:

A query that reads all artists from our graph. For each artist, we query DBPedia and retrieve the associated awards in dbpedia. See section 2.11.

A query that reads all artists from our graph. For each artist, we query DBpedia and retrieve the country of birth of the artist. Although this information is not needed in order to create inferences, it is necessary in order to create a complete profile of each one of the artists. See notebook section 2.3.

A query that reads all artists from our graph. For each artist, we query DBpedia and retrieve all artists that are considered as "associated artists" according to DBpedia. This information is valuable in order to make inferences between similar artists, and will be taken in account alongside songs genres and features. See section 2.2.

A query that reads all artists from our graph. For each artist, we query DBpedia and retrieve whether or not the predicate "dbo:activeYearsEndYear" exists. In case that it does, the predicate "gp:isActiveToday" is being created and assigned the boolean value "False". This allows us to map all the artists that are active. See section 2.4 in the notebook.

A query that reads all artists from our graph. For each artist, we query DBpedia and retrieve whether or not the predicate "dbo:activeYearsEndYear" exists. In case that it does not, the predicate "gp:isActiveToday" is being created and assigned the boolean value "True". This allows us to map all the artists that are active. See section 2.5 in the notebook.

Production of Inferences using SPARQL

A query to retrieve all characteristics of the songs. In this sense, we can create visualizations and compare songs. We opted for these queries because we intend to show meaningful output with this query. This is related to the main objective of our recommendation system. Namely, this query enables us to output all characteristics one looks for in the songs they ask for as input. The screenshots below demonstrate this. Songs are inserted as input, as well as the properties that different kinds of

songs have. The output, consequently, shows the strength of the relatability to those characteristics. This strength is on a scale from 0 to 1. Thus, if someone wants a strong relatability for the danceability_property, we are able to filter this in a descending order, so the most danceable songs will be given as output. The user retrieves in this way the songs they want to dance on with their friends.

```
%%sparql http://Aukes-MacBook-Pro.local:7200/repositories/groupprojectKD -s songs
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX dcb: <http://dbpedia.org/resource/Category:>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX gpp: <http://example.org/2021/groupproject/predicates/>
PREFIX ast: <http://example.org/2021/groupproject/artists/>
PREFIX gp: <http://example.org/2021/groupproject/>
PREFIX gnr: <http://example.org/2021/groupproject/genres/>
PREFIX gns: <http://example.org/2021/groupproject/song/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT * WHERE {
    ?song rdf:type gp:Song ;
    gpp:has_danceability_property ?danceability_property ;
    gpp:has_energy_property ?energy_property ;
    gpp:has_feelings_property ?feelings_property ;
    gpp:has_instrumentalness_property ?instrumentalness_property ;
    gpp:has_night_time_property ?night_time_property ;
    gpp:has_valence_property ?valence_property .
} LIMIT 10
```

song	danceability_property	energy_property	feelings_property	instrumentalness_property	night_time_property	valence_property
ject/song/road...	0.214773	0.684675	0.112851	0.653846	0.002770	0.181781
ject/song/vidage	0.383732	0.722714	0.140118	0.798583	0.001144	0.289984
ject/song/a_ca...	0.625257	0.462446	0.020408	0.000000	0.001012	0.784625
ject/song/cand...	0.606845	0.876873	0.001170	0.006923	0.001170	0.869126
ject/song/cher...	0.588433	0.606594	0.001196	0.000006	0.001196	0.844394
oject/song/like...	0.767140	0.459443	0.001074	0.003472	0.001074	0.968054
ject/song/thes...	0.594931	0.833829	0.041826	0.000206	0.001224	0.792869
ject/song/trou...	0.865699	0.385366	0.051469	0.000000	0.001032	0.484749
ject/song/you_...	0.662082	0.784778	0.032739	0.000001	0.001096	0.479596
oject/song/deli...	0.887361	0.744737	0.001350	0.000000	0.001350	0.788747

A query to retrieve all people that has the predicate “collaboratingWith”, that is under the domain of artist. Once a person has this predicate, the person is being inferred to be of type “artist”. This can assist us in building the ontology and eventually creating a person that has other artists the he collaborated with, has songs and albums the he wrote, has instrument that he plays and so on.

```
14 PREFIX gp: <http://example.org/2021/groupproject/>
15 PREFIX ast: <http://example.org/2021/groupproject/artists/>
16 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
17 PREFIX dbp: <http://dbpedia.org/property/>
18 PREFIX dbr: <http://dbpedia.org/resource/>
19 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
20 PREFIX owl: <http://www.w3.org/2002/07/owl#>
21
22 select ?person
23 where {
24     ?person gp:collaboratedWith ?artist_two.
25     ?person a ast:Artist.
26 }
27
```

Run

keyboard shortcuts

	person
1	:artists/cardi_b
2	:artists/cardi_b
3	:artists/cardi_b
4	:artists/cardi_b
5	:artists/cardi_b
6	:artists/cardi_b
7	:artists/cardi_b
8	:artists/kanye_west
9	dbr:Kanye_West
10	:artists/ty_dolla_ign
11	:artists/ty_dolla_ign
12	:artists/ashford
13	:artists/drake
14	:artists/drake

Bonus Section: OWL Wizard

Before implementing oud ontology, we had multiple sessions of brainstorming and sketching, aimed to perfect the end result, in which we will generate inferences in our graph. One of the things we invested time in doing was assigning two default predicates to each property, regardless of its function. The two predicates were "rdfs:range" and "rdfs:domain", which allowed us to expand the oncology even further.

2.8 Mapping artists from local graph to their associated songs in dbpedia using dbo:artist

the mapping happens via the gp:createdSong in the local and dbo:Song in dbPedia

```
construct {
    gp:createdSong a owl:ObjectProperty.
    gp:createdSong rdfs:domain ast:Artist.
    gp:createdSong rdfs:range gp:Song.
    ?artist gp:createdSong ?song.
}
where {
    ?artist rdfs:label ?artistLabel;
                a ast:Artist.
    ?artist owl:sameAs ?dbArtist.
    SERVICE <https://dbpedia.org/sparql/> {
    ?song dbo:artist ?dbArtist.
            ?song rdf:type dbo:Song.
    }
}
```

On top of these two predicates, we have of course assigned each property to be of type “owl:property”.

A second concept we have implemented is assigning language tags to literals in an efficient way. When forming the initial graph with python (section 1.x) we have iterated through all of our artists, and for rach of them, assigning the type “ast.Artists”, assigning the artist label company (with the language tag) and assigning its genre. This action is robust and effected all of the artists in the graph.

```
for i in range(len(artists_names)):
    g.add((URIRef(ast+artists_names[i]), RDF.type, ast.Artist))
    g.add((URIRef(ast+artists_names[i]), RDFS.label, Literal(artists_labels[i], lang='en')))
    g.add((URIRef(ast+artists_names[i]), gpp.makesMusicOfGenre, URIRef(gnr+genre)))
```

Bibliography

[1] https://digital.hbs.edu/platform-rctom/submission/spotify-can-machine-learning-drive- content-generation/

[2] https://pandas.pydata.org/pandas-docs/stable/user_guide/style.html

[3] https://www.dataversity.net/introduction-to-skos/

[4] https://www.w3.org/2006/07/SWD/SKOS/skos-and-owl/master.html

[5] https://zenodo.org/record/4778563

[6] https://data.world/typhon/billboard-hot-100-songs-from-1970-2017

[7] https://doi.org/10.17632/3t9vbwxgr5.2