Music Recommender Bot

Introduction

In this report, we will explain our Music-Recommender Bot, which can recognize what the user is asking to, extract keyphrases from sentences, and even store them in order to answer the user with previous information from the dialog.

The dataset we use and additional labels

The dataset we use is **music-genres-dataset**, uploaded by **trebi**[1]. The tags we use from each element of this dataset are:

- genre: It is the genre of a song
- artist: It is the artist of a song
- track: It is the name of the song
- position: It is the score of the song

We added two more tags for each element, that are randomly set:

- mood: Indicates the mood of the track/artist
- date: Indicates the date of the track/artist

We also created different arrays with all the labels, that will be useful to get keywords in a sentence:

- genre_labels: Contains all the genres
- mood_labels: Contains all the moods
- date_labels: Contains all the years
- new_date_keywords: Contains all adjectives to refer to a newer date
- old_date_keywords: Contains all adjectives to refer to an older date
- artist_labels: Contains all the artists

Transformer for embedding:

We use the transformer “all-MiniLM-L6-v2”, that is the transformer we used in the previous lab and it was the most precise when it comes to the cosine similarity calculus between sentences.

Questions dictionary

In order to make our bot to “understand” the user questions, we hardcoded some sentences in a dictionary:

- The **keys** of the dictionary are possible questions that can do the user

- The **values** of the dictionary are the tags that indicates the type of questions

The **tags** of the questions are:

- Top-artists in genre question: The user is asking for the top-artists given a genre.
- Top-tracks in genre question: The user is asking for the top-tracks given a genre.
- Song from mood question: The user is asking for a song/track given a mood.
- Artist from mood question: The user is asking for an artist given a mood.
- Song from date question: The user is asking for a song/track given a date.
- Artist from date question: The user is asking for an artist given a date.
- Similar artist question: The user is asking for a similar artist given another artist.
- Not valid: The user question can't be understood by the agent.

```
questions = {

    #'Top-artists in genre question'
    'Can you tell me which bands are the most popular in _?':'Top-artists in genre question',
    'Which are the best bands in _?':'Top-artists in genre question',
    'Do you know any _ popular bands?':'Top-artists in genre question',
    'Do you know any _ famous bands?':'Top-artists in genre question',
    'Which bands are the best in _?':'Top-artists in genre question',
```

(We have many more questions than the ones shown, to have solid cos-similarity scores.)

Answers dictionary

We also have a **dictionary** with all the answers that the agent can do:

- The **keys** of the dictionary are the tags that indicates the type of questions
- The **values** of the dictionary are the answers of the agent to the user

The answers has different some special characters that will be substituted when doing the answer generation:

- * : It corresponds to the **main keyword** of the question. For example, in 'Top-artists in genre question', the main keyword is a genre, and in 'Song from date question' the main keyword is a date.
- _ : It corresponds to the **content** of the answer. For example, in 'Top-artists in genre question', the content is a list of the top artists in a genre given by the user, and in 'Song from date question' the content is the name of the song that the agent recommends.
- + : It corresponds to **additional information** that the bots give to the user **related to the content**. For example, in 'Song from a mood question', it is the artist of the track that the agent recommends.

```
answers = {
    'Top-artists in genre question':['The top-artists in * are:\n_'],
    'Top-tracks in genre question':['The top-tracks in * are:\n_'],
    'Song from mood question':['When I feel *, I like to listen to the track _, from +.'],
    'Artist from mood question':['When I feel *, I like to listen to the artist _.'],
    'Song from date question':['You should listen to _, a track from the artist +, from *.'],
    'Artist from date question':['You should listen to _, an artist from *.'],
    'Similar artist question':['An artist similar to * could be _.'],
    'Not valid':['I don' + "'" + 't understand you. Could you ask me again please?']
}
```

(The answers are in a list because we wanted to add more default answers, but we finally didn't implement this.)

Dialog history

Every time the user makes a question, the agent extracts the keywords from the question (such as a genre, a mood, a date or an artist), and all the keywords extracted are stored in the **Dialog history**.

The dialog history consists of an array of 3 positions, where position 1, 2 and 3 are the previous genre, mood and date respectively.

In case the user makes a question in which there are no keywords, the agent will take the keywords stored in the Dialog history.

```
def get_and_update_genre(sent):

  result = get_genre_from_sentence(sent)

  if(len(result)>0):
    update_dialog_history(PREVIOUS_GENRE, result)
  else:
    result = get_dialog_history(PREVIOUS_GENRE)

  return result
```

(In this method, first we obtain the genre from the current sentence. In case there is a genre in the sentence, it updates the dialog history with that genre. In case there is no genre, it will get the one stored in the dialog history.)

The method of the previous image is applied also in mood and date.

Each time we run the chat_bot, the Dialog history values are reset.

Tag prediction of a sentence

In the last report, we had an agent that only made use of **DialogTag** to understand the user's input.

In this version of the agent, we still make use of **DialogTag** to know the nature of the utterance given by the user. Once we know it is not a greeting nor a closing statement such as goodbye, we call the method **get_type_of_sentence(input)** in order to predict the tag of the given sentence.

The method does the following:

First it gets a filtered dictionary of questions from the original questions dictionary, and then, from these questions, we return the tag of the question with the max cosine similarity.

The mentioned filter is necessary for better scores and it is the function **get_auxiliary_dictionary(input)**: Returns a filtered dictionary of questions that depend on the **keywords** found in the input sentence or the previous keywords (from the **dialog history**).

This filter has different layers:

The first group of layers are four conditions that check if there is an artist, a genre, a mood or a date in the input sentence of the user, respectively.

- In case there is an artist, it will return the questions that are tagged as ‘Similar artist question’.
- In case there is a genre, it will return the questions that have ‘genre’ in their tag.
- In case there is a mood, it will return the questions that have ‘mood’ in their tag.
- In case there is a date, it will return the questions that have ‘date’ in their tag.

The second group of layers are three conditions that check if there is a genre, a mood or a date in the dialog history, respectively.

- In case there is a genre, it will return the questions that are tagged as ‘Similar artist question.
- In case there is a mood, it will return the questions that have ‘mood’ in their tag.
- In case there is a date, it will return the questions that have ‘date’ in their tag.

In case none of these conditions are met, then it returns all the questions dictionary.

```
def get_auxiliar_dictionary(input):

  #First we look at the current sentence keywords

  if(len(get_artist_from_sentence(input)) > 0):
    return questions_artist_related;

  if(len(get_genre_from_sentence(input)) > 0):
    return questions_genre_related;

  if(len(get_mood_from_sentence(input)) > 0):
    return questions_mood_related;

  if(len(get_date_from_sentence(input)) > 0):
    return questions_date_related;

  #Get dialog history keywords

  if(len(get_dialog_history(PREVIOUS_GENRE)) > 0):
    return questions_artist_related;

  if(len(get_dialog_history(PREVIOUS_MOOD)) > 0):
    return questions_mood_related;

  if(len(get_dialog_history(PREVIOUS_DATE)) > 0):
    return questions_date_related;

  return questions
```

Methods to get keywords

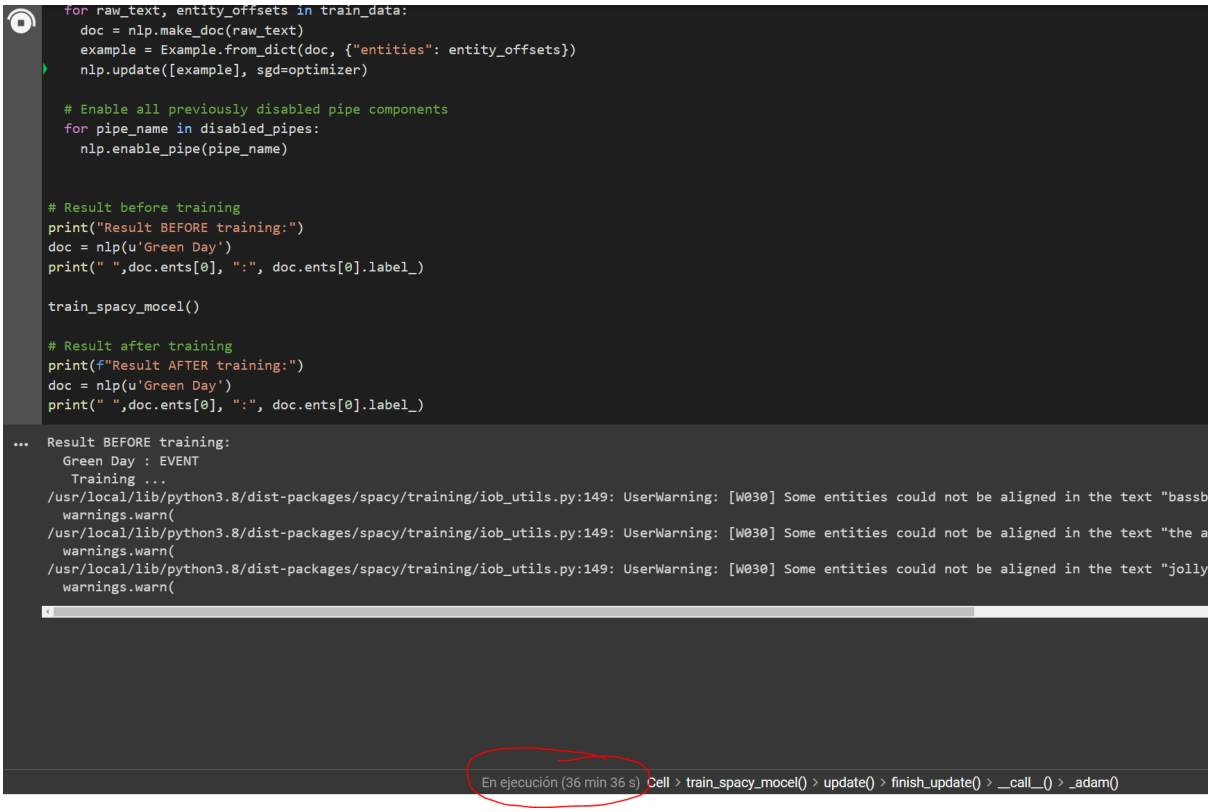
Ideal method to get keywords: spaCy

We tried to use spaCy for entity recognition as, in theory, would correctly label each keyword we use. Nonetheless we found out that we only needed the labeling process in the case of band/artists, which is a controversial field. The problem we were facing with the mentioned field is that while grouping artists with bands, band names differ a lot from artist names, for instance: "Green Day" would be labeled as "EVENT" whereas "Bob Marley" would be labeled as "PERSON". The problem doesn't end in that label misunderstanding as it doesn't recognize more than a half of the artists in 1000 examples.

```python
nlp = spacy.load('en_core_web_sm')

counter_artist_E = 0
counter_artist_P = 0
counter_artist_O = 0
counter_artist_R = 0

for artist in artist_labels[:1000]:
  doc = nlp(artist)
  for ent in doc.ents:
    if ent.label_ == 'EVENT':
      counter_artist_E += 1
    elif ent.label_ == 'PERSON':
      counter_artist_P += 1
    elif ent.label_ == 'ORG':
      counter_artist_O += 1
    else:
      counter_artist_R += 1

print('Num de EVENTS: ',counter_artist_E)
print('Num de PERSON: ',counter_artist_P)
print('Num de ORG: ',counter_artist_O)
print('Num de REST: ',counter_artist_R)
print('Total de artistas: ',len(artist_labels))
```

```
Num de EVENTS:  1
Num de PERSON:  365
Num de ORG:  155
Num de REST:  112
Total de artistas:  162806
```

Also, when we tried to train spaCy to use our custom labels, it had a really big time of execution:



These are the reasons why we didn't use this method to get the keyword from a sentence.

Final method to get keywords

For instance, when extracting the artist/band from a sentence, if any of the words in the tokenized sentence are in the artist_labels (where we store all of the artists/bands of the database) we would capture it in the result variable as shown in the image.

```
def get_artist_from_sentence(sent):
  result = ""

  sent = sentence_detection(sent)
  t_sent = sent_to_word_tokenizer(sent)[0]

  for word in t_sent:
    if word in artist_labels:
      result = word

  return result
```

This method is common for the rest of keywords, such as mood, genre.

The **date keyword is an exception**, because we first compare date_labels if there is a number in the sentence. If there is no number, we check if there is a keyword that is in new_date_keywords or old_date_keywords:

```
def get_date_from_keywords(sent):
  result = ""

  sent = sentence_detection(sent)
  t_sent = sent_to_word_tokenizer(sent)[0]

  for word in t_sent:
    if word in new_date_keywords:
      result = date_labels[len(date_labels)-1]
    if word in old_date_keywords:
      result = date_labels[0]

  return result

def get_date_from_sentence(sent):
  result = ""

  numbers = re.findall(r'\d+', sent)    # we search numbers

  if len(numbers) == 0:
    result = get_date_from_keywords(sent)
  else:
    for date in date_labels:
      if numbers[0] in date:
        result = date

  return result
```

Methods to create the answer to the user

For each question tag, there is a method that generates the answer of the agent to the question given by the user. Let's take **get_artist_from_genre(sent)**, the method that creates the answer of a 'Top-artists in genre question':

- Step 1: First of all, it gets the genre, the date and the mood that the user has written in the current sentence or in previous sentences (Dialog history). The only mandatory keyword to' activate' this answer is the genre.

- Step 2: After getting the keywords, it filters the data by genre, and then in case it got a mood or a date, it filters the data again by those parameters.

- Step 3: After that, it sorts the artists that it obtained by position, and it gets the 10 first artists.

- Step 4: Then, it takes the default answer from the answers dictionary, in this case the one that has as key ‘Top-artists in genre question’. Then, it replaces the **special characters** with the corresponding information.

```
def get_artists_from_genre(sent):
  genre = get_and_update_genre(sent)
  date = get_and_update_date(sent)
  mood = get_and_update_mood(sent)

  artists_from_genre = filter_data('genre', genre, data)

  if(len(date)>0):
    artists_from_genre = filter_data('date', date, artists_from_genre)

  if(len(mood)>0):
    artists_from_genre = filter_data('mood', mood, artists_from_genre)

  artists_from_genre.sort(key=sortPos,reverse=False)
  artists_from_genre = artists_from_genre[:10]

  result = ""
  for i in range(len(artists_from_genre)):
    result = result + "\n" + str(i+1) + "- " + artists_from_genre[i]['artist']

  default_answer = answers['Top-artists in genre question'][0]

  response = default_answer.replace('*', genre)
  response = response.replace('_', result)

  return response
```

In the case of **get_song_from_mood(sent)** method, that is the one that computes the answer from a ‘Song from mood question’, it doesn’t do step 3 of the previous method, what it does in that step is get a random song from the list of filtered tracks, but the rest of steps would work the same way.

```
def get_song_from_mood(sent):
  mood = get_and_update_mood(sent)
  genre = get_and_update_genre(sent)
  date = get_and_update_date(sent)

  tracks_from_mood = filter_data('mood', mood, data)

  if(len(genre)>0):
    tracks_from_mood = filter_data('genre', genre, tracks_from_mood)

  if(len(date)>0):
    tracks_from_mood = filter_data('date', date, tracks_from_mood)

  if(len(tracks_from_mood)>0):
    track_selected = random.choice(tracks_from_mood)

    artist = '"' + track_selected['artist'] + '"'
    result = '"' + track_selected['track'] + '"'

    default_answer = answers['Song from mood question'][0]

    response = default_answer.replace('*', mood)
    response = response.replace('+', artist)
    response = response.replace('_', result)
  else:
    response = ""

  return response
```

All the methods that generate the answers work in a similar way as the two previous methods explained.

Chatbot method

This is the proper function of the agent that consists in greeting the user and resetting the dialog history before entering the infinite loop with the boolean variable “status”.
Once the agent enters the while loop it will ask for input and react to it. First tries to understand the nature of the utterance whether the utterance is conventional-opening or closing, else, does the tag prediction with the mentioned methods above.

```
def chat_bot():

  reset_dialog_history() #to reset the dialog history each time you execute the chat_bot

  status = True

  print("Hello, I'm a music-recommender chatbot. Tell me what are you looking for and I will search in my music library.

  while status:

    input_text = input("\nUser text: ")
    output_tag = predict_tag(input_text)

    print("\n")

    if(output_tag == "Conventional-closing"):
      status = False
      print("Goodbye!")

    elif(output_tag == "Conventional-opening"):
      print("Hello, tell me what genre of music you are into or your current mood.")
```

(Infinite loop with the boolean variable “status”.)

```
else:
  output_tag = get_type_of_sentence(input_text)
  print("(Tag: " + output_tag + ")")

  print("\n--BOT ANSWER--\n")

  if(output_tag == question_labels[TOP_ARTISTS_FROM_GENRE]):
    print(get_artists_from_genre(input_text))

  if(output_tag == question_labels[TOP_TRACKS_FROM_GENRE]):
    print(get_tracks_from_genre(input_text))

  if(output_tag == question_labels[SONG_FROM_MOOD]):
    print(get_song_from_mood(input_text))

  if(output_tag == question_labels[ARTIST_FROM_MOOD]):
    print(get_artist_from_mood(input_text))

  if(output_tag == question_labels[SONG_FROM_DATE]):
    print(get_song_from_date(input_text))

  if(output_tag == question_labels[ARTIST_FROM_DATE]):
    print(get_artist_from_date(input_text))

  if(output_tag == question_labels[SIMILAR_ARTIST]):
    print(get_similar_artist(input_text))

  if(output_tag == question_labels[NOT_VALID]):
    print(get_not_valid())

  print_keyword_from_sentences(input_text)
  print_dialog_history()
```

(If the utterance is not opening-clause nor ending-clause.)

The loop stops when it detects a conventional closing sentence of the user.

Dialog Flow example

```
User text: Hello!
1/1 [==============================] - 0s 81ms/step


Hello, tell me what genre of music you are into or your current mood.
```

(The user greets the agent.)

```
User text: Can you tell me the top rock songs?
1/1 [==============================] - 0s 71ms/step


(Score: 0.6926136)
(Tag: Top-tracks in genre question)

--BOT ANSWER--

The top-tracks in rock are:

1- (Don't Fear) The Reaper
2- Feel Like Makin' Love - 2015 Remastered Version
3- Born To Be Wild
4- Sunshine Of Your Love
5- The Joker
6- Baba O'Riley - Original Album Version
7- You Really Got Me (Remastered)
8- She Talks To Angels
9- More Than a Feeling
10- Stairway To Heaven

--Keywords from CURRENT sentence--
Genre: rock
Mood:
Date:
Artist:

--Dialog history--
Previous genre: rock
Previous mood:
Previous date:
```

(User asks for a top-tracks in genre question.)

```
User text: Do you have some sad song?
1/1 [==============================] - 0s 68ms/step


(Score: 0.63545823)
(Tag: Song from mood question)

--BOT ANSWER--

When I feel sad, I like to listen to the track "Hold the Line", from "Toto".

--Keywords from CURRENT sentence--
Genre:
Mood: sad
Date:
Artist:

--Dialog history--
Previous genre: rock
Previous mood: sad
Previous date:
```

(The user asks for a sad song, and the agent taking into account the dialog history, it search also a song from the rock genre.)

```
User text: Can you recommend me something older?
1/1 [==============================] - 0s 63ms/step


(Score: 0.5397573)
(Tag: Artist from date question)

--BOT ANSWER--

You should listen to "Chicago", an artist from 1970.

--Keywords from CURRENT sentence--
Genre:
Mood:
Date: 1970
Artist:

--Dialog history--
Previous genre: rock
Previous mood: sad
Previous date: 1970
```

(The user asks for an older song. Again, the agent makes use of dialog history to find an older song with previous characteristics.)

```
User text: Do you know something similar but from the 80s?
1/1 [==============================] - 0s 94ms/step


(Score: 0.5605442)
(Tag: Artist from date question)

--BOT ANSWER--

You should listen to "Jefferson Starship", an artist from 1980.

--Keywords from CURRENT sentence--
Genre:
Mood:
Date: 1980
Artist:

--Dialog history--
Previous genre: rock
Previous mood: sad
Previous date: 1980
```

(The user can also ask with a numeric date.)

```
User text: bye
1/1 [==============================] - 0s 60ms/step


Goodbye!
```

(closing statement to end the conversation)

References

[1] Dataset: https://github.com/trebi/music-genres-dataset