POLITECNICO DI TORINO

Master degree course in Computer engineering

Master Degree Thesis

Multi-channel Conversational Agent for Personalized Music Recommendations

Supervisor:
Prof. Maurizio Morisio

Candidate
Giulio Verazzo
Student id: 225208

Internship Tutor
Dott. Ing. Giuseppe Rizzo

Academic Year 2017-2018

Contents

Chapter 1

Introduction

Recent technological advances in entertainment applications all move in the direction of providing the user with an increasingly tailor-made and personalized experience. The reasons for this phenomenon are to be found in the need to offer quality and relevant content to each user in order to increase the engagement and improve the experience (and therefore revenues). The amount of data available is enormous and the development of intelligent and automatic filters is one of the main technological challenges currently underway. In this scenario, different technology components are moving, such as recommender systems, chatbots and applications for affective computing. The aim of this thesis work is to explore and evaluate, in terms of design, usability and usefulness, the use of a conversational interface for a recommender system, whose input data is previously filtered through an affective computing software. In detail, the application developed is a multi-platform chatbot, named *Beat in a Bot*, whose purpose is to recommend music artists to listen to, based on music preferences shared among groups of people with similar personality traits. The result is a complete and usable product, currently available to the public on Telegram and Facebook Messenger. The work is presented as follows: the first chapter contains an overview of all the pieces of the project. The second chapter covers related work in the fields of chatbot and recommendations, music preferences and recommendations and personality of individuals and music preferences. The third chapter includes a description of the high-level architecture and the details of each component. The fourth chapter is dedicated to the implementation and explains all the technical aspects of the application. The fifth chapter shows the results of the evaluation process composed of the design, usability and usefulness test. The last chapter is dedicated to conclusions, future improvements and a possible provisional exploitation plan of the product.

Chapter 2

Related Work

In recent years, several studies have been carried out on the topics covered in this thesis project. As for chatbot and recommendations, (Ikemoto et al. 2018) proposed a conversation strategy for interactive recommendations using a chatbot. The strategy combines questions about user's preferences and recommendations soliciting user's feedback to them. (Kucherbaev et al. 2017) implemented a chatbot to connect citizens with policy-makers to improve the civic engagement of citizens providing recommendations based on open data sources, making it easy to contribute to the city and to be involved in discussions about urban issues. (Holotescu 2016) built a MOOC (Massive Open Online Courses) recommender system as a chatbot for Facebook Messenger called MOOCBuddy to find the best online learning resource. (Atzori et al. 2017) outlined a proof of concept to create a chat-based client named *Touristific*, for inserting and requesting information into an integrated system for planning travels. The approach consists of using a context-aware recommender system that uses attributes like position, social network connections, weather conditions, date and time of the visit to predict personalized recommendations; these are personalized packages built with semantic data integration and presented using a chatbot as user interface. (Narducci et al. 2017) from University of Bari developed a conversational movie recommender system implemented as Telegram Bot with the aim to integrate it as a component of an Internet of Things device such as a smart TV. The bot is based on the Linked Open Data (LOD) cloud and implements Personalized PageRank as recommender algorithm. (Costa & Macedo 2013) described an ongoing recommender system application, that implements a multi-agent system, with the purpose of gathering heterogeneous information from different sources and selectively deliver it based on: user's preferences, the community's trends, and on the emotions that it elicits in the user.

Several studies focused their attention on music preferences and recommendations: in particular (Nanopoulos et al. 2010, Symeonidis et al. 2008, Musto et al. 2012) have leveraged social media sources to music recommendations. One worth mentioning is a work by (Bu et al. 2010) who proposed a novel music recommendation algorithm that uses both multiple kinds of social media information and music acoustic-based content. They modeled objects and relations using an hypergraph and approached the music recommendation as a ranking problem on the hypergraph; (Baltrunas et al. 2011) have elaborated a system design methodology to build an effective context-aware mobile recommender system (CARS): such a system adapts recommendations to the specific situation in which the recommended item will be consumed. For instance, music recommendations while the user is traveling by car should take into account the current traffic condition or the driver's mood. This requires the acquisition of ratings for items in several alternative contextual situations, to extract from those data the true dependency of the ratings on the contextual situation; (Lu & Tseng 2009) proposed an interesting study to improve music recommendations described as "personalized hybrid music recommendation", which combines the content-based, collaboration-based and emotion-based methods by computing the weights of the methods according to users' interests.

Correlations between music preferences and the Five Factor Model (McCrae & John n.d.), a model used to describe aspects of personality such as openness to experience, agreeableness, conscientiousness, extraversion and neuroticism, has been investigated by (Rawlings & Ciancarelli 1997) and (Dollinger 1993) who involved university students to both NEO Personality Inventory and an update version of Litle and Zuckerman's Music Preference Scale, a questionnaire measuring music preference. They found that Extraverts obtained high scores on the Popular Music factor and Open individuals liked a wide range of musical forms outside the mainstream of popular and rock music. They also found that females liked popular music styles more than what did males. Music preference also correlates to Jungian types (Isabel Briggs Myers 1962) of personality, a model used to describe personality of individuals using four dichotomic indexes that indicates how people focus attention and energy (Extraversion-Introversion), how they extract information from the surrounding world (Sensation-iNtuition), take decisions (Thinking-Feeling) and relate to the external world (Judging-Perceiving). Such correlation was explored by (Pearson & Dollinger 2004) who hypothesized and demonstrated that the sensing–intuition dimension correlates with overall musical enjoyment, in particular, individuals who scored towards the sensing end, endorse more musical styles, particularly classical music, as well as they have greater musical training and involvement. Furthermore, extraversion also correlates with overall musical interest, particularly for popular/rock music. Finally, thinking–feeling correlates with liking for country and western music.

Chapter 3

Background

3.1 Chatbots

3.1.1 Introduction

Messaging platforms play an important role in today's communications as demonstrated by the enormous growth in terms of usage and time spent of messaging apps. An analysis conducted by Text Request[1], a company that offers enterprise solutions for communication strategies using text messages, reported 18.7 billion text messages sent every day with a growth from 2011 to 2014 of 140% (Burke 2016); moreover, Flurry Analytics [2], a company by Yahoo that offers developer tools to measure and analyze activity across mobile apps, published a report (Khalaf 2017) showing a grew of 394% between years 2015 and 2016 in time spent for Messaging & Social apps (Figure 3.1). Reasons of such popularity are to be found in the benefits of using text communication tools. Text messages are fast, easy to use, cheap or almost free, can be used in formal or informal contexts, they are not intrusive like a phone call and they can be ubiquitous: can be sent from desktop or mobile systems like smartphones, tablets and smartwatches. All these things explain why text messages are so popular globally and why messaging platforms are growing exponentially in terms of app usage and time spent. Chatbots, being part of the instant messaging world, are surfing the raising wave of such popularity.
A chatbot could be thought as someone to chat with, that is always available and

[1] https://www.textrequest.com/

[2] https://developer.yahoo.com/analytics/

will answer to all the questions in his sphere of understanding with the only exception that he is not human. From a technical perspective, a chatbot is a computer program that conducts a conversation via auditory or textual methods with the intention of simulating a human-like behavior[3]. Instant messaging apps like chatbots uses Web Socket or other real-time communication protocols that provide the reliability and speed required for real-time text transmission. There is a big variety of chatbot applications, from those that can handle few questions based on keywords for few tasks, to those that use Natural Language Processing and Artificial Intelligence algorithms, able to learn from their interlocutor to improve themselves over time.

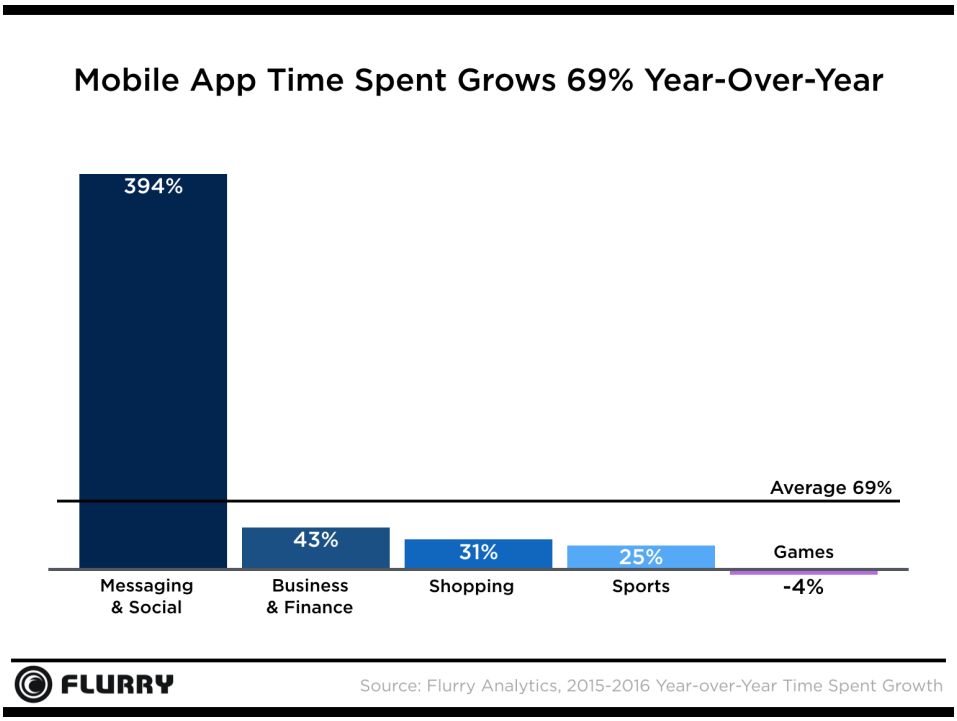


Figure 3.1. Flurry Analytics, 2015–2016 Year-Over-Year Time Spent Growth.

[3] http://searchdomino.techtarget.com/definition/IM-bot

3.1.2 History of chatbots

The first evidence of this kind of computer programs comes from Alan Turing, an Englishman known as the father of computer science, who was a computer scientist, crypt analyst, logician and mathematician. In 1950, he published his famous article *Computing Machinery and Intelligence* (Turing 1995) which is used nowadays as the basis of intelligence assessment of computer programs. In this article, he proposed a criteria to determine if a machine is able to think or not; this is known as the *Turing Test* and depends on the ability of a computer program to impersonate a human in a real-time written conversation with a human judge, sufficiently well that the judge is unable to distinguish, reliably on the basis of the conversational content alone, between the program and a real human. The criteria of Alan Turing inspired Joseph Weizenbaum, a German-American computer scientist and professor at Massachusetts Institute of Technology, who in 1966 published a program called ELIZA (Weizenbaum 1966). The ELIZA program was built with the purpose to give to his interlocutor, an illusion of intelligence, making people to think they were talking to a real human. His algorithm (used by all the chatbot makers) was able to analyze the input given by the users by searching from phrases and keywords, and give pre-planned responses. For instance, given a sentence ELIZA would do the following in order:

- Receive the input and store them in memory for further analysis;
- Search for keywords in the sentence;
- Give back the pre-programmed response if a keyword is matched;
- Give back the default answer "Sorry, don't know about that" if there is no clear match.

So, in a sentence containing the keyword "mother" a typical response by ELIZA would be "Tell me more about your family". Although the simplicity of the algorithm, this was enough to give an illusion of intelligence to a human judge showing the fact that such an illusion is surprisingly easy to generate, because humans are so ready to give the benefit of the doubt when conversational responses are likely to be interpreted as "intelligent".

The term "ChatterBot", that became then "chatbot" in the literature to follow, was coined in 1994 by Michael Mauldin, an American inventor and scientist, founder of Lycos Inc., who created the program Verbot, a popular chatbot that followed the path traced by ELIZA. Its latest version, which was called Sylvie, can be considered as the first intelligent animated virtual human: it incorporates real-time animations as well as speech and natural language processing. The idea was to develop a conversational agent that could act as a human-machine interface.

Thanks to the latest progress in the field of artificial intelligence, chatbots can develop their skills and learn from every single chat. So, the more conversations about different topics they have, the more skilled they become. Such a chatbot does not get stuck when he gets unknown questions because he can quickly make connections between various notions and come to an answer. A good example of a chatbot powered by artificial intelligence is Cleverbot[4] created by British AI scientist Rollo Carpenter. Cleverbot uses artificial intelligence to learn from human inputs in order to give responses that are not pre-programmed. It responds to the input by finding how a human responded to that input when it was asked. Cleverbot has reached a score of 59.3% to the Turing Test, compared to the rating of 63.3% human achieved by human participants. A score of 50.05% or higher is often considered to be a passing grade (Aron 2011). The introduction of artificial intelligence in the field of conversational agents, has attracted the attention of the big companies of the IT world, which have started to release commercial products that use this technology. Apple started the trend with the introduction of Siri, the first digital smartphone assistant, followed by products such as Amazon Alexa, Google Assistant, Microsoft Cortana and Samsung Bixby to cite the most popular. Their most common features are: send text and email, make phone calls, schedule appointments, check the weather, stocks and flight status, make conversions and translations, search on the web, find what song is playing the room, buy products on online stores, tells you when to leave based on the expected traffic and more. The interest of such big companies to the digital assistants world, is a sign of the enormous potential of this technology and how much it will have an impact in the future of our lives.

3.1.3 Types of chatbots

Chatbots can be distinguished by the logic of their back-end that defines their intelligence level and capabilities. There are three types of bots: rule-based chatbots, intent-based chatbots and conversational agents.

Rule-based chatbots

These bots were made to follow a pre-determined path on a logic tree. The developer writes a *pattern* and a *template*; when the bot encounters that pattern in a sentence from user, it replies with one of the templates. Those kind of bots are easy to create but it is incredibly difficult to make them answer to complex queries. The pattern matching is weak and hence time consuming and takes a lot

[4] http://www.cleverbot.com/

of effort to write the rules manually. A good example of this kind of bots is ELIZA. ELIZA consisted of a simple substitution rules to mimic a psychologist from 1960s. The main idea was that the bot simply replies to the questions by repeating back the words of the questioner. If the question was "should I buy an orange?" a possible answer was "tell me why you should buy an orange". They do not have the concept of statement meaning and only provide an appearance of conversation without understanding what is being said.

Intent-based chatbots

These types of bots do not relies upon a match between patterns and templates, instead they can handle free-text. This means that the user can simply enter any sentence and the bot will understand language as commands. The understanding is divided into two subproblems:

- Identifying what the user wants the machine to do (the "Intent");
- Figuring out the details of the Intent.

For instance, if the user asks to "Play Jazz", the bot first needs to understand that user wants to play music (the Intent) and then it must understand that, in particular, he wants to hear Jazz music (the details). Intents can be understood using Natural Language Processing (NLP) algorithms such as word embeddings (Mikolov et al. 2013), which converts words or phrases (called "tokens") into vectors of real numbers so that they can be used to train a deep learning algorithm to recognize text patterns. A typical implementation is a sequence-to-sequence model that consists of an encoder and a decoder. Both are implemented using recurrent neural networks. The encoder takes as input the set of tokens and generates a vector that goes as input into the decoder that generates a set of output tokens until it generates a special stop symbol. This approach is vastly used in language translation. A set of Italian words like "il libro è sul tavolo" is transformed into a vector and translated into "the book is on the table" thanks to the number of learning sessions the machine has done. Conversation can be seen as a translation if we substitute the source language with the question and the destination language with the answer. Intelligent models can learn from existing human conversations. Lots of dialogues dataset can be found on the web like OpenSubtitles, Ubuntu Dialog Corpus or replies to tweets from Twitter. There are several NLP/NLU engines available on the web, some of them are open-source and owned by some of the biggest companies in the planet; wit.ai [5](Owned by Facebook), which is now

[5] https://wit.ai/

turning into a Messenger only technology, Dialogflow [6] (Owned by Google), used as a component of this project and IBM Watson[7] to mention the most popular.

Conversational Agents

Those are expansion of Intent-based agents to add multi-turn conversation. This is done by keeping track of the state of the conversation and knowing when the person wants to talk about something else. A good implementation of this concept is the framework RavenClaw (Bohus & Rudnicky 2003) that uses a dialog stack and an expectation agenda. The dialog stack keeps track of all the thing the chatbot wants to talk about. The expectation agenda is a data structure to keep track of what the chatbot expects to hear. For example, the chatbot asks, "what is 4+5?"; on the top of the dialog stack there is the 4+5 question and the expectation agenda is filled with the answer "9". Let's say the user replies "buy me a pizza", the bot needs to switch the context of the conversation and so, the dialog stack is pushed with "order groceries" and the expectation agenda is updated with possible answers and questions about ordering food. Another possible implementation that can be done is using a machine learning approach with reinforcement learning (Serban et al. 2017). Reinforcement learning consists of a set of states, actions and a reward function that provides a reward for being in a state s and taking an action a. The idea is that the chatbot has states made of what the bot knows (questions it has answered), the last thing the bot said and the last thing the user said. The bot has to learn a policy (a rule) that gives the best action a for being in state s. The problem is that learning a policy requires a lot of training and also, it is hard to know exactly what state the agent is because of errors in speech-to-text or errors in understanding.

3.1.4 Current state of chatbots and limitations

The growing interest in chatbots can be explained viewing at three different perspectives. From a financial perspective, business owners realized the usefulness chatbots can provide to end users, especially when the information can be categorized into concrete and predictable subjects. For example, chatbots can be easily integrated into the customer support experience making the company available to users 24 hours 7 days a week. In addition to that, it is proven that customers prefer to chat with a fast responding customer support rather than find the number and wait to the phone until their turn to come. A chatbot can easily handle all the Frequently Asked Question and when the problem cannot be solved by the

[6] https://dialogflow.com/

[7] https://www.ibm.com/watson/services/natural-language-understanding/

machine, he could refer the users to the appropriate technician that is now freed from the hassle of repeating again the same things over and over. This is also good to separate the mechanic work from the brain work giving the machine and the human the jobs that suit them well both. From a user perspective, chatbots can finally give them the best interface to a machine for the human: the word, written or spoken. This opens to a whole new world of possibilities, users can now speak to a machine asking for the information they are searching for, in the most natural way. From a development perspective, big companies has announced and released their Software Development Kits and Application Programming Interfaces giving the developers the ability to make their own chatbot available for the users: Apple with his iPhone assistant Siri, IBM with the Watson project, Google with Google Home, Amazon with Echo, Microsoft with Cortana, Facebook with the Messenger platform and many others.

Bots are able to understand the intent of the user relatively well only if the interface is made of a set of pre-defined commands hidden in keywords. Problems come to rise when they have to fully understand human language.

Chatbots today are facing those main problems:

- They are devoid of meaning that means, they do not have any knowledge of the world. For instance, in the statement "buy me 3 apples" there is no difference between 3 apples or 300 for the machine and this may cause understanding and evaluating problems. When we refer to an object, the bot has never held the object or used it so it will have a limited understanding of it. Since meanings that chatbots have are largely fixed we currently can not negotiate meaning with them.

- They have problems with prosody (the way phrases are pronounced). Chatbots with speech interaction can not capture nuances of meaning from the way a phrase is pronounced. As example, the phrase "I know that you know" can hide a feeling of anger towards the interlocutor if a pause is inserted after the word "you". This is extremely difficult to encode in a chatbot working logic.

- They have problems with logical inference (make a logical assumption given what is been said). As example, in a normal conversation a simple sentence like "I'm American" brings a set of logical inferences, for instance, the interlocutor is born in America, speaks English, knows about American culture *et cetera*. Such information derives from a bag of shared experiences between humans, something that chatbots can only have in a limited amount.

In conclusion, the state of art is that we can build knowledge into chatbot that they can use to cooperate with us on tasks requiring minimal understanding. Building better chatbots will necessitate knowledge engineering and research into how agents can better follow and adapt to the subtleties of meaning and conversation.

3.2 Recommender systems

3.2.1 Introduction

On the Web, where the number of choices is overwhelming, there is need to filter, prioritize and efficiently deliver relevant information in order to alleviate the problem of information overload, which has created a potential problem to many users. Recommender systems solve this problem by searching through large volume of dynamically generated information to provide users with personalized content and services. In this chapter, there will be a section about fundamentals of recommender systems, what they are, how they generally works and why there is interest in such technology. A second section to describe different types of recommender systems and finally, what is the current state of the art, as well as, what are the current challenges and limitations.

3.2.2 Fundamentals of recommender systems

Recommender systems are information filtering systems that deal with the problem of information excess by filtering vital information fragment out of large amount of dynamically generated information according to user's preferences, interests, or observed behavior over time. Recommender systems are used today for large variety of tasks and can be beneficial to both service providers and users. They reduce transaction costs of finding and selecting items in an online shopping environment and it is proven that they can improve the decision making process and quality: showing users items that might interest them, increases the time spent on the platform and therefore the likelihood of a purchase being made. In scientific libraries, recommender systems support users by allowing them to move beyond catalog searches.

From a technical perspective, they are computer algorithms that uses linear algebra theorems to predict a user preference from a user-item matrix so, to implement a recommender system, a *user-item* matrix is first needed. Such matrix has all the users on the rows and all the items on the columns. Each value of the matrix can be the rating that a particular user gave to a particular item. Obviously, users have rated only a limited number of items and so the matrix is sparse. This brings us to the goal of the algorithm that is, to fill all the empty boxes with the predicted ratings for the users to the items. The way the user-item matrix is filled, depends on the typology of recommender systems: those are presented in the following section.

	$item_1$	$item_2$	$item_3$	...	$item_n$
$user_1$		5	2		1
$user_2$	3				
$user_3$	1		3		
.					
.					
.					
$user_{m-1}$	5		4		2
$user_m$		4			3

Figure 3.2. An example of user-item matrix

3.2.3 Types of recommender systems

Since recommender systems can be used in a large variety of applications, different types of algorithms have been developed each with its peculiarities, pros and cons. Recommender system algorithms can be divided into, Content-based filtering, Collaborative filtering and Hybrid filtering.

Content-based filtering

Content-based filtering algorithm main idea is based on the concept that two items are similar if their characteristics also are. For instance, two pairs of shoes are considered similar if they are of the same brand or belongs to the same category (i.e. boots); so items are represented in terms of their descriptors or attributes. This means that the algorithm needs data provided with meta-data that describes them. Such kind of data is called *Classified data*. Usually this type of filtering is used with text documents like books or articles because such kind of items are generally already classified for other purposes like cataloging.
Users and items need to be prepared before they can be used in a content-based filtering algorithm. The process is performed in three steps handled by separate components:

- *Content Analyzer*: In order to be used in a recommendation algorithm, information (e.g. text) needs to be in a structured form. This component analyzes data items and produce structured data for next processing steps by feature extraction techniques (e.g. Web pages represented as keyword vectors). This representation is the input to the Profile Learner and Filtering Component.
- *Profile Learner*: To match users with items, the system has to represent users in the same information space of the items. This module collects data

representative of the user preferences and tries to generalize it in order to construct the user profile. Usually, the generalization is realized through machine learning techniques like supervised learning algorithms, which generate a predictive model (the user profile) that represent user interests starting from liked or disliked items. For instance, the Profile Learner of a movie recommender system can implement an algorithm in which the learning technique combines vectors of positive and negative feedback into a prototype vector representing the user profile. Training examples are movies on which a positive or negative feedback has been provided by the user.

- *Filtering Component*: Once items and users are represented in the same information space, the user profile can be exploited to suggest him relevant items. Given a new item representation, the Filtering Component predicts whether it is likely to be of interest for the active user, by comparing features in the item representation to those in the representation of user preferences (stored in the user profile). The result is a binary or continuous relevance judgment i.e. a ranked list of potentially interesting items. The matching is computed using some similarity metric like cosine similarity that exploits the cosine between the two vectors to express distance.

The main problem with this type of algorithms is the difficulty to find data that is already classified: each item has to be classified with its attributes and this process is not always straightforward. First of all, there is the problem to choose the attributes of the items and then to find how to measure them, as example, given a movie, it is hard to know if it is drama or comedy without a manual classification. So, data needs to be manually classified and this can be very tedious. This obstacle makes content-based filtering approach less used compared to other algorithms of recommendation systems.
One successful application of content-based filtering is the Music Genome Project by Pandora Radio. The idea is to associate descriptors to each song. Each song is described by a vector of 450 "genes" manually generated by professional musical analysts. Genes are songs properties like the tempo, the pitch, the frequency and more. Once each song has its own attributes, users can start to like songs and the algorithm can do his recommendations.

Collaborative filtering

Collaborative filtering algorithms take inspiration from what happens to the real life when we are searching for recommendations and we end up usually asking a friend. This relies upon the idea that if two people have same opinions about a group of items, then it is likely that they have the same opinion on other items too. Every algorithm that is based on the user behaviors (history, reviews, similarity) belongs to the branch of collaborative filtering algorithms. Unlike content-based,

collaborative filtering does not require items to be classified, instead, this algorithm can be used even with absolute no knowledge about the items that are going to be recommended. These types of algorithms tries to predict the reviews of a user to a product he has not reviewed yet. Review is a generic term and can be referred to a positive-only feedback, where user has liked/not liked the item, an explicit review, where the user assigns a number (typically on the Likert scale from 1 to 5) to the item, or an implicit review made of all the meta-data about a user like the numbers of clicks, the time spent viewing a product, his temporary shopping cart, his researches or his history of purchased products or digital goods consumed. Of all the collaborative-filtering algorithms available, two in particular has gained popularity in the community: the Nearest Neighbor algorithm and the Latent Factor algorithm. The Nearest Neighbor algorithm finds user that are similar to other users, using a metric for the similarity and a metric for the distance between users. The Latent Factor algorithm tries to find common factors among users from the reviews they already gave to items.

Hybrid recommender system

As the name suggest, hybrid recommender systems combine collaborative filtering and content-based filtering techniques to obtain the best of both worlds and to overcome some of the common problems in recommender systems such as cold start, gray sheep and the sparsity problem (discussed in the following section). Several studies (Burke 2002) empirically compared the performance of the hybrid with the pure collaborative and content-based methods and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches. There are several ways to implement a hybrid approach: one can be making the content-based and collaborative-filtering prediction separately and then combining them; predictions are based on a weighted average of the content-based recommendation and the collaborative recommendation. The rank of each item being recommended could be a measure for the weight. In this way the highest recommendation receives the highest weights. Another technique is to add content-based capability to a collaborative-based approach or viceversa; since collaborative filtering looks for the correlation between user ratings to make predictions, finding such correlation can be unfeasible for users with few ratings. Adding content information to the recommender system can help to correlate users. For example, if one user liked the movie “Rocky” and another liked the movie “Rocky II” they would not necessarily be matched together in a standard collaborative filtering algorithm because there is no knowledge about items correlations. A hybrid approach deals with these issues by incorporating both the information used by content-based filtering and by collaborative filtering.

Netflix [8], a popular streaming media platform, uses an hybrid recommender system. The website makes recommendations by comparing the watching and searching habits of similar users (i.e., collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering).

3.2.4 Current state and limitations

Latest studies (Gouvert et al. 2018, Wu et al. 2018) have shown the utility of machine learning techniques in the area of recommendation systems and information retrieval. The general opinion is that of significant improvements over the conventional models. As example, deep learning can be used to help classifying data for content-based algorithms or to replace matrix factorization with deep neural networks for collaborative filtering. Most of the deep learning development has been biased towards entertainment industry such as in movie and music recommendation. This can be largely attributed to the availability of rich datasets for validation. Although this trend has covered all the three major variants of recommender system algorithms, collaborative filtering, content based and hybrid systems, it can be clearly seen from the number of publications made that the majority have leveraged deep learning to enhance collaborative filtering capabilities. Unfortunately, recommender systems have a set of problems and limitations (Adomavicius & Tuzhilin 2005) that has to be taken into account: *cold start, synonymy, gray sheep, data sparsity* and *shilling attacks* are the most frequent.

Cold start

The cold start problem addresses the case of new items or new users in the system: since they have no ratings history, find good recommendations can be tricky for them. One of the solutions is to use a hybrid system called content-boosted collaborative filtering that combines known attributes of items with demographic data of users to improve collaborative filtering recommendations.

Synonymy

Synonymy occurs when the algorithm has to deal with items that are practically the same but differs from a single or few attributes, like two different editions of the same book; latent factor collaborative filtering works well with these cases.

[8] https://www.netflix.com

Shilling attacks

Shilling attacks is the name given to the problem of dealing with users that want to fool the recommender system. As example, think at a restaurant owner that gives fake negative reviews to other restaurant while using fake accounts to give positive reviews to his restaurant. In that cases, precautions is the best weapon.

Data sparsity

Data sparsity occurs when a lot of users have reviewed only few items: this makes the user-item matrix very sparse, resulting in hardness to compute the prediction and unsatisfied results. The solution is to use dimensionality reduction to remove less important dimensions in vectors to reduce sparsity.

Gray sheep

Gray sheep are users with non-consistent opinions. The hypothesis on which collaborative filtering algorithms are based is that, if two users like the same item and one of them likes another item, that item may interest the other user too, but it is false with gray sheep. Content-boosted collaborative filtering can help dealing with this problem.

3.3 Personalization

3.3.1 Introduction

With the enormous amount of available data on the web today, there is the need to filter out what really interests users. Data filtering can improve user engagement in entertainment applications, can enhance revenues of online shopping portal (think about what Amazon does with "Customers who bought this item also bought") and can help search engines to give better query results. Recommender systems were built to overcome the problem of data filtering but they only relies upon items attributes and users history, with little "real-life" relation to the user. There are strong desires to provide online users more dedicated and personalized services that fits into individual's need, usually strongly depending on the inner personality of the user. Canada Peer Counseling Center (Chen 1998) considers that for most people they have investigated, recommendations from companion volunteers with same views as being the most effective. It means people with same personality tend to attract each other. As a result, analysis of different personality features can be the basis for building characterized service. For instance, an extrovert user may have a higher level of online activity that is more likely to use recommendation system to make new friends with strangers (Moore 2012).

3.3.2 Background

Personality is defined as the set of habitual behaviors, cognition and emotional patterns that evolve from biological and environmental factors (Corr & Matthews 2009). The study of the psychology in personality attempts to explain people differences in behavior. Many approaches have been taken to study personality, including biological, cognitive, learning and trait based theories, as well as humanistic approaches. Following, there will be a description of two approaches used for scientific applications thanks to their numerical nature: the Five Factor Model and the Myers Briggs Type Indicator.

Five Factor Model

The Five Factor Model (FFM), also known as the Big Five Model (Big5), is a model based on common language descriptors of personality. The model mainly relies upon two theories: the lexical hypothesis according to which is possible to derive a comprehensive taxonomy of human personality traits by sampling language, and a statistical approach that identifies the characterizing dimensions of individual differences through a factorial statistical analysis. Studies on the personality lexicon (John et al. 1988) started in 1884 by an English psychologist, Sir Francis Galton, who was the first person that have investigated the hypothesis that it is possible to derive a comprehensive taxonomy of human personality traits by sampling language (Galton 1883). A study by (W. Allport & S. Odbert 1936) put Galton's hypothesis into practice by extracting 4,504 adjectives from the dictionary that they believed were descriptive of observable human personality traits. This number was reduced in a successive study by (Cattell 1957) who brought it to 171 by eliminating synonyms and adjectives. Finally, (C. Tupes & E. Christal 1992), based on a subset of only 20 of the 36 dimensions Cattel had discovered, found just five broad factors to describe personality of individuals that were labeled: *Agreeableness, Conscientiousness, Extraversion, Neuroticism* and *Openness*, also known as OCEAN:

- *Agreeableness* refers to being helpful, cooperative, and sympathetic towards others. Low Agreeableness is related to being suspicious, challenging and antagonistic towards other people.

- *Conscientiousness* is determined by being disciplined, organized, and achievement-oriented. Low Conscientiousness individuals tend to be more tolerant and less bound by rules and plans.

- *Extraversion* is displayed through a higher degree of sociability, assertiveness, and talkativeness. People low in extraversion are reserved and solitary.

- *Neuroticism* refers to degree of emotional stability, impulse control, and anxiety. Those who have a low score in Neuroticism are more calm and stable.

- *Openness* is reflected in a strong intellectual curiosity and a preference for novelty and variety. People low in Openness tend to be more conservative and close-minded.

In 1985, psychologists McCrae and Costa developed a personality inventory called *The Revised NEO Personality Inventory* (NEO PI-R) (Costa & McCrae 1992) that makes the Big Five personality traits measurable. The inventory consists of 240 questions on a five-point Likert scale, and can accurately measure the five personality traits plus six underlying facets for each of them. Traits are expressed as dimensions of a vector in a scale between 0 and 1.

Myers Briggs Type Indicator

In 1942, two psychologists, Katharine Cook Briggs and her daughter Isabel Briggs Myers, have developed a convenient way to describe the order of Jungian preferences of a person. Their work (Isabel Briggs Myers 1962) is based on the Jungian theory of personality described in the book *Psychological types* (Jung 1921), in which he outlines the different characteristics of attitudes and behaviors. Jung mainly splits individuals in two categories, Introverts and Extroverts that represents the guidelines both in the objective and subjective world. Then, he postulated that individuals relate to the world through two sets of opposed functions, rational functions of thinking and feeling and irrational functions of sensation and intuition. Jung work is descriptive and general, which from one side leaves personal interpretation opened but on the other makes hard to outline personality attitudes precisely.
Myers and Briggs, facing these difficulties, have extracted the more easily distinguishable personality traits from Jung work on which they added a personal contribute based on a more traditional psychometric procedures. The result of their studies is a set of intrinsically consistent indices that measures the differences in behavior of individuals. The set is made by four indexes, Extraversion-Introversion, Sensing-Intuition, Thinking-Feeling and Judgment-Perceiving. The last index is their contribution to Jung's work. Each index is dichotomized, with a center-fixed zero point and represented by two letters that indicate the ends. They indicate how people:

- Focus attention and energy (Extraversion-Introversion)

- Extract information from the surrounding world (Sensation-iNtuition)

- Take decisions (Thinking-Feeling)

- Relate to the external world (Judging-Perceiving)

The combinations of these four indexes outline 16 possible psychological types. For instance, ESTJ means a profile made the following attributes: Extraversion (E), Sensing (S), Thinking (T), Judgment (J) while INFP means Introversion (I), iNtuition (N), Feeling (F), Perception (P).

3.3.3 Personality traits from social media platforms

Knowing the user personality can help to make better recommendations in recommender systems thanks to the *homophily* principle, which states that people with related attributes, such as age, interests, and personality, form similar ties (McPherson et al. 2001).
In psychological researches, most traditional personality analyzing experiments are based on self-reported inventory. Self-Report Measures are any method of data collection that is based on the participant to report his or her own behaviors, thoughts, or feelings. The advantage of this method is that the researcher can obtain information that is not easily observable, but the disadvantage is that participants' report may not be accurate or reliable. For example, asking students to report how many hours per week they used Facebook, they may under-report the time due to embarrassment or not realizing how much time they spend on the site. However, independent observation of Facebook use would be difficult and costly to implement. To solve these problems, several studies (Hughes et al. 2012, Bachrach et al. 2012, Gosling et al. 2011) has explored techniques of personality prediction from social media behaviors. The rising of social media and their growing popularity is a resource of enormous value for personality traits studies. Users fill them with textual data that tend to reflect many aspects of real life, including personality. They widely share their feelings, moods, and opinions, providing without knowing a rich collection of information that could be used for a variety of purposes. This phenomenon is known as subconscious crowdsourcing and indicates that people are unaware and are subconsciously sharing information to their network. Golbeck et al. have shown that Extraversion and Neuroticism were found to correlate with number of friends in the real world as well as on Facebook (Golbeck et al. 2011). Individuals high in Extraversion and low on Neuroticism tend to maintain persistent connections with their friends (Anderson et al. 2001, Berry et al. 2000). Extroverts people also find those platforms easy to use (Rosen & Kluemper 2008). Generally, users tend to be friend of people with higher Agreeableness and select contacts with similar Openness, Extraversion and Agreeableness (Schrammel et al. 2009).
There also are several tools to predict personality from social network behavior: (Bai et al. 2012) have built a model to predict personality of RenRen users, a Chinese social network. University of Cambridge Psychometrics Centre developed a

web application called Apply Magic Sauce[9], which uses artificial intelligence models based on over 6 million social media profiles and matching scores on psychometric tests; this gives the ability to predict user's personality from their digital footprint. Those are examples of the excellent body of work carried out by researchers in order to better understand the existing correlation between personality and social media platforms.

Unfortunately, collecting data about users personality can be a problem in terms of privacy. The recent Cambridge Analytica scandal [10] is a clear example of the delicacy of such data. They collected personally identifiable information of about 50 million Facebook users without explicit consent in 2014 using a Facebook app called *thisisyourdigitallife*. The data was used to influence American elections and the brexit vote. Such influence was conducted with news that exploited users hopes and fears to guide public opinion towards politicians that paid Cambridge Analytica for such data. As a result, Facebook sparked public outcry (a *#deletefacebook* campaign was born on Twitter) and lowered stock prices and had to publicly apologize both online and offline with posts and newspaper articles. Such episode teaches us that the power of this technology is to keep under control and should be limited to a scientific and conscious usage only in order to avoid this type of problems in future applications.

[9] https://applymagicsauce.com/

[10] https://en.wikipedia.org/wiki/Facebook_and_Cambridge_Analytica_data_breach

Chapter 4

Approach

4.1 Introduction

This chapter is dedicated to present the architectural work required before the actual chatbot implementation, first from a high-level perspective, showing how components are bound together to work and communicate, and then exploring the details of each single component, their description, how they work and the rationale for their choice.

4.2 High-level Architecture

To explore the various aspects that we had to take into consideration while architecting the chatbot application, is useful to start from the words of the thesis title. The first word, *Multi-channel*, describes a property of the chatbot. It means that it can be accessible from different platforms that offer the chatbot messaging service and that users, once logged in, must be able to use the service independent of the chosen chat interface. For the first functionality, two alternatives were possible: write the code for each platform using the specific SDK or use a middleware, a software that adds a level of abstraction above different SDKs thus making the unified development of multiple applications possible under the same source code. Fortunately, thanks to the growing popularity of chatbots, there are a lot of good middleware called *Chatbot Builders* that can handle both conversational logic and the multi-channel aspect such as Dialogflow, wit.ai, Chatfuel [1] and others. To allow users authentication instead, we decided to use a login system, built in an

[1] https://chatfuel.com/

external website, that directly communicates with the bot backend and can be easily accessed from a link provided in the first message of the bot.
The second word, *Conversational Agent*, denotes the nature of the application. This aspect is covered in part by the chatbot builder for routing user requests to correct responses, and in part by a web server that builds the responses hosted on a machine property of Istituto Superiore Mario Boella [2] research center.
As for *Personalized Music Recommendations*, first we have to explain what *Personalized* means in this application context. The chatbot recommends music artists to users and those are chosen on a preference inferred by personality traits of each user. To compute such traits we had two possibilities, a direct and an indirect one. The former consists of subjecting users to a psychological assessment where they directly express behaviors while responding to test questions, while the latter, is to make use of an affective computing service to indirectly predict personality from users online behaviors. Affective computing systems are technologies that can recognize, interpret, process and simulate human affects (Picard 1997). As discusses in the Background section, there already are commercially available technologies that can be classified as affective computing systems and that are able to infer personality traits of individuals from their social network behaviors.
Analyzing the two possibilities, we can say that, given the fact that a psychological assessment requires at least 10 to 15 minutes to be completed, the affective computing software appears a more viable way for an entertainment product like a music recommender chatbot.
The last aspect that we had to take into consideration is *Music Recommendations*. Here, the obvious choice is to use a modern recommender system, the state-of-the-art approach to give recommendations (Singhal et al. 2017). As for data storage, we decided to use a MySQL database mainly for the ecosystem: MySQL has 40 years of existence and this means an enormous amount of already solved problems and useful information are available today on the Web. Another reason is that MySQL also supports JSON object storage, which is fundamental to interact with REST webservices.
To recap, the architecture of *Beat in a Bot* is made of different components: a chatbot builder to handle the conversational flow and the multi-channel aspect, a website to provide shared authentication between the different platforms and to show the results of the personality prediction, a recommender system to provide personalized recommendations, a MySQL database to store users and artists data, a web server to programmatically build responses, manage the database operations, execute the recommender algorithm, make requests to the external web services

[2] http://www.ismb.it/en

and to manage the authentication requests that come from the website. A visual representation of the high-level architecture is depicted in Figure 4.1.

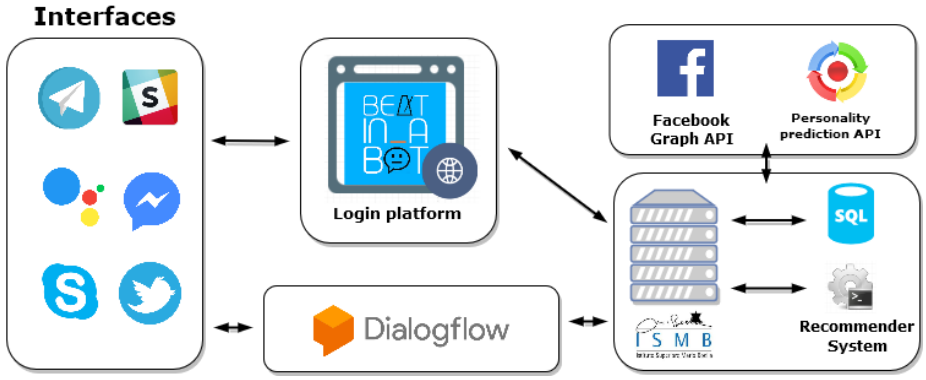


Figure 4.1. The system architecture

4.3 Building components

4.3.1 Dialogflow

Dialogflow [3], formerly known as *Api.ai*, is a developer of human-computer interaction technologies based on natural language conversations. The company, is backed by Google and runs on Google infrastructure, which means it can scale to millions of users thus making it suitable also for the enterprise environment. It allows developers to easily build conversational agents, powered by AI, that once designed can be easily deployed to a variety of platforms with a one-click integration. Companies like KLM, Domino's, Ticketmaster and others has built their conversation agent with Dialogflow. It supports more than 14 languages and it is available in two pricing options, a free standard edition and paid enterprise edition, making it so appealing for both companies and individual developers.

Fundamentals

Dialogflow basic concepts are: *Intents, Entities, Events, Contexts*, *Actions and Parameters*. In order to start a conversation with an agent, the user needs to invoke it. A user does this by asking to speak with the agent in a manner specified

[3] https://dialogflow.com

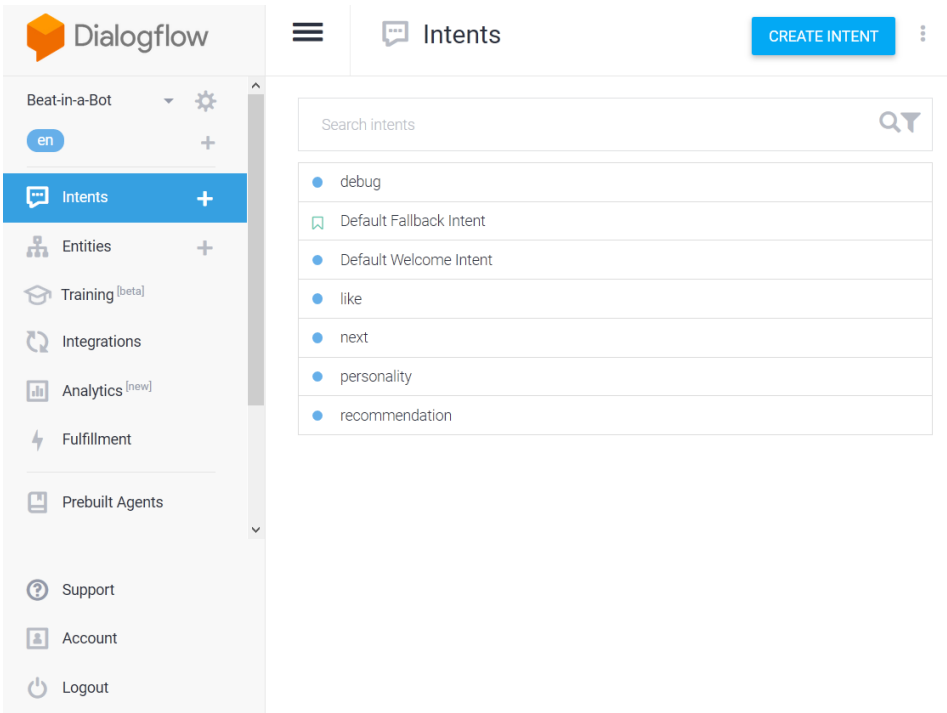


Figure 4.2. The Dialogflow console.

by the agent developer that can be an *Event* or a keyword that triggers an *Intent*. In Dialogflow, an intent houses elements and logic to parse information from users and answer to their questions. For the agent to understand the question, it needs examples of how the same question can be asked in different ways. Developers add these permutations to the *User Says* section of the intent console panel. The more variations are added to the intent, the better the agent will comprehend the user. Based on this data, Dialogflow builds a machine learning model (algorithm) for making decisions on which intent should be triggered by a user input and what data needs to be extracted. This algorithm is unique to each agent. The algorithm adjusts dynamically according to the changes made in the agent and in the Dialogflow platform. To make sure that the algorithm is improving, the agent needs to constantly be trained using real conversation logs.
Dialogflow agent needs to know what information is useful for answering users' requests. Those pieces of data are called Entities. Entities like time, date, and numbers are covered by system entities while entities like weather conditions or

seasonal clothing, needs to be defined by the developer so they can be recognized as an important part of the question.

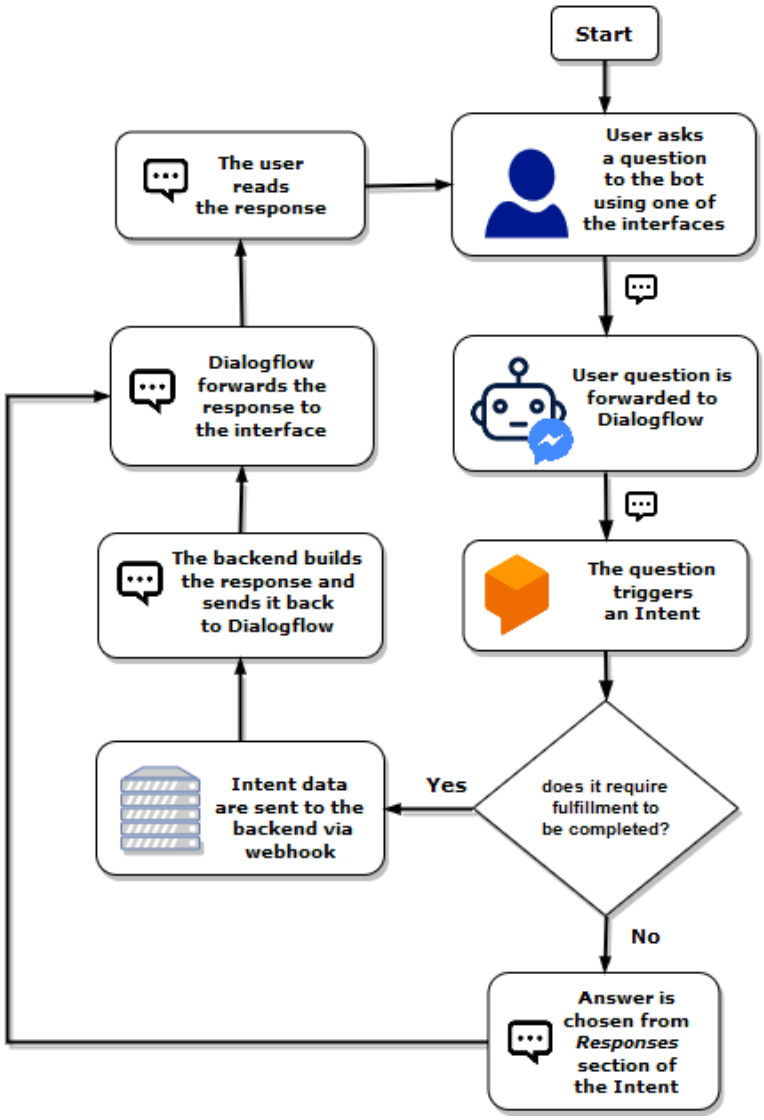


Figure 4.3. Dialogflow response logic

The agent can respond to user questions in two ways: with text responses directly provided in the *Text response* section of the intent console panel or by providing a *webhook* to an external webservice, which subsequently fetches the data needed, determines how it would like to respond and sends the response back to Dialogflow that forwards it to the chat interface (Figure 4.3).

What makes Dialogflow chatbots real conversational agents is the concept of *Contexts*. A Context can be used to remember something from one intent to another. For instance, if the first intent was triggered by "*What's the weather supposed to be like in San Francisco tomorrow?*", information can be saved in the form of *parameters*, which in this case can be, the *city* we would like to know the weather about and the time, *tomorrow*. The next intent can be *"How about the day after that?"* and because the agent knows previous information thanks to the context parameters, it can replies with the San Francisco forecast for the day after. Contexts are also used to repair a conversation that has been broken by a user or a system error, as well as branch conversations to different intents depending on the user's response.

Multi-channel

Dialogflow provides a one-click integration to deploy the conversational agent on many different messaging platforms. The process works as follows: when on *Integrations* section of the Dialogflow console, the developer selects the platform he wants to deploy on. Once selected, a window showing instructions on how to make the integration appears; after all is set in the correct way, the conversational agent starts to work on the platform. As example, if the user wants to deploy the agent on Telegram, he needs to start a conversation with the BotFather bot as stated in the Telegram documentation. BotFather provides an access token that has to be copied and pasted in the required box by Dialogflow. After that, the integration can be completed pressing the button *Start* that checks if the configuration is correct and makes the communication between Dialogflow and the platform active.

Rationale for the choice

We decided to use Dialogflow rather than other platforms because at the same time it is simple and powerful, reliable (thanks to the Google backend), and it has the possibility to easily deploy the conversational agent on many different platforms. This eliminates the need to re-write the code for each messaging platform making the developer work easier.

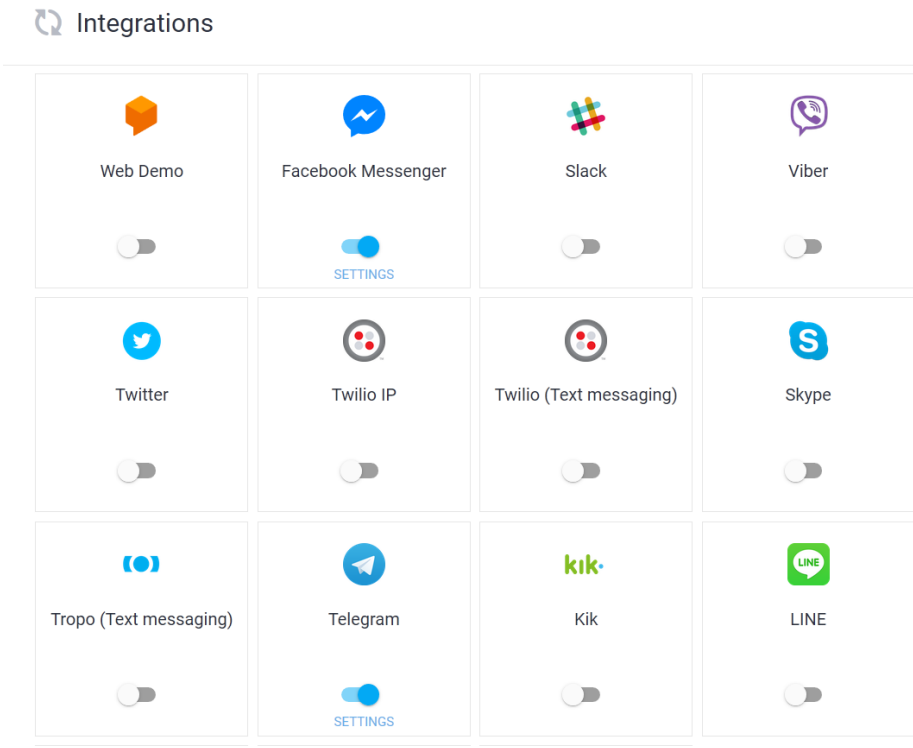


Figure 4.4. Integrations section of Dialogflow console.

4.3.2 MyMediaLite

MyMediaLite (Gantner et al. 2011) is a recommender system library for the Common Language Runtime (CLR, often called .NET). It is a free and open source software developed by Zeno Gantner, Steffen Rendle, Lucas Drumond, and Christoph Freudenthaler at University of Hildesheim. MyMediaLite can be used as an executable file or can be imported as a library in a personal project. It has many recommendation algorithms that can use collaborative and attribute/content data; since it is written in C# for the .NET platform, it can run on every architecture supported by Mono as Linux, Windows and Mac OS X. The core library of MyMediaLite, as the name says, is leaner and simpler and carries less overhead; it has a size around 150KB and does not require a database to work. MyMediaLite includes evaluation routines for rating prediction and item prediction; it can measure error metrics like Mean Absolute Error(MAE), Root Mean Squared Error

(RMSE) and precision metrics like prec@N, Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain (NDCG). Other additional features are:

- Serialization: save and reload recommender models
- Real-time incremental updates for many recommenders
- Attribute-based diversification of recommendation lists

MyMediaLite addresses the two most common scenarios in collaborative filtering:

- rating prediction (e.g. on a scale of 1 to 5 stars)
- item prediction from positive-only feedback (e.g. from clicks, likes, or purchase actions).

Most Popular and Item K-Nearest Neighbor algorithms

MyMediaLite offers many recommendation algorithms that can be easily selected specifying the command-line option `-recommender=METHOD`. Two algorithms in particular, the MostPopular and Item K-Nearest Neighbor, are used by the recommender system in the chatbot backend. One or the other is used depending on the decision-making process of the bot value chain discussed in the next chapter.

The MostPopular algorithm simply extracts from the user-item matrix the most popular items i.e. those that have the highest number of users' feedback. Results are given in descending order from the most popular item to the least popular one.

The Item K-Nearest Neighbor is a collaborative-filtering based algorithm. Given a target user and its positively (i.e., above a pre-defined threshold) rated items, the algorithm relies on the items' similarities for the formation of a neighborhood of nearest items. The choice of the K nearest neighbors for the neighborhood formation results in a trade-off: a very small K leads to few candidate items that can be recommended because there are not a lot of neighbors to support the predictions. In contrast, a very large K impacts precision as the particularities of user's preferences can be blunted due to the large neighborhood size. Usually, K is set to be in the range of values from 10 to 100, where the optimum K also depends on data characteristics such as sparsity.

The similarity can be measured with two different metrics, cosine similarity and Pearson correlation:

- *Cosine Similarity*: in the recommender system model, items are represented as vectors whose dimensions are the recommendations given by users. The

cosine of the angle between two vectors can be seen as a measure of how far they are. If the two vectors are aligned, it means that the two items are identical and the cosine value is 1. If they are perpendicular, the cosine is zero and the items are rather different. If they are aligned but opposite, the cosine is -1 and means that the two items are antithetical. The cosine similarity formula is the following:

$$CosSim(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} = \frac{\langle x, y \rangle}{\|x\| \, \|y\|} \tag{4.1}$$

where $x(x_1, x_2, ..., xi)$ is the vector of the first item and $y(y_1, y_2, ..., yi)$ the vector of the second item.

- *Pearson Correlation*: rather than considering the distance between item vectors as a similarity measure, we can consider the correlation between items. To compute the correlation between two items i and j, the first step is to isolate users who both rated i and j (the co-rated cases). The set of such users is denoted with U and the similarity between the two items is given by:

$$sim(i, j) = \frac{\sum_{u \epsilon U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \epsilon U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \epsilon U}(R_{u,j} - \bar{R}_j)^2}} \tag{4.2}$$

where $R_{u,i}$ denotes the rating of user u on item i , $\bar{R}_i$ is the average rating of the i -th item, same for item j.

The last step is to generate predictions. Once the set of most similar items are isolated, the next step is to look into the target users ratings and use a technique to obtain predictions. Two such techniques are considered, weighted sum and regression:

- *Weighted sum*: As the name implies, this method computes the prediction on an item i for a user u by computing the sum of the ratings given by the user on the items similar to i. Each rating is weighted by the corresponding similarity $s_{i,j}$ between the active item i and its similar items j. Predictions are computed with the following formula:

$$P_{u,i} = \frac{\sum_{allsimilaritems,N}(s_{i,N} * R_{u,N})}{\sum_{allsimilaritems,N}(|s_{i,N}|)} \tag{4.3}$$

Basically, this approach tries to capture how the active user rates the similar items. The weighted sum is scaled by the sum of the similarity terms to make sure the prediction is within the predefined range.

- *Regression*: This approach is similar to the weighted sum method but instead of directly using the ratings of similar items it uses an approximation of the ratings based on regression model. In practice, the similarities computed using cosine or correlation measures may be misleading in the sense that two rating vectors may be distant (in Euclidean sense) yet may have very high similarity. In that case using the raw ratings of the "so called" similar item may result in poor prediction. The basic idea is to use the same formula as the weighted sum technique, but instead of using the similar item "raw" rating values $R_{u,N}$'s, this model uses their approximated values $R'_{u,N}$ based on a linear regression model. If we denote the respective vectors of the target item i and the similar item N by R_i and R_N the linear regression model can be expressed as:

$$\bar{R'_N} = \alpha R'_i + \beta + \epsilon \tag{4.4}$$

  The regression model parameters α and β are determined by going over both of the rating vectors; ϵ is the error of the regression model.

The Cold Start problem

It is known that a recommender system in order to work properly needs a user history over which building recommendations. Suppose the case of a Collaborative Filtering algorithm: the system is based on similarity between users but how to deal with new users? New users are a problem for recommender systems because there are not enough data to define their profile so that similarity with other users (or between items) can be measured. This problem takes the name of the *Cold-start* problem because, as a car with a cold engine has troubles to start up, a recommender system algorithm does not do his best when started with few data. There are several solutions to the problem:

- *Popularity based strategy*: Global trends and popularity of products can be used as well as what is popular in the area of the user or in a particular hour of the day.

- *Contextual information*: Several information can be exploited from the user profile like his position, the website from witch he came, his operating system and his browser.

- *Use of a personality model*: Using a personality model like the Five Factor Model or the MBTI, a initial profile of the user can be created on his personality.

- *Community information*: Data can be extracted from a community the user belongs to. As instance, information about a user gathered from a recommender system that works on Amazon can be used for another that operates on eBay.

- *Decision tree*: a decision tree classification model can be used. The tree is build on the properties of the users that already are in the dataset.

The approach we take to solve the problem is the use of a personality model. Since the application requires the prediction of the user personality, we can leverage this information to build some initial recommendations. The idea is to start with the most popular artists of the musical genres that are bounded with users personality profiles as described by a study[4] by NERIS Analytics. Such study has found a correlation between Myers-Briggs Type Indicators of personality and music preferences. Details will be discusses in the Implementation section.

Rationale for the choice

We decided to use MyMediaLite as a recommender system because it is a ready-to-use package making it very easy to integrate in the solution. MyMediaLite it is also used by researchers of Istituto Superiore Mario Boella, the research center where the thesis was born, who strongly support usage of open-source software. This gave us the added value of having a live community support to the usage of the software.

4.3.3 Apply Magic Sauce

Apply Magic Sauce is an application developed by The Psychometrics Centre, a Strategic Research Network of the University of Cambridge, center of excellence in psychological, occupational, clinical and educational assessment. It is released as *Software as a Service (SaaS)* paradigm that includes a web application and a RESTful Web service. Apply Magic Sauce is powered by an artificial intelligence algorithm and its model is trained on over 6 million social media profiles and matching scores on psychometric tests. It takes as input a *digital footprint* that can be a list of Facebook pages IDs, Facebook statuses, tweets, user browsing data, open text and other and gives as output an individual profile containing psychographics and demographics.

[4] https://www.16personalities.com/articles/music-preferences-by-personality-type

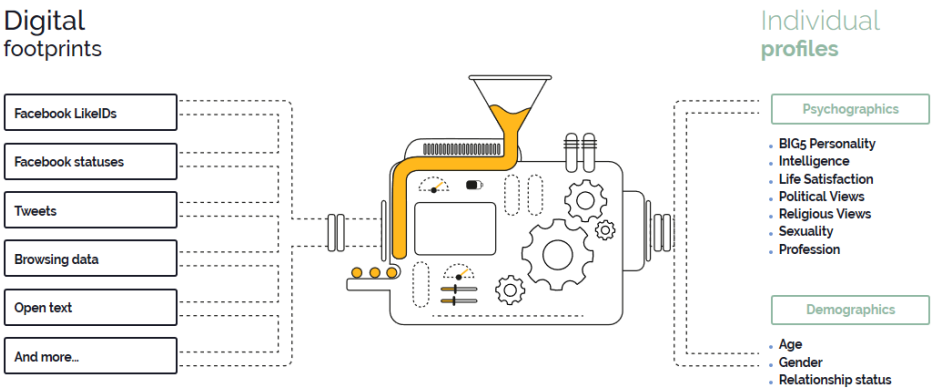


Figure 4.5. The Apply Magic Sauce Trait Prediction Engine. Image taken from https://applymagicsauce.com/about_us.html

Machine-learning algorithms are very useful to find patterns in data but often those data can be difficult to interpret. Apply Magic Sauce goal is to add psychological data points to any sample to help interpret machine-learning algorithms results: these algorithms are at risk of perpetuating historical prejudice, being overly domain-specific or prescribing norms that can feel impersonal. Apply Magic Sauce API enables conversation between businesses and consumers about how predictive technologies ought to be used. This could help algorithms explain themselves and learn from their mistakes, just like humans. Once predicted, psychological data points can be added to any sample data our devices records, making them a valuable resource that can be used to tailor online experience on the different personalities.

Rationale for the choice

As already discussed, to predict users' personality profiles we had to choose between subjecting users to a psychological test or use an affective computing system like Apply Magic Sauce. While the first choice has the advantage of not having any requirement, we had to reason about the best way to implement it in a chatbot application. One solution is to use the chatbot itself to submit the test questions to the users, gather the answers and compute the result while another can be, redirect users to an external resource where they can take the test and when finished, ask them to write the result back to the chatbot. The first case requires the construction of a psychological assessment that is beyond the objective of this

thesis, while the second case is considered bad practice in a chatbot implementation. However, both solutions will require a lot of time to be accomplished since in its shorter version, the Form M, the MBTI questionnaire is made of 93 items. The second choice, Apply Magic Sauce, is only feasible if the user has a enough "digital footprints" like a Facebook account with a sufficient number of likes or a Twitter account with enough tweets for the service to work. In the other end, it is fast and reliable (it depends on the "quality" of footprints). The approach chosen was Apply Magic Sauce for the speed of the process, taking into account the fact that users tend to abandon apps if they require a tedious process to start up (Jain 2016).

4.3.4 Facebook Graph API

On April 21, 2010, during the F8 Live event, Facebook presented a series of news about the way to develop applications for the social network. The most important were the adoption of the OAuth 2.0 authentication standard, three new SDK, for Javascript, Python and PHP and a new API called *Graph API*. Graph API is described as "the primary way for apps to read and write to the Facebook social graph"[5]. Technically speaking it is a low-level HTTP-based API that developers can use to programmatically query data, post new stories, manage ads, upload photos, and perform a variety of other tasks that an app might implement. The Facebook Graph API is based on the concept of *graphs*. Graphs are mathematical structures, composed by elements called *nodes* linked together by *edges*. Those structures extend their field of application way beyond mathematics, reaching computer science and sociology. The idea behind the Graph API is to consider elements like users, pages, photos, videos, comments and others as nodes of the graph (called *objects* in Facebook documentation) and connections between them as *edges*. There are also *fields* that represent information about objects. For instance, *friend* connection links together *user* nodes and *likes* connections links a user to the pages he likes. The Graph API is HTTP based so it can be used with any language that has an HTTP library. Each node has an unique ID that is used to access it via the Graph API. The following example shows how to make a request to the graph:

```
GET graph.facebook.com
    /{node-id}
```

Responses are always formatted as JSON objects. The response of a request made with Politecnico di Torino page-id, has the following format:

[5] https://developers.facebook.com/docs/graph-api

```
{
  "about": "Pagina ufficiale del PoliTo su Facebook",
  "name": "Politecnico di Torino",
  "fan_count": 60254,
  "picture": {
    "data": {
      "height": 50,
      "is_silhouette": false,
      "url": "https://scontent.xx.fbcdn.net/v/p50x50/polito.jpg
      "width": 50
    }
  },
  "id": "17570259916"
}
```


To request for edges instead, the syntax is:

```
GET graph.facebook.com
    /{node-id}/{edge-name}
```

It is also possible to push by making HTTP POST requests and to delete using HTTP DELETE requests to the same endpoints.

Rationale for the choice

The choice of using this tools is linked to the usage of Apply Magic Sauce web service. To predict users personality profiles, the list of page IDs that the user liked is used as his digital footprints. The only way to programmatically retrieve such list is to use the Facebook Graph API.

4.3.5 Website and login platform

The support website has two main purposes in the architecture: it allows chatbot users authentication and communicates with the server to start the prediction of user's personality profile. The authentication system is implemented using Facebook Login[6] while the communication with the server occurs through an HTTP GET request to the backend URL. Facebook, having joined the OAuth 2.0 standard for user authentication in 2010, provides a login system called *Facebook Login* that can be easily integrated into a web page with the Facebook JavaScript SDK.

[6] https://developers.facebook.com/docs/facebook-login/web

Facebook Login enables people to sign into a website using their Facebook credentials.
The access to the site is via the chatbot. The first time a user starts a conversation with the bot receives the website URL in the onboarding message. The page is structured as a to-do list with two entries: first it is asked to log in with Facebook Login and then to press a *Start* button. The first action allows to retrieve the user's Facebook username, ID and Access Token while the second action, which occurs when the user presses the *Start* button, sends those data to the bot backend. The server receives the data, starts the personality prediction and return the results back to the caller (the website). As consequence, the website shows information on the user's personality such as a brief description of his Myers-Briggs Type Indicator (taken from the official website of the Myers-Briggs Foundation[7]), the Facebook pages that has determined the four dimensions of the MBTI and a list of music artists who have the same personality as the user [8]. Finally, a *Back to the bot* button appears on the screen. Such button, automatically redirect users to the platform on which the bot has been started allowing conversation to continue.

Rationale for the choice

The choice of developing a supporting website for the application is essentially bound to two previous architectural choices: first one, the multi-channel support, which automatically implies to have a flexible authentication method that can be used independently of the messaging platform chosen to interact with the bot. This is necessary because every single platform has its own login system and it is up to developers to manage this aspect. The other, is a direct consequence of the choice to use Apply Magic Sauce web service, which requires data from the user's Facebook profile to make the personality prediction. Since Apply Magic Sauce also accept user's Tweets as input, a possible extension for the website could be to allow users authenticate with Twitter also. This soften the actual pre-requisites of the application, allowing more people to use the service. As for the website hosting, we chose GitHub Pages[9], a hosting service offered by GitHub. It is free and it allows developers to host directly from GitHub repositories, having all the benefits of the git versioning platform like commit, push and pull to easily maintain and update the website.

[7] http://www.myersbriggs.org

[8] http://intuitivemusician.com/

[9] https://pages.github.com/

4.3.6 Web server

A web server is a software application that runs on a computer called *server* and is in charge of handling requests from a *client*, typically a web browser. It uses the HTTP (*HyperText Transfer Protocol*) protocol, the basic network protocol on the World Wide Web. Main purpose of a web server is to process and deliver web pages to clients but they can also run operations written in a server-side scripting language as ASP, PHP, Python or Node.js. This allows to deliver dynamic web pages and to retrieve and modify information from databases. In my application context, the web server has four main purposes:

1. Act as webhook for the Dialogflow *Fulfillment* feature.
2. Communicate with Facebook Graph API.
3. Communicate with Apply Magic Sauce API.
4. Manage the database operations.

Webhook

Webhooks are described as "user-defined HTTP callbacks" (Fitz 2009) i.e. user defined callbacks made with HTTP POST. To support webhooks in a platform such a website, the developer has to allow users specify a URL where the application will post and on what events. For instance, when an event occurs such as pushing code to a repository or commenting a post on a blog, the source site makes a HTTP request to the URL configured as a webhook. The main reason behind webhooks is to augment or alter the behavior of a web page or a web application. Dialogflow, the chatbot builder chosen as a middleware for this project, allows developers to define a webhook URL in the *Fulfillment* section of the agent builder console. When an intent configured to reply using webhook is triggered, a RESTful POST request is sent to the URL previously specified. When this happens, Dialogflow sends to the webhook a properly formatted JSON object, that includes information about the triggered Intent, the original request, the conversation contexts and other metadata about the session and the user. When the web server receives the request, it parses the JSON and replies accordingly to the data received as discusses in the Dialogflow section.

Facebook Graph API

The first time a user interacts to the chatbot, a link to a website is presented to him; once on the website, the user has to first authenticate with Facebook Login and then, he has to press a button that starts the computation of his personality profile. An app, which implements Facebook Login as the website, is able to

obtain an access token that provides temporary, secure access to Facebook APIs. An access token is an opaque string that identifies a user, an app, or a Page and can be used by the app to make graph API calls. Basically, with an access token, information on a node (a user, a page or other) can be retrieved from Facebook. So, what the website does, is to first obtain the user access token and then send it to the web server to make him able to retrieve the list of user's likes to Facebook pages from the Facebook Graph.

Apply Magic Sauce API

Once obtained the list of user's likes to Facebook pages, it can be sent to the affective computing service *Apply Magic Sauce* that exposes a so-called "Prediction API" where information about user psychological traits can be retrieved from his "digital footprints". In this case, the web server sends the list of likes as input to the API (the digital footprints) and the Big5 personality traits together with the user's Myers-Briggs Type Indicator are returned as output (the psychological traits). The returned data is structured so that each Big5 trait is associated with a list of IDs of Facebook paged described as "positive" or "negative". Such distinction helps identify what pages had determined or excluded the related trait. Those data are sent back to the website as a result of the computation started when the *Start* button in pressed.

Database operations

The web server is in charge of establishing a connection with the MySQL database to fetch and store the data needed for the application. It has to fill and keep updated the users table, extract the recommended artists information from the ids given by the recommender system and keep track of the recommendations for each user. Moreover, it stores the JSON objects gained from *Apply Magic Sauce* to show users how the pages they liked on Facebook relates with the four dimensions of their personality, it checks if a user is already signed in the application, it stores the likes to recommended artists given by users on the chat and finally it manage all the "next recommendation" requests.

Rationale for the choice

We chose to add and develop a web server in the application architecture to implement and expand the functionalities offered by the chatbot. The web server is vital for the majority of the operations (described in the sections above) and without it, only a small amount of possibilities are available i.e. those offered by the chatbot builder alone. We chose Python as the main programming language for the server, mainly for the large variety of packages available, its compatibility

with all the other software used and the support of the huge community on the web. Python is also a preferred language between researchers at Istituto Superiore Mario Boella (ISMB). As for the hosting, the python script runs on a ISMB machine under Flask, a very popular micro web framework that allows to easily implement a web server with very few lines of code. The URL mapping instead, is managed by Apache, the most commonly used HTTP server over the net. The web server is reachable at the following URL:

`http://datascience.ismb.it/beatinabot`

and exposes the following endpoints:

Endpoint	**Method**	**Description**
/dialogflow	POST	It is the webhook endpoint where Dialogflow fulfillment requests are sent. Responses are JSON objects formatted as required by Dialogflow.
/login	GET	Login requests from the website are sent to this endpoint. The server replies with the JSON object from Apply Magic Sauce.

4.3.7 Database

A fundamental piece of the developed architecture (and every other web application) is the database. A database is a collection of information that is organized to be easily accessed, managed and updated. In a relational database, where information is modeled as relations between sets of objects, data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it.

There are several types of databases available today and, depending on the nature of the application and of course the developer preferences, one can be preferred over another to better fit the project needs. The database chosen for this chatbot application is a relational database called *MySQL*. MySQL is a Relational Database Management System (DBMS) made of a command line client and a server; both are available on Unix and Windows systems and it is released as an open source software. It supports a large variety of programming languages like Java, Mono, .NET, PHP, Python and many others. In the context of the application, the database contains data about artists, users and their personal recommendations.

Entity-Relationship model

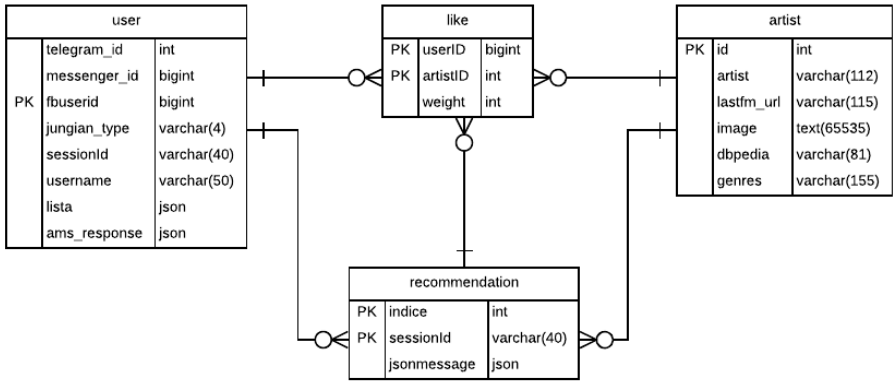


Figure 4.6. Beat in a Bot database ER model

Beat in a Bot database model is made of 4 entities (user, like, recommendation, artist) and 5 relations.
The *user* entity models information on a chatbot user and has 8 attributes:

- *telegram_id*: a unique id assigned by Telegram to the user. Retrieved when the bot is used on Telegram
- *messenger_id*: a unique id assigned by Messenger to the user. Retrieved when the bot is used on Facebook Messenger
- *fbuserid*: a unique id assigned by Facebook to the user. Retrieved after the Facebook Login. It is the table primary key.
- *jungian_type*: user's Myers-Briggs Type Indicator returned by Apply Magic Sauce prediction API
- *sessionId*: a unique id assigned by Dialogflow to the current chat session. Expires every 10 minutes
- *username*: user's Facebook username
- *lista*: JSON object containing the list of recommendations' IDs for the user
- *ams_response*: JSON object containing the result of the personality prediction by Apply Magic Sauce

The *like* entity represents a user feedback on a recommendation. It has 3 attributes:

- *userID*: foreign key that refers to the *fbuserid* attribute of the table *user*. Combined with *artistID* is the primary key for the table
- *artistID*: foreign key that refers to the *id* attribute of the table *artist*
- *weight*: the feedback count of the user to the artist

The *recommendation* entity models an artist recommendation. It has 3 attributes:

- *indice*: an integer number used as the recommendation index. Indices inside the attribute *lista* of the table *user* refers to this attribute.
- *sessionId*: foreign key that refers to the *sessionId* attribute of the table *user*
- *jsonmessage*: the recommendation itself. It is a JSON object properly structured to be compatible with Telegram and Messenger that includes a card with the artist image and the buttons *I like it* and *next*.

The *artist* entity represents an artist in the database. It contains the gold standard discussed in the next section. It has 6 attributes:

- *id*: the artist id
- *artist*: the artist name
- *lastfm_url*: URL to the artist's Last.fm profile
- *image*: artist's image URL
- *dbpedia*: pointer to the artist's DBpedia page
- *genres*: the list of artist's music genres

The *user* entity relates with the *like* entity in a one to many relation: a user can express zero or more likes while a like is express by one and only one user. *User* is also in a one to many relation with the *Recommendation* entity. This means that a recommendation can be assigned to only one user while a user can receive zero or many recommendations.
The *like* entity is in a many to one relation with *recommendation* and with the entity *artist*. A feedback (as a single entity) can be given to only one recommendation while a recommendation can receive zero or many feedback. The same logic applies to the *artist-like* relation.
Finally, an *artist* can be included in many recommendations while a single recommendation only includes one artist. So, the relation between *artist* and *recommendations* is one to many.

Dataset used

The database has been initialized with data from *hetrec2011-lastfm-2k* (Cantador et al. 2011), a gold standard built by Ignacio Fernández-Tobías with the collaboration of Iván Cantador and Alejandro Bellogín, members of the Information Retrieval group at Universidad Autonoma de Madrid. It contains social networking, tagging, and music artist listening information from a set of 2K users from *Last.fm*[10] online music system. The dataset has been enriched with the image URL of each artist retrieved with the usage of Last.fm API.
When the users of the chatbot application give their feedback on recommendations, those are added to dataset and used to build the models for the recommender system as described in the *Implementation* section. Statistics of the dataset are reported in the table 4.1.

Statistic	**Value**
Users	1892
Artists	17632
Bi-directional user-friend relations	12717
(user_i, user_j) pairs	25434
Avg. friend relations per user	13443
User-listened artist relations	92834
Avg. artists most listened by each user	49067
Avg. users who listened each artist	5265
Tags	11946
Tag assignments (tas)	186479
Avg. tas per user	98562
Avg. tas per artist	14891
Avg. distinct tags used by each user	18930
Avg. distinct tags used for each artist	8764

Table 4.1. Dataset statistics

4.3.8 Music Preferences by Personality Type

The connection between music preferences and personality is supported by several studies in the psychology literature. As example, (Daoussis & Mckelvie 1986, Litle & Zuckerman 1986, McCown et al. 1997) has found that sensation seeking, extraversion and psychoticism predicted liking for more stimulating music like rock-and-roll,

[10] https://www.last.fm

or “exaggerated bass” in music. Pearson et al. found that extraverts likes more types of music than introverts, and in particular they prefer popular/rockmusic. They also found that Feeling people tend to like country and western music more than those oriented toward Thinking (Pearson & Dollinger 2004).
In the context of this thesis application, such connection is used to solve the cold start problem of the recommender system: data about users are enhanced with music preferences obtained from their previously predicted personality type. Recommender algorithms that use previously enhanced data are known as Content-boosted Collaborative Filtering and studies has proven that such algorithms achieve better results than pure collaborative filtering, pure content-based and naive hybrid approach (Melville et al. 2002).

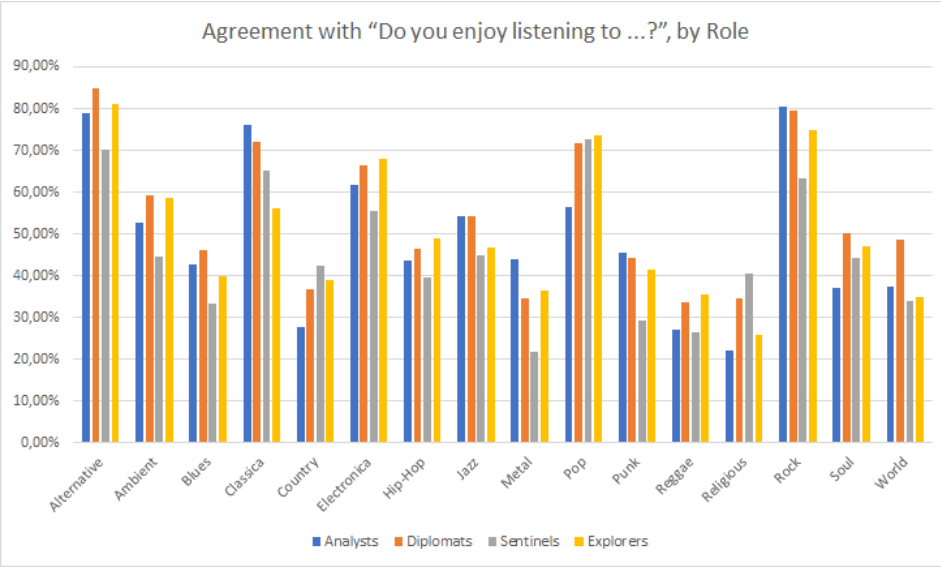


Figure 4.7. Agreement with “Do you enjoy listening to ...?”, by Role

The enhancing data used has been derived from a survey on musical preferences, made by an English company named NERIS Analytics Limited. NERIS Analytics Limited owns a website[11] called *16Personalities.com* on which they publish articles and studies about personality types and their impact on our lives, geographical distribution, social attitudes, relationships and more. Their model incorporates

[11] https://www.16personalities.com

the latest advances in psychometric research, combining time-tested concepts with robust and highly accurate testing techniques. The framework used by NERIS Analytics is based on the Myers-Briggs Type Indicator and the 16 personality described are grouped into four groups called *Roles*: Analysts (INTJ, INTP, ENTJ, ENTP), Diplomats (INFJ, INFP, ENFJ, ENFP), Sentinels (ISTJ, ISFJ, ESTJ, ESFJ) and Explorers (ISTP, ISFP, ESTP, ESFP). The survey was made on over 4000 respondents whose personality profile were already known. Questions was of the type: "*Do you enjoy listen to <music genre>?*". The survey has revealed the outlooks different personalities have on music. Results are presented in figure 4.7, which shows the percentage of agreement to the question "Do you enjoy listen to...?" of the four groups. Such results has been reported into a table in order to better classify Roles and their musical preferences. This allowed to easily link each Myers-Briggs Type Indicator predicted by *Apply Magic Sauce* with the relative vector of preferred musical genres.

	Analysts	**Diplomats**	**Sentinels**	**Explorers**
Alternative	78,88%	84,87%	70,22%	81,20%
Ambient	52,52%	59,30%	44,63%	58,47%
Blues	42,60%	46,08%	33,33%	39,92%
Classica	76,06%	71,95%	65,12%	56,05%
Country	27,74%	36,76%	42,51%	38,89%
Electronica	61,62%	66,34%	55,43%	67,86%
Hip-Hop	43,47%	46,32%	39,56%	49,00%
Jazz	54,36%	54,37%	44,93%	46,83%
Metal	43,83%	34,66%	21,64%	36,29%
Pop	56,34%	71,70%	72,55%	73,52%
Punk	45,53%	44,36%	29,16%	41,30%
Reggae	26,98%	33,50%	26,58%	35,46%
Religious	21,96%	34,41%	40,44%	25,70%
Rock	80,45%	79,63%	63,22%	74,90%
Soul	37,08%	50,30%	44,38%	46,99%
World	37,38%	48,63%	34,06%	34,94%

Following are the python arrays ordered by descending percentage of agreement, as they are declared within the server source code:

```
Analysts  = ['Rock','Alternative','Classica','Electronica','Pop','Jazz',
            'Ambient','Punk','Metal',Hip-Hop','Blues','World','Reggae'
            ,'Religious']

Diplomats = ['Alternative','Rock','Classica','Pop','Electronica','Ambient',
            'Jazz','Soul','World','Hip-Hop','Blues','Punk','Country',
```

```
            'Metal','Religious',Reggae']

Sentinels = ['Pop','Alternative','Classica','Rock','Electronica','Jazz',
            'Ambient','Soul','Country','Religious','Hip-Hop','World',
            'Blues','Punk',Reggae','Metal']

Explorers = ['Alternative','Rock','Pop','Electronica','Ambient',
            'Classica','Hip-Hop','Soul','Jazz','Punk','Blues',
            'Country','Metal','Reggae','World,'Religious']
```

Chapter 5

Implementation

5.1 Introduction

This chapter contains the details of how the application works from a technical perspective. The description follows the flow of a typical user experience.

5.2 Initial phase

Users can interact with the bot in two different ways: by typing his name in the search field of the supported messaging platforms or by following a link that redirects them to the chat. All the platforms share a common paradigm to start a conversation that consists of sending a welcome event to Dialogflow. Such event is a RESTful POST request to Dialogflow webhook that embeds a properly formatted JSON. The request triggers the entry point Intent called *Default Welcome Intent*; such Intent is configured so that the response is not directly managed by Dialogflow but instead, is delegated to the web server (the bot backend) which pointer is configured as the default Dialogflow webhook for fulfillment option. The backend receives from Dialogflow a JSON object that contains:

- Information on the triggered Intent (*Default Welcome Intent*)
- The resolved query
- The chat session ID
- The timestamp
- The language used

- The status of the HTTP response
- The original request (The JSON response as sent by the chat platform)

Once received, it acts according to the triggered intent read from the JSON, in this case, the Default Welcome Intent. In particular, it checks if the user has already used the application before. The check is performed looking for a match between the user's platform id (parsed from the JSON) and those already in the database. In case of a positive outcome, the user is recognized and the message *Happy to see you again <username>! To start, press the "Continue" button below!* is displayed in the chat. Otherwise, the bot shows the onboarding message with the authentication instructions. The picture below illustrates the logic of the onboarding phase in the case of a new user.

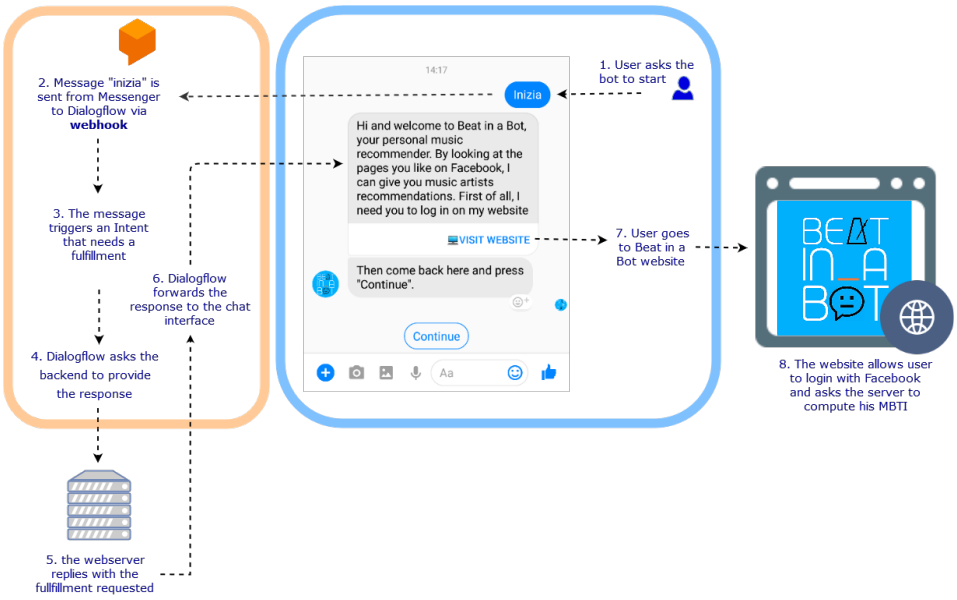


Figure 5.1. The initial workflow.

5.3 Authentication

Before moving on, we have to explain how users' authentication has been implemented. Users are identified with three different identifiers, the id assigned by the chat platform, the session id assigned by Dialogflow and the Facebook user id

taken from Facebook Login on the external website. The reason of using such set of identifiers is due to a lack of an integrated and shared way among all platforms to authenticate a user. Each of the three id has its own purpose: the session id, contained into the JSON from Dialogflow to the web server, is active for 10 minutes and it is used to recognize a user among all the different requests in the same chat session; the platform id, also embedded into the JSON object, is used to identify the user on a specific platform and since it does not expire like the session id, it can be used as his main identifier (the one that makes possible to recognize him at the first interaction with the bot). The last id, returned when the user logs in with Facebook Login, is always unique regardless of the chat platform. This uniqueness is due to the fact that the login operation is performed outside the chat. The Facebook user id, being independent of the chat platforms and Dialogflow, is used to associate all the others to the same user so that he can be recognized among all the chat platforms.
While some platforms provide a webview as an integrated messaging chat element but others do not, to maintain compatibility with as much platforms as possible, the access to the login website is provided with a button within the chat that embeds a customized website URL for each client. The link is built using the user's session id and a pointer to the bot, on the platform it has been opened, as URL parameters. The session id is simply sent back to the server after the Facebook Login phase and has the purpose of linking the external website session to a specific user, while the pointer to the bot is needed to provide a back button after a successful login. As example, if the bot is opened with Telegram, the website URL is built as follows:

```
https://giulioverazzo.github.io/beatinabot_website/
?sessionId=<sessionId>&bot_url=https://t.me/beatinabot
```

5.4 Website implementation

5.4.1 Technical aspects

The supporting website is developed using modern technologies for web developing like HTML5, CSS3, JavaScript and the jQuery library; rationale of choice is that these technologies are largely used by the web developers community and this represents an added value in terms of technical support. The site is built upon a template by HTML5 UP[1] that provides free, customizable and fully responsive templates for the web. This allows to develop websites that easily adapts to the screen form factors of smartphones and tablets as well as desktop and notebooks.

[1] https://html5up.net/

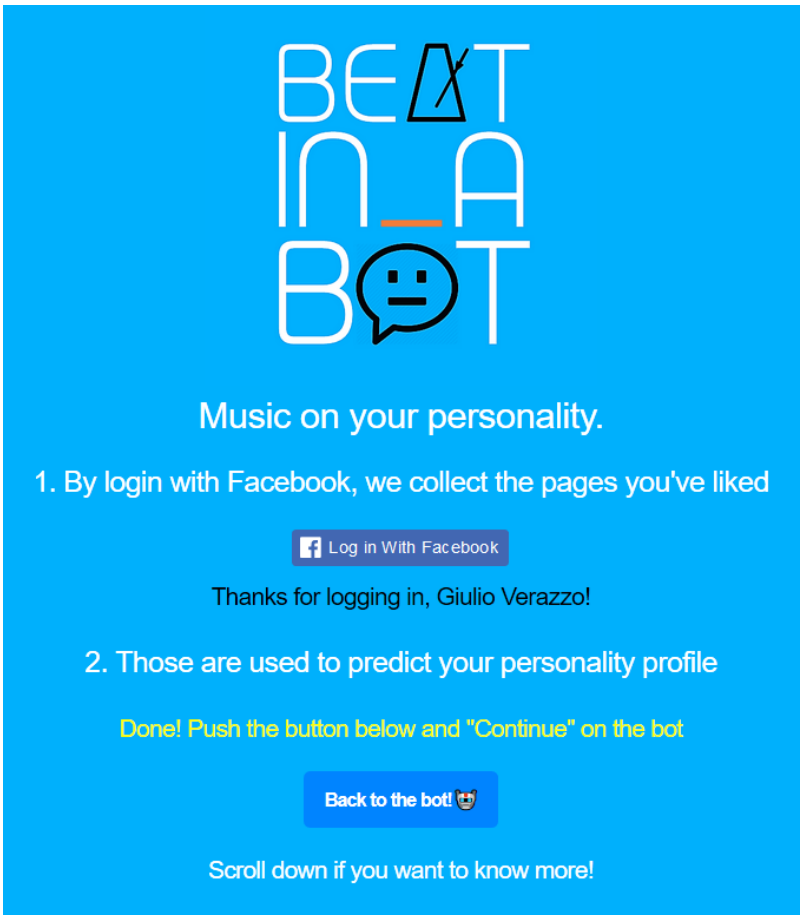


Figure 5.2. The supporting website.

As for the hosting, we chose GitHub Pages, a service that gives the opportunity to host directly from the GitHub repository for free. From what regards developing details, we used Facebook JavaScript SDK to implement Facebook Login authentication system and the jQuery library to make the asynchronous GET request to the server.

5.4.2 Working logic

When user clicks on the website button provided in the chat, a new window is opened and the page layout appears on the screen. The website is structured

as a list of instructions to follow in order to log in and start the prediction of user's personality type. The login functionality is implemented with Facebook Javascript SDK, which requires a registration on the Facebook Developers portal. The registration is free and allows developers to create apps that can benefit from Facebook services; when an app is created, an app id is assigned to it. Such id must be inserted into the appropriate field of the SDK used, in this case, the Javascript SDK. The SDK, asynchronously loaded inside the HTML page, provides a class called *FB* that has a method, *FB.init()*, which takes the app id as a parameter. After the SDK initialization, a *Log in with Facebook* button can be added to the page. The SDK automatically handles the login system and the user's login status. When the login button is pressed, the SDK handles it and a pop up window shows up asking for user's Facebook credentials. If provided credentials are correct, the user needs to grant the application with the permission to read his data such as the email, the public profile, the list of friends and the list of likes to Facebook pages. Such permission requests have to be properly set in the Facebook Developer console of the application. While the first three are approved by Facebook automatically as a predefined option, the permission to access user's likes to Facebook pages needs a special approval process. Developers must submit the app to a verification process that usually requires two days to be completed: if the app has all the requirements and passes Facebook internal tests, the *user_likes* permission can finally be asked to users.

As login confirmation, the website receives from Facebook an access token that can be used to retrieve user's information from the Facebook Graph. The token, together with the Facebook user ID, his username and the Dialogflow session id, is sent to the backend when user press the *Start!* button on the website that makes the following GET request:

```
GET http://datascience.ismb.it/beatinabot/hello
?userid=<Facebook-user-id>
&accessToken=<Facebook-user-access-token>
&name=<Facebook-username>
&sessionId=<chat-session-id>
```

At this point, the server receives the request and proceeds to check if the user has ever used the chatbot before: if so, it directly replies with the JSON object taken from Apply Magic Sauce while if not, using the access token, it first sends a request to the Facebook Graph to retrieve the list of user's likes to pages and then, sends it to Apply Magic Sauce in the body of the successive POST request.

```
GET https://graph.facebook.com/v2.11/<fbuserid>/likes?
access_token=<Facebook-user-access-token>

POST https://api.applymagicsauce.com/like_ids?
```

```
traits=BIG5&interpretations=true&contributors=true
```

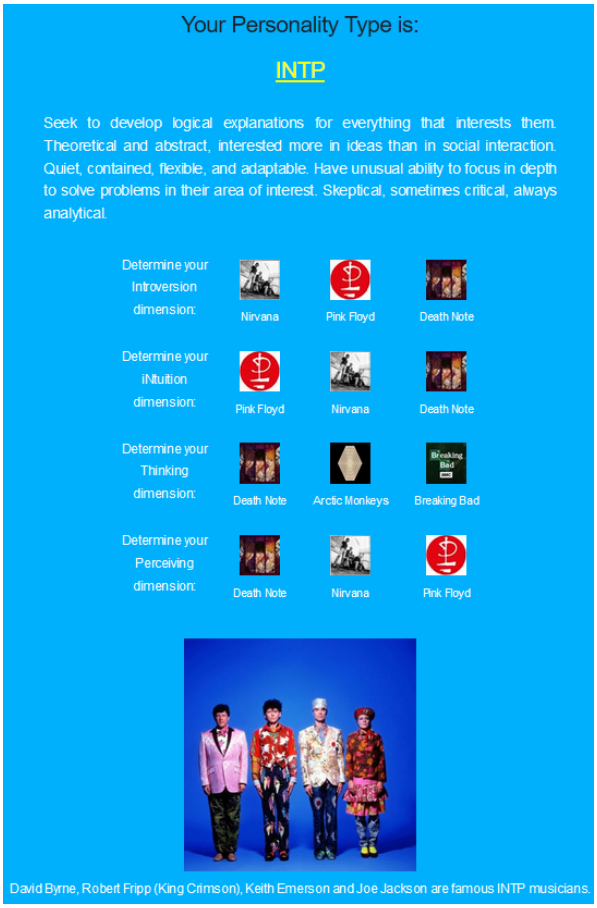


Figure 5.3. Additional information on user's personality.

The response received is a JSON object that contains values expressed in percentile of the Big5 factors, the user's Myers-Briggs Type Indicator and the sets of pages that has determined, in a positive or negative way, each value of the Big5 traits. The JSON object, together with user's MBTI, Facebook id, username and the session id are stored to the users table of the database and finally the backend replies to the original website request, with the JSON object from Apply Magic Sauce. When the website receives the response, the *Start!* button disappear and a *Back to the Bot!* button takes his place; such button has the purpose of

creating a natural flow that guides the user from the website back to the chatbot after the login phase. Its implementation depends on the chat platform used to talk with the bot. The website sets the logic of the button checking the parameter `bot_url` of its URL: if it is a Telegram URL, the button just points to `http://t.me/beatinabot`. If it is a Messenger URL instead, the button calls the method `requestCloseBrowser()` of the Facebook Messenger SDK that closes the webview.

After setting the back button, the website parses the JSON object received and calls two methods: `JungianTypeResult(mbti)` and `showLikes(obj, mbti)`. The former takes the Myers-Briggs Type Indicator read from the JSON and populates an empty DIV with a brief description of the user's personality type taken from the official Myers-Briggs Foundation website and a picture of one of four famous musician with the same personality type as the user. The latter converts the Big5 dimensions to MBTI dimensions and builds a table, which rows, are populated with the pictures of the Facebook pages that has determined the four dimensions of user's personality.

5.4.3 MBTI - Big5 Conversion

To show users what Facebook pages have determined the four dimensions of their Myers-Briggs Type Indicator, a conversion to the Myers-Briggs domain of the Big5 index values returned by Apply Magic Sauce is required. Such conversion is based on a study (McCrae & Costa 1989) that had the purpose of reinterpreting the Myers-Briggs Type Indicator from the perspective of the Five-Factor Model of personality. The study was conducted by submitting a sample of 267 men and 201 women, ages 19 to 93, to both the NEO-PI (the standard Big5 inventory) and the MBTI inventory with the aim of finding a correlation between the indices of the two models. The correlational analyses showed that the four MBTI indices did measure aspects of four of the five major dimensions of normal personality: in particular, a negative correlation was found between the Big5 Extraversion factor and the Extraversion-Intraversion factor of MBTI scale, same for Big5 Conscientiousness and the Judging-Perceiving factor, while a positive correlation showed up between Big5 Openness and the MBTI Sensing-iNtuition and between the Big5 Agreeableness and the Thinking-Feeling factor of MBTI. As for the Neuroticism factor, no significant correlation result was found.

The actual conversion is done by a JavaScript method inside the website. For each of the Big5 traits, a switch-case statement selects the single traits and stores in a vector the list of "positive" or "negative" pages depending of the first letter of user's MBTI.

Table 3
Correlations of Self-Reported NEO-PI Factors With MBTI Continuous Scales in Men and Women

MBTI scales	NEO-PI factor N	E	O	A	C
Men					
EI (Introversion)	16**	– 74***	03	– 03	08
SN (Intuition)	– 06	10	72***	04	– 15*
TF (Feeling)	06	19**	02	44***	– 15*
JP (Perception)	11	15*	30***	– 06	– 49***
Women					
EI (Introversion)	17*	– 69***	– 03	– 08	08
SN (Intuition)	01	22**	69***	03	– 10
TF (Feeling)	28***	10	– 02	46***	– 22**
JP (Perception)	04	20**	26***	05	– 46***

Note $N = 267$ for men, 201 for women NEO-PI = NEO Personality Inventory, MBTI = Myers-Briggs Type Indicator N = Neuroticism, E = Extraversion, O = Openness to Experience, A = Agreeableness, C = Conscientiousness
$^{*}p < 05$
$^{**}p < 01$
$^{***}p < 001$

Figure 5.4. Correlations of Self-Reported NEO-PI Factors With MBTI Continuous Scales in Men and Women (McCrae & Costa 1989)

```
for (var c of contrib){
    switch(c.trait){
        case ‘‘BIG5_Extraversion’’:

            if(mbti[0] === ‘‘I’’){
                first = c.negative;
                p1 = ‘‘Introversion’’;
            }else{
                first = c.positive;
                p1 = ‘‘Extroversion’’;
            }
```

```
                break;
    ...
    ...
    }
```

At the end of the loop, the four vectors named `first,second,third` and `fourth`, will contain the IDs of the Facebook pages that has determined the four dimensions of user's personality. Such IDs are used to request from the Facebook Graph the names and pictures that populates the table in the website.

5.5 Second phase: personality and recommendations

When the bot acquires the focus again after pressing the *Back to the bot!* button on the website, the next step consists of sending the keyword *Continue* to the bot with the provided `quick reply` button. Quick replies are in-conversation buttons that appear prominently above or inside the composer; when a quick reply is tapped, the buttons are dismissed, and the title of the tapped button is posted to the conversation as a message. This particular design solution finds his meaning to the fact that Dialogflow does not foresee sending events from external resources (the website in this case) and so, there is no way to notify the bot about the successful authentication.

When the keyword is received by Dialogflow, it triggers an Intent that uses the webhook for its fulfillment. The server checks if the chat session id is in the database and from here, three possible scenario can occur:

1. User has gone to the website, followed the instructions given and tapped on the quick reply button.

2. User has tapped the quick reply button without first visiting the website or has wrote something unexpected to the bot.

3. User has gone to the website and then has wrote something unexpected to the bot.

On the first scenario (Figure 5.6), the conversation follows its natural flow and the message *"Cool! Your Facebook digital footprints tells me that you are an <MBTI> person. Do you want to know more about your personality or go straight to your music recommendations?"* is returned as a positive outcome by the web server.
Users can reply with open phrases like *"Tell me more about my personality"* or *"Recommendations please"*. Association between open phrases and Intents is managed by Dialogflow with a proprietary Natural Language Processing algorithm

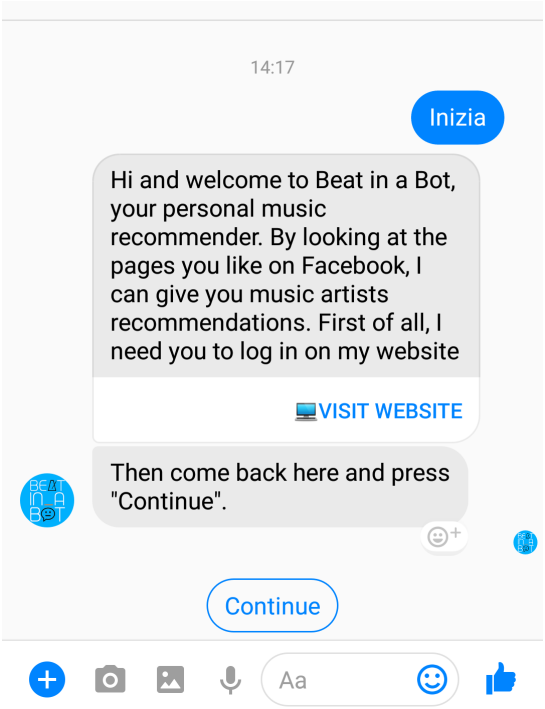


Figure 5.5. Bot onboarding message and the quick reply button.

that mixes a rule-based approach with a machine learning model[2]. Based on the response given by the user, either the *Personality - more* Intent or the *Recommendation* Intent can be triggered: in the first case, response is given by the backend that replies with the same description of the user's Myers-Briggs Type Indicator shown on the website and a quick reply button with the text *"Now recommend me!"*. In the other case, the web server computes the recommendations and replies with the first artist's card. Details on how recommendations are built will be explained in the following section.

On the second scenario, that occurs when user has tapped the quick reply button without first visiting the website or has wrote something unexpected to the bot, the conversation flow breaks and instructions on how to correctly proceed are given to the user. Instructions includes a text message, a new button to the website

[2] https://dialogflow.com/docs/machine-learning

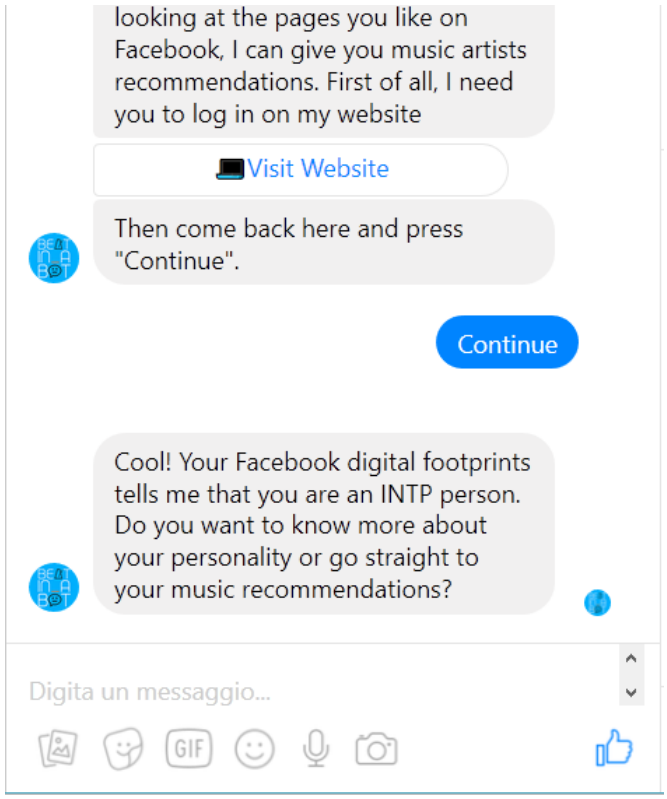


Figure 5.6. First scenario.

and a new *Continue* quick reply button. The third and last scenario is managed by Dialogflow using the *Fallback Intent* option. Fallback Intents are triggered if a user's input is not matched by any of the regular Intents; the conversational flow is built setting the output and input contexts of the various Intents properly and fallback Intents follows the same rule. In this particular case, the *Personality* Intent, that is triggered by the *Continue* keyword, has the *personality* context both as input and output context and the *recommendation* context as additional output context. If no user's input matches the *Personality* Intent, its relative fallback Intent that has *personality* as input context, triggers and the response is delegated to the backend that replies with the message *"it is not so hard, just push the continue button below."* and the *Continue* quick reply button to recover from the error and restore the conversation flow.

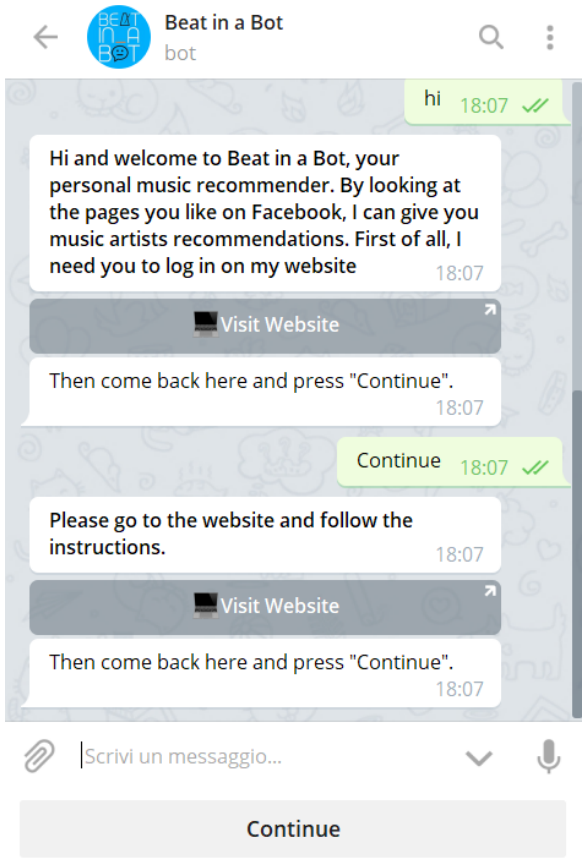


Figure 5.7. Second scenario.

5.6 Recommendations

The recommendations phase starts when the user asks the bot to compute them: this triggers the *Recommendation* Intent inside Dialogflow and the response is once again built by the web server. Using the session id as a key, the web server extracts the user's MBTI from the database and passes it as an argument to the function `genresListFromJungianType(...)`. The function returns a list whose elements are the musical genres associated to the given MBTI by the study conducted by NERIS Analytics Limited. Next, the server calls the `artistsIDsFromFile(...)` method that takes the user's Facebook id and his MBTI as arguments and implements the following value chain:

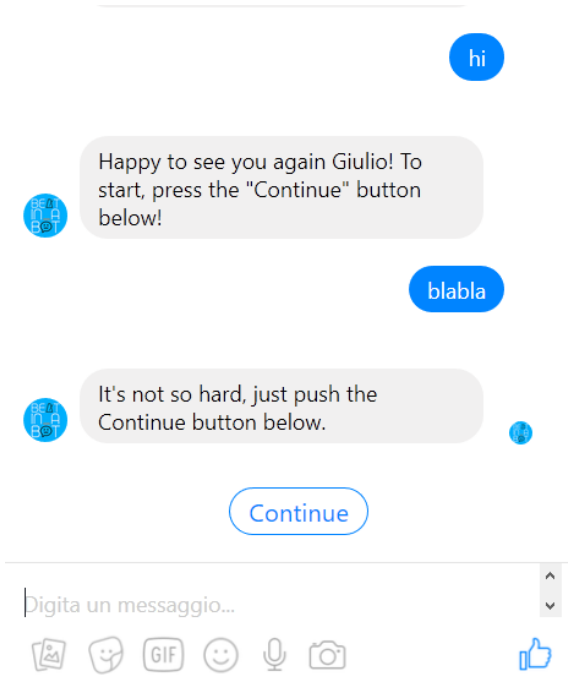


Figure 5.8. Third scenario response.

5.6.1 Recommender models generation

Before moving on with the details of the value chain, it is necessary to explain how the recommender system works. As written in the approach section, the recommender system used is *MyMediaLite*. MyMediaLite is released as a binary file that can be executed from the command line passing a set of arguments that specifies the possible operations available. The binary file is programmatically executed using the `call(...)` method of the `process` library for Python. The recommender system executable is mainly used for two operations: to save recommender system models and to load the models for prediction making. The first operation is executed by a Python script that is programmed to run every 30 minutes using the Unix command *cron*. Cron is a Unix utility that allows tasks to be automatically run in the background at regular intervals by the cron daemon. The tasks, called *cron jobs* are written into a file called *CronTab* which contains the schedule of cron entries to be run and at specified times. The string added to the server crontab is the following:

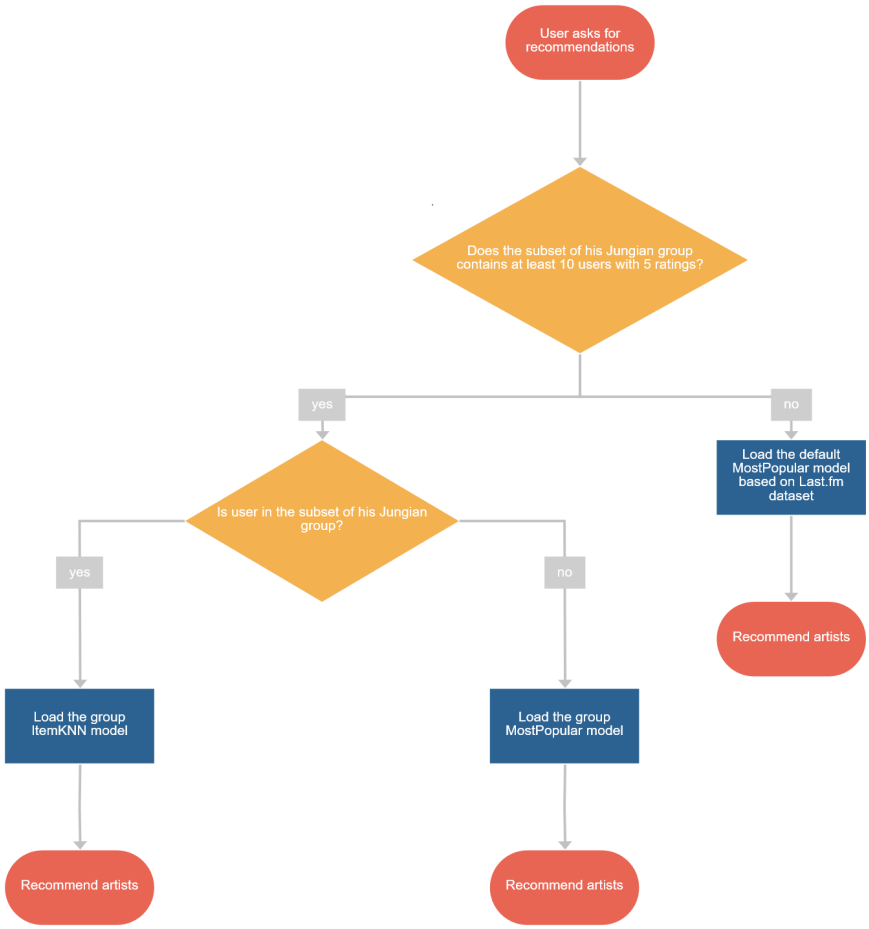


Figure 5.9. The value chain of Beat in a Bot recommendation process.

```
/30    * /home/musicbot/MusicBot/cron.sh
```

The execution time is empirically set to 30 minutes for the initial test phase; it will be adjusted based on future analysis on the bot usage. The cronjob runs a script that creates five Comma-Separated Values (CSV) files: one for each personality group for a total of four, and one for all the users. Those files contain the user-like entries of the database table *likes*.

```
cat all.csv
```

```
userID,artistID
967939456704050,412
967939456704050,441
967939456704050,691
967939456704050,986
...
```

Those are used to generate the prediction models for both the MostPopular and the Item K-Nearest Neighbor algorithms. The result of this operation consists of nine files: four Most Popular models and four ItemKNN models for the personality groups and one Most Popular model for all the users.
To save models, the recommender system takes as input the CSV file of user's group, used to train the model, and generates as output, the model file. If no `-recommender` option is specified, the MostPopular algorithm is used by default. It runs using the following command:

```
./mymedialite/item_recommendation
--training-file=mymedialite/data/<group_name>.csv
--save-model=mymedialite/models/<group_name>_MPmodel
```

where the first argument specifies the path to the input file used to train the model and the second argument where to save the model.

5.6.2 Recommendation value chain

The idea behind the recommendation process is to give users the best recommendations possible based on the data available on his ratings. Since users' personalities are grouped into four categories, the first decision is taken on the amount of groups user-likes data available in the database: if data of the group to which the user belongs to are enough i.e. there are at least 10 users who gave at least 5 ratings, the algorithm proceeds with a second decision process otherwise, it loads the Most Popular model file based on the Last.fm dataset, which can be used to predict general recommendations. The second decision process checks if the active user is part of the user-likes data of his group. If so, it means that there is a user history and so, the ItemKNN model can be loaded and used: this allows to find the recommendations with the highest precision possible. If the user is not part of the user-likes data, it means that he is new to the system and he will be present when the next cron job executes; in this case, the personality group Most Popular model is loaded and used to predict optimal recommendations.
Recommendations prediction is executed running MyMediaLite with the following command:

```
./mymedialite/item_recommendation
--training-file=mymedialite/data/<group_name>.csv
--load-model=mymedialite/models/<group_name>_ItemKNNmodel
--prediction-file=mymedialite/mml_output/<fbuserid>.prediction
```

It takes three arguments: the training file used to train the model, the model that has to be used to make predictions and the name of the output prediction file. As a design choice, each prediction file is named with the active user's Facebook id in order to be unique for each user.
When the web server receives the request by Dialogflow with *Recommendation* as the triggered Intent, it calls the method `artistsIDsFromFile(...)` that reads the output prediction file of the active user and returns a list with the artists ids predicted for him. At this point, for each user's favorite music genres, the algorithm uses the ids in the prediction list to extract from the database 50 random artists of which only the first two are chosen to be inserted in the recommendations list. As example, a user with a personality of the type INTP belongs to the *Analysts* group and so is linked with the following vector of genres in descending order of preference:

```
Analysts  = ['Rock','Alternative','Classica','Electronica','Pop','Jazz',
            'Ambient','Punk','Metal',Hip-Hop','Blues','World',
            'Reggae','Religious']
```

the recommendations list for such user will contain 2 Rock artists, 2 Alternative artists and so on. In parallel, a list of indices for each recommendation is maintained. Such list is shuffled at each iteration (and so for each music genre) to add some randomness to recommendations. The algorithm checks if there are doubles in the recommendations' list and removes it. When the loop ends, the recommendation indices' list, which will contain 2 ids for each music genre, is first reversed and then inserted in the database table as a JSON object in the relative user's row of the *Users* table. Finally, the *Recommendations* table of the database is filled with the JSON objects of the blobs that Dialogflow sends to Messenger and Telegram to show the recommendations. Such objects are indexed using the recommendations indices' list.

The solution of reverting the list of recommendations indices is due to the fact that recommendations are given with a stack logic. This derives from the way recommendations are presented to users in the chatbot interface: it shows one artist recommendation at a time, in a form of a card containing the artist name, his music genre, an image taken from his Last.fm profile and two buttons *Next* and *I like it!*: the first is used to navigate recommendations and the second to express a preference.

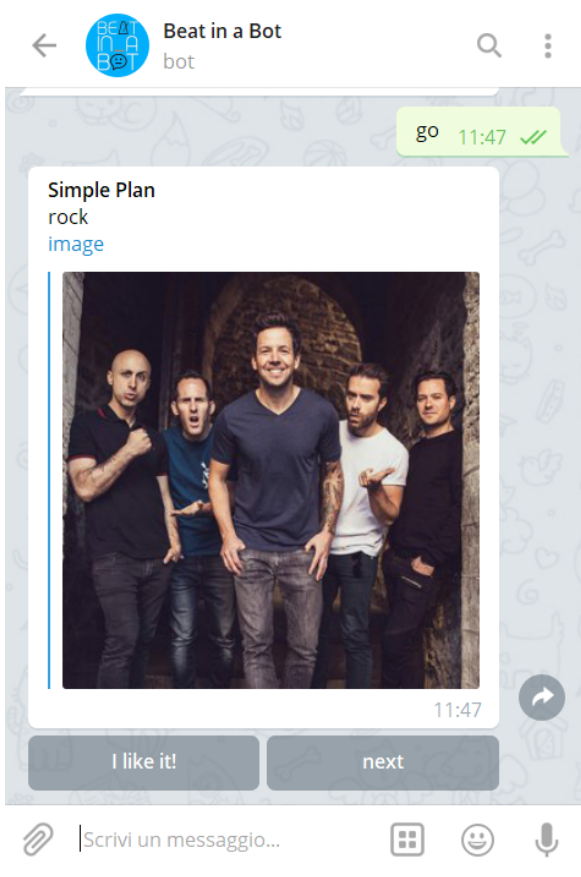


Figure 5.10. The card with the recommended artist

The logic behind the operation of the *Next* button is as follows: when the button is pressed, the *Next* Intent is triggered in Dialogflow and the response managed by the web server as usual. The web server takes the list of recommendations indices (which is reversed) from the database, pops out the id of the next recommendation to extract the blobs from the *Recommendation* table, puts the list back into the database and sends the blob to Dialogflow that forwards it to the chat interface which will shows the artist's card to the user; this logic provides a natural way to take the count of recommendations in order to signaling the user when they are over with the message: *"You reached the bottom. This means your recommendations for today are over. You can start back saying "hi" again or come visit me tomorrow for new ones :)"*.

5.6.3 User preferences

Users can express their positive-only feedback tapping the *I like it!* button above the artist's card. Each like button is built to send a special string in a postback to the bot. Such string is encoded as follows: `"<artist_id>;<user_id>"`. An Intent named *like* inside Dialogflow, is triggered only by this specific string and the two IDs contained are caught as system entities. As a consequence, the two values will have a dedicated space in the JSON object sent to the webhook. When the web server receives the request from Dialogflow, the two IDs can be easily read from JSON object.

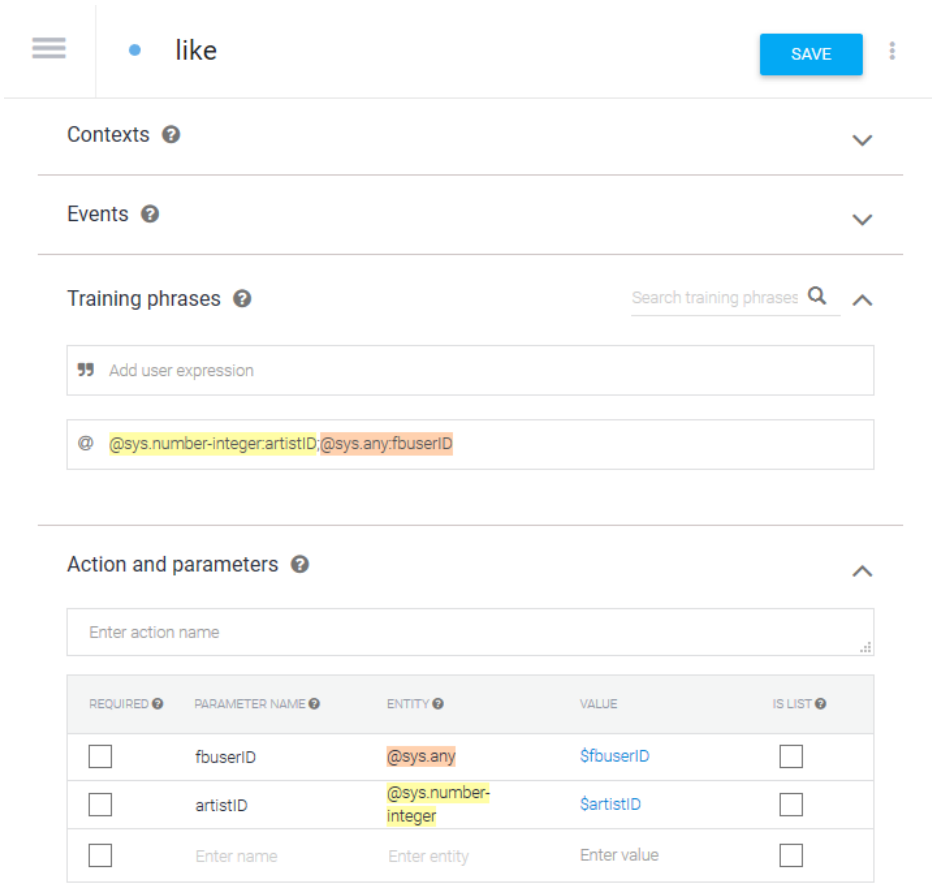


Figure 5.11. The like Intent inside Dialogflow

At this point, the server stores the `user-artist` pair in the database table *Likes*, that represents the user feedback. Then it replies with a randomly chosen message, taken from a list of variants, and a quick reply button *Next artist* that guides the user in the bot interaction.

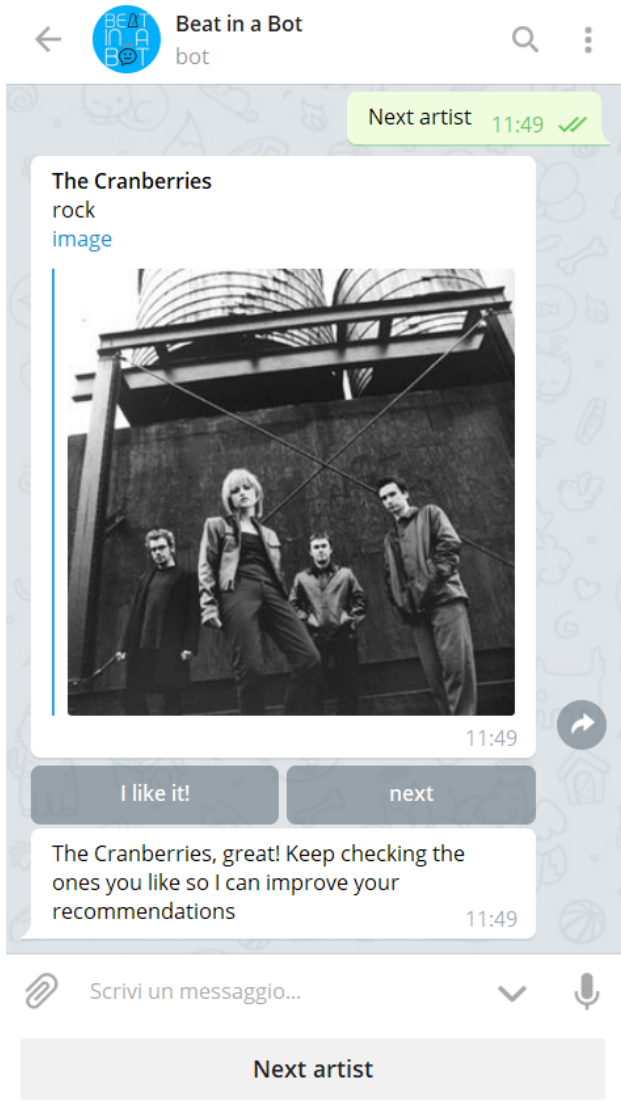


Figure 5.12. Bot response to user feedback

If the user writes something unexpected to the bot, a fallback Intent named *next - fallback*, entirely managed by Dialogflow, triggers and instructions for error recovering are given to the user.

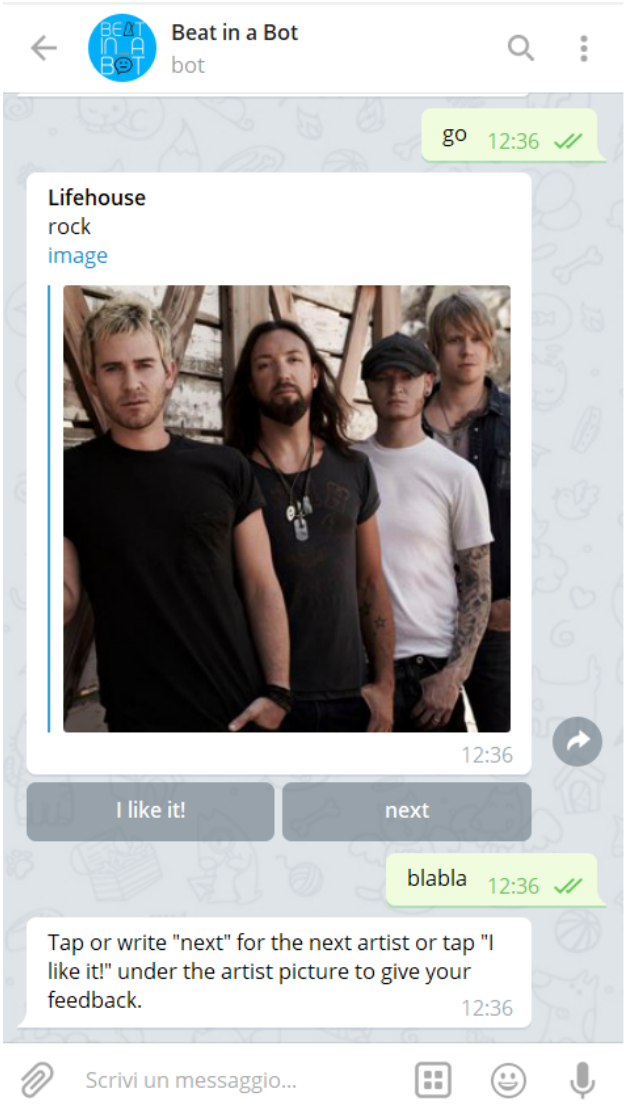


Figure 5.13. Beat in a Bot response to unexpected input during recommendations phase

Chapter 6

Experimental validation

6.1 Introduction

This chapter is dedicated to the experimental validation of the chatbot application developed. The validation is composed by three different tests: a design test, a usability test and a usefulness test. The design of the application has been tested with an online tool called *Alma, the chatbot test*[1] . It is based on a heuristic evaluation of 7 metrics of chatbots design. A test reliability study has been conducted to validate the tool. The application usability has been tested with a qualitative survey based on Nielsen's heuristics. This subsection includes a background and an analysis of the test results. The usefulness of the application has been evaluated with an experiment: it has been asked to 43 people, between the ages of 20 and 35, to use the bot and express their preferences on recommendations considered interesting. The subsection contains an analysis of the results obtained.

6.2 Design test

6.2.1 Introduction

To test chatbot design an online tool called *Alma* has been used. Alma is a free Chrome extension released as an open source project by Jesús Martín (Product Designer for BEEVA Labs), Carlos Muñoz-Romero (Digital Product Manager for monoceros.xyz) and Nieves Ábalos (Conversational Interfaces expert for monoceros.xyz) that acts like a chatbot itself. Alma asks 33 questions that helps to

[1] http://chatbottest.com/

find problems in the human-chatbot interaction. The test is based on a heuristic evaluation of 7 metrics of chatbot design:

- *Personality*: Alma's questions tries to understand if the chatbot has a personality that can be recognized as unique, if that personality is coherent on the whole conversation, if it is tailored on the chatbot audience and last, if the tone adapts to the ongoing conversation.

- *Onboarding*: This metric will measure the clarity of the chatbot introduction message. The onboarding has to be short and effective at the same time, making clear what the chatbot can do and how users can get the most out of it, avoiding errors and frustration.

- *Understanding*: Testing the understanding of a chatbot is something that mainly deals with technical aspects of the development. This part really depends on the type of service the chatbot offers: if it pretends to give a human-like experience, the level of understanding must be high, otherwise a command based conversation can be acceptable if the bot is mainly a single purpose service.

- *Answering / Speaking*: Questions related to this metric will test how much effectively chat elements are used inside the conversational interface. Such elements are plain text, buttons, emojis, pictures, video, cards and any other media and can be integrated to the chat to make the conversation much more interesting and engaging.

- *Navigation*: As any good interface cannot lack of navigation tools like back buttons and search boxes, a conversational interface has to provide ways to help users go from some parts of the interaction to others. Navigation questions will measure how well the chatbot implements this important interaction tools.

- *Error management*: Conversational interfaces give a lot of freedom to users but at a cost: users expects that the chatbot will understand everything due the illusion of a human-like interaction. This inevitably brings errors and this metric will test how well the chatbot can manage them.

- *Intelligence*: The test includes some questions that check the ability of a chatbot to know things not necessarily said by the user, remember others that appear during the interaction, and use that knowledge to adapt the conversation to the user and the specific situation.

6.2.2 Design test results

Test results are presented in a circular diagram in which each of the seven metrics is expressed in a range between 0 and 100. Beat in a Bot design test has given the following results:

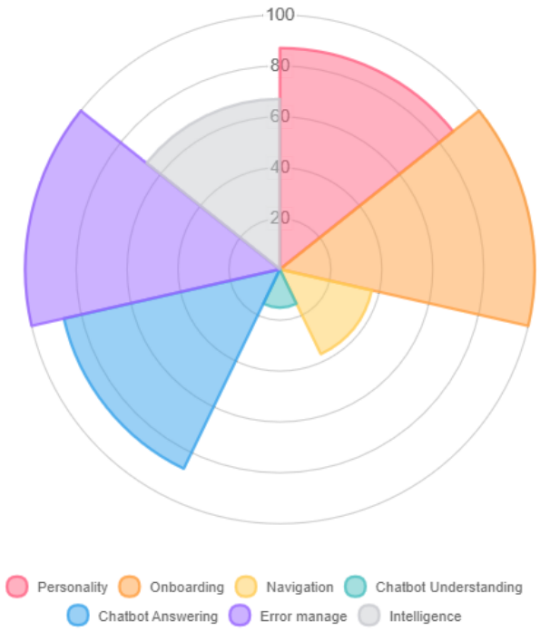


Figure 6.1. A circular diagram reporting the scores of each metric of the test results.

Before analyzing the test results, it is necessary to have in mind that there is a difference in the weight of the measures depending of the type of bot being tested. The *Understanding* metric is much more important for a bot that has the purpose of entertaining a human-like conversation rather then for a single-purpose bot that mainly uses a command-based interaction. As for the former, it is expected to answer to a large set of questions about different topics, but for the other, users only expect replies to questions about his main activity.

Personality

As for the personality of Beat in a Bot, the score marks a value between 80 and 100, indicating a recognizable "persona". The bot tends to maintain the same tone during the conversation and always replies consistently with what is asked over

time. Furthermore, the answers are always relevant, in the sense that, they reflect user's expectations.

Onboarding

Onboarding scores a mark of 100. The chatbot greets the user as soon as he writes the first message explaining what his purpose is and giving the instruction on how to continue the interaction.

Navigation

Navigation scores a mark just below 40. The reason is that, the chatbot, because of the nature of the service it offers, does not need to provide the user with options to undo operations since there is no critical task that needs to be undone (like for instance, in a online shopping bots).

Chatbot Understanding

Here the chatbot has his lowest score, under 20. As for the metric "Navigation", the reason for this result is due to the intrinsic single-service nature of the bot, whose conversational interface is mainly command-based. It does not need to have a vast natural language understanding level, since available commands are presented directly to user from the bot; this is a design choice to make the conversational flow as effective as possible while reducing the risk of understanding errors.

Answering/Speaking

The answering/speaking metric settles on an high value, between 80 and 100. The chatbot always gives the right answer to the given command and, once the onboarding process is completed, it shows a card-buttons interface that allows to navigate between recommendations or give a preference for an artist (Figure 5.10).

Error Manage

Test has given the mark of 100 for the "Error manage" metric. The bot correctly recognizes and manages error cases and reports instructions on how to continue. A representative example is the case in which the user tries to jump the log in phase. The bot acknowledge that and asks the user to please log in before continue with the conversation (Figure 5.13).

Intelligence

Intelligence scores a mark just above 60. Since the bot records the chat session and uses an authentication system, it has the ability to remember data and preferences

not explicitly said by the user. Those are used to adapt the conversation to the user and to the specific situation, tailoring recommendations as preferences are expressed.

6.2.3 Test reliability study

Tests aim to measure dimensions (or factors) of constructs through indicators. It is known that every measurement brings errors that can be random or systematic. To keep incidence of errors under control, the concept of reliability is introduced. Reliability is the degree to which an assessment tool produces stable and consistent results i.e. the degree of coherence between independent measurements of the same construct. A measure is said to have a high reliability if it produces similar results under consistent conditions. *Usability* is the construct of the test whose validity is to be measured and its indicators are:

1. Personality
2. Onboarding
3. Navigation
4. Chatbot understanding
5. Chatbot answering
6. Error management
7. Intelligence

The *reliability coefficient* of the test expresses the proportion of true variance in the measurement with respect to the total variance.

$$r_{tt} = \frac{\sigma^2(V)}{\sigma^2(X)} = \frac{\sigma^2(X) - \sigma^2(E)}{\sigma^2(X)} = 1 - \frac{\sigma^2(E)}{\sigma^2(X)} \tag{6.1}$$

The variance, expressed as $\sigma^2(X)$ measure how far a set of (random) numbers are spread out from their average value. It is an index of spreading. There are several methods to compute reliability of a test:

Test-Retest Method

The Test-Retest method is the simplest way of testing the stability and reliability of an instrument over time. Consists of submitting the test at two different times, T1 and T2, and then calculate the correlation between the two scores. This method relies upon the calculation of the Pearson correlation coefficient.

Parallel-forms method

The Parallel-forms method consists of submitting two equivalent version of the test (same average and same standard deviation) that both measure the same construct, knowledge or skill. The test versions are given to the same sample of people within a short period of time and an estimate of reliability is calculated from the two sets. The correlation of the scores from the two tests is a measure of their reliability.

Split-half method

The Split-half method consists of first submitting the test once, then subdivides the test in two equal halves and consider them as two parallel-forms (same average and same standard deviation). The reliability is then calculated as a correlation between values of the two forms.

Internal coherence method

The internal coherence test consists of submitting the test only once, at time T1. Each item is considered as a separate test. The average correlation between all the items is estimated (with special formulas), and from it, an evaluation of the reliability coefficient is derived.

6.2.4 Reliability test and results

The method chosen to evaluate the reliability of the chatbot design test is the Test-Retest method because it is the only one that does not require additional work on the test itself: the test is made by external source and there is no way to subdivide it into two equal halves; furthermore, there is no equivalent test available that can be used as a different form. The following table shows the values of the indices obtained in the two tests taken at time T1 and T2:
The test-retest coefficient is calculated with the following formula:

$$r_{tt} = \frac{Cov(T1, T2)}{Dev.St(T1) * Dev.St.(T2)} \tag{6.2}$$

The covariance is equal to:

$$Cov(T1, T2) = 812{,}53 \tag{6.3}$$

The test-retest coefficient is equal to:

$$r_{tt} = 1 \tag{6.4}$$

	T1 (01/25/2018)	T2 (02/5/2018)
Personality	82	79
Onboarding	100	100
Navigation	38	40
Understanding	16	22
Answering	82	87
Error management	100	100
Intelligence	64	67
Average	68,86	70,71
Std. deviation	29,41	27,72

Table 6.1. Indices values obtained in tests T1 and T2

6.2.5 Result analysis

The test-retest reliability coefficients (also called coefficients of stability) vary between 0 and 1, where:

- 1 : perfect reliability,
- >= 0.9: excellent reliability,
- >= 0.8 < 0.9: good reliability,
- >= 0.7 < 0.8: acceptable reliability,
- >= 0.6 < 0.7: questionable reliability,
- >= 0.5 < 0.6: poor reliability,
- < 0.5: unacceptable reliability,
- 0: no reliability.

On this scale, a correlation of .9 (90%) would indicate a very high correlation (good reliability) and a value of 10% a very low one (poor reliability). The result obtained with the test-retest method used to evaluate the reliability of *chatbottest.com*, is a reliability coefficient equal to 1 indicating a perfect reliability of the test.

6.3 Usability test

To test the usability of *Beat in a Bot* it has been asked to 50 individuals, between the ages of 20 and 35, to fill a survey [2] of 9 questions based on a custom review of Jakob Nielsen's usability heuristics for evaluating user interfaces. Jakob Nielsen, Ph.D., is a User Advocate and principal of the Nielsen Norman Group which he co-founded with Dr. Donald A. Norman (former VP of research at Apple Computer). Nielsen established the "discount usability engineering" movement for fast and cheap improvements of user interfaces and has invented several usability methods, including heuristic evaluation. He holds 79 United States patents, mainly on ways of making the Internet easier to use. In 1995 he developed 10 usability heuristics (Nielsen 1995) for evaluating user interfaces. These heuristics have stood the test of time, providing designers with a quick and easy way of evaluating the usability of software interfaces against a set of universal design principles.

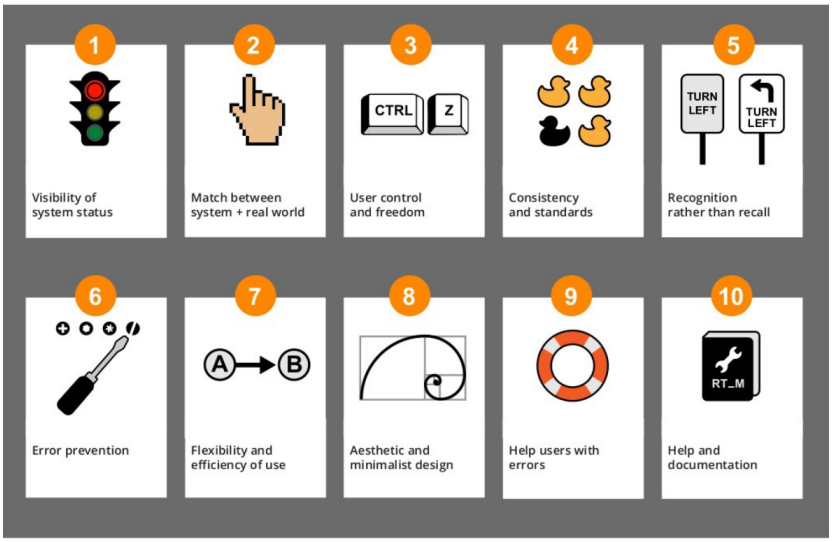


Figure 6.2. Nielsen's Ten Heuristics.

[2] https://goo.gl/forms/cjmWnKvhaqSW79rf1

6.3.1 Nielsen's Ten Heuristics.

This subsection reports the Nielsen's ten heuristics for user interfaces and for each of them, an analysis is done to adapt them to the world of chatbots.

Visibility of system status

"The system should always keep users informed about what is going on, through appropriate feedback within reasonable time."

The medium of bots is a conversation, and conversation is governed by messages. Since messages are ephemeral, they become stale with time and old messages are less valuable than ones from few seconds ago.
This heuristic can be ported to chatbots in a slightly different form: the system should allow user to *request* information about what is going on, through appropriate feedback within reasonable time.

Match between system and the real world

"The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order."

The main obstacle to this heuristic is that, despite chatbots are made to chat, they have problems with language understanding. Even those built atop the latest tech are limited in what they can understand and how well they can respond. The key to best follow this heuristic is to know the chatbot audience. Some users will appreciate a command line interaction style, and others will expect to converse in natural language. Still, others might speak in slang or abbreviations. Bots should be built with a solid understanding of the audience they seek to appeal to.

User control and freedom

"Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo."

Chatbots can easily implement "emergency exits", keeping the user aware of valid options during any stage of the interaction.

Consistency and standards

"Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions."

Unfortunately, at the time, there are no platform conventions for chatbots but this heuristic can be interpreted to mean that bots should be internally consistent; a bot should stick to a single style of language, whether that is natural language, command line, or something in between.

Error prevention

"Even better than good error messages is a careful design that prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action."

Thanks to the conversational nature of chatbots, it is quite easy to follow this design principle. Bot designers should build interactions with the assumption that errors will happen early and often, given the ambiguity and impreciseness of most human dialogue. Bots should ask users for confirmation before any critical step in an interaction.

Recognition rather than recall

"Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate."

One big problem of user interface design is that users do not seem to read much, if at all. This problem intensifies with chatbots given that the medium is mostly text. The majority of chatbot platforms provides structured messages that includes button, menus and quick replies; while this seems a solution, an over-reliance on them can feel contrived and can denatures the promise of talking to chatbots using the human language.

Flexibility and efficiency of use

"Accelerators-unseen by the novice user-may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions."

Bots are supremely well positioned to provide these types of invisible accelerators to power users providing both speaking and command interactions. While one user might talk to the bot using well structured phrases, another may prefer to use commands to achieve the same result. Here lies an open question: how to teach users to become power users without resorting an help menu?

Aesthetic and minimalist design

"Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility."

This principle is a little ambiguous for chatbots. Since they pretend to give an human-like interaction, users are also led to ask questions unrelated to the core competency of the bot. If a bot fails to reply such type of questions, the experience will seem subpar but at the meantime, bots have a restricted area of competency and it is impossible to cover all kinds of questions. The solution is in the middle: bots should reply to all the questions related their area of competency plus should be able to keep a small talk.

Help users recognize, diagnose, and recover from errors

"Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution."

Easily applicable with bots. If an error occurs, bots should inform users about that explaining what happened and how to recover.

Help and documentation

"Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large."

Also still applicable. Help and documentation should be accessible via the bot itself with a menu or a command.

6.3.2 The survey

The survey has been built keeping in mind the Nielsen's heuristics and adapting them to this specific thesis object application. It is composed of the following 9 items:

1. The chatbot explains its purpose clearly
2. During the conversation, the chatbot explains clearly what the user can do
3. The onboarding message (the initial greetings) makes you want to continue
4. The login system that involves the use of an external website is complicated
5. It is clear what the external web site does
6. How do you judge the usefulness of the recommendations?
7. How do you judge the overall experience?
8. Would you use it again?
9. Would you recommend it to your friends?

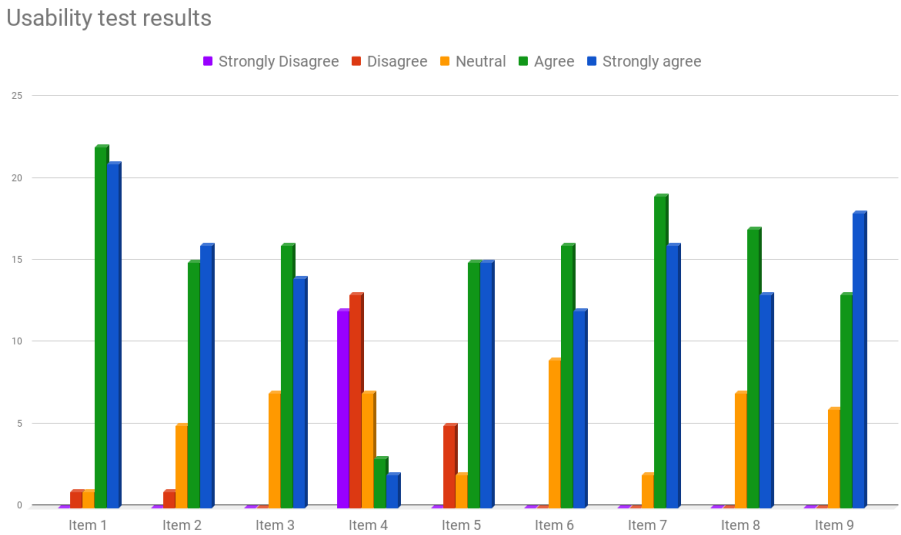
Users responses are structured using the Likert Scale, a psychometric scale commonly used to scaling responses in survey research. The scale was invented by psychologist Rensis Likert and allows respondents to specify their level of agreement or disagreement on a symmetric agree-disagree scale for a series of statements.

6.3.3 Result analysis

The following chart shows the number of times a particular answer was given for all the items of the survey in the form of vertical bars histograms.

The first survey item, *"The chatbot explains its purpose clearly"*, aims to evaluate if the onboarding message explains clearly what the bot can do for the user. With 21 *Strongly agree*, 22 *Agree*, 1 *Neutral*, 1 *Disagree* and zero *Strongly Disagree* responses, results confirms the item statement.

The purpose of the second item, *"During the conversation, the chatbot explains clearly what the user can do"*, is to evaluate if users can easily understand what options the bot makes available to them i.e. what users can do during the various steps of the conversation. Results reveal that 16 participants responded with *Strongly agree*, 15 with *Agree*, 5 with *Neutral*, only 1 with *Disagree* and zero with *Strongly disagree*. It can be said that user available options during the chat are fairly clear.



The third item, *"The onboarding message (the initial greetings) makes you want to continue"*, has the goal to measure the engagement power of the onboarding message. Results are quite positive with 14 *Strongly agree*, 16 *Agree*, 7 *Neutral*, zero *Disagree* and zero *Strongly disagree* responses. The message do its job but it also leaves space for future improvements.

The object of the fourth item, *"The login system that involves the use of an external website is complicated"*, is to see if users find the login system, which we consider as the weak point of the application, complicated. Surprisingly, results show that the procedure is well built from an usability perspective, and the majority of users did not find it particularly cumbersome. Response numbers are 2 *Strongly agree*, 3 *Agree*, 7 *Neutral*, 13 *Disagree* and 12 *Strongly disagree*. A note on this item: some users did not understand that the statement had a negative meaning and the answers had to be given backwards relative to other items to express appreciation.

The fifth item whose statement is *"It is clear what the external web site does"*, has the goal to understand if the bot explains clearly what is the role of the external website in the login procedure and if the website itself is built in a way that does not confuse the user during the experience. This item, together with the fourth, serves to understand the impact on usability of the particular access

system due to the multi-platform nature of the bot. The results shows a number of responses of 15 for both *Strongly agree* and *Agree*, 2 for *Neutral*, 5 for *Disagree* and zero for *Strongly disagree*. The outcome can be interpreted as positive because, even if there are 5 *Disagree* responses, the two positive responses marks a triple score.

Item number six, *How do you judge the usefulness of the recommendations?* is an explicit request to users to evaluate the quality of the recommendations in terms of usefulness, that is the actual approval of the recommended artists. Although positive responses score 12 for *Strongly Agree* and 16 for *Agree*, the results show a significant number of *Neutral* responses, 9 to be precise; the reasons for this result are due to the fact that many users did not know the artists proposed and responded to the survey without first having listened to them. Unfortunately, this phenomenon has been difficult to control, but despite this, the overall result is positive.

Regarding the seventh item, *How do you judge the overall experience?*, results are quite positive with a total of 16 *Strongly agree* responses, 19 *Agree* responses, only 2 *Neutral* and zero responses for *Disagree* and *Strongly disagree*. This is a clear sign of the goodness of *Beat in a Bot* it terms of usability.

The eighth item, *Would you use it again?*, aims to measure if users feel satisfied of the overall experience and consider *Beat in a Bot*, a valid tool for the recommendation of new music artists to listen to. Results are: no negative responses (*Strongly disagree* and *Disagree*), 7 *Neutral*, 17 *Agree* and 13 *Strongly agree*, showing a good level of user appreciation.

The last item, *Would you recommend it to your friends?*, aims to evaluate the application as a whole and to check if it is interesting enough to make people talk about it. An high number of *Strongly agree* and *Agree* responses, 18 and 13 respectively, confirms that the bot has aroused curiosity between users and the 6 *Neutral* responses can be seen as an incentive for a further improvement of the service.

6.4 Usefulness test

The usefulness test, carried out together with the usability test, aims to implicitly measure how much users have found the tool useful, from the number of positive-only feedback given to the proposed recommendations. Participants were asked to use the bot, scroll through at least 10 suggestions and mark those considered interesting. The actual users of the application were 27, 4 women and 23 men aged between 20 and 35. The distribution of users based on their belonging to a certain Myers-Briggs Type Indicator is shown in the following graph:

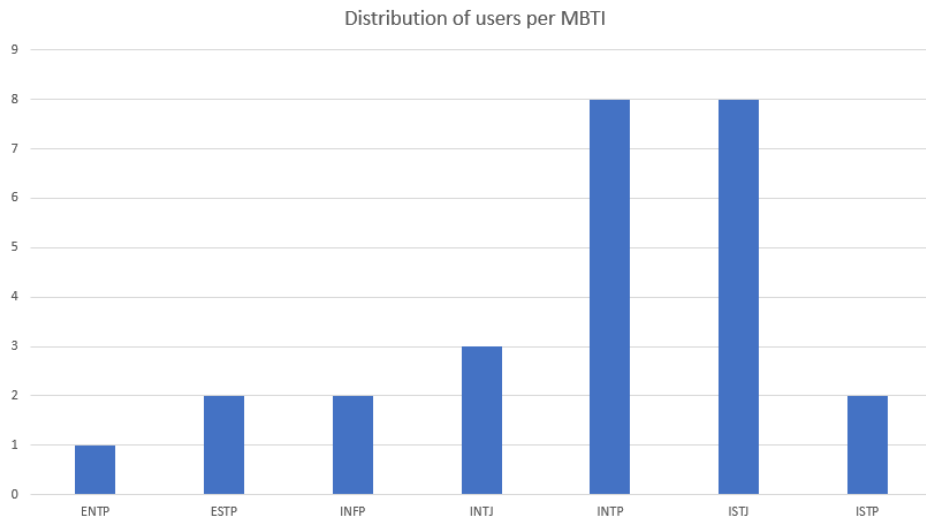
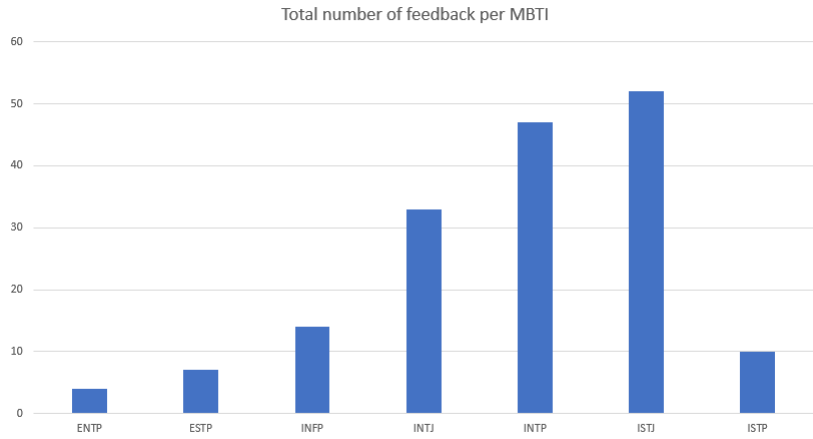
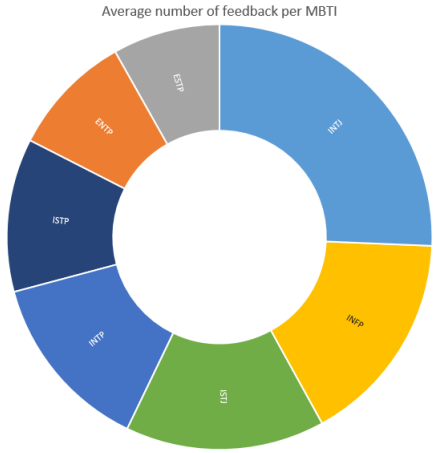


Figure 6.3. Distribution of users per MBTI.

Bars indicates a distribution of 8 INTP users, 8 ISTJ users, 3 INTJ users, 2 INFP, 2, ISTP, 2 ESTP, and 1 ENTP user. Adding the feedback from users belonging to the same MBTI group, we obtain a result of 47 feedback from INTP users, 52 feedback from ISTJ users, 33 feedback from INTJ, 14 feedback from INFP, 10 feedback from ISTP, 7 feedback from ESTP and 4 from ENTP. The result is summarized in the figure below. Bars indicates the total number of feedback given by each group of users. The similarity between the distribution of users graph and the total number of feedback graph, shows that the number of preferences is directly proportional to the number of users and that the average number of preferences given for each user within his group is similar among the various groups.



This is clearly shown by the dimension likeness of the colored part of the radial graph above, where each of them represents the average number of feedback given by users of each group. Numbers are 5.9 for INTP users, 4 for ENTP users, 3.5 for ESTP users, 7 for INFP, 11 for INTJ, 6.5 for ISTJ and 5 for ISTP. We can conclude that, since it has been asked to go through at least 10 recommendations and participants had the freedom of withdraw at each moment, a number of average feedback greater than 5 for the majority of groups is a good result in terms of application usefulness.



Chapter 7

Conclusions

The goal of this project was to experiment the usage of a multi-platform chatbot as an interface for a recommender system and an affective computing service. The first step was to outline an high-level architecture of the application then we went to choose the individual components that best suited the solution defined: chatbot logic management and multi-channel deployment were compassed with Dialogflow, a conversational agent builder by Google. Facebook Login were used as authentication system, implemented with Facebook Javascript SDK into a HTML5 website hosted on GitHub Pages. Recommendations were computed with MyMediaLite choosing the algorithm on the basis of a value chain that takes into account the size of the dataset and its content; the cold start problem were mitigated using data about user's personality, predicted with the affective computing service Apply Magic Sauce and a study about the connection between music preferences and personality by NERIS Analytics. As for music artists data, the *hetrec2011-lastfm-2k* gold standard by Ignacio Fernández-Tobías, Iván Cantador and Alejandro Bellogín, were used. Dataset and data about users authentication and artist preferences were stored using a MySQL database. User-tailored responses, database queries and HTTP requests are managed by an Apache/Flask backend written in Python and hosted on a server of the ISMB research center. After the architecture configuration, the bot was released on Facebook Messenger and Telegram with the name *Beat in a Bot*.
Bot validation were done with three tests: a design test made with *Alma, the chatbot test*, an usability test taken with a Nielsen's heuristic-based survey and a usefulness test based on usage analytics of the bot. Test results has revealed bot strength and weaknesses and a good overall level of user's appreciation. The developed chatbot lends itself to a variety of future improvements. The main one concerns the prediction phase of user's personality. At the moment, it relies upon two external web services, Apply Magic Sauce API and Facebook Graph API and

has the user requirement of a Facebook account. External web services provide a fast and easy way to obtain a result, but they may become a problem for the application maintaining due to the fact that their functionality may change over time, they have limitations and they are not under the complete control of the chatbot developer. The Facebook account requirement can be an obstacle for privacy concerned people that do not want to share their personal information and also for others that do not have a Facebook account. A solution to this problem is represented by the possibility of predicting user's personality from the chat itself, using only text. This can be achievable expanding the onboarding experience with a little conversation before moving to artist recommendations. Several studies have explored the possibility of personality prediction from plain text. One worth mentioning is TwitPersonality by Carducci et al. (Carducci G. 2018) who presented a supervised learning approach to compute personality traits by only relying on what an individual tweets about his thoughts publicly. There also are several products already available that are able to find user's personality from written text, like IBM Personality Insights [1], Apply Magic Sauce itself and Textgain[2] but, while they eliminate the need of a social account, they still have all the problem of external services already discussed. Furthermore, to generate a reliable personality prediction, 100 to 200 words are required and this is difficult to achieve in a entertainment chatbot application.

From a financial perspective, the chatbot can be used to generate earnings applying a freemium pricing strategy. Freemium is a pricing plan by which a product or service is provided free of charge, but money (premium) is charged for additional features, services, or virtual goods. The plan includes two versions of the chatbot: a free version that provides only the names and images of artists as recommendations and a paid version that gives the possibility to receive personalized music playlists composed of the best songs of the recommended artists. Playlists can be created for all the different music streaming platforms available, so that the chatbot can be used by a wider audience, rather than being proposed as an individual service.
In conclusion, chatbots represent a good alternative to websites and apps and are flexible enough to be used in conjunction with artificial intelligence, data science, recommender systems and other fields of technology. They find their place in a wide variety of applications, from entertainment to customer care, from e-commerce to even psychotherapy. The chatbot developed for this thesis project has aroused interest between users and can be considered a valid tool for personalized music recommendation.

[1] https://www.ibm.com/watson/services/personality-insights/

[2] https://www.textgain.com/

Bibliography

Adomavicius, G. & Tuzhilin, A. (2005), 'Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions', *IEEE Trans. on Knowl. and Data Eng.* **17**(6), 734–749.
URL: *https://doi.org/10.1109/TKDE.2005.99*

Anderson, C., John, O. P., Keltner, D. & Kring, A. M. (2001), 'Who attains social status? effects of personality and physical attractiveness in social groups', *Personality & Social Psychology* (81), 116–132.

Aron, J. (2011), *Software tricks people into thinking it is human*, New Scientist.

Atzori, M., Boratto, L. & Spano, L. D. (2017), 'Towards chatbots as recommendation interfaces'.

Bachrach, Y., Kosinski, M., Graepel, T., Kohli, P. & Stillwell, D. (2012), Personality and patterns of facebook usage, *in* 'Proceedings of the 4th Annual ACM Web Science Conference', WebSci '12, ACM, New York, NY, USA, pp. 24–32.
URL: *http://doi.acm.org/10.1145/2380718.2380722*

Bai, S., Zhu, T. & Cheng, L. (2012), 'Big-five personality prediction based on user behaviors at social network sites'.

Baltrunas, L., Kaminskas, M., Ludwig, B., Moling, O., Ricci, F., Aydin, A., Lüke, K.-H. & Schwaiger, R. (2011), Incarmusic: Context-aware music recommendations in a car, *in* C. Huemer & T. Setzer, eds, 'E-Commerce and Web Technologies', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 89–100.

Berry, D. S., Willingham, J. K. & Thayer, C. A. (2000), 'Affect and personality as predictors of conflict and closeness in young adults' friendships', *Journal of Research in Personality* **34**(1), 84 – 107.
URL: *http://www.sciencedirect.com/science/article/pii/S0092656699922717*

Bohus, D. & Rudnicky, A. I. (2003), Ravenclaw: dialog management using hierarchical task decomposition and an expectation agenda., *in* 'INTERSPEECH',

ISCA.
URL: *http://dblp.uni-trier.de/db/conf/interspeech/interspeech2003.html#BohusR03*

Bu, J., Tan, S., Chen, C., Wang, C., Wu, H., Zhang, L. & He, X. (2010), Music recommendation by unified hypergraph: Combining social media information and music content, *in* 'Proceedings of the 18th ACM International Conference on Multimedia', MM '10, ACM, New York, NY, USA, pp. 391–400.
URL: *http://doi.acm.org/10.1145/1873951.1874005*

Burke, K. (2016), Text Request.
URL: *https://www.textrequest.com/blog/many-texts-people-send-per-day/*

Burke, R. (2002), 'Hybrid recommender systems: Survey and experiments', *User Modeling and User-Adapted Interaction* **12**(4), 331–370.
URL: *https://doi.org/10.1023/A:1021240730564*

C. Tupes, E. & E. Christal, R. (1992), 'Recurrent personality factors based on trait ratings', **60**, 225 – 251.

Cantador, I., Brusilovsky, P. & Kuflik, T. (2011), 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011), *in* 'Proceedings of the 5th ACM conference on Recommender systems', RecSys 2011, ACM, New York, NY, USA.

Carducci G., Rizzo G., M. D. P. E. M. M. (2018), 'Twitpersonality: Computing personality traits fromtweets using word embeddings and supervisedlearning', *Information, to appear.* .

Cattell, R. (1957), *Personality and Motivation: Structure and Measurement*, World Book Company.
URL: *https://books.google.it/books?id=zNl-AAAAMAAJ*

Chen, G. (1998), *The peer help centre in Canadian colleges and its inspiration*, Researches on Higher Education.

Corr, P. J. & Matthews, G. (2009), The cambridge handbook of personality psychology, Cambridge University Press. -0–9, Cambridge, U.K., pp. 521–86218. ISBN 978.

Costa, H. & Macedo, L. (2013), Emotion-based recommender system for overcoming the problem of information overload, *in* J. M. Corchado, J. Bajo, J. Kozlak, P. Pawlewski, J. M. Molina, V. Julian, R. A. Silveira, R. Unland & S. Giroux, eds, 'Highlights on Practical Applications of Agents and Multi-Agent Systems', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 178–189.

Costa, P. T. & McCrae, R. R. (1992), *Revised NEO Personality Inventory (NEO Pl-R) and NEO Five-Factor Inventory (NEO-FFI) Professional Manual*, Psychological Assessment Resources, Odessa, FL.

Daoussis, L. & Mckelvie, S. (1986), 'Musical preference and effects of music on a reading comprehension test for extraverts and introverts', **62**, 283–289.

Dollinger, S. J. (1993), 'Research note: Personality and music preference: Extraversion and excitement seeking or openness to experience?', *Psychology of Music* **21**(1), 73–77.

Fitz, T. (2009), *What webhooks are and why you should care.*
URL: *http://timothyfitz.com/2009/02/09/what-webhooks-are-and-why-you-should-care/*

Galton, F. (1883), *Inquiries into human faculty and its development*, Macmillan.

Gantner, Z., Rendle, S., Freudenthaler, C. & Schmidt-Thieme, L. (2011), MyMediaLite: A free recommender system library, *in* 'Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011)'.

Golbeck, J., Robles, C. & Turner, K. (2011), Predicting personality with social media, *in* 'CHI '11 Extended Abstracts on Human Factors in Computing Systems', CHI EA '11, ACM, New York, NY, USA, pp. 253–262.
URL: *http://doi.acm.org/10.1145/1979742.1979614*

Gosling, S., Augustine, A., Vazire, S., Holtzman, N. & Gaddis, S. (2011), 'Manifestations of personality in online social networks: Self-reported facebook-related behaviors and observable profile information', *Cyberpsychology, Behavior, and Social Networking* **14**(9), 483–488.

Gouvert, O., Oberlin, T. & Févotte, C. (2018), 'Negative binomial matrix factorization for recommender systems', *CoRR* **abs/1801.01708**.

Holotescu, C. (2016), 'Moocbuddy: a chatbot for personalized learning with moocs'.

Hughes, D. J., Rowe, M., Batey, M. & Lee, A. (2012), 'A tale of two sites:twitter vs. facebook and the personality predictors of social media usage', *Computers in Human Behavior* **28**(2), 561–569.
URL: *http://opus.bath.ac.uk/28062/*

Ikemoto, Y., Asawavetvutt, V., Kuwabara, K. & Huang, H.-H. (2018), Conversation strategy of a chatbot for interactive recommendations, *in* N. T. Nguyen, D. H. Hoang, T.-P. Hong, H. Pham & B. Trawiński, eds, 'Intelligent Information and Database Systems', Springer International Publishing, Cham, pp. 117–126.

Isabel Briggs Myers, L. (1962), *Introduction to Type®*, Cpp.
URL: *https://books.google.it/books?id=weICwouLaesC*

Jain, A. (2016).
URL: *https://clevertap.com/blog/10-reasons-why-users-uninstall-your-mobile-app/*

John, O. P., Angleitner, A. & Ostendorf, F. (1988), 'The lexical approach to personality: A historical review of trait taxonomic research', *European Journal of Personality* **2**(3), 171–203.

Jung, C. G. (1921), *Collected Works of C.G. Jung, Volume 6: Psychological Types*, Princeton University Press.
URL: *http://www.jstor.org/stable/j.ctt5hhqtj*

Khalaf, S. (2017), Flurry analytics.
URL: *http://flurrymobile.tumblr.com/post/155761509355/on-their-tenth-anniversary-mobile-apps-start*

Kucherbaev, P., Psyllidis, A. & Bozzon, A. (2017), 'Chatbots as conversational recommender systems in urban contexts', *CoRR* **abs/1706.10076**.
URL: *http://arxiv.org/abs/1706.10076*

Litle, P. & Zuckerman, M. (1986), 'Sensation seeking and music preferences', *Personality and Individual Differences* **7**(4), 575 – 578.
URL: *http://www.sciencedirect.com/science/article/pii/0191886986901364*

Lu, C.-C. & Tseng, V. S. (2009), 'A novel method for personalized music recommendation', *Expert Systems with Applications* **36**(6), 10035 – 10044.
URL: *http://www.sciencedirect.com/science/article/pii/S0957417409000591*

McCown, W., Keiser, R., Mulhearn, S. & Williamson, D. (1997), 'The role of personality and gender in preference for exaggerated bass in music', *Personality and Individual Differences* **23**(4), 543 – 547.
URL: *http://www.sciencedirect.com/science/article/pii/S0191886997000858*

McCrae, R. & Costa, P. (1989), 'Reinterpreting the myers-briggs type indicator from the perspective of the five-factor model of personality', *Journal of Personality* **57**.

McCrae, R. R. & John, O. P. (n.d.), 'An introduction to the five-factor model and its applications', *Journal of Personality* **60**(2), 175–215.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-6494.1992.tb00970.x*

McPherson, M., Smith-Lovin, L. & Cook, J. M. (2001), 'Birds of a feather: Homophily in social networks', *Annual Review of Sociology* **27**(1), 415–444.
URL: *https://doi.org/10.1146/annurev.soc.27.1.415*

Melville, P., Mooney, R. J. & Nagarajan, R. (2002), Content-boosted collaborative filtering for improved recommendations, *in* 'Eighteenth National Conference on Artificial Intelligence', American Association for Artificial Intelligence, Menlo Park, CA, USA, pp. 187–192.
URL: *http://dl.acm.org/citation.cfm?id=777092.777124*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. (2013), Distributed representations of words and phrases and their compositionality, *in* C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K. Q. Weinberger, eds, 'Advances in Neural Information Processing Systems 26', Curran Associates, Inc., pp. 3111–3119.
URL: *http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf*

Moore, J. C. M. K. (2012), 'The influence of personality on facebook usage, wall postings, and regret. computers in human behavior', **28**, 267–274.

Musto, C., Semeraro, G., Lops, P., de Gemmis, M. & Narducci, F. (2012), Leveraging social media sources to generate personalized music playlists, *in* C. Huemer & P. Lops, eds, 'E-Commerce and Web Technologies', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 112–123.

Nanopoulos, A., Rafailidis, D., Symeonidis, P. & Manolopoulos, Y. (2010), 'Musicbox: Personalized music recommendation based on cubic analysis of social tags', *IEEE Transactions on Audio, Speech, and Language Processing* **18**(2), 407–412.

Narducci, F., de Gemmis, M., Lops, P. & Semeraro, G. (2017), Recommender systems in the internet of talking things (iott), *in* 'Proceedings of the Poster Track of the 11th ACM Conference on Recommender Systems (RecSys 2017), Como, Italy, August 28, 2017.'.

Nielsen, J. (1995), '10 Usability Heuristics for User Interface Design'.
URL: *http://www.nngroup.com/articles/ten-usability-heuristics/*

Pearson, J. L. & Dollinger, S. J. (2004), 'Music preference correlates of jungian types', *Personality and Individual Differences* **36**(5), 1005 – 1008.
URL: *http://www.sciencedirect.com/science/article/pii/S0191886903001685*

Picard, R. W. (1997), *Affective Computing*, MIT Press, Cambridge, MA, USA.

Rawlings, D. & Ciancarelli, V. (1997), 'Music preference and the five-factor model of the neo personality inventory', *Psychology of Music* **25**(2), 120–132.

Rosen, P. & Kluemper, D. (2008), 'The impact of the big five personality traits on the acceptance of social networking website', **2**.

Schrammel, J., Köffel, C. & Tscheligi, M. (2009), Personality traits, usage patterns and information disclosure in online communities, *in* 'Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology', BCS-HCI '09, British Computer Society, Swinton, UK, UK, pp. 169–174.
URL: *http://dl.acm.org/citation.cfm?id=1671011.1671031*

Serban, I. V., Sankar, C., Germain, M., Zhang, S., Lin, Z., Subramanian, S., Kim, T., Pieper, M., Chandar, S., Ke, N. R., Mudumba, S., de Brébisson, A., Sotelo, J., Suhubdy, D., Michalski, V., Nguyen, A., Pineau, J. & Bengio, Y. (2017), 'A deep reinforcement learning chatbot', *CoRR* **abs/1709.02349**.
URL: *http://arxiv.org/abs/1709.02349*

Singhal, A., Sinha, P. & Pant, R. (2017), 'Use of deep learning in modern recommendation system: A summary of recent works', *International Journal of Computer Applications* **180**(7), 17–22.
URL: *http://www.ijcaonline.org/archives/volume180/number7/28811-2017916055*

Symeonidis, P., Ruxanda, M., Nanopoulos, A. & Manolopoulos, Y. (2008), Ternary semantic analysis of social tags for personalized music recommendation, *in* 'In Proceedings of ISMIR '08', pp. 219–224.

Turing, A. M. (1995), Computers & thought, MIT Press, Cambridge, MA, USA, chapter Computing Machinery and Intelligence, pp. 11–35.
URL: *http://dl.acm.org/citation.cfm?id=216408.216410*

W. Allport, G. & S. Odbert, H. (1936), 'Trait-names: A psycho-lexical study', **47**.

Weizenbaum, J. (1966), 'Eliza—a computer program for the study of natural language communication between man and machine', *Commun. ACM* **9**(1), 36–45.

Wu, H., Zhang, Z., Yue, K., Zhang, B., He, J. & Sun, L. (2018), 'Dual-regularized matrix factorization with deep neural networks for recommender systems', **145**, 46–58.