Emotion learning with a chatbot

Proof of Concept

Luis Mienhardt and Till Holzapfel

Target group

The target group of our application can be characterized by the following attributes. First, we target persons that like the board game diplomacy [1]. Second and more important in the context of our seminar, we target persons that want to improve their emotional communication, want to detect and udnerstand the emotional reactions of others and want to learn appropriate emotional reactions. We might specify even further and say, that our target group consists of persons with difficulties in navigating social life, that have difficulties to understand the emotions of others or difficulties with proper emotional reactions. The primary example might be a person with an Asperger syndrome.

Learning scenario

You are sitting alone at home and think about the last board game evening. As you and your friend are interested in cognition and the mind in general, you often play psychological games like “Werewolf” together. But as you are bad in detecting others emotions you always kill the wrong people and your friends get angry with you. You have similar problems at work, as you misinterpret the mood of your boss and in your personal life when you are dating someone. You decide, that you have to improve your perceptibility and understanding of emotions and how to appropriately express them. In theory you know what to do, because you are seeing a therapist, but you need some practice. As you are scared to hurt other people, you need a way to do this on your own.

Requirements

From our target group and learning scenario, but also from the requirements of our course, we extracted this requirements:

- the requirements of our course is to create a language controlled interface that does make use of artificial intelligence in its implementation;
- correct detection of emotion and appropriate emotional reaction is the learning goal of our target group and it should be rewarding to learn them
- we want our users to learn emotional reaction in a playful way, as this helps to overcome anxiety in interacting with others.

Software

The requirements we defined led to the decision to use the Telegram Chatbot API as language controlled interface for our application and the IBM Watson Tone Analyzer for emotion detection. As we wanted our application to be easy accessible for our target group, we decided to develop it as an chatbot. This also made it possible for us to fulfill the course requirement of a language controlled interface. Telegram was used, as it supports the development of chatbot in their messenger. The python-telegram-bot API [3] had the advantage, that it enabled us to use python as programming language that we already had experience with. We added the IBM Watson Tone Analyzer, as we needed a program for emotion detection to analyze the expressed emotions of our target group, but also to create emotional adequate text as response. As we knew, it would be out of scope to program an analysis tool, we decided to make use of IBM's service.

Telegram chatbot API

Telegram is a messaging app that is used for written online communication. It also provides an interface for chatbot communication and encourages the creation of chatbots by external developers. Telegram provides a chatbot API that can be used for the creation of chatbots [2]. Because the telegram API relies on HTTPS requests, we to use the API of python-telegram-bot [3]. This and our experience with it lead to the decision to use Python as programming language.

IBM Watson Tone Analyzer

We also used the IBM Watson Tone Analyzer in our application [4]. The IBM Watson Tone Analyzer is a cloud service of IBM to analyze the emotions expressed in a text. The text analysis is able to detect the dimensions *anger*, *fear*, *joy*, *sadness*, *analytical*, *confident* and *tentative*. How exactly the IBM Watson Tone Analyzer analysis works and the dimension values are produced is not disclosed.

Design

The design of our application had two goals. First, we wanted a game with mechanics reminding of the diplomacy board game [1]. Second, we wanted a language controlled interface for the game, making use of the IBM Watson Tone Analyzer to create a believable agent that expresses and detects emotions.

Didactic

Our didactical design focuses on three pillars. We wanted a game-like training application to teach emotion detection and adequate emotional response. This training application should take the form of a realistic and believable conversation with a chatbot. It was important for us, that this conversation provided feedback for the players emotional reaction.

Our first didactic principle based on the requirements for a safe-space for emotional exploration. We evolve this principle by balancing the reward for correct emotion detection and adequate responses against the drawback of loosing the game. The reward should be high enough to motivate the player exploring possible reactions of the chatbot, while the drawback of loosing should feel hard enough to discourage the player from taking the training to lightly. In our application we use direct feedback to motivate the player and make it a necessity to cooperate with the chatbot to make the player take the training serious.

Our second didactic principle is a realistic and believable conversation that make sure, that the emotional learning is transferable to real life situations. The didactic principle is realized through appropriate emotional responses by the chatbot. The emotion analysis of the users messages by the IBM Watson Tone Analyzer is one of the methods we used to create this responses. A further possibility is to implement natural language generation, which we propose for future improvement.

Interaction

As we wanted the conversation to be realistic, but the application to feel game-like we decided to split the possible interactions into two phases. A game phase that is characterized by menu interaction familiar from other video games and a conversation phase that contains an artificial intelligence guided emotional conversation.

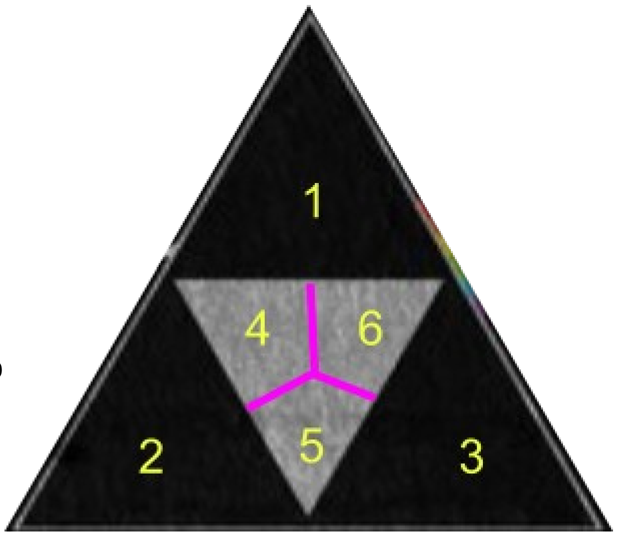
The game-phase was designed with video game menu mechanics in order to keep the interactions as intuitive as possible. We used the telegram API’s Keyboard Markups to build a menu through which one can navigate the game. The menu allows the user to read the tutorial, start a conversation with one of its two opponents, decide on a move, end the turn and quit the game. Through this interface the whole game can be controlled.

The conversational-phase was more complex to design. We had to weight of the requirement for believability with that for direct feedback. Detailed feedback during the conversation-phase would mean to break the believable setting. To keep the conversation believable we came up with a feedback that is incorporated into the conversation. In our example this was an generic answer depending on wether the player responded adequately or did not. Ideally, we would create an adaptive response to the players message that incorporates feedback relating to this message. Additional natural language processing tools are necessary for this.

Technical

The game-phase consists out of a menu the user can navigate through. The application provides information through text messages and pictures send by the chatbot. This format was chosen, as it resembles the navigation in a computer game and is easy to understand.

The game rule design tries to replicate the playing feel of a diplomacy game but in a simplified form. The game consists out of a board with 6 tiles, as in the picture to the right. The player starts on field 1, the two opponents on field 2 and 3. Every turn each player can move by one field. The move is only successful, if the field is free and no other player wants to move on the same field this turn. Otherwise the player stays on his field. The game is won by two players together, if they both move on the field another player is occupying. This rules make cooperation necessary to win the game. Thus the player is motivated to convince his virtual opponent to cooperate with him and win the game.



The conversational-phase design made sure, that it is possible to convince a virtual opponent to cooperate. In the conversational-phase with the virtual opponent, the player is asked to send a message. It is analyzed by the IBM Watson Tone Analyzer. A score between zero and one is given for the emotional dimensions of *anger*, *fear*, *joy*, *sadness*, *analytical*, *confident* and *tentative*. Each virtual opponent has his own preferred emotional set-up in these dimensions. The analysis of the player message is matched with this set-up. A score is calculated according to rules you can see in the code snippet 6 below. The attitude of the virtual opponent is then updated by taking the mean of the prior attitude and the calculated score. Finally, the virtual opponent sends a feedback message to the player, whether his attitude has in- or decreased and which emotion had the greatest difference from the preferred set-up. Then the player can send a new message. Through this design it is necessary for the player to detect the emotional set-up of the virtual opponent in order to win the game.

The virtual player does additionally react to the player with a message that express the emotions opposite of his preferred set-up. This is supposed to give the player the impression of an emotional agent. We choose the opposite, as it seemed most believable that for example someone tentative is looking for a confident person to trust and vice versa.

Implementation

Pytho-telegram-bot API

We used the python-telegram-bot API to create the structure of the game-phase. 6 You can see its implementation in the attached code snippets. The API uses conversation states that the user pass while using the chatbot. The API identifies a text message send by the user and compares it to the states conversation handlers. If there is a handler that can handle the message, the function that accompanies the handler is executed. We realized the menu-like feeling of our game, as we only handle messages that are predefined by string variables that we use for the text of our menu buttons. If one of the menu buttons is pressed, the chatbot reacts. The only state where this is different is the BOTCON state. In this state the user is in the conversational-phase and should freely interact with the chatbot. This is done by letting the chatbot react to every message the user sends. Therefor we have a message handler with no requirements at all.

Game rules

The rules that are important for the calculation of the moves and to determine if the player has won are implemented through a single function, covering all possible moves. You can find the game rules explained in the technical design.

Virtual Players

The virtual players have been implemented as class objects 6. Every instance can have its own attributes, that are important to save their current attitude to the player or there position on the board. As you can see, the player message evaluation is a function of the class object. This made it easy to store the calculated score in the classes attributes.

Database

For the database we used a python dictionary, this dictionary adds an entry with the players telegram ID at the beginning of a game. Subsequently all information concerning that player are saved inside his dictionary entry. Part of his entry are the ki_bot classes for his virtual opponents, his desired move and the fields occupied by the player and his opponents. At the end of the game the data is deleted, such that it does not interfere with a new started game.

Evaluation

Our main goal was to implement an application that creates an atmosphere were the user can experiment with his emotional behavior and his emotion detection ability. Nevertheless we wanted to keep it necessary to learn emotional behavior in order to succeed. This goals were achieved. During our tests we recognized, that it is absolutely necessary to try to conform to the emotional set-up of the virtual opponents, while is was still fun to experiment with their reaction.

What we could not implement are fully adaptive and thus believable reactions of the virtual opponents. In our current state the responses are generically generated. The believability of the virtual opponents would thus profit from a more sophisticated natural language processing and production tool.

The part of our program that was most difficult to implement was the emotion evaluation. This is mainly due to the fact, that the IBM Watson Tone Analyzer does not keep with its promises. He is not reliable, neither for short nor for longer messages. This would be fine, if the analysis would be more transparent, such that an improvement could be implemented. But IBM is keeping the methodology completely enclosed. This meant, we had to guess which messages would provide which emotional evaluation outcome and thus were not able to improve the performance ourselves. In future implementations of the program we would replace the IBM Watson Tone Analyzer, with a more transparent or a self created tool for emotion detection.

[1] „Play Diplomacy Online ::: web version of the classic Diplomacy board game“. https://www.playdiplomacy.com/help.php (zugegriffen Apr. 21, 2020).
[2] „Bots: An introduction for developers“. https://core.telegram.org/bots (zugegriffen Apr. 21, 2020).
[3] „python-telegram-bot“. https://python-telegram-bot.org/ (zugegriffen Apr. 21, 2020).
[4] „Watson Tone Analyzer - Overview“, Aug. 15, 2019. https://www.ibm.com/cloud/watson-tone-analyzer (zugegriffen Apr. 21, 2020).

```
con_handler = ConversationHandler(
        entry_points = [CommandHandler('start', start_game)],
        states = {
            START: [MessageHandler(Filters.regex(phrase_start),start_new_game),
                    MessageHandler(Filters.regex(phrase_tutorial),tutorial),
                    MessageHandler(Filters.regex(phrase_end_game),done),
            ],
            TURN: [MessageHandler(Filters.regex(phrase_bot_A_reply),enter_bot_con),
                    MessageHandler(Filters.regex(phrase_bot_B_reply),enter_bot_con),
                    MessageHandler(Filters.regex(phrase_quit_game),quit_game),
                    MessageHandler(Filters.regex(phrase_end_turn),end_turn)
            ],
            BOTCON:
[CallbackQueryHandler(first_response_bot_con,pattern=key["callback_data"]) for key in
keyboard_moves]
                    +
                    [MessageHandler(Filters.regex(phrase_return),leave_bot_con),
                    MessageHandler(Filters.regex(""),bot_con)
            ],
            RESOLVETURN: [MessageHandler(Filters.regex("^continue$"),evaluate_commands)]
                    +
                    [CallbackQueryHandler(evaluate_commands,pattern=key["callback_data"])
for key in keyboard_moves
            ]
        },
    )
```

Conversation Handler

```
class ki_bot:
def evaluate_emotions_answer(self,update,context):
        evaluation = 0
        emotions_detected = extract_emotions(message)
        matched_emotions = {}
        matched_emotions_abs = {}

        for emotion in emotions_detected:
            matched_emotions[emotion] = self.values[emotion]-emotions_detected[emotion]
            matched_emotions_abs[emotion] = abs(self.values[emotion]-
emotions_detected[emotion])
            if abs(matched_emotions[emotion]) <= 0.5:
                Evaluation += 1

        evaluation = evaluation / 7
```

Emotion evaluation