Chatbots for Daily Conversations and Song Recommendations

Huijun Hao
hhao004@ucla.edu
UID: 105863846

Jingwei Huang
j284huan@ucla.edu
UID: 805525466

Syed Nawshad
snawshad@ucla.edu
UID: 805871517

Yuefu Liu
liuyf@ucla.edu
UID: 405711899

Abstract

Abstract - In this paper, we explore chatbots: different implementations, models and uses. In our work, we have found that there are three approaches to designing a chatbot. The first model is the naive method where a model is trained on specific responses. The second, the sequential model is more flexible than the naive model but performs poorly. Finally we looked into pre-trained models, particularly DialoGPT, which is pre-trained on reddit discussion threads and is able to perform the most humanlike conversation and has the most flexibility in its use.

1. Introduction and Motivation

1.1. Industry Motivation

Chatbots are typically used by businesses to perform simple customer service tasks and automate conversations. The customer service tasks for chatbots include opening transaction reports, navigating websites, and performing common requests. These simple chatbots utilize a "Naive" method which will be used in this paper.

More complex chatbots that can perform daily human-like conversations are used for customer outreach tasks like setting up appointments, product promotion and public engagement. These utilize large datasets, pre-trained models and transfer learning. These complex models will also be utilized in this paper, particularly the DialoGPT model.

1.2. The Naive Method

The naive method solves a multi-class classification problem by utilizing a linear neural network that trains on a small or large intents file. The intents file contains input tokens and automated responses, which act as the classes for those tokens. Intents files exist in public space for chatbots to train on. This makes the "naive" method flexible from the design stage. In addition, this paper found that the naive method also achieved very small loss. While these benefits were observed, the naive model had major issues.

While the Naive model is customizable for any task from the design stage, once it is trained on the task designed in the intents file, the chatbot can no longer operate for any other application. For example if an intents file was written to take coffee orders, it can only do that task and nothing further. This is in stark contrast to more complex chatbot systems like Siri and Alexa which can perform daily conversations and be asked many and varying random questions.

1.3. Sequential and Pre-trained Models

While the naive method can accomplish a large number of specific tasks, a single naive model won't be dynamic enough to handle multiple types of tasks. For instance, if a chatbot is specifically designed for ordering coffee, the intents file for that chatbot will be written with common phrases for ordering coffee. This type of model can't generalize to other tasks like daily conversations.

This is where sequential and pre-trained models come in, models that can be applied to more general tasks like daily conversation. The sequential and pre-trained models are trained on large datasets like typical neural network models. Because of this they are more flexible.

2. Workflow of Project

Our project implements a chatbot that could handle human-like daily conversations and give music

recommendations on request based on the emotion analysis of the user's chat history.

2.1 Chatbot

To create a chatbot for daily conversation, we first create our own intents file and train our model based on it. We also tried another pre-trained model DialoGPT. The specific workflow for this part is introduced in the following sections.

2.1.1 Create Intents File

In order for the naive method to work, an intents file needs to be created. The intents file is composed of three subcomponents: tags, patterns, and responses. Patterns are the tokens that the chat bot looks for from the user, which is also the input. The responses attached to the same tag that patterns are attached to are what the chatbot responds with and are thus the output of the model.

2.1.2 Processing Input

When a user types a message to the chatbot, or in other words provides input to the chatbot model, the chatbot first pre-processes the input. The first step of the pre-processing pipeline is tokenizing the input sentence into words. Then, using nltk's word stemming function, obtaining the stem of each of those words, and then finally a word embedding via a bag of words. After this preprocessing pipeline, the data can be input into a neural network.

2.1.3 Model

Naive Model: For the naive model, we use nltk package for word tokenization and create a simple neural network with epoch number as 1000, batch size as 8, learning rate as 0.001 and a hidden layer with 8 units.

Sequential Model: We created a 3-layer sequential model with 96% model accuracy. First layer has 128 neurons and the second layer has 64 neurons. The third output layer contains the number of neurons that is equal to the number of intents to predict output intent tag with softmax function. We used the relu activation function for the first and the second layers and set the dropout probability to 0.5.

DialoGPT: DialoGPT is a fine-tuned pre-trained model for multi-turn conversations. Unlike the above two models which choose a predefined response, DialoGPT is a generative model on the basis of the GPT-2 model architecture. The model is trained on 147M multi-turn dialogue based on Reddit discussion thread. We loaded the pre-trained model and the tokenizer of DialoGPT-medium from AutoTokenizer. We encoded the sentences and combined the history sentence encoded embeddings as the input.

2.1.3 Prediction

Naive Model: The Naive model, unlike the other two models, is purely trained on the intents file that are typically created for the specific application. The inputs, tags and patterns and outputs, responses are created rather than taken from a large dataset and formulated into a multi-class classification problem.

Sequential Model: Our sequential model takes the word embeddings of the dialog created by the bag-of-word model and generates multi-class classification predictions. The predicted results are sorted from high probability to low probability and predictions lower than an error threshold (we set 0.25) are filtered out. Then the result ranked top will be used to map to a class defined in our intents file and the chatbot response is then randomly picked from all the replies we have pre-written in the corresponding class.

DialoGPT: For DialogGPT, the multi-turn dialog within a dialog session is concatenated into a long text and the response generation task is treated as language modeling (Zhang et al., 2020 [1]). To generate open-domain text, DialoGPT implements a maximum mutual information (MMI) scoring function. MMI employs a pre-trained backward model to predict source sentences from various given response (P(Source|target)). A set of hypotheses of the replies is first generated using top-K sampling. The probability of P(Source|Hypothesis) is calculated for each hypothesis and the hypotheses are re ranked based on their probabilities. Responses are then selected based on their ranking.

2.2 Song Recommendation

The song recommendation part mainly includes two steps, we first use two kinds of emotion analyzer to classify the emotion tag based on the conversation history and then post the tag to the last.fm API to get a song recommendation list.

2.2.1 Emotion Analyzer

IBM Tone Analyzer API: We at first wanted to use the IBM Watson Tone Analyzer to detect emotional tones from the dialogue. However, the IBM Watson Tone Analyzer has stopped the service and we turned to IBM Natural Language Understanding API. It can handle multiple tasks including classifying tones and emotions. We merge the history sentences and send it to the analyze method as parameter text. Natural

Language Understanding returns a JSON object with a classifications field containing class_name and confidence score. The class_name stores the emotion tag among {excited, frustrated, impolite, polite, sad, satisfied, sympathetic} and the confidence score represents the probability of being classified as class_name. The classification objects are sorted in the descending order by confidence score, so we choose the first class_name as our emotion tag.

Robertuito-Emotion-Analysis: This model is a Python toolkit for sentiment analysis and social NLP tasks. It is designed based on RoBERTa model architecture. We import the create_analyzer with the task, emotion classification, as our analyzer. The analyzer produces a prediction dictionary with the key as emotion and value as probability. The dictionary is sorted by the probability in descending order and chooses the first one as the emotion prediction. The emotion label list is {joy, surprise, disgust, sadness, fear, anger, others}.

Twitter-Roberta-Base-Emotion: It is a RoBERTa based model trained on over 58M tweets. We tokenized all history sentences and sent them to the model. This model only has 4 emotion labels, {joy, sadness, anger and optimism}. The output is a list of probability scores and we will choose the one with the highest probability. Although there are fewer labels in this model compared to the previous ones, this model has a higher prediction accuracy, which will be discussed in later sections.

2.2.2 Song API

To get a song recommendation list, we used the Last.fm song API. Last.fm provides free access to their music dataset and we post the request with an emotion tag through APIs. Last.fm song API returns a dictionary with keys as names of songs and values as song urls. We choose the top 5 song options as our recommendation for users.

3. Application

3.1 Frontend UI

We used React as our frontend to implement the user interface.

Daily Conversation: We designed a chat window to interact with our chatbot. Users can type in the sentences in the bottom input box. The conversation example is shown in Fig 1.

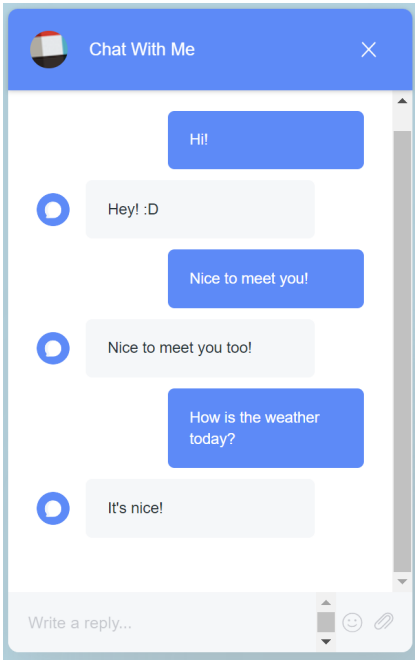

Fig 1. Dialogue example with the chatbot in chat window

Song recommendation: Press the "Start Song Recommendation" button below the chat window, then our chatbot will provide a prediction of the user's emotion and a recommendation list of songs with both names and urls.

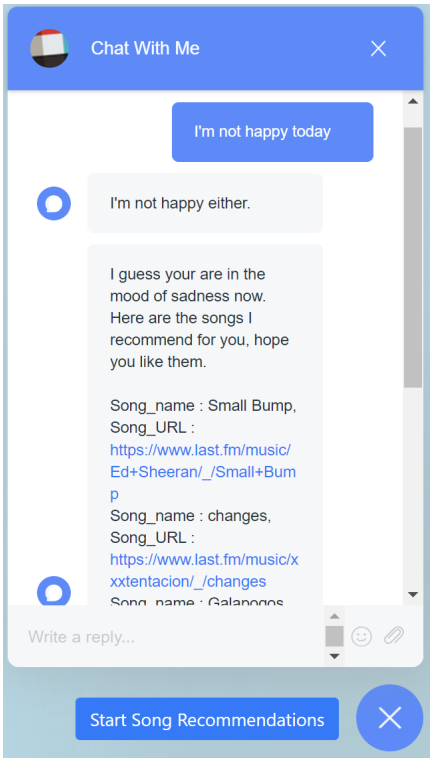

Fig 2. Song recommendation by the chatbot in chat window

3.2 Backend API

We've tried to fit several combinations of our chatbot models and emotion classification models to the backend, that is, the sequential model with all three emotion analyzers and the DialoGPT model with all

three emotion analyzers. Based on their performance, we choose the combination of the DialoGPT model and the Twitter-Roberta-Base-Emotion model as our final version.

4. Comparison and Novelty

For chatbot, we compared three models, including different dataset size and accuracy. The emotion analyzers were compared based on the results of emotion classifications and song recommendations. According to the analysis of the strengths and weaknesses of different models, we select the model that best analyzes the mood.

4.1 Model Comparison

Naive model: This is the most basic model with the smallest intents file and simplest structure of the neural network, which can only carry out simple conversations and contains only a little information. Prediction is not always accurate.

Sequential Model: The sequential model is constructed based on the naive model. More layers are added and the model prediction is more precise by using the activation function relu and dropout in the hidden layer. Although the structure of the model has been adjusted, the contents of the intents file are limited, so the chatbot can only select random responses from the file with a higher probability of prediction. As a result, the responses are not correct all the time.

DialoGPT: DialoGPT was trained on 147M multi-round conversations based on Reddit discussion threads. So it contains more responses, and with it the chatbox can communicate with users more smoothly. The domain of conversation content is also more extensive. Also, this model uses a different forecast approach from the naive model and sequential model. MMI is implemented as the scoring function which employs a pre-trained backward model to generate rankings of hypothetical responses. Overall, DialoGPT has the best performance in the daily conversation task.

4.2 Emotion Analyzer Comparison

IBM Tone Analyzer API: The advantage of this API is it can handle a variety of tasks, including classifying tones and sentiments analysis. But at the same time, the disadvantages are also significant**,** classification stores the emotion tag among {excited, frustrated, impolite, polite, sad, satisfied, sympathetic}. We found that the polite, impolite, sympathetic, and other characteristics of the tags are not likely to be used to classify music. This suggests that some of the emotions may be misclassified and thus cannot recommend the best-fitted songs to users.

Robertuito-Emotion-Analysis: This analysis is specifically used for sentiment analysis so tagging is more straightforward and the classification is more comprehensive. This makes the classification of emotions more reliable than the IBM Tone Analyzer API. The problem with Robertuito-Emotion-Analysis is that it cannot identify negative words. For example, when a user says "I'm not happy today", only happy is correctly placed and music related to "happy" is recommended. This leads to the possibility that users may get songs that are the opposite of their mood. However, we did not find an analysis that both identifies negatives and includes a lot of emotion tagging.

Twitter-Roberta-Base-Emotion: The RoBERTa-based model is trained on over 58 million tweets and has only four sentiment tags {happy, sad, angry, and optimistic}. A large amount of data can make the classification more reasonable, and simple tagging can make the classification clear. And it can identify the negative words in the conversation, such as not enjoy, and unsatisfied can be accurately identified to improve the accuracy of prediction.

4.3 Novelty

Through our research and studies, we found that existing applications can usually perform only one task, such as a chatbot that can have a daily conversation with the user. Or a chatbot can provide song proposals based on the consumer's music preferences and their mood at the moment. Chatbots that can only do a single task can cause boredom and tedium for the user. So based on the application shown earlier, our chatbot can do two tasks at once. First, it satisfies the task we need to do as a daily chatbot, which is to communicate with users and suggest songs based on their emotional feedback, allowing them to accept the songs they like in their existing sentiments.

5. Conclusion and Discussion

In summary, we built an application combining two main tasks, a chatbot for daily conversations and song recommendation. We compared different models both in generating dialogue responses and emotion analysis. Among all the combinations, we choose the DialoGPT and Twitter-Roberta-Base-Emotion due to their flexibility and accuracy in use. There are still some limitations in our

application. For future work, the emotion classification labels can be expanded and we can fine-tune the model for higher accuracy in emotion prediction.

6. Reference

[1] Zhang, Y., Sun, S., Galley, M., Chen, Y.-C., Brockett, C., Gao, X., Gao, J., Liu, J., & Dolan, B. (2020, May 2). *Dialogpt: Large-scale generative pre-training for conversational response generation*. arXiv.org. Retrieved June 8, 2022, from https://arxiv.org/abs/1911.00536

[2] Music&me Chatbot Song Recommender System: https://github.com/Srishti20022/Music-me-Chatbot_song_recommendor_system-

[3] Simple Chatbot Implementation with Pytorch: https://github.com/python-engineer/pytorch-chatbot

[4] Pérez, J. M., Giudici, J. C., & Luque, F. (n.d.). *Pysentimiento/pysentimiento: A python multilingual toolkit for sentiment analysis and social NLP tasks*. GitHub. Retrieved June 8, 2022, from https://github.com/pysentimiento/pysentimiento/