Weather Chatbot

09/27/2019

Jiacheng He

Contents

1.Project Overview

In recent years, Natural Language Processing, as a subfield of combination of Artificial Intelligence (AI) and Linguistic, has become more and more important in the processing of human-computer interaction. The purpose of natural language processing is to enable people and machines to communicate through natural language, so that human manipulation of the machine will become very simple and practical. However, the human language is often ambiguous, contextual and implicit and the computer program itself could not clearly recognize the true semantics of the human language. Therefore, we need to analyze and process natural language through a series of methods, let the machine understand the meaning that human users want to express and make reasonable feedback automatically.

2.Knowledge Review

2.1. Simple Echo Chatbot

A simple Echo chatbot will give users a very direct feedback. For instance, if a user type in “hello”, the echo chatbot could directly make the response “hello” as well. If you want to make the echo chatbot more personalized and interested, the program can build a dictionary containing many alternative responses and randomly select one response for the user. Furthermore, the program could create rules of dictionary to match the input statements through the regular expression to get corresponding response for users.

2.2. Intention Recognition

The task of Intention Recognition is to figure out the intention from human languages. It seems very simple, but in practice, human language often contains implicit, contextual or ambiguous statements. It is very difficult to extract intent from these statements.

This research mainly introduces three methods for intention recognition:

- The first method is really simple, it let the program creates a dictionary with some intents as keys and corresponding regular expressions as values to identify the intent of a statement by searching through the keywords to match the correct intent. Although this method is very simple, the range covered by regular expressions is very limited and does not easily match the true intent. For example, two sentences "I agree with this" and "I disagree with this" are easily recognized as the same intent because the overall structure of the two sentences is very similar.
- The second method uses the word vector to extract the words in each sentence to compare the similarities of word vectors between two sentences to determine the intent of the sentence. This method uses the Support Vector Machine (SVM) to analyze the word vector features using a classifier and uses the data and corresponding intent to train the Support Vector Classifier (SVC), and then, use the SVC to predict the intent of the target statements. The intention of this method is more accurate and less restrictive, but this method often requires a large number of training data areas to train Support Vector Classifier, which makes this method not very practical.
- The third method also uses the concept of the word vector, but it uses a package called “rasa_nlu" to do machine learning on a series of json data to analyze the intent of each sentence in the data and train a interpreter to extract the intent of any sentence. After training the interpreter, the program could use it to directly extract the intention of the

input message by comparing the similarity of input sentence and sentences from json data. This method is relatively practical because it uses mechanical learning to distinguish the intent of similar statements as long as the training data is sufficient. In addition, the training data does not need to be very large under certain circumstances such as this project due to it only requires searching for one type of stuff.

2.3. Named Entity Recognition

The task of Named Entity Recognition (NER) is to locate and classify name entity from unstructured text into predefined categories such as three major classes (entity, time and number) and seven subclasses (name, institution name, place name, time, date, currency, and percentage). The reason of why NER is used to extract the entities from the statement is that the keywords in the statement are an important basis for our judgment of semantics.

Similarly, we have been introduced with three NER methods:

- The first method also uses regular expressions, but this time we only need to extract named entities, so we only use regular expressions to analyze words instead of sentences. However, the limitations of regular expressions are still very large. For example, names of people, places, countries, etc. start with a capitalization, and words followed by a series of lowercase subtitles all have the same regular expression. This makes it difficult to distinguish which type of entity is extracted.
- The second method uses a python package called spacy, which is a written package that can help us extract the named entities in the sentence and classify them. It is very practical, but the entity class that can be extracted is relatively fixed, and it is often impossible to extract some specific entities such as the distinction between units and the

extraction of language. However, the extraction of names such as cities and countries is very effective.

- The third method still relies on the rasa_nlu package, which extracts and analyzes the entities you want in sentences from json data, so that the interpreter could understand that when the user enters a similar sentence, the entity to be extracted is the same one from in the sentence from json data. For instance, if json data has a sentence “what are available languages” and entities contains the word “languages”, then the program could extract the entity “languages” from a user input sentence “what languages are available” no matter with the index of word “languages” in the sentence. This method maximizes the accuracy of the extracted entities.

2.4. Database and API

Data support is very important in our project, because without a certain amount of data, we can't accurately query the information we want. In the research, we mainly introduced two forms of data acquisition, one is to extract data through the database established by ourselves, and the other is to obtain data through the network API link.

- **Database**: Building a database allows us to easily deal with more different kinds of questions without being limited to a fixed format. By creating a database with corresponding data, we can more easily match all the entities in the user-entered statements with the keywords in the database to get the most appropriate information we want. Only with the large amount of data support in the database, we have the ability to implement the query search function. In this research, we have been introduced the method of extracting data from a SQL database, the most common database. But building our own database is not very practical in our project, since our project requirement is to

be able to query a certain kind of things, which requires a lot of data support, and it is difficult for us to build such a database.

- **API**: The API is another way to extract data. This is also a way to extract data, it can easily get a lot of data through the URL link address and return in json format. Since the API obtains data through the URL link address, it has a little demand for the state of the network link, and if the server state of the API is unstable, it will cause us to fail to obtain data. Although this method is very convenient, it may fail to get data because you don't have a good network connection. And the security status of many free API data is very unstable since they do not require any authorization to access it.

2.5. Multiple rounds of dialogue & Negation

With the support of the data, we can achieve more dialogue functions instead of letting the user input a sentence and then the machine answers, such a boring chat. The first thing that can be achieved with data support is multiple rounds of dialogue. Our conversation can only be done in a question-and-answer manner without data support, but with data support, we can qualify entities to match the goals we want. For example, in the restaurant search process, we can limit the consumption of water, geographical location and other entities of the restaurant, and then use these limited steps to match the data to get the restaurant we want. In this process, we record the limits of the entities in each round of dialogue to narrow the results extracted from the database to find the most matching results. The main purpose of multiple rounds of dialogue is to let the program remember the restrictions on entities in your previous rounds of conversation.

In human-computer interaction, the negation semantics are very difficult to extract. At present, we decide whether an entity is negative or not only by checking whether there is "not" or "n't" before the entity. However, many words that negate the meaning, such as hate, cannot be

identified as negative, and cannot be judged as negative if the negative word is no longer an entity. Multiple rounds of dialogue involving affirmation and negation can more accurately identify the result for the same intent. So in my weatherbot project, I mainly use rasa_nlu to train statements containing negative meanings to achieve the identification of negative intent.

2.6. State Machine

The state machine is a very powerful concept. Through the change of state and the change of intention, we can time more functions in natural language dialogue. There are often different states and intentions in the process of communicating with each other in natural language. For example, when a person ask "how are you?", then he is in the state of greeting. Recording the difference between different states and intentions can effectively help the chatbot to distinguish the user's current actual intent. When people say "yes", the status of affirmative answers to different questions is different. We can't treat all affirmative answers as the same state, otherwise we can't let chatbot communicate in natural language. With the state machine, chatbot can analyze and react to the user's current actual intent through state and intent.

2.7. Pending Action

A pending action is a function implemented by the concept of state machine that allows the current state to be pending. Generally, when the items we want to ask are not available, we will inevitably withdraw from the current state, but we have not achieved the initial purpose. For example, when we want to buy a coffee and ask salesperson to order a latte, the salesperson tells you that there is no latte. If there is no pending action, we will inevitably enter the next state, such as leaving, but with the pending action, the salesperson can tell us “we do not have latte, would you like try mocha?”. In this way, we could get suggestion back and choose coffee again,

which means we stay in our last state. Following the same rule, the chatbot can implement the process of automatically making suggestions. Therefore, our chatbot could become more intelligent and give us a more natural and humanized chat process with pending action.

3.Program Implementation

After a month of learning, I designed a simple query chatbot, whose main function is to provide weather query service, which can check the current weather conditions and give corresponding feedback in real time at any place around the world.

In my program I got three states which are INIT, GET_WEATHER, and FINISH and shown in the following picture #1:

- INIT: initial state, which start the chat and do anything before user want to start searching the weather
- GET_WEATHER: getting weather state, which retrieving the weather information from API and wait the user to confirm or deny.
- FINISH: finishing state, which wait user to restart the search or exit program directly.

```
33  INIT = 0
34  GET_WEATHER = 1
35  FINISH = 2
```

Picture #1

Moreover, I set a policy rules using these three states and all of my defined intents from “weather_rasa.json” file. Since my program doesn't have many situations where I can make similar answers but have different intentions, I don't need too many state transitions to ensure the accuracy of the intent. The following picture#2 is my policy rules dictionary:

```
135     policy_rules = {
136         (INIT, "greet"): (INIT, random.choice(responses["greet"])),
137         (INIT, "ask_explanation"): (INIT, explain_text),
138         (INIT, "add_restriction"): (INIT, restrict_text),
139         (INIT, "start_searching"): (INIT, random.choice(responses["start_searching"])
140                                    .format("You can search by city names, coordinates and US postal code.\n"
141                                            "Hint: If you want to enter negative latitude or longitude, "
142                                            "please separate negative sign from the number. Such as \"- 10\".")),
143         (INIT, "search_weather"): (GET_WEATHER, weather_text),
144         (GET_WEATHER, "affirm"): (FINISH, random.choice(responses["affirm"])),
145         (GET_WEATHER, "deny"): (FINISH, random.choice(responses["deny"])
146                                .format("You may restart search or exit the program.")),
147         (FINISH, "ask_else"): (INIT, random.choice(responses["ask_else"])),
148         (INIT, "goodbye"): (FINISH, random.choice(responses["goodbye"])),
149         (GET_WEATHER, "goodbye"): (FINISH, random.choice(responses["goodbye"])),
150         (FINISH, "goodbye"): (FINISH, random.choice(responses["goodbye"]))
151     }
```

Picture #2

In this picture #2, it also shows all of my intent that defined in “weather_rasa.json” file, which are “greet”, “ask_explanation”, “add_restriction”, “start_searching”, “search_weather”, “affirm”, “deny”, “ask_else”, and “goodbye”. According to three states and nine intent, my chatbot will do the following steps, when it is running:

1. Firstly, waiting for a greeting sentence
2. Secondly, waiting for user ask explanation, add restriction or input start searching directly.
3. Thirdly, after start the searching system, user could directly ask weather with a city name, multiple city names, city names following by country name, latitude and longitude values, and US postal code.
4. Then, the chatbot replies to the weather in the corresponding city or region, and responds to an error message if the weather is not found.
5. After that you can enter a positive or negative answer to indicate whether you are satisfied with the results of the feedback.

6. Finally, whether or not you are satisfied with the results of the feedback, chatbot will provide you with two options, either re-searching or exiting the program.

In addition, you could exit the program at any state by saying something with intent “goodbye” to the chatbot. This is the principle of how my chatbot works.

In this project, I extracted the named entities in two ways, which are rasa_nlu and spacy. I extracted the restriction requirements, language type, unit type, latitude and longitude and zip code through rasa_nlu package methods. Then, the city name and country name from the input message are extracted through the spacy package. Please check the chatbotMain.py, line 86 interpret method for detailed steps on extracting entities.

As the picture #3 shown below, it shows how I access the API link address to obtain data by using requests package in python. I get my weather data from a site called “Open Weather Map” and I retrieve the json data of weather from the API address. For weather parameters in API response are shown below in picture #4. For more information about the usage of this API, please check the website https://openweathermap.org/current.

```
282  # this function get api link with my api key and set up the url properly to extract the weather info in json format
283  def get_weather_info(lat, lon, location, postal, lang, unit):
284      api_key = "8f418eec31daa053b149f4363269e47d"
285      url = "https://api.openweathermap.org/data/2.5/weather?appid={}".format(api_key)
286      if location != "":
287          url += "&q={}".format(location)
288      if lat != "" and lon != "":
289          url += "&lat={}&lon={}".format(lat, lon)
290      if postal != "":
291          url += "&zip={}".format(postal)
292      if lang != "":
293          url += "&lang={}".format(lang)
294      if unit != "":
295          url += "&units={}".format(unit)
296      response = requests.request("GET", url)
297      return response.json()
```

Picture #3

Parameters:

- `coord`
  - `coord.lon` City geo location, longitude
  - `coord.lat` City geo location, latitude
- `weather` (more info Weather condition codes)
  - `weather.id` Weather condition id
  - `weather.main` Group of weather parameters (Rain, Snow, Extreme etc.)
  - `weather.description` Weather condition within the group
  - `weather.icon` Weather icon id
- `base` Internal parameter
- `main`
  - `main.temp` Temperature. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.pressure` Atmospheric pressure (on the sea level, if there is no sea_level or grnd_level data), hPa
  - `main.humidity` Humidity, %
  - `main.temp_min` Minimum temperature at the moment. This is deviation from current temp that is possible for large cities and megalopolises geographically expanded (use these parameter optionally). Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.temp_max` Maximum temperature at the moment. This is deviation from current temp that is possible for large cities and megalopolises geographically expanded (use these parameter optionally). Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.sea_level` Atmospheric pressure on the sea level, hPa
  - `main.grnd_level` Atmospheric pressure on the ground level, hPa
- `wind`
  - `wind.speed` Wind speed. Unit Default: meter/sec, Metric: meter/sec, Imperial: miles/hour.
  - `wind.deg` Wind direction, degrees (meteorological)
- `clouds`
  - `clouds.all` Cloudiness, %
- `rain`
  - `rain.1h` Rain volume for the last 1 hour, mm
  - `rain.3h` Rain volume for the last 3 hours, mm
- `snow`
  - `snow.1h` Snow volume for the last 1 hour, mm
  - `snow.3h` Snow volume for the last 3 hours, mm
- `dt` Time of data calculation, unix, UTC
- `sys`
  - `sys.type` Internal parameter
  - `sys.id` Internal parameter
  - `sys.message` Internal parameter
  - `sys.country` Country code (GB, JP etc.)
  - `sys.sunrise` Sunrise time, unix, UTC
  - `sys.sunset` Sunset time, unix, UTC
- `timezone` Shift in seconds from UTC
- `id` City ID
- `name` City name
- `cod` Internal parameter

Picture #4

The last part about the running of the program is how to interact with the telegram in order to receive message and send message back. I retrieve the chatbot user input message by add getUpdates at the end of telegram API's URL address. The last element of getUpdates is the most recently message that user has sent. In addition I use the method in picture #5 below to get_updates with an offset value which is last update ID. Then, it will automatically clear the data in getUpdates method for URL address by setting the offset with last update ID. The reason I making this step is that the getUpdates data will keep growing and make the search to last updates very hard when it grows really large. Thus, it is always better to clear the old messages.

```
47  # this function retrieves a list of "updates" from telegram url
48  # if the offset is specified, it always does not receive any messages with smaller IDs than this
49  def get_updates(offset=None):
50      url = URL + "getUpdates?timeout=100"
51      if offset:
52          url += "&offset={}".format(offset)
53      content = get_url(url)
54      js = json.loads(content)
55      return js
```

Picture #5

Sending message back would be much more easier, I simply add sendMessage attribute to the end of API address with corresponding chat ID and a response message. The methods is shown below in the picture #6.

```
77  # this function sent message back which let the chatbot could speak to you
78  def send_message(message, chat_id):
79      url = URL + "sendMessage?text={}&chat_id={}".format(message, chat_id)
80      get_url(url)
```

Picture #6

4.Conclusion

After testing, my weatherbot has a certain level of intelligence to help users search for weather information with any place around the world. Finishing this research gave me a deeper understanding of artificial intelligence and natural language processing, which let me really understand how the intelligent voice system analyzes natural language in the process of human-computer interaction. This research project not only taught me a lot of new knowledge, such as intent recognition, named entity recognition, state machine, API calling method, but also gave me a deeper understanding of how the program implements functions step by step to increase the degree of intelligence. There are still many places in my program that are not very rigorous. I plan to continue to learn the techniques of natural language processing, and make my weatherbot more intelligent in the furture.

Before signing up for this research, I couldn't imagine that I could make a simple chatbot and provide a certain search function for it. I am very grateful to our instructor, Fan Zhang, for guiding us this month and letting us to be able to complete such a simple chatbot program independently.