Natural Language Processing Chatbot

Lingxiao Li
Department of Computer Science
University of Liverpool
Liverpool, United Kingdom
sglli17@student.liv.ac.uk

***Abstract*—This report aims to briefly analyze the working principle of natural language processing chatbot and introduce an implementation of a chatbot helping people to search stock information.**

Keywords—artificial intelligence, natural language analysis, Rasa-NLU, machine learning, chatbot

I. INTRODUCTION

With the development of computer science, Artificial Intelligence has been applied in an increasing number of areas. Natural language analysis is one of the import areas which aims to program computers to process and analyze large amounts of natural language data [1]. There are serval challenges in natural language processing which includes speech recognition, natural language understanding, and natural language generation. This report focuses on the application of natural language understanding and gives and example on how to implement it in a chatbot which helps to search the stock price using Python.

II. WORKING PRINCIPLE

A. *Echo Respond*

Elizabeth which is known as the first automated conversation and natural language processing program designed to provide an enjoyable and easy way into natural language processing [2]. The essence of Elizabeth is an Echobot. In other words, it extracts the content of users' inputs and responds using this content in a pre-defined format. There are three stages to achieve an Echobot.

The first stage simply returns the pre-stored format string and user's input message. Example:

User: Hello!

Bot: I can hear you! You said Hello.

The second stage is making the Echobot more personalized. The main purpose of this stage is to make the Echobot different from other chatbots; it also improves users' interests to use it. The first solution is creating a dictionary whose keys are pre-defined questions and values are the answers to these questions. Some of these answers are rhetorical questions which interest users. In this case, the Echobot can respond to some specific questions. The second solution is defining serval answers for a question and using the random function to select one of the answers randomly. In this case, the answer is more flexible and human-like. Example:

User: What's your name?

Bot: My name is Echobot.

The third stage is using a regular expression. There are three steps to achieve this stage. The first step is mating messages against known patterns. The second step is extracting key phrases. Specific content is extracted if they match the regular expression. The last step is transforming sentences grammatically. For instance, represents the pronouns of the phases. Example:

User: When you last watched the movie?

Bot: How could I forget when I last watched the movie?

B. *Intent Recognition and Entity Extraction*

It is important to understand the intent and extract entities in natural language. This part of work can also be achieved by regular expression. It is simpler and has higher efficiency than a machine learning approach. However, it is difficult to define a good regular expression. This report mainly focuses on the machine learning approach. The machine will understand users' intents by learning from the training data. To make the data readable to machine, it is essential to transform natural language into word vectors.

- Intent recognition

Supervised learning is used to recognise the intent in natural language. The training data is a set of word vectors with labelled intents. There are kinds of the algorithm can be used to train the model. For example, KNN (K nearest neighbour) and SVM (Support-Vector machine). The work efficiency depends on the size of the training data set.

- Entity Extraction

For extracting entities in natural language, the keyword is not suitable for extracting unknown entities. In other words, it is impossible to pre-define every entity. Others method such as spelling, context and after specific words need to be used. In addition, It is important to identify the roles entities are playing. For example:

I want a flight from Tei Aviv to Bucharest.

Show me flights to Shanghai from Singapore.

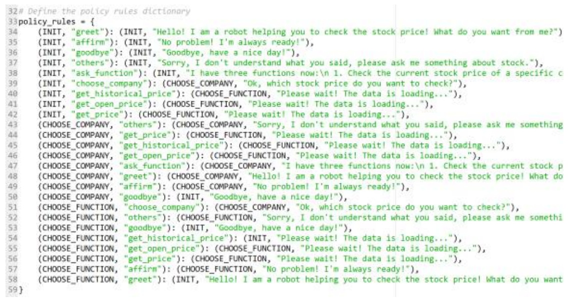
In this example, Tei Aviv, Bucharest, Shanghai and Singapore are all location entities while the role they played is different: place of departure and destination. Dependency analysis is another factor need to be considered. When considering the role of the entity plays, words related to it also need to be considered. More specific examples will be shown in the implementation of part.

III. Implementation

In this project, a chatbot to help users search stock information is created using Python. The IDE used in this project is Spyder from Anaconda. The process of implementing this chatbot is divided into five parts.

A. *Define State Rules*

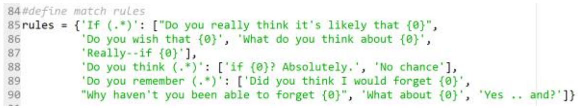
To achieve rounds of dialogue, defining the state of the chatbot is important. First, create a dictionary in Python; the key of the dictionary is a pair of values. The first value in the key is the current state of the chatbot; the second value is the intent of the current sentence. Since it is only a demo with not many functions, only three states are defined which are INIT, CHOOSE_COMPANY and CHOOSE_FUNCTION. The chatbot does not do any work about stock information search when the state is INIT. When the state is CHOOSE_COMPANY, users have already provided information about the name of the company which need to be searched. The state CHOOSE_FUNCTION means users have selected a specific function provided by the chatbot and get the result. The value of the dictionary is also a pair of values. The first value is the next system state; the second value is the reply should be sent to users. The code is as below shows:



```
# Define the policy rules dictionary
policy_rules = {
    (INIT, "greet"): (INIT, "Hello! I am a robot helping you to check the stock price! What do you want from me?")
    (INIT, "affirm"): (INIT, "No problem! I'm always ready!"),
    (INIT, "goodbye"): (INIT, "Goodbye, have a nice day!"),
    (INIT, "others"): (INIT, "Sorry, I don't understand what you said, please ask me something about stock."),
    (INIT, "ask_function"): (INIT, "I have three functions now:\n 1. Check the current stock price of a specific c
    (INIT, "choose_company"): (CHOOSE_COMPANY, "Ok, which stock price do you want to check?"),
    (INIT, "get_historical_price"): (CHOOSE_FUNCTION, "Please wait! The data is loading..."),
    (INIT, "get_open_price"): (CHOOSE_FUNCTION, "Please wait! The data is loading..."),
    (INIT, "get_price"): (CHOOSE_FUNCTION, "Please wait! The data is loading..."),
    (CHOOSE_COMPANY, "others"): (CHOOSE_COMPANY, "Sorry, I don't understand what you said, please ask me something
    (CHOOSE_COMPANY, "get_price"): (CHOOSE_FUNCTION, "Please wait! The data is loading..."),
    (CHOOSE_COMPANY, "get_historical_price"): (CHOOSE_FUNCTION, "Please wait! The data is loading..."),
    (CHOOSE_COMPANY, "get_open_price"): (CHOOSE_FUNCTION, "Please wait! The data is loading..."),
    (CHOOSE_COMPANY, "ask_function"): (CHOOSE_COMPANY, "I have three functions now:\n 1. Check the current stock p
    (CHOOSE_COMPANY, "greet"): (CHOOSE_COMPANY, "Hello! I am a robot helping you to check the stock price! What do
    (CHOOSE_COMPANY, "affirm"): (CHOOSE_COMPANY, "No problem! I'm always ready!"),
    (CHOOSE_COMPANY, "goodbye"): (INIT, "Goodbye, have a nice day!"),
    (CHOOSE_FUNCTION, "choose_company"): (CHOOSE_COMPANY, "Ok, which stock price do you want to check?"),
    (CHOOSE_FUNCTION, "others"): (CHOOSE_FUNCTION, "Sorry, I don't understand what you said, please ask me somethi
    (CHOOSE_FUNCTION, "goodbye"): (INIT, "Goodbye, have a nice day!"),
    (CHOOSE_FUNCTION, "get_historical_price"): (INIT, "Please wait! The data is loading..."),
    (CHOOSE_FUNCTION, "get_open_price"): (CHOOSE_FUNCTION, "Please wait! The data is loading..."),
    (CHOOSE_FUNCTION, "get_price"): (CHOOSE_FUNCTION, "Please wait! The data is loading..."),
    (CHOOSE_FUNCTION, "affirm"): (CHOOSE_FUNCTION, "No problem! I'm always ready!"),
    (CHOOSE_FUNCTION, "greet"): (INIT, "Hello! I am a robot helping you to check the stock price! What do you want
}
```

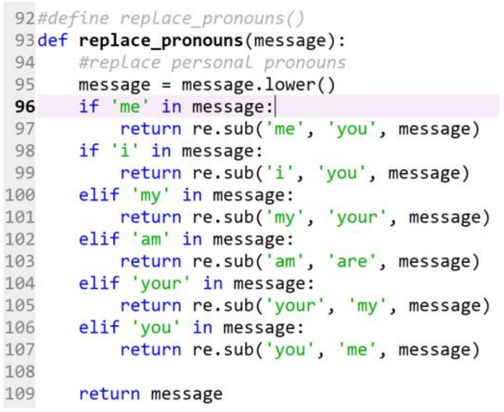
B. *Define Echo Respond*

To make the chatbot's respond more human-like. It is essential to define echo respond to respond to the question not related to the stock information. As mentioned in the working principle part, there are three functions, and one dictionary rules need to be defined to achieve echo response function. The match rules are defined using a dictionary with the specific questions as keys and standard respond format as values.



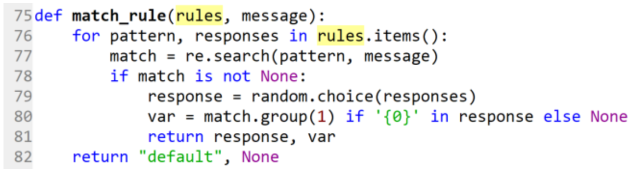
```
#define match rules
rules = {'If (.*)': ["Do you really think it's likely that {0}",
          'Do you wish that {0}', 'What do you think about {0}',
          'Really--if {0}'],
          'Do you think (.*)': ['if {0}? Absolutely.', 'No chance'],
          'Do you remember (.*)': ['Did you think I would forget {0}',
          "Why haven't you been able to forget {0}", 'What about {0}', 'Yes .. and?']}
```

The first function needs to be defined **replace_pronouns(message)** which can represent the pronouns in the input messages, codes are shown below:



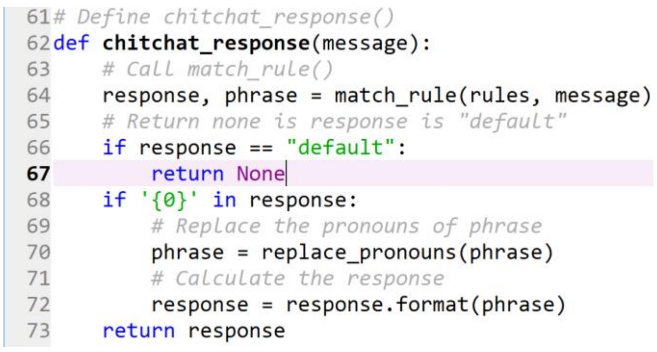
```
#define replace_pronouns()
def replace_pronouns(message):
    #replace personal pronouns
    message = message.lower()
    if 'me' in message:
        return re.sub('me', 'you', message)
    if 'i' in message:
        return re.sub('i', 'you', message)
    elif 'my' in message:
        return re.sub('my', 'your', message)
    elif 'am' in message:
        return re.sub('am', 'are', message)
    elif 'your' in message:
        return re.sub('your', 'my', message)
    elif 'you' in message:
        return re.sub('you', 'me', message)

    return message
```

The second function to be defined is **match_rule(rules, message)** which can match the content using pre-defined rules and extract the mating content and put the content in standard format response. Codes are shown below:



```
def match_rule(rules, message):
    for pattern, responses in rules.items():
        match = re.search(pattern, message)
        if match is not None:
            response = random.choice(responses)
            var = match.group(1) if '{0}' in response else None
            return response, var
    return "default", None
```

The last function to be defined is **chitchat_response(message)** which can call the other two functions and return the response.



```
# Define chitchat_response()
def chitchat_response(message):
    # Call match_rule()
    response, phrase = match_rule(rules, message)
    # Return none is response is "default"
    if response == "default":
        return None
    if '{0}' in response:
        # Replace the pronouns of phrase
        phrase = replace_pronouns(phrase)
        # Calculate the response
        response = response.format(phrase)
    return response
```

The chat below is the result of using these three functions.

```
USER : do you remember your last birthday
BOT : What about mine last birthday
USER : do you think humans should be worried about AI
BOT : No chance
USER : I want a robot friend
BOT : Why do you want a robot friend
USER : what if you could be anything you wanted
BOT : Do you really think it's likely that me could be anything me wanted
```

C. *Define Training Data and Model*

The approach to achieve extracting intent and entities is machine learning. Rasa-NLU is selected as the main tools. First, install Rasa-NLU packages in the environment and import it in the project. Rasa-NLU accepts different kinds of training data. The data type used in this project is Json which is more readable and has a clear structure. The standard format of JSON type data is shown below:

```
      },
{
        "text": "I want to know the stock price of GOOGLE at 2015.12.10",
        "intent": "get_historical_price_intraday",
        "entities": [
          {
            "start": 42,
            "end": 48,
            "value": "GOOGLE",
            "entity": "company"
          },
          {
            "start": 52,
            "end": 62,
            "value": "2015.12.10",
            "entity": "date"
          }
        ]
      },
```

The training data in JSON presents the intent and entities in sentences. Ideally, the training data set should as large as possible. However, only ten sentences are defined for an intent in this project due to the limited time. With the help of Rasa-NLU, an interpreter can be trained in only serval lines code.

```
# Create a trainer that uses this config
trainer = Trainer(config.load("config_spacy.yml"))

# Load the training data
training_data = load_data('training-data.json')
interpreter = trainer.train(training_data)
```

A well-trained interpreter can extract the intent and entities in different natural language sentences.

D. *Extract Intent and Entities*

After training an interpreter, **interpreter.parse(message)** can be used to extract intent and entities of a sentence; the result is shown below:

```
USER : What can you do for me?
{'intent': {'name': 'ask_function', 'confidence':
0.4157279600028467}, 'entities': [], 'intent_ranking': [{'name':
'ask_function', 'confidence': 0.4157279600028467}, {'name':
'choose_company', 'confidence': 0.13271447222593913}, {'name':
'get_price', 'confidence': 0.10679387232125058}, {'name':
'get_historical_price_intraday', 'confidence':
0.08389960911445586}, {'name': 'get_historical_price',
'confidence': 0.06733500848174852}, {'name': 'others',
'confidence': 0.05714672852318785}, {'name': 'affirm',
'confidence': 0.04781574552792431}, {'name': 'get_open_price',
'confidence': 0.038002976378851805}, {'name': 'goodbye',
'confidence': 0.029360446743718047}, {'name': 'greet',
'confidence': 0.021203180680077253}], 'text': 'What can you do for
me?'}
```

The result shows the confidence of different intent. The intent with the highest confidence is chosen as the intent of the sentence. To make the result more usable in the next steps, the intent and entities information need to be stored in the dictionary. This process can be achieved by the following code:

```
entities = interpreter.parse(message)["entities"]
intent = interpreter.parse(message)["intent"]["name"]
# Initialize an empty params dictionary
#params = {}
# Fill the dictionary with entities
for ent in entities:
    params[ent["entity"]] = str(ent["value"])
```

E. *Connect to iexfinance*

The chatbot created in this project aims to help the user check the information about the stock. This function is achieved by using iexfinance API. Like installing Rasa-NLU, the first step is installing iexfinance to the Python environment and import to the project.

```
from iexfinance.stocks import Stock
from datetime import datetime
from iexfinance.stocks import get_historical_data
from iexfinance.stocks import get_historical_intraday
import matplotlib.pyplot as plt
```

With the help of iexfinance, it is easy to get a different stock price of companies. There are three main functions used in this project.

(1) check the current stock price; this can be achieved by the following code:

```
from iexfinance.stocks import Stock
tsla = Stock('TSLA')
tsla.get_price()
```

(2) check the historical price between a period of time

```
from datetime import datetime
from iexfinance.stocks import get_historical_data

start = datetime(2017, 1, 1)
end = datetime(2018, 1, 1)

df = get_historical_data("TSLA", start, end)
```

(3) check the open price

```
stock = Stock(company)
price = str(stock.get_open())
```

F. *Connect to WeChat*

The final stage is to connect the program to WeChat which is known as the most popular online chat application in China. The tools used in this project is WXPY which is a powerful tool to process messages on WeChat. As usual, first, install the packages of WXPY to the Python environment and import to the project.

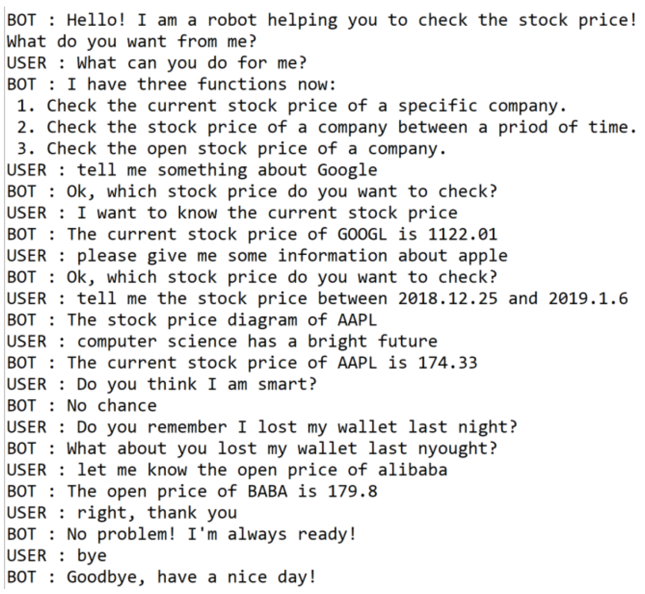
```
pip install -U wxpy
```

First, initialize a **Bot()** which can provide a QR code for the user to log in their WeChat account. In order to keep receiving the message from a specific friend or a group of them, registering the bot is a good choice. Then define a method called **reply_my_friend(msg)** to accept messages and send the answers to users. The codes are shown below:

```
my_friend = bot.friends().search('Jethro', sex=MALE, city='青岛')[0]
# sent a start message
my_friend.send('Hello! I\'m here!')
# reply the message send by my_friend
@bot.register(my_friend)

def reply_my_friend(msg):
    global state, params
    state, params = send_message(state, msg.text, params)
    return None
```

IV. Evaluation

The process of implementing a stock chatbot using Python is not too complexed. An example dialogue between user and chatbot is shown below:



```
BOT : Hello! I am a robot helping you to check the stock price!
What do you want from me?
USER : What can you do for me?
BOT : I have three functions now:
 1. Check the current stock price of a specific company.
 2. Check the stock price of a company between a priod of time.
 3. Check the open stock price of a company.
USER : tell me something about Google
BOT : Ok, which stock price do you want to check?
USER : I want to know the current stock price
BOT : The current stock price of GOOGL is 1122.01
USER : please give me some information about apple
BOT : Ok, which stock price do you want to check?
USER : tell me the stock price between 2018.12.25 and 2019.1.6
BOT : The stock price diagram of AAPL
USER : computer science has a bright future
BOT : The current stock price of AAPL is 174.33
USER : Do you think I am smart?
BOT : No chance
USER : Do you remember I lost my wallet last night?
BOT : What about you lost my wallet last nyought?
USER : let me know the open price of alibaba
BOT : The open price of BABA is 179.8
USER : right, thank you
BOT : No problem! I'm always ready!
USER : bye
BOT : Goodbye, have a nice day!
```

The stock chatbot can answer different questions about searching for stock information. For the season not related to this topic, the chatbot can also reply. However, there is still some limitation for this simple chatbot, such as the response is still a little bit stiff. This section aims to evaluate the natural language processing chatbot implemented in this project from two criteria which are application range and real effect.

A. Application Range

There is a wild range for natural language analysis chatbot since it can understand different intent if related training data is defined. For example, it can help people manage their schedules. However, a large number of training data need to be defined to make the chatbot cover different areas, which is a hard work needs to be finished by a large group of engineers.

B. Real Effet

The effect of natural language chatbot in practical is pretty good, like Siri and Cortana, they can help users do a lot of works. On the contrary, some people complain about speech assistants are not smart enough. The reason for this is the training data set is not large enough. In fact, the performance of the chatbot is partly depended on the size of the data set. However, it is impossible to define a perfect data set covering all aspects. In order to improve the performance of the chatbot, on the one hand, refine the training data set, one the other hand, focus on a new machine learning algorithm which is harder to achieve.

V. Conclusion

In conclusion, this report explains some of the working principles of a usual chatbot and gives the process of implementing a simple stock chatbot. The result shows that good chatbot can help users do numbers of works which improves their work efficiency. It can be applied in various area and have great potential. It also shows that it is not easy to build and smart chatbot. There are still some limitations on how to improve the performance of the chatbot. With the development of techniques and scientists' effects, natural language analysis chatbot will become real smart in the near future.

References

[1] Online, Available: https://en.wikipedia.org/wiki/Natural_language_processing

[2] Online, Available: http://www.philocomp.net/ai/elizabeth.htm