Yue Wang

Dr. Fan Zhang

AI15

03-07-2019

Project Report

Introduction

When you are a decision maker, you first have a question. Then you need to distill to 2 to 3 keywords for finding documents containing keywords in a search engine like Google. Based on popularity, search engines deliver documents to you. Next, you read documents to find an answer. At last, you analyze if it's correct or not.

But for the robot, it's another different process. You ask a natural language question, the machine understand the question first. Then, it produces possible answers and offers evidence. Based on multiple answers, machine order them by evidence. It's called computing confidence. Machine deliver responses, evidence, and confidence to you. Last, you consider the answer.

Nowadays, we are in automatic open-domain question answering, which is a long-standing challenge in Artificial Intelligence to emulate human expertise. Our goal is to give machine rich natural language questions over a broad domain of knowledge. And the machine delivers us precise answers, accurate confidences, consumable justifications and fast response time.

This project is to build chatbot in Python, using regular expressions and machine language to extract meaning from free-form text to querying stock data. The chatbot is integrated into Wechat, based on iexfiance API.

Part 1

At the beginning of the project, I handle text with <u>regular expressions</u>. The regular expression is to match messages against known patterns, extract key phrases and transform sentences grammatically.

The following code has set a pattern. It will achieve multiple selective answers to the same question. For instance, if the user message contains "I want", the chatbot will return the value of the key 'I want (.*)' randomly. But I only set four patterns, it's limited for a chatbot.

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

Then, I check if the user message has matched the rules I wrote above. If it has matched, it will return a random response. If no pattern has matched, chatbot will reply "I don't know :(".

```python
# Define match_rule()
def match_rule(rules, message):
    response, phrase = "I don't know :(", None

    # Iterate over the rules dictionary
    for key,value in rules.items():
        # Create a match object
        match =re.search(key,message)
        if match is not None:
            # Choose a random response
            response = random.choice(rules[key])
            if '{0}' in response:
                phrase = match.group(1)
    # Return the response and phrase
    return response, phrase
```

Later, I wrote the code to swap the pronouns in order to transform the answer from the question. For example, if 'you' occurs in the user message, the bot message will replace 'you' to 'me'.

```python
# Define replace_pronouns()
def replace_pronouns(message):

    message = message.lower()
    if 'me' in message:
        # Replace 'me' with 'you'
        return re.sub('me','you',message)
    if 'my' in message:
        # Replace 'my' with 'your'
        return re.sub('my','your',message)
    if 'your' in message:
        # Replace 'your' with 'my'
        return re.sub('your','my',message)
    if 'you' in message:
        # Replace 'you' with 'me'
        return re.sub('you','me',message)

    return message
```

At last, in respond(message) function, I combine all the processed I said above to return a chatbot message from line63 to line 70.

```python
57 # Define respond()
58 def respond(message):
59     # Call the match_intent function  "greeting"
60     intent = match_intent(message)
61     if intent is not None:
62         return responses[intent]
63     # Call match_rule
64     response, phrase = match_rule(rules, message)
65     if '{0}' in response:
66         # Replace the pronouns in the phrase
67         phrase = replace_pronouns(phrase)
68         # Include the phrase in the response
69         response = response.format(phrase)
70         return response
```

You may notice the code from line 59 to line 62. These codes extract the intent of the user message by regular expression.

The following code illustrates how I extract the intent from the user message. I create a dictionary called "keywords", it contains three intents: greet, thank you and goodbye. These three intents each has some keywords. If the user sends 'bye' in the message, this message will be included in 'goodbye' intent. Then I compile these

keywords into a regular expression. So, I can use re.search() in match_intent() function.

I also define a response dictionary according to these three intents. For example, if the program detects the intent is 'greet', the chatbot will reply 'Hello you! :'. The match_intent(message) will return the intent the program has detected. I initialize the intent is None. In the respond(message) function, if the intent called from match_intent is not None, the chatbot will return the response based on the user's intent.

```
73 keywords = {
74              'greet': ['hello', 'Hi', 'hi','Hello','hey',"what's up"],
75              'thankyou': ['thank you', 'Thank you','thx','Thanks'],
76              'goodbye': ['bye', 'farewell','88']
77          }
78 # Define a dictionary of patterns
79 patterns = {}
80 # Iterate over the keywords dictionary
81 for intent, keys in keywords.items():
82     # Create regular expressions and compile them into pattern objects
83     patterns[intent] = re.compile("|".join(keys))
84
85
86 responses = {'greet': 'Hello you! :)',
87              'thankyou': 'you are very welcome',
88              'default': 'default message',
89              'goodbye': 'goodbye for now'
90          }
91 # Define a function to find the intent of a message
92 def match_intent(message):
93     matched_intent = None
94     for intent, pattern in patterns.items():
95         # Check if the pattern occurs in the message
96         if pattern.search(message):
97             matched_intent = intent
98     return matched_intent
99
```

Part 2

We can extract intent from user message. We can also extract the entities, right? How to express " Can you help me please" in computer language? The computer use vectors to express, the units can be characters, words or sentences.

The word vector attempts to represent the meaning of a word with a fixed-length numerical vector. Words that appear in similar contexts have similar vectors. Words with similar meaning have similar vectors. Training word vector requires a lot of data. More data, more precise of the prediction. We use spaCy to use word vector. And use Support Vector Machines to calculate the similarity for prediction. The following code first loads the spaCy model: nlp, en_core_web_md. Then create a support vector classifier.

```python
100 import spacy
101 # Import SVC
102 from sklearn.svm import SVC
103 import pandas as pd
104 import warnings
105 warnings.filterwarnings("ignore", category=FutureWarning, module="sklearn", lineno=196)
106 def entity_train():
107     # Load the spacy model: nlp, en_core_web_md
108     nlp = spacy.load("en_core_web_md")
109     # Create a support vector classifier
110     clf = SVC()
111     X_train = pd.read_csv('X_train.csv')
112     y_train = pd.read_csv('y_train.csv')['label']
113     # Fit the classifier using the training data
114     clf.fit(X_train, y_train)
115     return nlp
116
```

Hence, I use spacy(nlp) to extract the entities from the user message. Using spaCy's entity recogniser, I define extract_entities(). In this function, it only extracts the entities if it is a date, organization or person.

```python
136 # Define extract_entities()
137 def extract_entities(nlp, message):
138     # Define included entities
139     include_entities = ['DATE', 'ORG', 'PERSON']
140     # Create a dict to hold the entities
141     ents = dict.fromkeys(include_entities)
142     # Create a spacy document
143     doc = nlp(message)
144     for ent in doc.ents:
145         if ent.label_ in include_entities:
146             # Save interesting entities
147             ents[ent.label_] = ent.text
148     return ents
149
150 def check_Org(entities):
151     if entities['ORG'] is not None:
152         return entities['ORG']
153     return None
```

We are only interested in the company which the user ask. So there is a check_Org function to check if the entities in the message contain ORG, if so, this function returns the name of ORG, which is a company name.

We'd like to explore the database using natural language. Rasa NLU is an open-source natural language processing tool for intent classification and entity extraction in chatbots. I wrote wy.json which contains the training data. The interpreter_train() create an interpreter by training. Then interpreter function will be called in the main by "intent=interprete(interpreter,msg.text)['intent']['name']". Because there are lots of intents, I assign each intent to a letter.

```python
117 from rasa_nlu.training_data import load_data
118 from rasa_nlu.config import RasaNLUModelConfig
119 from rasa_nlu.model import Trainer
120 from rasa_nlu import config
121 '''training for interpreter'''
122 def interpreter_train():
123     # Create a trainer
124     trainer = Trainer(config.load("config_spacy.yml"))
125     # Load the training data
126     training_data = load_data('wy.json')
127     # Create an interpreter by training the model
128     interpreter = trainer.train(training_data)
129     return interpreter
130
131 def interprete(interpreter,message):
132     intp=interpreter.parse(message)
133     return intp
154
155 '''assign each intent to a letter and return the letter of user's intent '''
156 def check_intents(intent):
157     intents=[['a','greet'],['b','price search'],[
158             'c','trading volume search'],['d','market value search'],
159            ['e','appreciate'],['f',None],['g','quit'],['h','specify company'],
160            ['i','confirm'],['j','deny']]
161     for i in intents:
162         if i[1]==intent:
163             return i[0]
164
```

Part 3

In order to achieve a robot which can ask the price, marketing value and trading volume of stock, I apply the iexfinace API. Also, I have a function to convert the company name to the stock symbol, which is efficient to check value in iexfinance API.

```python
from iexfinance import Stock
import requests
def get_ticker_symbol(name):
    url = "http://d.yimg.com/autoc.finance.yahoo.com/autoc?query={}&region=1&lang=en".format(name)
    result = requests.get(url).json()
    for x in result['ResultSet']['Result']:
        if x is not None:
            return x['symbol']
        else:
            return 0
```

Sometimes, it cannot search the price of some company like Samsung, the chatbot will send information of company not found. This goal is achieved by error_check function.

```python
def error_check3(CPN):
    try:
        c_CPN=CPN.get_market_cap()
    except Exception:
        error='information of company not found'
        return error
    else:
        return 1
```

When meeting the situation that the user continues to ask different things of the same company, or the information is not found, the chatbot needs to reply to different information. I need to achieve state machine. This machine can have six states, and I assign a number to different states. Also, it has pending and pending_action. In the dictionary, the value of the company is the company the user want to ask, the function's value is which information the user ask about.

```python
294        dictionary={'company':None,'function':None}
295        pending=0
296        pending_action=None
297        #numerical representation of state
298        INIT,SPECIFY_COMPANY,SPECIFIED,FOUND,NOTFOUND,END=0,1,2,3,4,5
299        state=INIT
```

Using the state machine, we need to change state. If the intent is confirm, we

do the pending_action. Or the intent is deny, we clear the pending_action.

```
165 '''pending actions according to pending type'''
166 def pending_actions():
167     dic={0:'Which company are you asking? Please say "The name is...".',
168      1:'What would you like to ask?',
169      2:'Sorry, I dont understand. Maybe you have typed wrong letter or words.' ,
170      3:'What do you want to ask about this company?'
171      }
172     return dic
173
174 '''actions at each state'''
175 def state_change_action(state):
176     dic={ 0:'',
177      1:'Which company are you asking? Please say "The name is..."',
178      2:"Company specified",
179      3:"Ok, I have found it." ,
180      4:"Sorry, I cannot find that information"
181      }
182     answer=dic[state]
183     return answer
```

In the main program, the bot_reply function will be called if there is no answer

get from respond function.

"dictionary,pending,pending_action,state=bot_reply(intent_,state,pending,Org,pendin

g_action,dictionary)"

```
if pending is not 0:
    my_friend.send(action[pending_action])
    return
if state is not 0 and state is not 3 and state is not 5 and state is not 2:
    my_friend.send(state_change_action(state))
if (dictionary['function'] is not None) and (dictionary['company']is not None):
    CPN=Stock(get_ticker_symbol(dictionary['company']))
    if dictionary['function']=='b':
        if error_check1(CPN)==1:       |
            p_CPN=CPN.get_price()
            state=FOUND
            my_friend.send('Ok,I have found it! The price of {0} is ${1}'.format(dictionary['company'],p_CPN) )
        else:
            my_friend.send('Price Information of company not found')
            state=NOTFOUND
```

Unless pending is 0, the chatbot will always call pending_actions. When the

state is 1 or 4, I will call the state_change_action to ask for more information or say I

cannot find that information. After calling bot_reply, if the value is dictionary is not

None, I will distinguish which function the user wants to ask and return the

information.

You may curious how I change the state in bot_rely function. I will use the state=3 to illustrate this process using the following code.

```python
elif state==3:#information found
    if intent=='j':   # no
        state=5
    elif intent=='i':   # thanks
        pending,pending_action,state=1,1,0
        d['company']=None
        d['function']=None
    elif intent in 'bcd'and Org is None:
        if d['company'] is not None:
            pending,pending_action,state=0,None,2
            d['function'] = intent
        else:
            pending,pending_action,state=0,None,1
            d['company'],d['function']=Org,intent
    elif intent in 'bcd' and Org is not None:
        pending,pending_action,state=0,None,2
        d['company'],d['function']=Org,intent
    else:
        pending,pending_action,state=1,1,0
        d['company'],d['function']=None,None

return d,pending,pending_action,state
```

When chatbot found information already, if the intent from the user is denying, the state will change to end. Or the intent is appreciate, the machine will fo back the initialized state and clear dictionary. The third situation is that the user continues to ask the question, but do not mention the company. There are two situations inside it. One is the company already save in dictionary, it's multiple round questions. The other one is that chatbot ask for the user to specify the company. Fourth, the user asks a complete question which contains company name the function. The machine goes back to state 2. Expect these situations, the machine goes to initialized state and clear dictionary.

Part 4

In order to deal with the audio sent from the user, I install Python Speech Recognition module and Pydub to implement the following code. msg.get_file will

download the contents of the audio. Then I converted that file to be a wav file. I use

the Google Speech Recognition to transfer the audio to text.

```python
from pydub import AudioSegment
import speech_recognition as sr
r = sr.Recognizer()
from io import BytesIO
def txt_recog(msg):

        audio = AudioSegment.from_file(BytesIO(msg.get_file()))
        export = audio.export('file.wav', format="wav")#change the path
        AUDIO_FILE = 'file.wav'#The same path as above
        user='empty'
        with sr.AudioFile(AUDIO_FILE) as source:
            print('say something')
            audio = r.record(source)  # read the entire audio file
        try:
            user=r.recognize_google(audio)
            print("Google Speech Recognition thinks you said " + user)

        except sr.UnknownValueError:
            print("Google Speech Recognition could not understand audio")
        except sr.RequestError as e:
            print("Could not request results from Google Speech Recognition service; {0}".format(e))

        return user
```

Conclusion

In part 1, I achieve three goals. First one is multiple selective answers to the

same question and provides a default answer. Second, chatbot can answer

questions through regular expressions, pattern matching, keyword extraction, syntax

conversion. Third, it can extract users' intents through regular expressions.

In part 2, the user's entities can be extracted by a support vector machine.

Based on predefined entities' types to identify an entity. Also, I construct a local

basic chat robot system through Rasa NLU to explore the database using natural

language.

In part 3, I implement multiple rounds of multi-query technology for state

machines and provide explanations and answers based on contextual issues.

In part 4, the chatbot can achieve receiving the audio from the user and reply

the information.

Through these four parts and integrated into Wechat, there is a chatbot that can answer questions on stock price, trading volume, and marketing value. Users can send audio or text message to ask multiple questions about the stock.

Works Cited

https://pypi.org/project/iexfinance/

https://docs.python.org/3/library/re.html

https://spacy.io/

https://rasa.com/docs/nlu/

https://github.com/jiaaro/pydub

https://github.com/youfou/wxpy