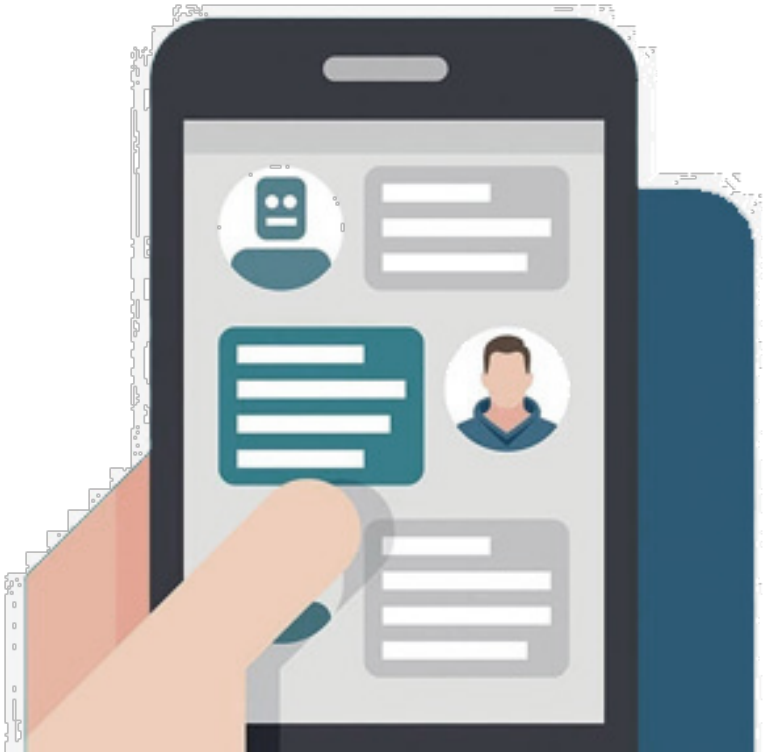
Project Report

Chatbot for Stock Information Queries



Zhang Chenyu

1/7/2019

Table of content:

Introduction

Industry Overview

Chatbot refers to a computer program which is able to conduct conversations with human beings using techniques provided by Artificial Intelligence, NLP (Natural Language Processing) particularly. The classic historic early chatbots are ELIZA (1966), which were used exclusively to simulate typed conversation while many chatbots now include functional features such as games and web searching abilities. So far, the Conversational AI has achieved a lot of breakthroughs especially in application. The virtual assistant built by IBM Watson caught my eye all of a sudden in class. Industrial chatbots can be mainly classified into usage categories such as conversational commerce (e-commerce via chat), analytics, communication, customer support, education, entertainment, finance, health, news, personal, productivity, shopping, social, sports, travel and utilities.

Though the progress, There still a long way to go for the development of Chatbots. As the database, used for output generation, is fixed and limited, chatbots may fail while dealing with an unsaved query. Chatbot's efficiency highly depends on language processing and is limited because of irregularities, such as accents and mistakes that can create an important barrier for international and multi-cultural organisations. Chatbots are unable to deal with multiple questions at the same time and so conversation opportunities are limited. As it happens usually with technology-led changes in existing services, some consumers, more often than not from the old generation, are uncomfortable with chatbots due to their limited understanding, making it obvious that their requests are being dealt machines. Besides, there are still malicious uses of chatbots.

But as a computer science student, solving problem whenever meeting one is always part of daily routine. As the old saying goes, “where there is a will, there is a way.” I

hold the view that one day the field of Chatbot (NLP) will flourish, and I hope to be one of the boosters.

Project Overview

In this project, a basic chatbot for stock information queries was built by using the knowledge I obtained during the previous four weeks. The program were firstly constructed and run using python on JupyterLab during to its aesthetic and interactivity, then it was completed, perfected on Spyder and the final step was to test its performance on Wechat. Several methods and algorithms including those obtained from extension packages were used to build the main framework.

Implementation

Rasa NLU

Rasa is an open source machine learning framework to automate text-and voice-based conversations. As part of Rasa, Rasa NLU (Natural Language Understanding) is a tool for understanding what is being said in short pieces of text. In other words, it can help us conduct intent classification and entity extraction, though the training data needs to be provided by ourselves.

In my framework, the Rasa NLU was selected as the basic tool to perform entity and intent extraction. The Rasa NLU-related packages were installed and imported to the specific environment firstly, after which the training data *stock-data. json* was created and filled. Many text related to the users' input was included though the total size of the training data set was not ideally large to realize high extraction accuracy due to the limited time and energy. Here's part of the training data:


```
{
  "text": "Show the close price of FACEBOOK on 2018-07-08",
  "intent": "history_data_close",
  "entities": [
    {
      "start":24,
      "end":32,
      "value":"FB",
      "entity":"company"
    },
    {
        "start":9,
        "end":14,
        "value":"close",
        "entity":"type"
    },
    {
        "start":45,
        "end":47,
        "value":"08",
        "entity":"day"
    },
    {
        "start":42,
        "end":44,
        "value":"07",
        "entity":"month"
    },
    {
        "start":37,
        "end":41,
        "value":"2018",
        "entity":"year"
    }
  ]
},
{
  "text": "Tell me the close price of Netflix on 2018-05-15",
  "intent": "history_data_close",
  "entities": [
    {
      "start":27,
      "end":34,
      "value":"NFLX",
      "entity":"company"
    },
    {
        "start":12,
        "end":17,
        "value":"close",
        "entity":"type"
```

```
    "intent": "greet",
    "entities": []
  },
  {
    "text": "good morning",
    "intent": "greet",
    "entities": []
  },
  {
    "text": "good evening",
    "intent": "greet",
    "entities": []
  },
  {
    "text": "dear sir",
    "intent": "greet",
    "entities": []
  },
  {
    "text": "right, thank you",
    "intent": "end",
    "entities": []
  },
  {
    "text": "bye",
    "intent": "end",
    "entities": []
  },
  {
    "text": "goodbye",
    "intent": "end",
    "entities": []
  },
  {
    "text": "good bye",
    "intent": "end",
    "entities": []
  },
  {
    "text": "farewell",
    "intent": "end",
    "entities": []
  },
```

Intent and Entity Extraction

An interpreter was obtained through the training process. Then, the interpreter can be used to extract intent and entities. The command *interpreter. parse()* is able to parse a sentence and generate useful messages such as positions, confidence. An example is given as shown below:

```
message = "And what about the open price of Facebook on 2019-03-05"
data = interpreter.parse(message)
data
```

```
{'intent': {'name': 'history_data_open', 'confidence': 0.3749416298096962},
 'entities': [{'start': 19,
   'end': 23,
   'value': 'open',
   'entity': 'type',
   'confidence': 0.9461382115376856,
   'extractor': 'CRFEntityExtractor'},
  {'start': 33,
   'end': 41,
   'value': 'FB',
   'entity': 'company',
   'confidence': 0.9658311906662297,
   'extractor': 'CRFEntityExtractor',
   'processors': ['EntitySynonymMapper']},
  {'start': 45,
   'end': 49,
   'value': '2019',
   'entity': 'year',
   'confidence': 0.9593173874927559,
   'extractor': 'CRFEntityExtractor'},
  {'start': 50,
   'end': 52,
   'value': '3',
   'entity': 'month',
   'confidence': 0.873281750075303,
   'extractor': 'CRFEntityExtractor',
   'processors': ['EntitySynonymMapper']},
  {'start': 53,
   'end': 55,
   'value': '05',
   'entity': 'day',
   'confidence': 0.8938543880569061,
   'extractor': 'CRFEntityExtractor'}],
 'intent_ranking': [{'name': 'history_data_open',
   'confidence': 0.3749416298096962},
  {'name': 'history_data_close', 'confidence': 0.2246084506623609},
  {'name': 'history_data_volume', 'confidence': 0.13000038688987245},
  {'name': 'current_price', 'confidence': 0.07705706190921874},
  {'name': 'end', 'confidence': 0.07361999908131925},
  {'name': 'greet', 'confidence': 0.06606565998473225},
  {'name': 'intro', 'confidence': 0.053706811662800065}],
 'text': 'And what about the open price of Facebook on 2019-03-05'}
```

By using *interpreter. parse()*, intent and entities of the inputted messages were recognized and stored for subsequent exploitation.

```
# Define function to extract intent
def intent_extraction(message):
    return interpreter.parse(message)["intent"]["name"]

# Define function to extract entities
def entity_extraction(message):
    entities = interpreter.parse(message)["entities"]
    parameters = {}
    for ent in entities:
        parameters[ent["entity"]] = str(ent["value"])
    return parameters
```

Stock Data Acquisition

The *iexfinance* is a Python module to get data for Stocks, ETFs, Mutual Funds, Forex/Currencies, Options, Commodities, Bonds, and Cryptocurrencies from IEX Cloud and IEX API.

Stock and *get_historical_intraday* was imported from *iexfinance* to obtain real-time quotes and historical data, respectively. *datetime* was used to generate compatible parameter and *token* was created to get access to the stock data. Since there were basically two jobs for my chatbot to do, that is, to acquire the current price or to obtain the historical data (the open/close price, the volume), related functions were defined accordingly.

```
# Define functions to obtain current price, open/close price, volume of a specified stock
def current_price(parameters):
    return Stock(parameters["company"], token = "sk_da231175d2cb425bbf943b1946966cfb").get_price()

def open_price(parameters):
    date = datetime(int(parameters["year"]),int(parameters["month"]),int(parameters["day"]))
    op = get_historical_intraday(parameters["company"], date, output_format = 'pandas', token = "sk_da231175d2cb425bbf943b1946966cfb")["open"][0]
    return op

def close_price(parameters):
    date = datetime(int(parameters["year"]),int(parameters["month"]),int(parameters["day"]))
    cp = get_historical_intraday(parameters["company"], date, output_format = 'pandas', token = "sk_da231175d2cb425bbf943b1946966cfb")["close"][0]
    return cp;

def volume(parameters):
    date = datetime(int(parameters["year"]),int(parameters["month"]),int(parameters["day"]))
    vl = get_historical_intraday(parameters["company"], date, output_format = 'pandas', token = "sk_da231175d2cb425bbf943b1946966cfb")["volume"][0]
    return vl;
```

State Definition and Policy Rules

A finite-state machine (FSM) or simply a state machine, is a mathematical model of computation, which can change from one state to another in response to some external inputs. Finite-state machine are significant in linguistics since it makes multiple rounds of queries possible and easy to realize. Seven states (INIT, FUNC, CURRENT_PRICE, HISTORY_DATA_open, HISTORY_DATA_close, HISTORY_DATA_volume, END) were defined. The amount of states might be a little bit large but the accuracy and speed of responses were improved. Besides, the policy rules were enlarged to realize fast and

comprehensive state transition. In addition, when the state receive an “end” intent, the imported *random* module will randomly choose one sentences from *([responses[6], responses[7], responses[8]])* to reply, which make the chatbot more human.

State List:

```
# Define the states
INIT = 0
FUNC = 1
CURRENT_PRICE = 2
HISTORY_DATA_open = 3
HISTORY_DATA_close = 4
HISTORY_DATA_volume = 5
END = 6
```

Random Choice:

```
# Random choose the sentences for ending
(CURRENT_PRICE, "end"):(FUNC, random.choice([responses[6], responses[7], responses[8]])),
(HISTORY_DATA_open, "end"):(FUNC, random.choice([responses[6], responses[7], responses[8]])),
(HISTORY_DATA_close, "end"):(FUNC, random.choice([responses[6], responses[7], responses[8]])),
(HISTORY_DATA_volume, "end"):(FUNC, random.choice([responses[6], responses[7], responses[8]])),
(FUNC, "end"):(FUNC, random.choice([responses[6], responses[7], responses[8]]))
```

Main functions

In this part, main functions for receiving messages and replying were defined:

1) *respond (policy_rules, state, message)* was responsible for taking the message and parsing it to extract the entities and intent, then it would return the new state and the perfected response according to the extracted intent and the certain policy rules.
2) *send_message (policy_rules, state, message)* played a role in receiving the message and send it together with the current state to *respond* function, and then taking the return from it.
3) *send_messages (messages)* was able to handle multiple messages and it would initialize the state to INIT.

Wechat Automated Chatbot

The *wxpy* is a python module and it was a updated version of *itchat.* By installing it, many functions related to Wechat APP such as sending messages or pictures, automatically replying messages of various types can be achieved.

Firstly, a bot was initialized during the process of logging in by scanning the generated QR code. Then a specified friend named “Stockbot Coco” was found and the bot was registered. Subsequently, *reply_message(message)* was defined to automatically reply the messages sent by users.

```
# Import the necessary wechat module
from wxpy import *

# Initialize the Bot
bot = Bot()

# Search the specified Wechat account
my_friend = bot.friends().search('Stockbot Coco')[0]

@bot.register(my_friend, TEXT)

# Reply messages sent by my_friend Stockbot Coco
def reply_message(message):
    # my_friend.send('Received: {} ({})'.format(message.text, message.type))
    msg=message.text
    state = FUNC
    state, response = send_message(policy_rules, state, msg)
    my_friend.send(response)
```

Simulation

To evaluate the performance of the designed chatbot, a series of simulated messages was inputted. Here's the simulated result:

```
USER : Hi
BOT : Welcome! My name is Coco and I can share stock information with you.
USER : What can you do?
BOT : I can tell you:
  1) The current price
  2) The historical data:
        1. the open/close price
        2. the volume
USER : Could you please tell me the current price of Nike?
BOT : Got it! The current price of NKE is 83.95.
USER : And what about the open price of Facebook on 2019-03-05
BOT : U wanna know the open price! It's 167.36.
USER : Please show me the close price of Alibaba on 2019-03-06
BOT : The close price on that day was 183.06.
USER : Now show the volume of Apple on 2019-03-07
BOT : On that day, the volume was 3113.
USER : Thanks, bye
BOT : Au Revoir! My friend @_@
USER : Oh! I forgot, please show me the current price of Facebook
BOT : Got it! The current price of FB is 193.
USER : Farewell
BOT : Au Revoir! My friend @_@
```

It can be seen that the replies of the built chatbot was basically satisfying though some errors occurred during the other simulations. The reasons are summarized:

1) The training data for the Rasa NLU interpreter used in this project was quite small compared with industrial ones, hence some intent or entities cannot be recognized during the execution process.
2) The historical data was not completely accessible by *iexfinance*. For instance, the open price of JD on 2015 was None since it had not decided to sell shares to public at that time.

Learning Process Record

Before the official lectures (Preparation)

Date : Before 3rd June 2019

Time : ------

Task	**Description**
Software Download & Preliminary Use	Installed Ubuntu 16.04 (Dual System), Zoom (Video Conferencing), Anaconda (packages)
Python	1) Studied, reviewed and practiced the basic knowledge of python 2) Read the books “Python for scientific computing”, “Python Cookbook” and “Deep Learning with python”
RESTful API	Learned the basic calls of RESTful API

Week 1

Date : 3rd June 2019 – 9th June 2019

Time : 9:00 a.m. - 12:00 p.m. 3rd June

Task	**Description**
Introduction of the syllabus	What we will learn and what we should do in the following six weeks
IBM Watson NLP	A brief overview and thoughts of IBM Watson's basic algorithms
Chatbots	Introduction of Chatbots and the realization of a basic one by using python on Jupyter Notebook
	Obtained ELIZA style simple chat robot by extracting meaning from free-form text using regular expressions and machine learning language
Codes	Perfected the codes on Jupyter Notebook, built the complete architecture on Spyder and uploaded to Github
Shell command	Mastered basic shell commands of Git, Linux and Anaconda
IBM Assistant	Basic understanding of the details about the realization and use of IBM Assistant
Review & Preview	Reviewed the comprehended knowledge and the video of the first lecture, previewed related knowledge

Week 2

Date : 10th June 2019 – 16th June 2019

Time : 8:00 p.m. – 11:00 p.m. 9th June

Task	**Description**
Introduction to this week's tasks	What we will learn and what we should do in this class and this week for self-studying
Regular Expression	A detailed explanation about regular expression using slides and examples
Word Vectors	Introduction of word vectors and python library spaCy for Natural Language Processing, the command to upload spaCy
Intent Recognition	Method I: Using regular expression (e.g. Names: ('[A-Z]{1}[a-z]*'))
	Method II: Using cosine_similarity
	Method III: Using SVM (supporting vector machine)
Entity Recognition	Method I: Using pre-built model (spaCy's entity recognizer)(e.g. nlp, ent, ent.text, ent.label_)
	Method II: Using regular expression (e.g. ('.* from (.*) to (.*)'))
	Method III: Assigning roles using spaCy's parser (ancestors)

Codes	Perfected the codes on Jupyter Notebook, built the complete architecture on Spyder and uploaded to Github
Review & Preview	Reviewed the comprehended knowledge and the video of the second lecture, previewed related knowledge

Week 3

Date : 16th June 2019 – 24th June 2019

Time : 9:00 a.m. – 12:00 p.m. 17th June

Task	**Description**
Introduction to this week's tasks	What we will learn and what we should do in this class and this week for self-studying
Access data	Two ways to access real-world data: Built a local datebase or require information by interacting with outside databases or APIs
Virtual Assistant	Use the information leant during the previous two weeks to build a personal assistant that can use real-world data to genuinely help us address practical issues (task-oriented) rather than just taking chit-chats
Basic SQL statements	SELECT, FROM ,WHERE, AND, OR

Get response using natural language	Creating queries from parameters (ssing your custom function to find hotels), Creating SQL from natural language (rasa_nlu, interpreter)
Incremental slot filling and Negation	Incremental slot filling allows users to add filters incrementally, just in case they don't specify all of their preferences in one message
Ultimate Code	Put together all the taught ideas in order to allow users to tell the chatbot about what they do and do not want to do, split across multiple messages. Perfected the codes on Jupyter Notebook, built the complete architecture on Spyder and uploaded to Github
Rasa	Studied rasa tutorials roughly
Additional Example	Referred to "JointSLU: Joint Semantic Parsing for Spoken/Natural Language Understanding" on Github and practiced personally
Review & Preview	Reviewed the comprehended knowledge and the video of the second lecture, previewed related knowledge

Week 4

Date : 24th June 2019 – 30th June 2019

Time : 9:00 a.m. – 12:00 p.m. 23th June

Task	**Description**

Introduction to this week's tasks	What we will learn and what we should do in this class and this week for self-studying
Access data by API	Access information by interacting with outside databases or APIs or using python packages
Stateful Bots	Answering contextual questions and dealing with rejections
Queuing answers	Pending actions and pending state transitions
Integration	Putting all the methods learned in four weeks to build an integrated chatbot
Frontiers of dialogue research	Introduction of the frontier technology on dialogue research and industry
Recap on previous lectures	Review the total nine means of building a chatbot
Additional Slides	Machine learning, LSTM, Seq2seq and other methods or algorithms about Natural Language Processing (NLP)
Review & Preview	Reviewed the comprehended knowledge and the video of the second lecture, previewed related knowledge