Intelligent chatbot for financial purpose report

Qianzeng Yang

Content

1 Background 3

2 Tools and environment 3

3 Project ideas..... 3

4 Implementation 4

4.1 Intention and entities extraction..... 5

4.2 Policy making and multiple turns conversation..... 8

4.3 Negation entities extraction 10

4.4 Data manipulation and API calling 11

4.5 Final Deployment 12

5 Conclusion 13

6 Demo..... 14

1. Background

Intelligent chatbot is a useful tool in many business domains. For example, the Amazon custom service robot can help the customer deal with some basic problem and assign the human operators, which save lots of cost.

In this project, we mainly focus on the application in finical domain and try to implement the intelligent stock query robot. This robot is aimed to response the user meaningfully and facilitate them to fulfill their tasks such as stock price information querying, volume information querying, market value querying, etc.

2. Tools and environment

Programming language: Python 3.7

IDE: Pycharm and Jupyter Notebook

Nature language process (NLP): Spacy

Nature language Understanding (NLU): RASA NLU

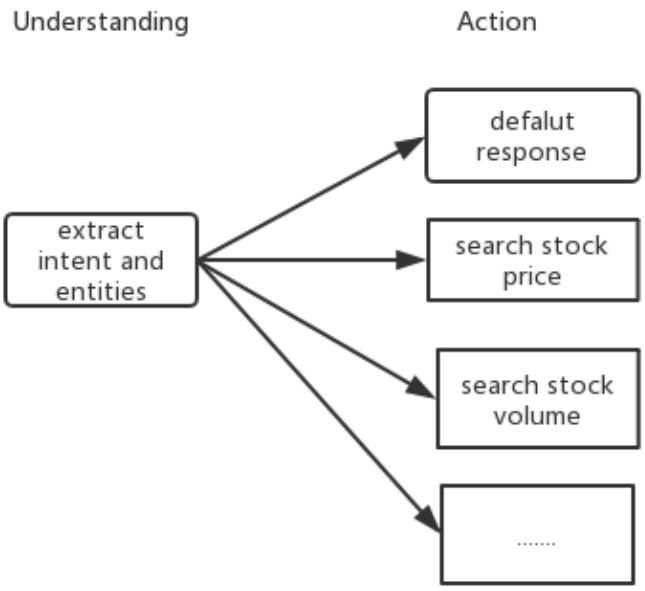
Stocks information query API: iexfinance

Robot deployment platform: Wechat by WXPY API

3. Project ideas

For a single round conversation, the basic idea of the robot behaviors is that response the user text with related meaningful actions. Therefore, there are two kind of problems should be considered. The first one is understanding of user speech which means the robot extract the intents and entities of the user speech and recognize what they want to do. The second problem is about the action, which the robot should response to the user intent with

relative behavior. The actions of the robot can be various. They can simply be affirmative message to user or the searching result of user intent task. However, all the actions finally express in a response text. For example, the user says "Hello" to the robot and the robot should recognize that the user is greeting to the robot and then it should give the user feedback of their greeting actions. Another example is that if user ask the robot that "Please help me to check the stock price for Apple". The robot should first to understand that the user what to search the stock price and the entity of this intention is to search the price for Apple company. Then, the robot will give feedback to the user with this searching result.



In most conditions, however, conversation may not be a single round. The robot has the ability to facilitate the user to use it functions, therefore, it could have multiple rounds for user to complete their task. The multiple rounds conversation can be little different from the single round. The understanding of the user intent should consider the context rather than the only one sentence. Ideally, the intent and entities may appear

consequently in a series of sentences. A feasible solution is that classify the possible user intents into different groups. Each group is referred to one robot state. Each robot state is the context to understanding its related specific user intention, which means the robot only accept the intention by the state policy. The policy is the intention-state key paired rules which specify which state will be transmit when a intention is extracted by bot. In each state, the robot behaves like the single round, will jump to the next state by the policy rules.

For example, an initial state is used to waiting user stock searching intention. And we make a policy that only searching for stock price will jump to the searching state. If the user intent to greet with the robot, the state will not change due to the policy. Once user instruct the bot to search the stock price, the robot will transit to the searching state.

4. Implementation

I. Intention and entities extraction

The extraction of intention and entities is the way to understand what the user what to do and which object user what to apply for it. In this project, we are not to supposed to extract all kind of the intentions but only the designed five intentions which are greeting, stock searching, volume information querying, market value querying and farewell. To extract intentions and entities from user's text, there are three methods used in the project.

1. Regular expression

   A regular expression is an expression defines a search pattern. It is the simplest the way to extract a specified character pattern from a defined rule. This method can be applied to a simple regular pattern such as phone number. In this project, the regular expression is used to extract the stock symbol, which is sequential capital characters. For instance, "AMZN" is the stock symbol for Amazon company.

```
import re
text = "I Want to check the stock for APPL, IBM and TSLE"
# extract the stock symbol by regex, return an empty list if not matched.
def extract_entities_re(text):
    pattern = re.compile(r"([\w]+[A-Z]+[\w]+)|(^[A-Z]+$)")
    results = pattern.findall(text)
    intent_list = []
    for item in results:
        for intent in item:
            if intent is not "":
                intent_list.append(intent.strip())
    return intent_list
print(extract_entities_re(text))
```

```
['APPL', 'IBM', 'TSLE']
```

Fig 4.1.1.1 entities extraction by re.

2. RASA NLU

   Rasa NLU is a useful tool for nature language understanding, it provides useful functions for intent classification and entities extractions. In this project, most of the intents are classified by Rasa NLU.

   Before the robot working, we first to design the data set for RASA to train. The data set contains the example sentences for each intent that should be classified by the chatbot. Also, in each sentence multiple entities can be labeled. The data is in the format of JSON.

```
    {
      "text": "please check Tesla stock",
      "intent": "stock_search",
      "entities": [
        {
          "start": 13,
          "end": 18,
          "value": "Tesla",
          "entity": "stock_name"
        }
      ]
    },
    {
        "text": "No",
        "intent": "negative",
        "entities": []
    },
    {
        "text": "No,thank you",
        "intent": "negative",
        "entities": []
    },
    {
      "text": "I want to check the stock for IBM.",
      "intent": "stock_search",
      "entities": [
        {
          "start": 30,
          "end": 33,
          "value": "IBM",
          "entity": "stock_name"
        }
      ]
    }
  ]
}
```

Fig 4.1.2.1 The sample training data

The RASA NLU will train from the dataset and interpret the new message from user to extract the entities and intents.

```
: # Import necessary modules
from rasa_nlu.training_data import load_data
from rasa_nlu.config import RasaNLUModelConfig
from rasa_nlu.model import Trainer
from rasa_nlu import config

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

# Load the training data
training_data = load_data('/Users/alexander/PycharmProjects/myrobot/testData2.json')

# Create an interpreter by training the model
interpreter = trainer.train(training_data)
def interpret (message):
    intent = interpreter.parse(message)["intent"]["name"]
    entities = interpreter.parse(message)["entities"]
    return intent, entities
```

```
Fitting 2 folds for each of 6 candidates, totalling 12 fits
```

Fig 4.1.2.2 Load training data and interpret message

II. Policy making and multiple turns conversation

For multiple turns conversation, we classify the user intends to 3 different stage. Each stage is referred to a robot state.

In the project, there are four states:

INIT state: For greeting user and answer the explanation of the robot functions.

SEARCH_STOCK state: For searching stock information.

Farewell state: For saying good bye.

Also, we have 4 pending states. These pending states will prevent the robot jump to next state and used to waiting for user additional message.

REQUIRE_ENTITIES: Waiting for more entities from user message.

ENTITIES_OK: Entities input complete.

DATA_OK: Getting data from internet successful.

DATA_FAILED: Getting data from internet failed.

```
# define the robot state
INIT = 0
STOCK_SEARCH = 1
FAREWELL = 2

# define the pending state
REQUIRE_ENTITIES = 0
ENTITIES_OK = 1
DATA_OK = 2
DATA_FAILED = 3
```

Fig 4.2.1 System state and pending state

To make the transition of each system state, we make a policy to achieve that. The policy is a dictionary to specify the next state for the current intent and pending status. It's a structure like below

```
# define state transition policy
policy = {
    (INIT,None,None):(INIT,False,robot_behaviors["default"]),
    (INIT,None,"greet"):(INIT,False,robot_behaviors["greet"]),
    (INIT,None,"ask_functions"):(INIT,False,robot_behaviors["explain_functions"]),
    (INIT,REQUIRE_ENTITIES,"stock_search"):(STOCK_SEARCH,True,robot_behaviors["confirm_without_
    (INIT,ENTITIES_OK,"stock_search"):(STOCK_SEARCH,True,robot_behaviors["confirm_with_stock_en
    (STOCK_SEARCH,REQUIRE_ENTITIES,None):(STOCK_SEARCH,False,robot_behaviors["ask_more_entities
    (STOCK_SEARCH,ENTITIES_OK,None):(STOCK_SEARCH,True,robot_behaviors["confirm_with_stock_enti
    (STOCK_SEARCH,ENTITIES_OK,"stock_search"):(STOCK_SEARCH,True,robot_behaviors["confirm_with_
    (STOCK_SEARCH,DATA_OK,None):(STOCK_SEARCH,False,robot_behaviors["answer_stock_info"]),
    (STOCK_SEARCH,DATA_OK,"stock_search"):(STOCK_SEARCH,False,robot_behaviors["answer_stock_inf
    (STOCK_SEARCH,DATA_OK,"confirm"):(STOCK_SEARCH,False,robot_behaviors["ask_more_entities"]),
    (STOCK_SEARCH,DATA_OK,"negative"):(INIT,False,robot_behaviors["farewell_ok"]),
    (STOCK_SEARCH,DATA_FAILED,None):(STOCK_SEARCH,True,robot_behaviors["search_failed"]),
    (STOCK_SEARCH,DATA_FAILED,"confirm"):(STOCK_SEARCH,False,robot_behaviors["ask_more_entities
    (STOCK_SEARCH,DATA_FAILED,"negative"):(INIT,False,robot_behaviors["farewell_failed"]),
    (FAREWELL,None,"farewell"):(INIT,False,robot_behaviors["farewell_ok"])

}
```

Fig 4.2.2 Policy

The structure of the policy is:

(current_state, Pending_state, intent)->(next_state, isInternalState, robot_behavior)

The robot behavior is the action for each transition. The internal state is the state that robot may automictically transit to next state without user's interaction.

The pending function judging the process of each system state. It allow user to query multiply rounds with robot.

```
# judge the pending stage
def pending(state,intent,current_entities,current_data):
    pendingState = None
    if state is INIT and intent != "stock_search":
        pendingState = None
        current_entities = []
        current_data = []
    elif state is INIT and intent == "stock_search":
        if len(current_entities) != 0:
            pendingState = ENTITIES_OK
        else:
            pendingState = REQUIRE_ENTITIES
    elif current_data is None:
        pendingState = DATA_FAILED
        current_data = []
    elif state is STOCK_SEARCH and len(current_entities) == 0 and len(current_data) is 0:
        pendingState =  REQUIRE_ENTITIES
    elif state is STOCK_SEARCH and len(current_entities) > 0 and current_data is not None: #anc
        pendingState = ENTITIES_OK
    elif state is STOCK_SEARCH and len(current_data) is not 0:
        pendingState = DATA_OK
    elif state is STOCK_SEARCH and len(current_entities) > 0 and current_data is None :
        pendingState = DATA_FAILED
    elif state is FAREWELL:
        pendingState = None
        current_entities = []
        current_data = []

    return pendingState
```

Fig 4.2.3 Pending

The execution of the policy allow robot to transit the system state according to the user intention.

```
# execute the policy
import random
def execute_policy(state,intent,current_entities,current_data):
    pendingState = pending(state,intent,current_entities,current_data)
    if intent == "confirm" or intent == "negative":
        print("clear data")
        current_entities = []
        current_data = []
    try:
        next_state, isInternal, behaviors = policy[(state,pendingState,intent)]
        if "{}" in random.choice(behaviors):
            print("bot: "+random.choice(behaviors).format(format_info(current_data)))
            my_friend.send(random.choice(behaviors).format(format_info(current_data)))
        else:
            print("bot: "+random.choice(behaviors))
            my_friend.send(random.choice(behaviors))
    except:
        if intent == "farewell":
            next_state,current_entities,current_data = execute_policy(FAREWELL,intent,current_e
        else:
            next_state,current_entities,current_data = execute_policy(state,None,current_entiti
    else:
        if isInternal:
            if pendingState is ENTITIES_OK:
                current_data = query_stock_info(current_entities)
                if current_data is not None:
                    current_entities = []
                next_state,current_entities,current_data =  execute_policy(next_state,None,curr
    return next_state,current_entities,current_data
```

Fig 4.2.4 Execution of policy

III. Negation entities extraction

In some situation, the user may indicate negated entities in the message. For example, the user may say "I want a stock for Apple Company but not the IBM". In this situation, there are two entities "Apple Company" and "IBM", however, the IBM is not what user expect to search for. The robot should consider that negation in the sentence. To achieve that, we use the negation entities extraction strategies. Here is the basic idea to extract the negated entities.

1. Extract the entities as usual.
2. Locate the entities in the sentence.
3. Split the sentence into sub-sentences by the end of each entities.
4. Search the negative symbol like "not, n't" and mark the entities in

the sentence to negated sentence.

```
: # extract the entities with negation entities return entities, neg_entities
def extract_all_entities(text):
    entities = extract_entities(text)
    neg_policy = ["n't","not"]
    # get the index of the final character of each entities
    ends_index = sorted([text.index(e)+len(e) for e in entities])
    # initialise a list to store sentence chuncks
    chunks = []
    start = 0
    for end in ends_index:
        chunks.append(text[start:end])
        start = end
    pos_entities = []
    neg_entities = []
    for chunk in chunks:
        for entity in entities:
            if entity in chunk:
                if "not" in chunk or "n't" in chunk:
                    neg_entities.append(entity)
                else:
                    pos_entities.append(entity)
    return pos_entities,neg_entities
text = "I want the stock for AAPL but not IBM"
print(extract_all_entities(text))
```

```
(['AAPL'], ['IBM'])
```

Fig 4.3.1 Extraction of negative entities

IV. Data manipulation and API calling

In this project, the chatbot is able to check the stock information online. This is implemented by the interface provide by the iexfinance. Iexfinance is useful tools to get the on time updated stock information online. Before implement the stock searching functions, an API key is required to authenticate for the service. After register the account and API key, the information of the stock can be achieved by the function stock.getquote().

```
In [2]: from iexfinance.stocks import Stock

        appl = Stock('AAPL', token="sk_5949b3f3377c4d278e80fa93394b9bda")

        quote = appl.get_quote()
        print(quote)
```

```
{'symbol': 'AAPL', 'companyName': 'Apple, Inc.', 'primaryExchange': 'NASDAQ', 'calculationPri
ce': 'close', 'open': 204.17, 'openTime': 1566999000885, 'close': 205.53, 'closeTime': 156696
4800000, 'high': 205.72, 'low': 203.32, 'latestPrice': 205.53, 'latestSource': 'Close', 'late
stTime': 'August 28, 2019', 'latestUpdate': 1566964800000, 'latestVolume': 15896313, 'iexReal
timePrice': 205.56, 'iexRealtimeSize': 100, 'iexLastUpdated': 1567022394558, 'delayedPrice':
205.65, 'delayedPriceTime': 1567023347475, 'extendedPrice': 205.26, 'extendedChange': -0.27,
'extendedChangePercent': -0.00131, 'extendedPriceTime': 1567035896671, 'previousClose': 204.1
6, 'previousVolume': 25897344, 'change': 1.37, 'changePercent': 0.00671, 'volume': 15896313,
'iexMarketPercent': 0.024316519182781566, 'iexVolume': 386543, 'avgTotalVolume': 29763354, 'i
exBidPrice': 0, 'iexBidSize': 0, 'iexAskPrice': 0, 'iexAskSize': 0, 'marketCap': 92882706540
0, 'peRatio': 17.37, 'week52High': 233.47, 'week52Low': 142, 'ytdChange': 0.299516, 'lastTrad
eTime': 1567022328226, 'isUSMarketOpen': False}
```

Fig 4.4.1 Acquire the data by API

Sometimes, the parameter stock symbol may be not correct, the error will be occurred in this situation. To handle this, use try-except sentence

to handle the error.

```
In [12]: from iexfinance.stocks import Stock
         stock = Stock('IPIPOP', token="sk_5949b3f3377c4d278e80fa93394b9bda")
         try:
             quote = stock.get_quote()
         except:
             print("Error: not found the stock")

         Error: not found the stock
```

Fig 4.4.2 Error handles

V. Final Deployment

The robot is deployed to the Wechat by WXPY API. WXPY is a tool for automatic chat bot on the Wechat platform. Before using the WXPY, please make sure that the web Wechat of your account is available.

First, import the package and make an instance of bot.

```
In [*]: from wxpy import *
        bot = Bot()

        Getting uuid of QR code.
        Downloading QR code.
        Please scan the QR code to log in.
```

Fig 4.5.1 make a friend instance

Second, register the friend to the robot and implement the function "reply_my_friend()".

```
def respond(message):
    intent, entities = interpret(message)
    entities = extract_all_entities(message)[0]

    next_state,new_entities,data = execute_policy(state,intent,entities,current_data)

    return next_state,new_entities,data
@bot.register(my_friend)
def reply_my_friend(msg):
    global state, current_entities,current_data
    state,current_entities,current_data = respond(msg.text)

    return ""
```

Fig 4.5.1 register the friend and implement the reply function

The registered friends will receive the robot reply defined by "reply_my_friend" function when they send a message to the robot.

If the robot replies the registered friend, the robot is successfully

deployed to Wechat.

5. Conclusion

In this project, I have an understanding for the chatbot. Here, I sincerely thank Dr. Zhang for his careful guidance. I have learnt many useful methods to implement an intelligent robot. Due to the time limit, I regret that not applied all of them. Here is the summary of this methods.

For the response of the robot, I learned the multiple selective answers to the same question and provide a default answer. Also, using pattern matching, keyword extraction to increase the diversity of answers.

For the intent and entities extraction, I have learnt to use regular expression, nearest neighbor classification and support vector machine to extract the user intent and entities. Furthermore, I applied the useful NLP/NLU tools to help me extract the intent. Also, I learnt how to extract the negation entities by splitting the sentence.

For the data manipulation, I Learned how to use SQLite and API to explore database content and acquire data from website. However, due to the time limit, this project only uses the iexfinance API to retrieve data.

For multiple round conversation, I used the multiple states to achieve multiple round conversation.

During the project, I also learn the python programming. I used to be a Java programmer, and at the first of the project, it's really hard to handle some python problems. I practice a lot for my python programming and feel confident about to do more hard jobs. I will keep going to learn more and search more about the application for intelligent chatbot. Also, keep maintenance of this project.

6. Demo

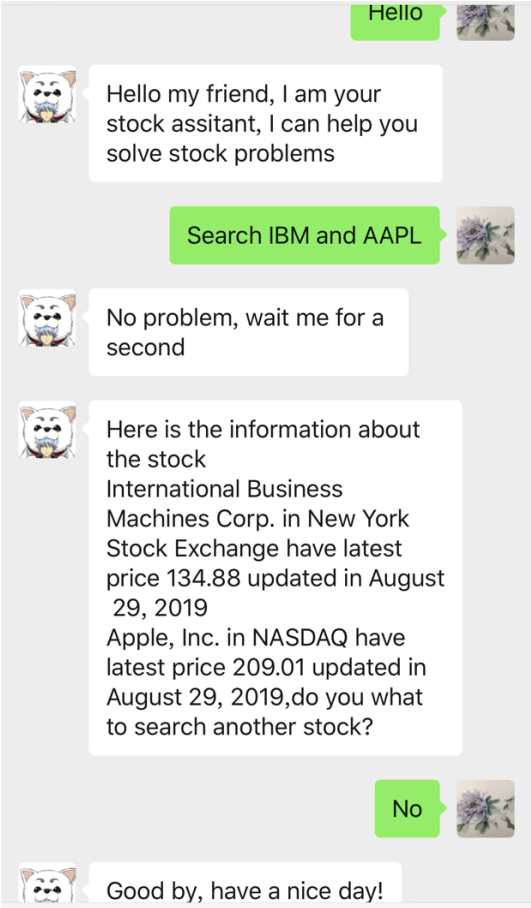


Fig 6.1 Multiply entities and negation intent

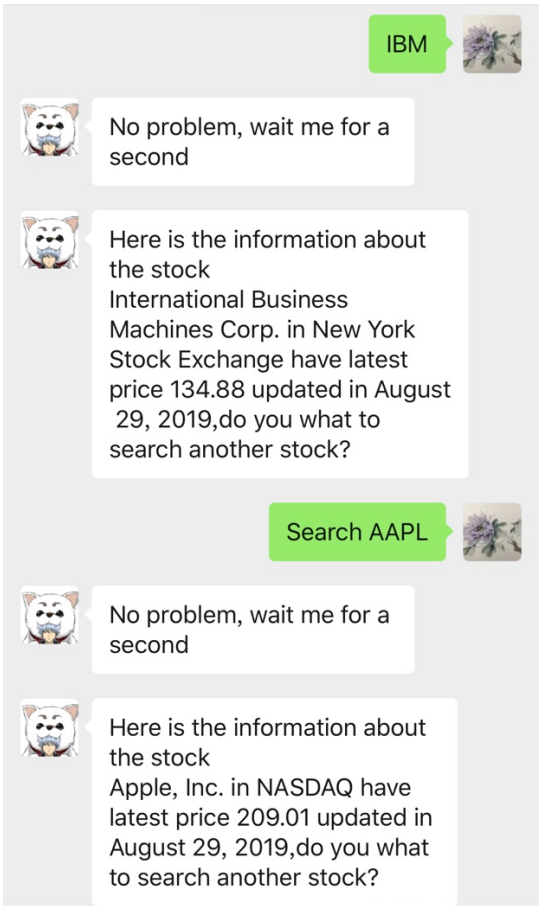


Fig 6.2 Multiply rounds conversation and pending