

# Chatbot Project Report

Fanzeng (Alex) Xia

## Contents

Introduction.....	3
Procedures.....	3
Background .....	3
Detailed Steps .....	3
Rasa NLU framework: Dialogue Data and Model Training .....	3
Stock Query API: iexfinance .....	5
Main code of the chatbot.....	5
Results.....	8
Python Console .....	8
WeChat .....	8
Conclusion .....	9
Appendix.....	9

## Introduction

This is a project followed by Dr. Fan Zhang at MIT IBM Watson AI Lab about researching an intelligent financial chat bot that is accurate and able to provide quick responses to posed questions as well as stock queries. The bot was finally deployed into WeChat. As leader, I took charge of most duties, ranging from using the Rasa NLU framework, implementing the intent recognition and named entity recognition, to enriching its response modes by means of incremental slot filling and negation.

## Procedures

### Background

In recent years, image recognition applications have made great progress due to the leap of deep learning.

However, due to too many semantic uncertainties, the development of Natural Language Processing (NLP) is much slower than computer vision. In this project, we understood the advantages and disadvantages of computers in NLP tasks and learned what we are already good at and what is still unresolved. Besides, we got to know how chat bots like IBM Watson, Deep Blue, Alpha Go, etc. work. In order to practice and integrate the NLP related contents we have learned, we decided to make a chat robot that can implement intelligent stock inquiry.

### Detailed Steps

#### Rasa NLU framework: Dialogue Data and Model Training

Rasa NLU is an open-source natural language processing tool for intent classification and entity extraction in chatbots. For example, taking a sentence like:

```
"I am looking for a Mexican restaurant in the center of town"
```

The implementation of model will return structured data (.json) like:

```
{
  "intent": "search_restaurant",
  "entities": {
    "cuisine": "Mexican",
    "location": "center"
  }
}
```

Due to the specific task on stock queries, we tried to come up with all possible conversation scenarios to create our own training data set with 3 different intents, example as shown below:

```

## intent:stock_search
- [AAPL] (company)
- Show me [AAPL] (company)
- Show me [TSLA] (company)
- [stock price] (price)
- [the price] (price)
- [price of it] (price)
- [price] (price)
- [volume] (volume)
- umm, [the trading volume] (volume)
- [volume] (volume) please
- i'm looking for a stock

## intent:greet
- hey
- hello
- hi
- good morning
- good evening
- hey there
- how are you
- how are u
- what's up
- wassup
- how's it going
- what's popping
- what's good
- how are we doing today
- what can you do for me?

## intent:thankyou
- thanks!
- thank you
- thx
- thanks very much
- great!
- awesome, thank you
- got you! Thanks
- alright
- hurray
- good night thanks
- nice
- nice job

```

After finishing the preparation of our NLU training data and saving it in a file called *nlu.md*, we can define our machine learning model inside Rasa framework using the *nlu\_config.yml* file. Next, type **python -m rasa\_nlu.train -c nlu\_config.yml --data nlu.md -o models --fixed\_model\_name nlu --project current --verbose** in cmd to train the model based on all the prepared files above.

The final interpreter model will be saved at *./models/current/nlu* relative to our working directory.

With the created model, we tried it through Python code to test whether we have implemented the function of named entity recognition well:

```

from rasa_nlu.model import Interpreter
import json

interpreter = Interpreter.load("C:/xfz/chatbot/models/current/nlu")
message = "show me Tesla"
result = interpreter.parse(message)
print(json.dumps(result, indent=2))

```

Result:

```

INFO:tensorflow:Restoring parameters from C:/xfz/chatbot/models/current/nlu\intent_classifier_tensorflow_embedding.ckpt
{
  "intent": {
    "name": "stock_search",
    "confidence": 0.9619808197021484
  },
  "entities": [
    {
      "start": 8,
      "end": 13,
      "value": "Tesla",
      "entity": "company",
      "confidence": 0.49892920091333937,
      "extractor": "ner_crf"
    }
  ]
}

```

## Stock Query API: iexfinance

Before writing the whole code logic of the chatbot, firstly we tested the function of stock query using iexfinance api in order to get the stock information (volume, open price and close price) instantly. It turned to be working really well using `get_open()`, `get_price()`, and `get_volume()` functions:

```
In [3]: from iexfinance import Stock
        tsla = Stock("TSLA").get_volume()
        print(tsla)
```

```
Out[3]: 8101194
```

## Main code of the chatbot

1. `send_message()`: to control the overall state transition based on the input message and print the conversation information on Python console.

```
def send_message(policy, state, message):
    global response
    print("USER : {}".format(message))
    new_state, response = respond(policy, state, message)
    print("BOT : {}".format(response))
    return new_state
```

2. `respond()`: generate our response based on our pre-defined policy.

```
def respond(policy, state, message):
    new_msg = interpret(message)
    policy = policyrule()
    (new_state, response) = policy[(state, new_msg)]
    return new_state, response
```

3. `interpret()`: achieve the intent recognition and named entity recognition tasks based on our Rasa NLU model.

```
def interpret(message):
    global flag
    global remember

    result = interpreter.parse(message)
    intent = result.get('intent').get('name')
    if intent == None:
        intent = 'none'
    if intent == 'stock_search':
        if result.get('entities')[0].get('entity') == 'company' or flag:
            stock_pending(message)
        else:
            stock_return(message)
    return intent
```

4. `stock_return()`: the response logic when there is no recognized pending state in user message:

```

def stock_return(message):
    result = interpreter.parse(message)

    global stock_re
    global setting

    if 'open' in result.get('entities')[0].get('value'):
        company = result.get('entities')[1].get('value')
        for i in setting:
            if company in i.get('name'):
                company = i.get('symbol')
        open_price = Stock(company).get_open()
        stock_re = "The open price is {}".format(open_price)

    if 'close' in result.get('entities')[0].get('value'):
        company = result.get('entities')[1].get('value')
        for i in setting:
            if company in i.get('name'):
                company = i.get('symbol')
        close_price = Stock(company).get_price()
        stock_re = "The close price is {}".format(close_price)

    if 'volume' in result.get('entities')[0].get('value'):
        company = result.get('entities')[1].get('value')
        for i in setting:
            if company in i.get('name'):
                company = i.get('symbol')
        volume = Stock(company).get_volume()
        stock_re = "The volume is {}".format(volume)

```

## 5. stock\_pending(): the response logic when there is pending state in user message

```

def stock_pending(message):
    result = interpreter.parse(message)

    global stock_re
    global remember
    global flag
    global setting

    company = ''

    if 'open' in result.get('entities')[0].get('value'):
        if remember != '':
            company = remember
        else:
            company = result.get('entities')[1].get('value')
        for i in setting:
            if company in i.get('name'):
                company = i.get('symbol')
        open_price = Stock(company).get_open()
        stock_re = "The open price is {}".format(open_price)

    if 'close' in result.get('entities')[0].get('value'):
        if remember != '':
            company = remember
        else:
            company = result.get('entities')[1].get('value')
        for i in setting:
            if company in i.get('name'):
                company = i.get('symbol')
        close_price = Stock(company).get_price()
        stock_re = "The close price is {}".format(close_price)

    if 'volume' in result.get('entities')[0].get('value'):
        if remember != '':
            company = remember
        else:
            company = result.get('entities')[1].get('value')
        for i in setting:
            if company in i.get('name'):
                company = i.get('symbol')
        volume = Stock(company).get_volume()
        stock_re = "The volume is {}".format(volume)

    if result.get('entities')[0].get('entity') == 'company':
        stock_re = "Do you wanna look at open price, close price or volume?"
        remember = result.get('entities')[0].get('value')
        for i in setting:
            if remember in i.get('name'):
                remember = i.get('symbol')

    flag = True

```

6. Define 4 transition states that will be used in the response policy state logic, as well as the `policyrule()` for generating the response pattern:

```
# Define the states
INIT = 0
PENDING = 1
CHOOSSED = 2
THANK = 3

# Define the policy rules dictionary
def policyrule():
    global stock_re
    policy = {
        (INIT, "none"): (INIT, "I'm sorry - I'm not sure how to help you"),
        (INIT, "greet"): (PENDING, "Hello. I'm a stock inquiry bot. You can ask questions about the stock of a specific company"),
        (INIT, "thankyou"): (THANK, "You are very welcome"),
        (INIT, "stock_search"): (CHOOSSED, stock_re),

        (PENDING, "none"): (PENDING, "I'm sorry, which stock do you wanna look at?"),
        (PENDING, "greet"): (PENDING, "I'm sorry, which stock do you wanna look at?"),
        (PENDING, "thankyou"): (PENDING, "I'm sorry, which stock do you wanna look at?"),
        (PENDING, "stock_search"): (CHOOSSED, stock_re),

        (CHOOSSED, "none"): (CHOOSSED, "I'm sorry, which stock do you wanna look at?"),
        (CHOOSSED, "greet"): (CHOOSSED, "I'm sorry, which stock do you wanna look at?"),
        (CHOOSSED, "thankyou"): (THANK, "You are very welcome"),
        (CHOOSSED, "stock_search"): (CHOOSSED, stock_re),

        (THANK, "none"): (INIT, "I'm sorry - I'm not sure how to help you"),
        (THANK, "greet"): (PENDING, "Hello. I'm a stock inquiry bot. You can ask questions about the stock of a specific company"),
        (THANK, "thankyou"): (THANK, "You are very welcome"),
        (THANK, "stock_search"): (CHOOSSED, stock_re),
    }
    return policy
```

7. Introduce the companies' name-symbol json file to achieve the transition to stock code (e.g.: Tesla to TSLA)

```
f = open("C:/xfz/chatbot/symbols.json", encoding='utf-8')
global setting
setting = json.load(f)
```

8. Get the message from WeChat using wxpy api and test all the functions:

```
# Send the messages
global state
state = INIT

from wxpy import *

bot = Bot(cache_path=True)

my_friend = bot.friends().search('Shao')[0]

@bot.register(my_friend)
def my_friendnd_message(msg):
    global state
    global response
    message = str(msg.text)
    state = send_message(policyrule(), state, message)
    return response
```

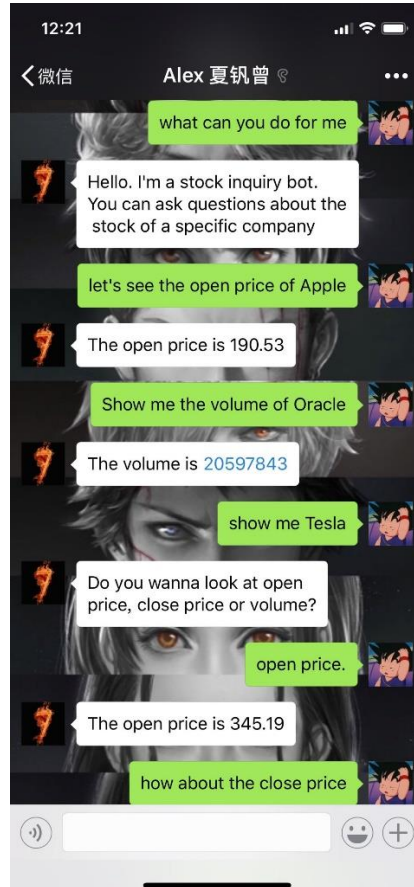
```
Getting uuid of QR code.  
Downloading QR code.  
Please scan the QR code to log in.  
Please press confirm on your phone.  
Loading the contact, this may take a little while.  
Login successfully as Alexfz
```

## Results

### Python Console

```
USER : what can you do for me  
BOT : Hello. I'm a stock inquiry bot. You can ask questions about the stock of a specific company  
USER : let's see the open price of Apple  
BOT : The open price is 190.53  
USER : Show me the volume of Oracle  
BOT : The volume is 20597843  
USER : show me Tesla  
BOT : Do you wanna look at open price, close price or volume?  
USER : open price.  
BOT : The open price is 345.19  
USER : how about the close price  
BOT : The close price is 354.31  
USER : what about the volume  
BOT : The volume is 7275706  
USER : thank you! bye  
BOT : You are very welcome
```

### WeChat





## Conclusion

My chat bot works well, it can implement the intent recognition and named entity recognition, and achieve its response modes by means of incremental slot filling and negation.

In the future, I would consider training another Seq2seq chatbot by TensorFlow's RNN (LSTM or GRU) api to achieve better functions and hone my skills.

## Appendix

All the code can be seen on <https://github.com>