智能聊天机器人项目总结报告

项目：智能聊天机器人

学校：北京理工大学

专业：软件工程

报告人：林壑峰

项目时间：2018 年 10 月——2018 年 11 月

一、项目背景

人工智能已经成为了现如今最热门的先进技术领域之一。AlphaGo，AlphaGo-Zero 等等都是人工智能前沿技术的代表。人工智能依托发展较为完善的机器学习和深度学习技术，逐渐演变成了多分支的高端产业，如智能语音识别，智能图像处理等等。而本次项目所集中探讨和实现的是智能聊天机器人。智能聊天机器人其实就是一种模拟人类对话交流的程序。最早的图灵测试就已经就智能聊天机器人的一个最终目的阐述了出来。简单来说，图灵测试就是测试一个聊天机器人能不能比较好地胜任和人类聊天这项任务。所以聊天机器人在计算机发展的初期就已经有初步的定义了。智能聊天机器人主要依赖的技术之一就是 NLP（自然语言处理）。自然语言处理的主要目的就是为了减少人机交互的障碍，让机器人听懂人类的语言。最早的聊天机器人是六七十年代的 ELIZA 和 PARRY，虽然只是模拟笔谈，但是也是行业中迈出的一大步。随着相关技术的迅速发展，智能聊天机器人技术也在飞速地更新。近年来，基于聊天机器人系统的应用层出不穷。从应用场景的角度来看，可以分为在线客服、娱乐、教育、个人助理和智能问答五个种类。

在线客服聊天机器人系统的主要功能是同用户进行基本沟通并自动回复用户有关产品或服务的问题，以实现降低企业客服运营成本、提升用户体验的目的。其应用场景通常为网站首页和手机终端。代表性的商用系统有小 I 机器人、京东的 JIMI 客服机器人等。用户可以通过与 JIMI 聊天了解商品的具体信息以及反馈购物中存在的问题等。值得称赞的是，JIMI 具备一定的拒识能力，即能够知道自己不能回答用户的哪些问题以及何时应该转向人工客服。

娱乐场景下聊天机器人系统的主要功能是同用户进行开放主题的对话，从而实现对用户的精神陪伴、情感慰藉和心理疏导等作用。其应用场景通常为社交媒体、儿童玩具等。代表性的系统如微软“小冰”、微信“小微”、“小黄鸡”、“爱情玩偶”等。其中微软“小冰”和微信“小微”除了能够与用户进行开放主题的聊天之外，还能提供特定主题的服务，如天气预报和生活常识等。

应用于教育场景下的聊天机器人系统根据教育的内容不同包括构建交互式的语言使用环境，帮助用户学习某种语言；在学习某项专业技能中，指导用户逐步深入地学习并掌握该技能；在用户的特定年龄阶段，帮助用户进行某种知识的辅助学习等。其应用场景通常为具备人机交互功能的学习、培训类软件以及智能玩具等。这里以科大讯飞公司的开心熊宝（具备移动终端应用软件和实体型玩具

两种形态）智能玩具为例，“熊宝”可以通过语音对话的形式辅助儿童学习唐诗、宋词以及回答简单的常识性问题等。

个人助理类应用主要通过语音或文字与聊天机器人系统进行交互，实现个人事务的查询及代办功能，如天气查询、空气质量查询、定位、短信收发、日程提醒、智能搜索等，从而更便捷地辅助用户的日常事务处理。其应用场景通常为便携式移动终端设备。代表性的商业系统有 Apple Siri、Google Now、微软 Cortana、出门问问等。其中，Apple Siri 的出现引领了移动终端个人事务助理应用的商业化发展潮流。，Apple Siri 随着 IOS 5 一同发布，具备聊天和指令执行功能，可以视为移动终端应用的总入口，然而受到语音识别能力、系统本身自然语言理解能力的不足以及用户使用语音和 UI 操作两种形式进行人机交互时的习惯差异等限制，Siri 没能真正担负起个人事务助理的重任。

智能问答类的聊天机器人主要功能包括回答用户以自然语言形式提出的事实型问题和需要计算和逻辑推理型的问题，以达到直接满足用户的信息需求及辅助用户进行决策的目的。其应用场景通常作为问答服务整合到聊天机器人系统中。典型的智能问答系统除了 IBM Watson 之外，还有 Wolfram Alpha 和 Magi，后两者都是基于结构化知识库的问答系统，且分别仅支持英文和中文的问答。

另外，很多的公司都提供的 API 接口供客户体验和编写聊天机器人，比如 IBM 的 Watson Assistant。大部分处理自然语言的库和训练需要的语言数据都有提供，体验者只需在线上进行简单的操作就可以搭建出自己所需要的聊天机器人。而且，聊天机器人的成品可以作为产品进行出售。这一领域的利润不可小觑。很多电商平台也都使用了聊天机器人来提升自己的服务质量。所以聊天机器人技术非常热门也非常重要。

二、相关技术回顾

1 正则表达式

正则表达式就是一个字符串，用来匹配、检索或替换符合一定规则的字符串。也就是说，正则表达式是一个自定的规则。编写正则表达式也有一定的限制。正则表达式在聊天机器人中的应用就是在语言中提取有一定规则的信息，比如电话号码、邮编、ID 等等；或者直接匹配语言中的句子来判断一个句子的意图，也就是聊天机器人中的模式匹配问题，比如匹配一个句子中的否定词等等。正则表达式虽然形式简单，但是能在很多情况下简化程序，达到很好的识别效果。

2 最近邻分类

自然语言处理中，我们需要把一个由单词组成的句子转化成能被程序处理的词向量。词向量就是来自词汇表的单词或短语被映射到实数的向量。而这些

数据经过人工标记分类到了对应的意图分类中，如喜欢，厌恶，中立等等。而测试所用的句子经过相同的处理后，也转化成了词向量。将测试数据对应的词向量和训练数据对应的词向量进行距离计算（距离公式根据情况而定，可以用欧几里得公式）。与测试词向量距离最近的训练词向量对应的意图在这种情况下可以被认为是测试词向量对应的意图。这种方法和机器学习的 KNN 方法很类似，都是通过相同空间下的距离计算出意图。

3 支持向量机分类

支持向量机分类就是在上述的最近邻分类的基础上引入了机器学习的支持向量机方法，以进行意图识别。

传统的支持向量机在解决小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中。主要思想是将线性不可分的一组样本数据映射到高维空间上使其线性可分，核函数又可以在不知道高维空间的数据表示时对于数据进行运算处理。

而在课上介绍的方法中，在词向量构建完成后，就对已有的数据进行训练并分类，再对测试数据进行预测，是一种典型的监督学习的方法，因为源域和目标域的标签都已知。这种分类对于词向量特征的提取会更加准确和到位，但是如果数据过多，效率可能会很低。

4 实体抽取

对于自然语言处理来说，很重要的一环就是对于句子实体的理解。有了句子中的实体，才能分析实体所包含的特征以及句子整体的语义。而 rasa_nlu 就提供了这样一个自然语言处理的库，用于训练数据、抽取实体、理解意图等等。需要注意的是，实体的抽取很依赖于前期训练数据的构建，所以前期的准备和数据的构建、处理非常关键。

5 数据库技术

很显然，有一部分聊天和交流是以检索为目的的，比如搜索最近的一家图书馆、好吃又实惠的饭店等等。这就需要在储存数据的数据库中进行检索并返回检索结果。数据库技术实际上是要和实体抽取紧密结合的，因为检索数据库需要相应的实体标签和对应的检索数值或者范围。

而数据库本身分为关系型数据库和非关系型数据库。关系型数据库，是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。关系模型由关系数据结构、关系操作集合、关系完整性约束三部分组成。非关系型数据库，也就是 NoSQL，有四种分类：键值存储数据库，列

存储数据库，文档型数据库，图像数据库。无论是哪种数据库，都不需要预定义模式，不需要共享架构，且弹性可扩展。

6 Docker

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。一个完整的 Docker 有以下几部分组成：dockerClient 客户端，Docker Daemon 守护进程，Docker Image 镜像，Docker Container 容器。Docker 主要是根据用户需要来虚拟一些环境，管理组件以达到项目要求。

三、项目实现概况

本次项目的主题是基于 rasa_nlu 等 python 库的智能聊天机器人构建。项目的目标是构建一个简单的金融聊天机器人，以实现一些简单的关于股票信息问题的对话。我构建的金融聊天机器人主要是运用了意图识别、实体抽取、多轮多次查询等等技术，实现对于股票任意时间（以线上数据库为准）的交易额、市值以及股价问询的回答。主要程序结构如下：

为了训练模型，我们需要自行构建 json 数据，并用 rasa_nlu 进行训练。json 文件部分结构如下所示：

```
},
{
  "text": "I want to know the Tesla's close price",
  "intent": "close",
  "entities": [
    {
      "start": 19,
      "end": 24,
      "value": "Tesla",
      "entity": "company"
    }
  ]
},
{
  "text": "I want to know the Apple's close price",
  "intent": "close",
  "entities": [
    {
      "start": 19,
      "end": 24,
      "value": "Apple",
      "entity": "company"
    }
  ]
},
```

训练程序如下所示：

```
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("/Users/linhefeng/config_spacy.yml"))

# Load the training data
training_data = load_data('/Users/linhefeng/demo-rasa.json')

# Create an interpreter by training the model
interpreter = trainer.train(training_data)
```

首先定义了一些常量，用于在各个函数之间进行传递使用，比如 month_list 字典就是用来把句子中的英文单词形式的月份信息转换成数字表示的月份信息，；policy 是规定了包括初始状态、查询收盘价状态等等状态的跳转规则。下面是全部的常量定义：

```
import re

pattern_day = re.compile('[0-9]{1,2}[a-z]{2}')
pattern_num = re.compile('[0-9]{1,2}')
pattern_year = re.compile('[0-9]{4}')
```

```
In [350]: company_dict = {'apple': 'AAPL', 'tesla': 'TSLA', 'microsoft': 'MSFT', 'alcoa': 'AA'}
          month_list = {}
          month_list['January'] = '1'
          month_list['February'] = '2'
          month_list['March'] = '3'
          month_list['April'] = '4'
          month_list['May'] = '5'
          month_list['June'] = '6'
          month_list['July'] = '7'
          month_list['August'] = '8'
          month_list['September'] = '9'
          month_list['October'] = '10'
          month_list['November'] = '11'
          month_list['December'] = '12'

          INIT = 0
          CLOSE = 1
          TIME = 2
          VOLUME = 3
          VALUE = 4
          PRICE = 5

          policy = {
              (INIT, 'close'): (CLOSE, 'Searching...'),
              (INIT, 'none'): (INIT, 'You can ask me for what you want.'),
              (CLOSE, 'time'): (TIME, 'Searching for details...'),
              (CLOSE, 'deny'): (CLOSE, 'Already delete. Searching...'),
              (INIT, 'volume'): (VOLUME, 'Searching...'),
              (VOLUME, 'time'): (TIME, 'Searching for details...'),
              (VOLUME, 'deny'): (VOLUME, 'Already delete. Searching...'),
              (INIT, 'value'): (VALUE, 'Searching...'),
              (VALUE, 'time'): (TIME, 'Searching for details...'),
              (VALUE, 'deny'): (VALUE, 'Already delete. Searching...'),
              (INIT, 'price'): (PRICE, 'Searching...')
          }
```

send_message 函数定义了对话的基本框架，先是输出 USER 的请求，再是通过调用 respond 函数对请求信息 message 进行相应的处理，之后在 respond 函数体里输出处理过后的回答。这里需要注意的是，函数需要返回参数 params

和当前状态 state，因为这涉及到下一次状态信息的跳转和请求信息分析。下面是 send_message 的函数体：

```
# 将message传递给bot
def send_message(message, policy, state, params, intent_dict, answer):
    print("USER : {}".format(message))
    answer = []
    new_state, response, params, answer = respond(policy, state, message, params, intent_dict, answer)
    return new_state, params, answer
```

respond 函数中先是进行了状态的跳转（这里利用到了 interpret 函数的意图识别功能），同时输出了状态跳转的时候的必要信息（这个信息是储存在 policy 中的，不是通过参数传递进来的信息）。之后对于请求信息进行分析，也就是用到了 analyse 函数。值得注意的是，这里对于参数 params 进行了一个清空，因为我们默认在进行时间条件的限制之后，用户会进行下一轮不同 intent 的问询。下面是 respond 函数体：

```
# 根据message进行回复
def respond(policy, state, message, params, intent_dict, answer):
    (new_state, response) = policy[(state, interpret(message))]
#     print("BOT : {}".format(response))
    answer.append(response)
    params, answer = analyse(message, params, intent_dict, answer)
    # 默认查询完特定时间的股票信息后，状态重置
    empty = {}
    if new_state == TIME:
        if params == empty:
            new_state = INIT
        else:
            new_state = state
    return new_state, response, params, answer
```

interpret 函数主要是对信息进行意图识别，并返回相应的状态信息。每一个状态信息都与 policy 中的状态信息对应。下面是 interpret 函数：

```
# 识别意图并将状态返回
def interpret(message):
    intent = interpreter.parse(message)['intent']['name']
    words = [str(i) for i in message.split()]
    for i in range(len(words)):
        if words[i] == 'no' or words[i] == 'not':
            intent = 'deny'
            break
        else:
            continue
    # 根据意图返回对应状态
    if intent == 'close':
        return 'close'
    elif intent == 'time':
        return 'time'
    elif intent == 'volume':
        return 'volume'
    elif intent == 'value':
        return 'value'
    elif intent == 'deny':
        return 'deny'
    elif intent == 'price':
        return 'price'
    else:
        return 'none'
```

最核心的部分就是 analyse 函数了。它对于信息进行分析，抽取实体，并根据识别出的意图调用 API 进行查询，最后返回结果。值得一提的是，在 analyse 里还包含了对于否定信息的处理方式，主要是通过上一次 intent 确定查找范围，并且根据用户输入删除相应信息，进行这一轮的查询。另外，对于时间的处理也是一门学门。我们的方法是先用正则表达式抽取出月和日，再对月和日进行处理，把它们转化成数字形式并存入 params 中。下面是 analyse 函数体：

```
from iexfinance import get_historical_data
from datetime import datetime

# 抽取实体，并对于message进行解析、回复
def analyse(message, params, intent_dict, answer):
    day = month = 0
    year = '2018'
    entities = interpreter.parse(message)['entities']
    intent = interpreter.parse(message)['intent']['name']
    words = [str(i) for i in message.split()]
    for i in range(len(words)):
        if words[i] == 'no' or words[i] == 'not':
            intent = 'deny'
            break
        else:
            continue
    # 查询意图是查询实时股价
    if intent == 'price':
        for ent in entities:
            params[str(ent['entity']) + '_' + str(ent['value'])] = str(ent['value'])
        for company in params:
            stock_search = Stock(company_dict[str(params[company])])
#             print("BOT : The price of {0} is {1}".format(params[company], stock_search.get_price()))
            answer.append("The price of {0} is {1}".format(params[company], stock_search.get_price()))
        intent dict.append(intent)
```

```
    # 查询意图是查询股票收盘价
    if intent == 'close':
        for ent in entities:
            params[str(ent['entity']) + '_' + str(ent['value'])] = str(ent['value'])
        for company in params:
            stock_search = Stock(company_dict[str(params[company])])
#             print("BOT : The close price of {0} is {1}".format(params[company], stock_search.get_close()))
            answer.append("The close price of {0} is {1}".format(params[company], stock_search.get_close()))
        intent_dict.append(intent)
    # 查询意图是查询股票交易额
    elif intent == 'volume':
        for ent in entities:
            params[str(ent['entity']) + '_' + str(ent['value'])] = str(ent['value'])
        for company in params:
            stock_search = Stock(company_dict[str(params[company])])
#             print("BOT : The volume of {0} is {1}".format(params[company], stock_search.get_volume()))
            answer.append("The volume of {0} is {1}".format(params[company], stock_search.get_volume()))
        intent_dict.append(intent)
    # 查询意图是查询股票市值
    elif intent == 'value':
        for ent in entities:
            params[str(ent['entity']) + '_' + str(ent['value'])] = str(ent['value'])
        for company in params:
            stock_search = Stock(company_dict[str(params[company])])
#             print("BOT : The value of {0} is {1}".format(params[company],
#                                                          stock_search.get_financials()[0]['totalAssets']))
            answer.append("The value of {0} is {1}".format(params[company],
                                                           stock_search.get_financials()[0]['totalAssets']))
        intent_dict.append(intent)
```

```
    # 查询意图是对于之前的一些查询条件进行否定
    elif intent == 'deny':
        flag = False
        for i in range(len(words)):
            words[i] = words[i].lower()
            if words[i] == 'no' or words[i] == 'not':
                flag = True
            key = 'company_' + str(words[i])
            if key in params and flag:
                params.pop(key)
        # 根据上一次状态进行相应回复
        if intent_dict[len(intent_dict)-1] == 'close':
            for company in params:
                stock_search = Stock(company_dict[str(params[company])])
#                 print("BOT : The close price of {0} is {1}".format(params[company], stock_search.get_close()))
                answer.append("The close price of {0} is {1}".format(params[company], stock_search.get_close()))
        elif intent_dict[len(intent_dict)-1] == 'value':
            for company in params:
                stock_search = Stock(company_dict[str(params[company])])
#                 print("BOT : The value of {0} is {1}".format(params[company],
#                                                              stock_search.get_financials()[0]['totalAssets']))
                answer.append("The value of {0} is {1}".format(params[company],
                                                               stock_search.get_financials()[0]['totalAssets']))
        elif intent_dict[len(intent_dict)-1] == 'volume':
            for company in params:
                stock_search = Stock(company_dict[str(params[company])])
#                 print("BOT : The volume of {0} is {1}".format(params[company], stock_search.get_volume()))
                answer.append("The volume of {0} is {1}".format(params[company], stock_search.get_volume()))
        elif intent_dict[len(intent_dict)-1] == 'price':
            for company in params:
                stock_search = Stock(company_dict[str(params[company])])
#                 print("BOT : The price of {0} is {1}".format(params[company], stock_search.get_price()))
                answer.append("The price of {0} is {1}".format(params[company], stock_search.get_price()))
```

```
    # 根据时间进行给定时间的股票信息查询
    elif intent == 'time':
        # 对于句子中的月和日进行解析
        tmp = [str(i) for i in message.split()]
        for i in range(len(tmp)):
            if tmp[i] in month_list:
                month = int(month_list[tmp[i]])
            if re.search(pattern_day, tmp[i]) is not None:
                day = int(pattern_num.findall(pattern_day.findall(tmp[i])[0])[0])
            if re.search(pattern_year, tmp[i]) is not None:
                year = int(pattern_year.findall(tmp[i])[0])
            else:
                year = 2018
                if year >= 2019:
#                     print('BOT : Sorry. Wrong about input date.')
                    answer.append('Sorry. Wrong about input date.')
                    return params, answer
        start = datetime(year, month, day)
        end = datetime(year, month, day)
        if int(month) < 10:
            month = '0' + str(month)
        if int(day) < 10:
            day = '0' + str(day)
        params['time'] = str(year) + '-' + str(month) + '-' + str(day)
        if intent_dict[len(intent_dict)-1] == 'value':
            for company in params:
                if company == 'time':
                    continue
                value_search = Stock(company_dict[str(params[company])])
                data = value_search.get_financials()
```

```
                for k in range(len(data)):
                    if data[k]['reportDate'] <= params['time']:
#                         print('BOT : The {3} of {0} at {1} is {2}.'.format(params[company], params['time'],
#                                                                            data[k]['totalAssets'],
#                                                                                intent_dict[len(intent_dict)-1]))
                        answer.append('The {3} of {0} at {1} is {2}.'.format(params[company], params['time'],
                                                                             data[k]['totalAssets'],
                                                                                 intent_dict[len(intent_dict)-1]))
                        break
                    else:
                        continue
        else:
            for company in params:
                if company == 'time':
                    continue
                info = get_historical_data(company_dict[str(params[company])],
                                           start, end)
                info_details = info[company_dict[params[company]]]
                if str(params['time']) not in info_details:
#                     print('BOT : Sorry. Wrong about input Date.')
                    answer.append('Sorry. Wrong about input Date.')
                    params['time'] = ''
                    print(answer)
                    return params, answer
                result = info_details[str(params['time'])][intent_dict[len(intent_dict)-1]]
#                 print('BOT : The {3} of {0} at {1} is {2}.'.format(params[company], params['time'],
#                                                                    result, intent_dict[len(intent_dict)-1]))
                answer.append('The {3} of {0} at {1} is {2}.'.format(params[company], params['time'],
                                                                     result, intent_dict[len(intent_dict)-1]))

            params = {}
    print(answer)
    return params, answer
```

对话结果：

```
USER : hello
['You can ask me for what you want.']
USER : I would like to know the close price of Tesla and Apple.
['Searching...', 'The close price of tesla is 354.31', 'The close price of apple is 193.53']
USER : not tesla
['Already delete. Searching...', 'The close price of apple is 193.53']
USER : How about March 8th
['Searching for details...', 'The close of apple at 2018-03-08 is 175.0336.']
USER : I want to know the Apple's close price.
['Searching...', 'The close price of apple is 193.53']
USER : How about February 20th in 2017
['Searching for details...', 'Sorry. Wrong about input Date.']
USER : How about February 21th in 2017
['Searching for details...', 'The close of apple at 2017-02-21 is 133.1211.']
USER : I want to know the price of Apple and Tesla
['Searching...', 'The price of apple is 193.53', 'The price of tesla is 354.31']
USER : not Apple
['Already delete. Searching...', 'The price of tesla is 354.31']
```

集成到微信的全部代码（比较简单和基础）：

In [230]:
```
from wxpy import *

bot = Bot()
```

```
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 Mccpiggy
```

In [231]:
```
#coding=utf8
my_friend = bot.friends().search('孙誉铭', sex=FEMALE)
print(my_friend)
```

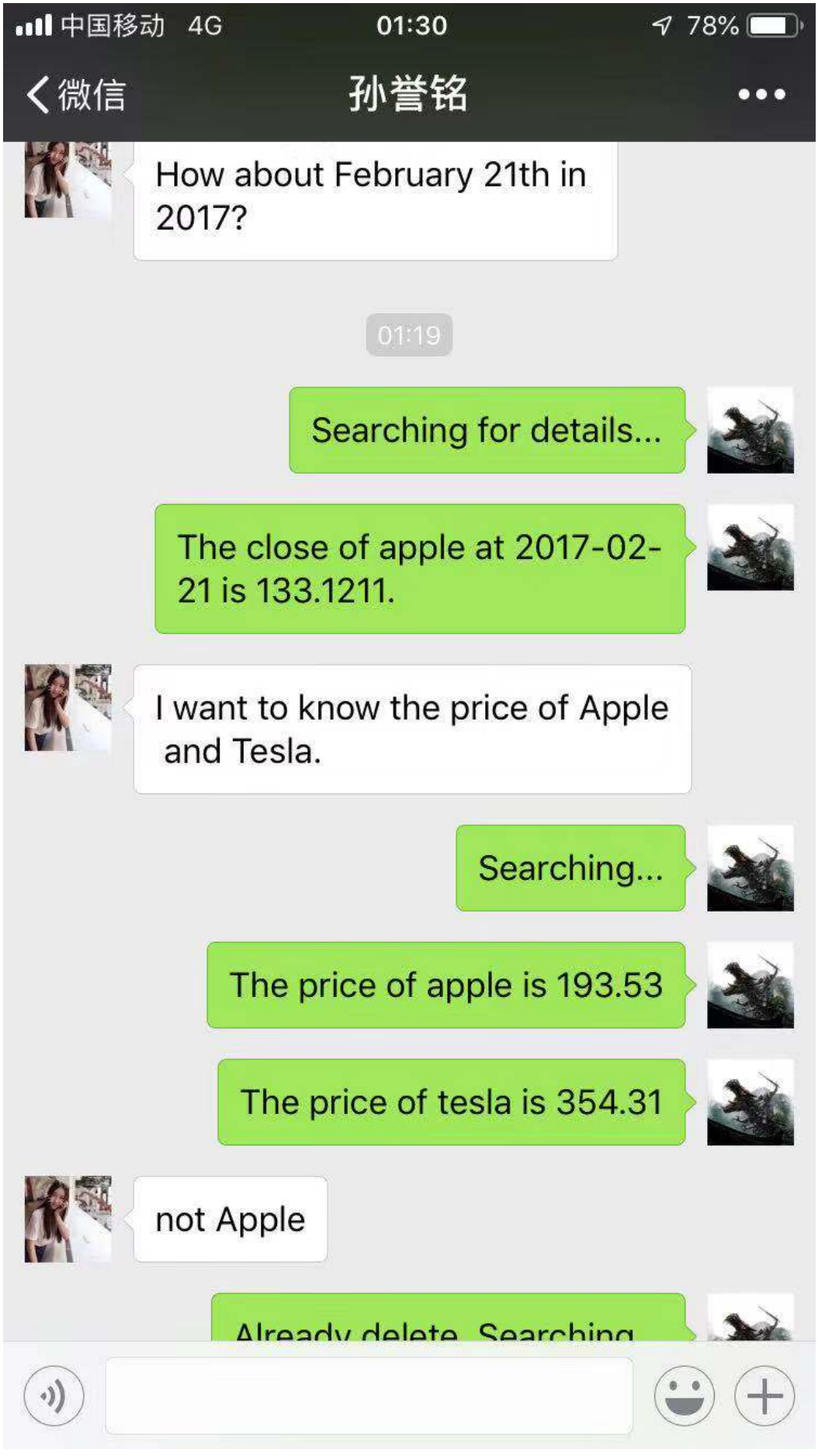
```
[<Friend: 孙誉铭>]
```

In [237]:
```
state = INIT
params = {}
intent_dict = []
answer = []

def test(msg):
    global state, params, answer
    state, params, answer = send_message(str(msg.text), policy, state, params, intent_dict, answer)
    return answer
```

In [238]:
```
@bot.register(my_friend)
def reply_my_friend(msg):
#     global state, params, answer
#     state, params, answer = send_message(str(msg.text), policy, state, params, intent_dict, answer)
    result = test(msg)
    for i in range(len(result)):
        msg.reply(result[i])
bot.join()
```

微信上对话结果部分示例：


中国移动 4G
01:30
78%
微信
孙誉铭
How about February 21th in 2017?
01:19
Searching for details...
The close of apple at 2017-02-21 is 133.1211.
I want to know the price of Apple and Tesla.
Searching...
The price of apple is 193.53
The price of tesla is 354.31
not Apple
Already delete. Searching

Searching...

The close price of tesla is 354.31

The close price of apple is 193.53

not Tesla

Already delete. Searching...

The close price of apple is 193.53

How about March 8th?

Searching for details...

The close of apple at 2018-03-08 is 175.0336.

值得一提的是，2017 年 2 月 20 日是周末，所以当天没有开盘，而聊天机器人给出了错误提示。

四、项目感想与收获

虽然本次项目只有短短的四周时间，但是和老师、同学交流的过程让我学到了非常多。我也更倾向于这种多人交流的课堂氛围，因为自己的疑问和想法能够第一时间表达出来，和老师能够交流得更加透彻，让自己对于技术和项目理解更加深刻。

在项目开始初期，我觉得本次项目比较简单，因为我是软件工程专业的学生，而且对于数据库、python、NLP 等等都有一定的了解和实践经验，所以感觉这次的项目会完成得很轻松。果然，在项目的初期教学过程中，我完成得很轻松，对于项目结果很是期待。

到了项目的中后期，由于前期的积累，我对于聊天机器人相关知识的掌握很到位，对于程序的理解和编写都游刃有余。所以在自己动手编写项目的时候显得不是很困难，可以熟悉运用 API 并且利用老师上课教授的知识进行拓展。我自己对于否定查询和 API 调用有着自己的想法，所以程序结构和上课时候所说的会有些不一样。

本次项目和孙誉铭同学合作得相当愉快，我们分工明确，团队协作，充分沟通，让项目推进得很顺利。我负责自然语言处理的代码编写，孙誉铭同学负责微信上的集成工作。项目结果也很令我们欣慰。

这次项目对我的影响很深远，首先让我学到了聊天机器人的相关知识，让我能更好地了解了人工智能的又一大应用。但是俗话说：师傅领进门，修行在个人。聊天机器人更加深入的知识和原理需要自己不断地学习和领悟，比如 spacy 和 rasa 的高级使用等等。其次我明白了团队合作和沟通的重要性。和同学一起合作项目的时候，我深刻明白一个团队中需要很深入的沟通和交流，才能让一个项目更好地向前推进。最后我想说，跟导师的沟通也是很关键的。和导师的一次次交流中，我明白了自己对于项目理解的问题，也对于聊天机器人的细节有了更深的认识。这对我编写整个项目有很大的帮助。并且导师会给我一些建设性的指导意见，对于我修改程序结构也有大有裨益。

我想在未来的学习和工作中我会对于聊天机器人这个领域继续深入地研究下去，保持对这个领域的热爱和好奇，让自己更加地充实和强大。