

Flight-Search Chatbot

PROJECT REPORT

Huixuan Wang | Aug 17, 2019

ABSTRACT

In this chatbot project, we build an intelligent chatbot using natural language that can help you search flights and get their information on Telegram.

For the interpretation of natural language, we mainly use Rasa NLU framework, spaCy package and pattern matching to extract intention and entities. Besides, some other techniques have been applied: SQL for temporary flights storage and suggestion providing; State machine for implement multi-round query technique; Incremental slot filling for constructing params in filtering.

Furthermore, the project uses python-telegram-bot library for Telegram access, lxml library for processing HTML, and pytesseract library for optical character recognition in the website.

In conclusion, our chatbot is able to get information about what you asked from the Internet and add filters by your request, thus provide you with various details about the flight you need.

Key words: natural language processing; chatbot

1 Introduction

With the blossom of artificial intelligence, natural language processing (NLP), as a primary part of AI, has made a great progress recently. From text embedding to machine translation, from recurrent neural network to virtual assistant, NLP has proved its significance and universality. Though not exceeding people's imagination, many commercial chatbots like Apple Siri and Google Now has already made our life different: more efficient and convenient.

Variety of circumstances can benefit from chatbots, from a simple one like stock-price acquirement to a complicated one like home control network. As a result, we choose to apply our chatbot related technique to a common field: flight search. Whether when we need to take an airplane, pick up friends from airport, or just for travel planning, we have to check the information of the timetable and the status of flights. Now, these can all be done in Telegram by a conversation with our chatbot.

Our chatbot is capable of such functions:

- ✧ Search all the flight by its departure and arrival city in one certain day
- ✧ Search flights by its flight number in one certain day
- ✧ Filter the flight results by the airline, departure/arrival time range, or the on-time performance
- ✧ Get details about scheduled departure/arrival time, actual departure/arrival time, historical on-time performance, status (preparing, cancel, arrived, etc.), type of aircraft, length of route, etc.

2 Technique

2.1 Intend Extraction

Intend extraction is always the first step for natural language understanding: get to know what the users want to say.

2.1.1 REGULAR EXPRESSION

Regular expression is a search pattern designed for string searching algorithms. By using regular expression, it will be much easier to find some message in certain type, like zip code and flight number. Besides, regular expression can help locate useful information in well-organized file like an HTML file

2.1.2 SPACY

spaCy is a pre-trained model for natural language processing, which provides a vector for every word. Thus, spaCy is capable of calculate the similarity of two sentences, extract entities like Date, Organization, Place, Person, etc., even find the ancestors of a certain word.

2.1.3 SUPPORT VECTOR MACHINE (SVM)

SVM are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings.

Rasa NLU in our project uses SVM with GridSearchCV for classifying intents, also Rasa is capable of extracting entities.

2.2 State Machine

State Machine is used for Multiple round multiple query technique. When having control of the state, the bot can have a more accurate understanding and response. For example, a message intended to affirm may mean nothing in the initial state, but does mean something in the option-choose state.

2.3 SQLite

SQL is a database management system, which can storage and query data simply and efficiently. We use a SQL database to store the flights that user required, and thus make a quick response when added filters.

2.4 Optical Character Recognition (OCR)

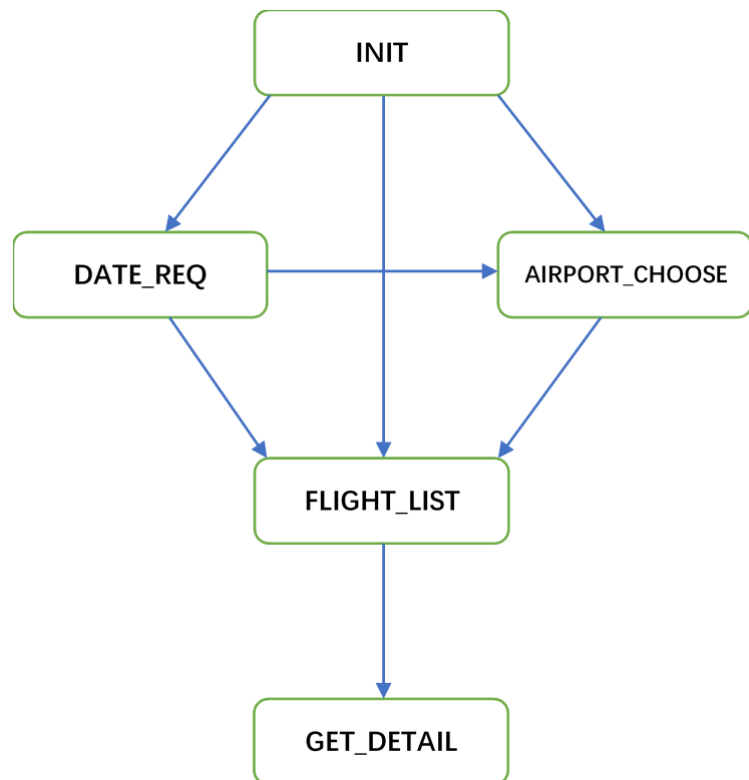
Since the website we used for data requirement has no API provided, we have to deal with the HTML file by ourselves. Unfortunately, some of the information like actual departure time and on-time performance are not in characters type, instead, image type. Thus, we have to use Python-tesseract (pytesseract) in our project, which is an optical character recognition (OCR) tool for python. That is, it will recognize and read the text embedded in images.

3 Implementation

3.1 STATE SETTING

In our project, five states have been set:

- **INIT:** Initial state, ready to start the chat
- **DATE_REQ:** Turn to this if the users provide no date, and mean to require for the date
- **AIRPORT_CHOOSE:** Turn to this if multiple airports is available in a city, thus get a precise airport.
- **FLIGHT_LIST:** Turn to this if departure/arrival airports (or the flight number), along with date are all set. Designed for showing flights list and adding filters as users required.
- **GET_DETAIL:** Turn to this if users choose a certain flight. Able to provide with more details about the users require.



Besides, particular response will be made when state changes (including remain itself). Due to the separation of their locations in the code, we shall not present them here.

3.2 Rasa for Intent Extraction

In our training data for Rasa NLU, we separate the intent into 7 categories: Greet, Goodbye, Affirm, Deny, Search_flight, Add_filter and Ask_detail.

Here show some typical examples in the training data for each category:

```
{
  "text": "hey there",
  "intent": "greet"
},
{
  "text": "hello",
  "intent": "greet"
},
{
  "text": "find me all the flights from beijing to shanghai in Jun 4",
  "intent": "search_flight"
},
{
  "text": "can you search the flight from Harbin to Shenzhen today?",
  "intent": "search_flight"
},
{
  "text": "I want some information about today's flights from Tokyo to Singapore",
  "intent": "search_flight"
},
{
  "text": "I prefer a midnight flight operated by British Airways",
  "intent": "add_filter",
  "entities": [
    {
      "start": 39,
      "end": 46,
      "value": "British",
      "entity": "airlines"
    },
    {
      "start": 11,
      "end": 19,
      "value": "midnight",
      "entity": "time"
    }
  ]
},
{
  "text": "yeah",
  "intent": "affirm"
},
{
  "text": "indeed",
  "intent": "affirm"
},
{
  "text": "i don't like it",
  "intent": "deny"
},
{
  "text": "it is not good",
  "intent": "deny"
},
{
  "text": "how about the flight status?",
  "intent": "ask_detail",
  "entities": [
    {
      "start": 21,
      "end": 27,
      "value": "status",
      "entity": "detail_type"
    }
  ]
},
{
  "text": "what's the type of the aircraft?",
  "intent": "ask_detail",
  "entities": [
    {
      "start": 11,
      "end": 15,
      "value": "type",
      "entity": "detail_type"
    }
  ]
},
]
```

Though Rasa is capable of extract entity, sometimes when it comes to proper nouns, Rasa doesn't work that well. Therefore, when intent is search_flight, Rasa is mainly responsible for intent extraction, not for entity extraction.

3.3 Entity Extraction

When entity comes with special patterns like a Flight Number, we use regular expression to extract it directly from message:

```
flight_num = re.search(r"\b[A-Za-z]{2}[0-9]{1,4}\b|\b[A-Za-z]{1}[0-9]{2,5}\b"
                      r"|\b[0-9][A-Za-z][0-9]{1,4}\b", message)
if flight_num:
    flight_num = flight_num.group(0)
```

Otherwise, we use Rasa and spaCy for entity extraction.

When Rasa tag a sentence with “search_flight”, spaCy takes over:

- Entities with the label of **GPE**, like {Beijing, Tokyo}: We now know that one of them is departure city and the other one is arrival city, so we use function **docs.ancestors()** from spaCy to find its ancestral words. If ancestors include {"from", "depart", "departure", "leave"}, we know it is departure city; if ancestors include {"to", "arrive", "reach", "get", "go"}, we know it is arrival city.

```
ancestor_list = list(doc[n].ancestors)
dep_word = {"from", "depart", "departure", "leave"}
arr_word = {"to", "arrive", "reach", "get", "go"}
for i in range(len(ancestor_list)):
    if str(ancestor_list[i]) in dep_word:
        dtype = "dep"
    elif str(ancestor_list[i]) in arr_word:
        dtype = "arr"
```

- Entities with the label of **DATE**, like {Jun 4}: We first use regular expression to separate days from months, then use a dictionary to transform month to its index number.

When Rasa tag a sentence with “add_filter” or “ask_detail”, we can get the entities directly from Rasa result, like the airlines choice, preferred time period, or the type of details that users required.

3.4 Data Obtaining on Internet

3.4.1 SEARCH AIRPORTS OF A CITY

API from *flightstats.com* is available for city-airports searching. By typing in the city name, we can get the results of airports available in Json type.

```
import requests

city="Shanghai"

url = 'http://www.flightstats.com/v2/api-next/search/airline-airport?query={}&type=airport&rqid=wz0b9ty387c'.format(
r = requests.get(url)
print(r.json())
```

```
{'data': [{'iata': 'PVG', 'fs': 'PVG', 'name': 'Shanghai Pudong International Airport'}, {'iata': 'SHA', 'fs': 'SHA', 'name': 'Shanghai Hongqiao International Airport'}]}
```

3.4.2 GET AIRLINE CODE BY ITS NAME

API from *flightstats.com* is available for airlines code requirement. By typing in the airlines name, we can get the results of its IATA code in Json type.

```
airline="Air China"

url = 'http://www.flightstats.com/v2/api-next/search/airline-airport?query={}&type=airline&rqid=wz0b9ty387c'.format(
r = requests.get(url)
print(r.json()["data"][0]["fs"])
```

```
CA
```

3.4.3 SEARCH FLIGHTS DATA

Because no suitable API has been found, we have to extract data from HTML file of the website, *variflight.com*.

As shown below, we can get almost every detail from this website:

北京首都---上海浦东 8月09号周五 共18次航班

航班信息	计划起飞	实际起飞	出发地	计划到达	实际到达	到达地	准点率	状态
 东方航空 MU5183	07:35	07:41	北京首都T2	09:50	09:30	上海浦东T1	77.42%	到达

Non character type

Using *lxml*, a feature-rich and easy-to-use library for processing HTML, we can easily get the data we need, except for the three items in image type:

actual time of departure/ arrival, and the on-time performance. We use Google's Tesseract-OCR to transform it into string type.

```
f5 = selector.xpath('div[@class="li_com"]/span[3]/img/@src') # 实际起飞
if f5:
    url = base_url + f5[0]
    resp = r.get(url)
    filename = './pictures' + '.png'
    with open(filename, 'wb') as f:
        f.write(resp.content)
    f5 = pytesseract.image_to_string(Image.open(filename)) → Image to string
    os.remove(filename)
    if len(f5) < 5: # 若识别不出‘.’或者‘.’ 进行拼接
        f5 = f5[:2] + ':' + f5[2:]
else:
    f5 = '--:--'
```

Here gets an url

3.5 SQL Database

3.5.1 STORAGE DATA

After get data from the Internet, we immediately add it to the SQL database, with each flight as a record. Each record has 14 elements, including flight number, departure airport, departure time, etc.


```
conn = sqlite3.connect('flight.db', check_same_thread=False)
c = conn.cursor()
c.execute("CREATE TABLE IF NOT EXISTS flight(f0 int,f1 text,f2 text,f3 text,f4 text,f5 text,"
        "f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 int,f13 int)")
c.execute("DELETE from flight")
value = "VALUES({}, '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', {}, {})" \
        .format(num, f1, f2, f3, f4[0], f5, f6[0], f7[0], f8, f9[0], f10, f11[0], f12, f13)
c.execute("INSERT INTO flight(f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13) {}".format(value))
c.execute("commit")
```

14 elements from website

3.5.2 DATA ACQUIRE

When presenting the flights result, we create SQL statement from natural language by adding params and negative params.

```
query = 'SELECT * FROM flight'
text = []
...
t = ' AND '.join(text)
if t:
    p = query + ' WHERE ' + t
else:
    p = query
c.execute("{}".format(p))
results = c.fetchall()
```



text including params and negative params

3.6 API for Telegram

With package python-telegram-bot and original Telegram bot API, our project can run with Telegram in 3 steps:

- Search BotFather in telegram, send '/newbot'
- Follow directions and register a new bot
- Copy the TOKEN to our project

```
def telegram(update, context):
    context.bot.send_message(chat_id=update.message.chat_id, text=main(update.message.text))

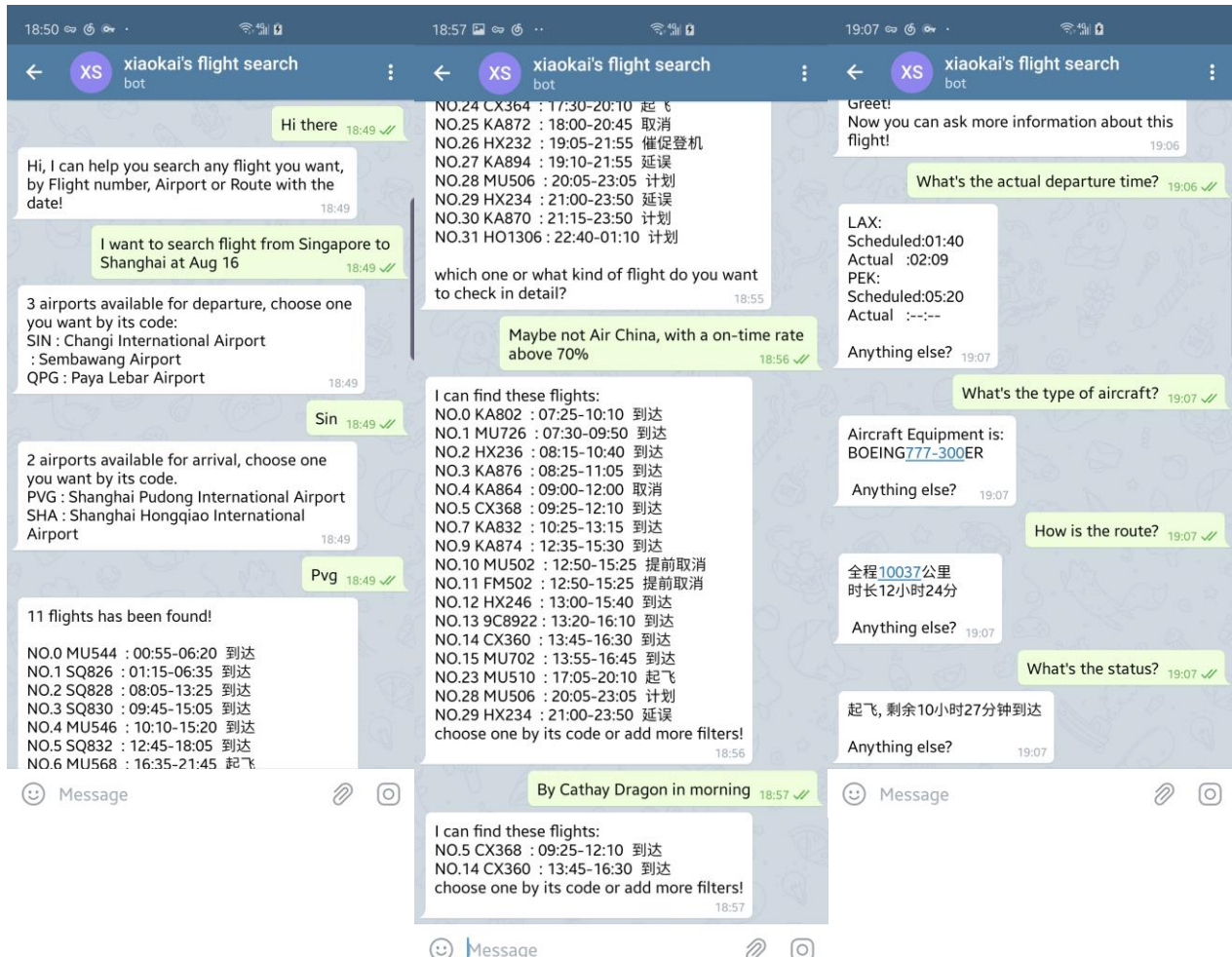
updater = Updater(token=TOKEN, use_context=True)
dispatcher = updater.dispatcher
handler = MessageHandler(Filters.text, telegram)
dispatcher.add_handler(handler)
updater.start_polling()
```



Our function

4 Results and Discussion

After many tests and modification, our chatbot has reached a relatively high level of intelligence and robustness. Here present some typical dialogues between users and our bot:



It is clearly seen that our chatbot can get user's meaning and continue the process by the state changing policy. Beyond that, our bot is always ready to give users additional guides when people get confused or type staff wrong. Thus, you can always ask what to do and how to do, also have no need to worry about making the chatbot confused or failed.