Building a FAQ Chatbot in Python – The Future of Information Searching

NLP PYTHON

YOGESH KULKARNI , JANUARY 21, 2018 / 0

SHARE

Introduction

What do we do when we need any information? Simple: "*We Ask, and Google Tells*".

But if the answer depends on multiple variables, then the existing Ask-Tell model tends to sputter. State of the art search engines usually cannot handle such requests. We would have to search for information available in bits and pieces and then try to filter and assemble relevant parts together. Sounds time consuming, doesn't it?

Source: Inbenta

This Ask-Tell model is evolving rapidly with the advent of chatbots (also referred to as just "bots").

This article talks about the development of a bot for extracting information related to the recently introduced Goods and Services Tax (GST) in India. Being a relatively new concept, a lot of people are still trying to understand how it works. Chatbots can provide such information in a natural and conversational way. This article demonstrates building a chatbot for answering queries related to GST. Let's call this a GST-FAQ Bot!

Table of Contents

1. Chatbots and NLP
2. Why Rasa-NLU?
3. Building GST FAQ bot architecture
4. Installation
5. Server
    - Steps to build the server of the chatbot
6. Client
7. Engine
8. Our chatbot in action

Chatbots and NLP

To know more about GST, like how to apply for registrations, tax slabs, etc., companies have posted Frequently Asked Questions (FAQs) on their websites. Going through that amount of information can be a painstaking process. In these situations, chatbots come in handy, effective and thus, have become enormously popular.

These days, Natural Language Processing (NLP), especially its component Natural Language Understanding (NLU), has allowed bots to have a greater understanding of language and context. They are becoming more intelligent in understanding the meaning of the search and can return very specific, context-based information.

Applications like WhatsApp, Facebook Messenger, Slack, etc. are increasingly being used by businesses. Bots are starting to replace websites-interface as well. From the considerable number of choices available for building a chatbot, this particular implementation uses the RASA-NLU library in Python.

Why RASA-NLU?

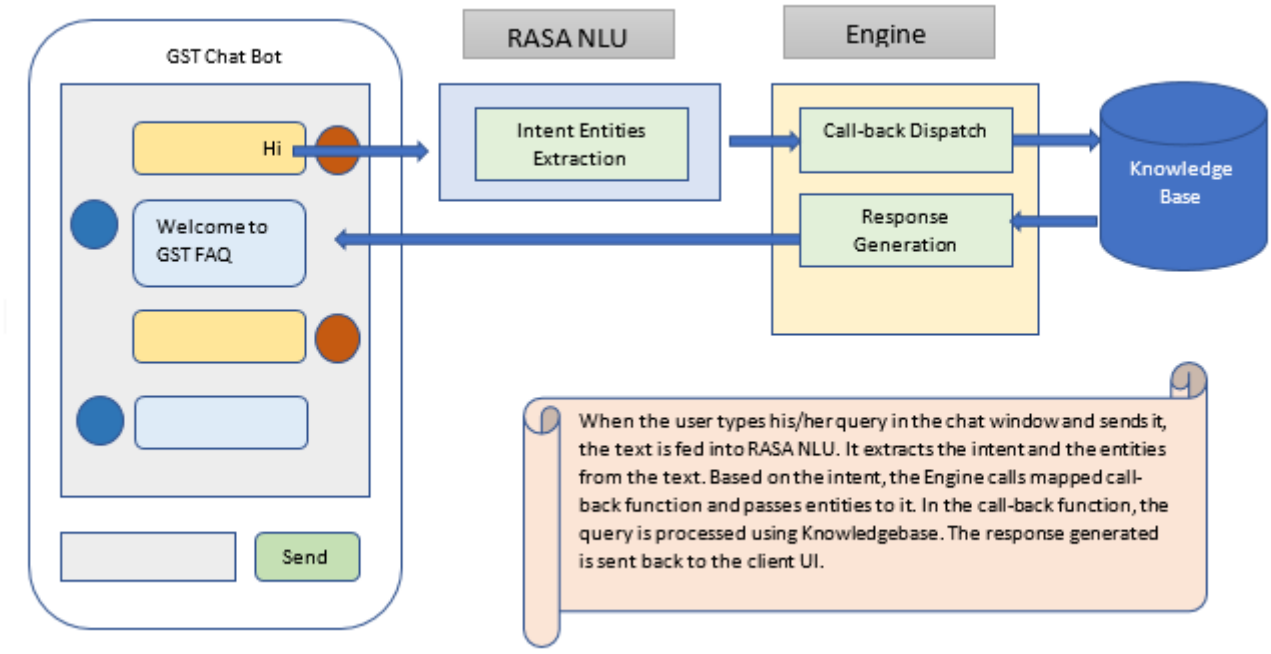
Many chatbot platforms are currently available, from rudimentary rule-based AIML (Artificial Intelligence Markup Language), to highly sophisticated AI bots. Some popular chatbot platforms are API.ai, Wit.ai, Facebook APIs, Microsoft LUIS, IBM Watson, etc.

RASA-NLU builds a local NLU (Natural Language Understanding) model for extracting intent and entities from a conversation. It's open source, fully local and above all, free! It is also compatible with wit.ai, LUIS, or api.ai, so you can migrate your chat application data into the RASA-NLU model.

Below is a demonstration on how to install RASA-NLU and build a simple FAQ bot in Python.

Building GST FAQ bot architecture

A chatbot is a client-server application. In the case of RASA-NLU, even the server can be local. The client is nothing but the chabot UI. The interaction and architecture can be understood by the following diagram:



Installations

RASA can be installed and configured on a standalone machine. Steps to follow:

1. Ready installation can be done by: `pip install rasa_nlu`
2. You can view the latest documentation here

RASA-NLU is made up of a few components, each doing some specific work (intent detection, entity extraction, etc.). Each component may have some specific dependencies and installations. Options like MITIE (NLP + ML), Spacy and Sklearn are available to choose from. We will be using Spacy-Sklearn here.

Client UI can be a web page (using frameworks like Flask in Python) or a mobile app. Flask is simple to code and runs locally. Use *pip install flask* and follow this tutorial to get a basic understanding of the framework.

Server

A RASA-NLU platform needs to be trained before we start using it. We need to supply it with a few sentences and mention which are the intents and entities in it. **Intents are the actions/categories of the sentences and entities are the necessary variables needed to fulfil the actions**.

For example, “I wish to book a flight from Mumbai to Pune on 27 March” has “flight-booking” as the intent and “Mumbai”,” Pune” and “27 March” as the entities. Similarly, many training examples can be used so that the RASA-NLU model is trained on different ways of extracting intents/entities from our domain conversations. This training data is stored in a json, a sample of which can be seen here.

It contains many entries. One of the sample entries is shown below:

```
{
  "text": "show me a mexican place in the centre",
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 31,
      "end": 37,
      "value": "centre",
      "entity": "location"
    },
    {
      "start": 10,
      "end": 17,
```

```
            "value": "mexican",
            "entity": "cuisine"
          }
        ]
}
```

Following is the explanation of some of the fields mentioned in the above code:

- **text:** the input sentence.
- **intent:** action or category, in this instance “restaurant_search”. This is typically the call-back function name.
- **entities:** array of entities. Here, there are two. One is of type ‘location’ with value as ‘centre’, whereas the other is of type ‘cuisine’ with value ‘mexican’. ‘start’ and ‘end’ specify beginning and ending indices of the word in the sentence.

You can use the below online tool as well, to generate this json file:

https://rasahq.github.io/rasa-nlu-trainer/

Steps to build server side of the GST chat bot application:

1. Create a new directory and navigate to it.
2. Create the following things in it:
    1. *data directory*
    2. *data/demo_gst.json*: This has GST FAQ training examples as shown above
    3. *json*: This has settings for RASA-NLU as shown below:

```
{
  "pipeline": "spacy_sklearn",
  "path" : "./",
  "data" : "./data/gstfaq-data.json"
}
```

3. Train the model in python
    -

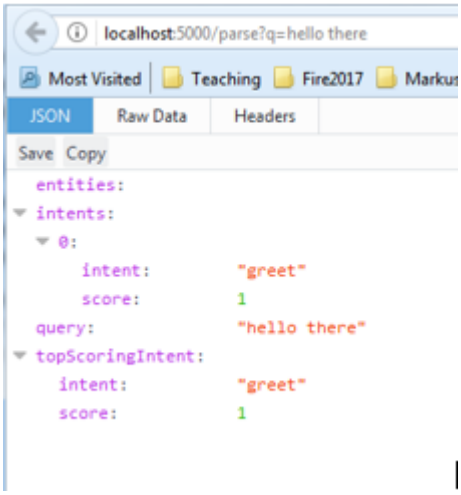
```
-m rasa_nlu.train -c config.json
```

4. This will create a model_YYYYMMDD-HHMMSS folder
5. Run:

```
-m rasa_nlu.server -c config.json --server_model_dirs=./model_YYYYMMDD-HHMMSS
```

- This starts the RASA-NLU server and let's us know the port, for instance

6. Once the server starts running you can GET/POST via curl to post or use it as the HTTP server
7. For HTTP server, add "-e luis" while running the server
8. Then, in the browser, type: *http://localhost:5000/parse?q=hello%20there*
9. The output is shown below:



We can now look at the remaining components of our GST FAQ bot.

Client

Our chatbot client UI has been built using Flask framework. It uses two html templates to render the UI (i.e. the chat window). The outer UI is built with **base.html** as shown below:

```
<!DOCTYPE HTML>
<html lang="en">
<head>
…
</head>
    <body>
         <h1 align="center">GST FAQ Chat</h1>
        <div class="container">
            {% block content %}{% endblock %}
        </div>
</body>
    <footer>
```

```
        {% block other_footers %}{% endblock %}
    </footer>
</html>
```

The **content** and **other_footers** blocks are defined in **home.html** as shown below:

```
{% block content %}
<div class="row">
    <div class="col-sm-6">
        <div class="row">
            <div class="chat_window">
                <ul class="messages"></ul>
                <div class="bottom_wrapper clearfix">
                    <div class="message_input_wrapper">
                        <input id="msg_input" class="message_input" placeholder="Say Hi to begin cha
t..." />
                    </div>
                    <div class="send_message">
                         <div class="text">send</div>
                    </div>
                </div>
            </div>
            <div class="message_template">
                <li class="message">
                    <div class="avatar"></div>
                    <div class="text_wrapper">
                        <div class="text"></div>
                    </div>
                </li>
            </div>
        </div>
    </div>
</div>
{% endblock %}



{% block other_footers %}
```

```
<script src="{{ url_for('static', filename="js/bind.js") }}"></script>
{% endblock %}
```

Engine

This is the heart of the chatbot. Based on the intent received from RASA-NLU, it dispatches the entities to the mapped call-back functions. The function in turn, depending on the entities, calls Knowledgebase to get the response. Once the response is received, it is sent back to the UI.

Knowledgebase can be as simple as a dictionary of questions and answers, or as sophisticated as one can imagine/require (like databases, internet sources, etc.). This article, being minimalistic for demonstration purposes, fetches pre-coded responses from the dictionary.

Let's take a look at the sample dictionary:

```
intent_response_dict = {
    "intro": ["This is a GST FAQ bot. One stop-shop to all your GST related queries"],
    "greet":["hey","hello","hi"],
    "goodbye":["bye","It was nice talking to you","see you","ttyl"],
    "affirm":["cool","I know you would like it"],
    "faq_link":['You can check all the events here <a href="https://www.cbec.gov.in/resources//htdoc
s-cbec/deptt_offcr/faq-on-gst.pdf</a>']
}
```

The engine's use of RASA-NLU for intent-entities extraction and dispatching call-backs can be seen below:

```
@app.route('/chat',methods=["POST"])
def chat():
    try:
        user_message = request.form["text"]
        response = requests.get("http://localhost:5000/parse",params={"q":user_message})
        response = response.json()
        response = response["topScoringIntent"]
```

```
        intent = response.get("intent")
        entities = response.get("entities")
        if intent == "gst-info":
            response_text = gst_info(entities
        elif intent == "gst-query":
            response_text = gst_query(entities)
        else:
            response_text = get_random_response(intent)
        return jsonify({"status":"success","response":response_text})
    except Exception as e:
        print(e)
        return jsonify({"status":"success","response":"Sorry I am not trained to do that yet..."})
```

User text is sent to the RASA-NLU server using http://localhost:5000/parse. Its response contains the intent and the entities. Depending on the intent, functions like ***gst-info*** and ***gst-query***are called. Their responses are then sent back to the UI.

The source code for this app is available on github.

Our chatbot in action

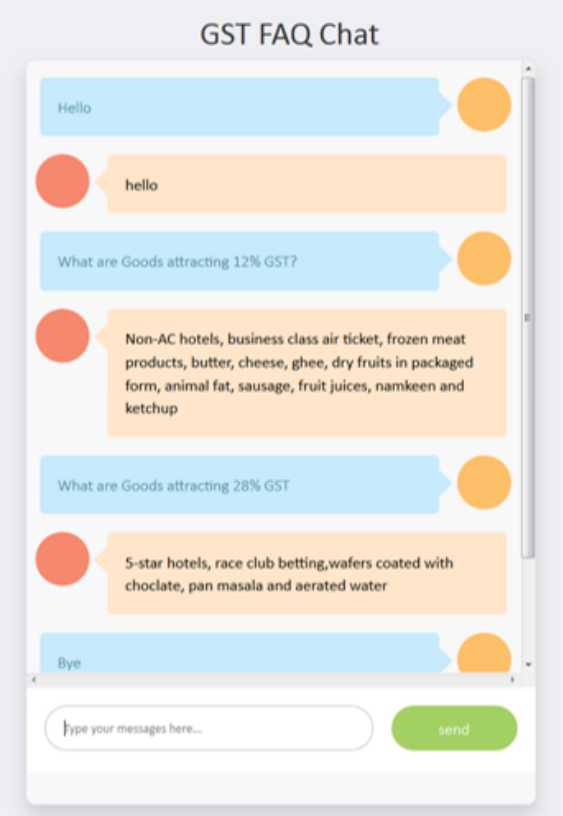
Steps to operate:

1. Start the RASA-NLU server by executing “run_server.bat” script. It loads the custom trained model and starts listening to port 5000

2. Execute the Flash app, by running the localhost at the given port, say 8000.

3. Start typing commands in the bottom chat window and click “Send”.

4. Typed messages and their responses appear in the window above, as seen in the adjoining picture.



You can view the video demonstration here.

End Notes

This tutorial presents just a small example, demonstrating the potential to develop something full-fledged and practically useful. Our GST Q&A bot can be enhanced on various fronts, such as expansion of knowledgebase (i.e. number of questions and answers), better training to find more intents and entities, Natural Language Generation of the responses to have a human language feel, etc.

GST FAQ Bot is just one example of building an intuitive frontend using government information. With the availability of more APIs and open public data, we can build similar (if not better) bots for those databases. Imagine interacting with government departments using a chatty bot!

References

- RASA-NLU setup, installation, https://github.com/RASAHQ/rasa_nlu