upGrad
#LifeKoKaroLift

Post-Graduate Diploma in ML/AI

upGrad

Course : Machine Learning

Lecture On : Chatbot

Instructor : Manish Kumar

Today’s Agenda

- What are chatbots?
- Subparts of Chatbot
- RASA Installation
- RASA Architecture
- Natural Language Understanding
- Dialogue Management system

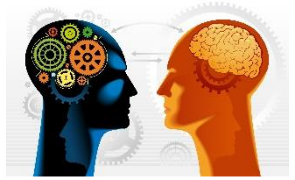
Chatbots: Introduction

upGrad

“ Chatbots are computer programs that maintain a conversation with a user in natural language, understand the user’s intent and send responses based on the organization’s business rules and data “

Building blocks of chatbots

A **front-end interface**, which connects to a messaging applications such as Facebook Messenger or Slack, through which users interact with the chatbot.



Natural language Understanding, responsible for recognizing the user’s intent. This element uses natural language processing and machine learning to parse user messages, collect relevant parameters from words and sentences, and map those to actions to take

Dialogue Management, manages the dialogue by maintaining a representation of the conversational logic and keeping track of context.

Evolution of a chatbot

upGrad

Basic Scripted

- Simplistic chatbot
- Looks for key phrases & give scripted responses

Intent Recognizer

- These systems try to identify parts of speech for each word and how the words relate to each other to extract meaning from a request

Dialogue Manager

- Maintains past conversations
- Can integrate gesture understanding with language understanding

Intelligent Advisor

- Ability to understand what a human is trying to achieve.
- Could advise on better paths.

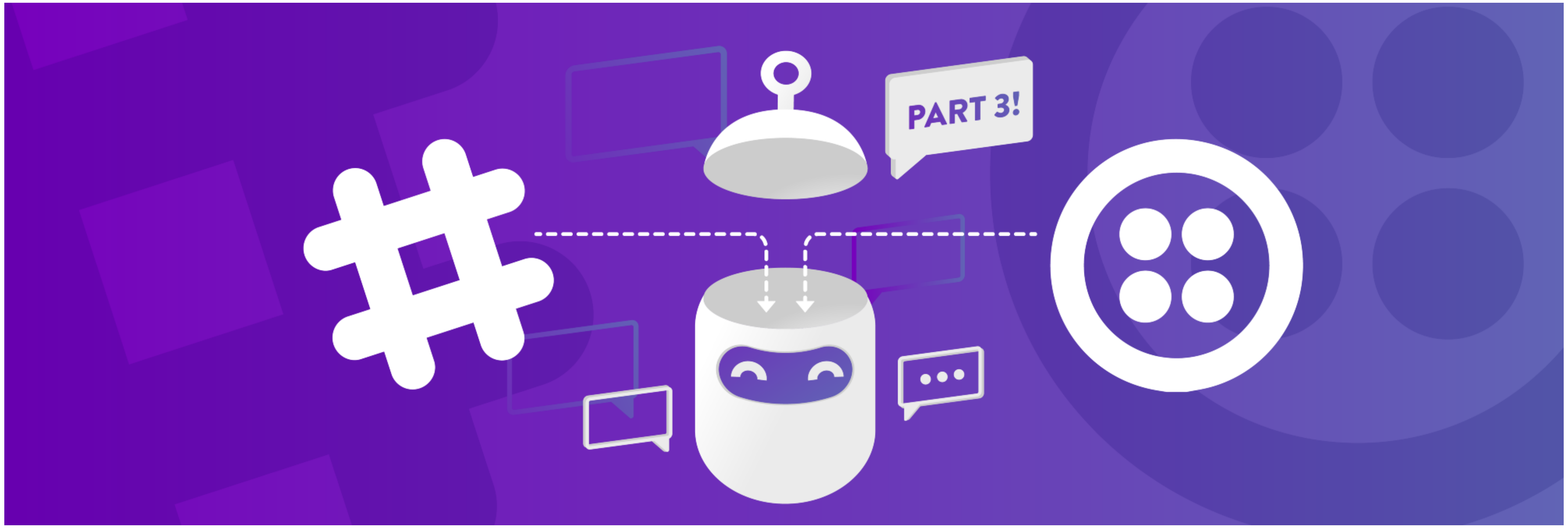
Human Advisor

- Gerneal AI (mythical stage)

Getting Started with Rasa

upGrad

Rasa NLU and **Rasa Core** are two open source libraries for building conversational agents. **Rasa NLU** is the tool used for **intent classification** and **entity extraction**.



Version Details :
rasa==1.5.1
rasa-sdk==1.5.1

https://rasa.com/docs/getting-started/

Rasa – Installation (Requirements)

upGrad

- ✓ Python 3.6.0
- ✓ Rasa NLU
- ✓ Rasa Core
- ✓ Spacy – en models

Python 3.6.5

upGrad

ANACONDA®

Creating an environment

```
conda create --name rasa
```

https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

DEMO !

Rasa & Rasa X

upGrad

Step 1 : Open anaconda prompt with administrator rights

Step 2 : Activate Virtual environment

```
>> activate rasa
```

Step 3 :

```
>>  pip install rasa-x --extra-index-url https://pypi.rasa.com/simple
>>  pip install rasa
```

Step 4 : Install rasa-nlu with spacy using

```
>> pip install rasa[spacy]
```

Step 5 : Download spacy model and link it

```
>> python -m spacy download en_core_web_md
>> python -m spacy link en_core_web_md en
```

RASA - Architecture

upGrad

1. The message is received and passed to an Interpreter, which converts it into a dictionary including the original text, the intent, and any entities that were found. This part is handled by NLU.

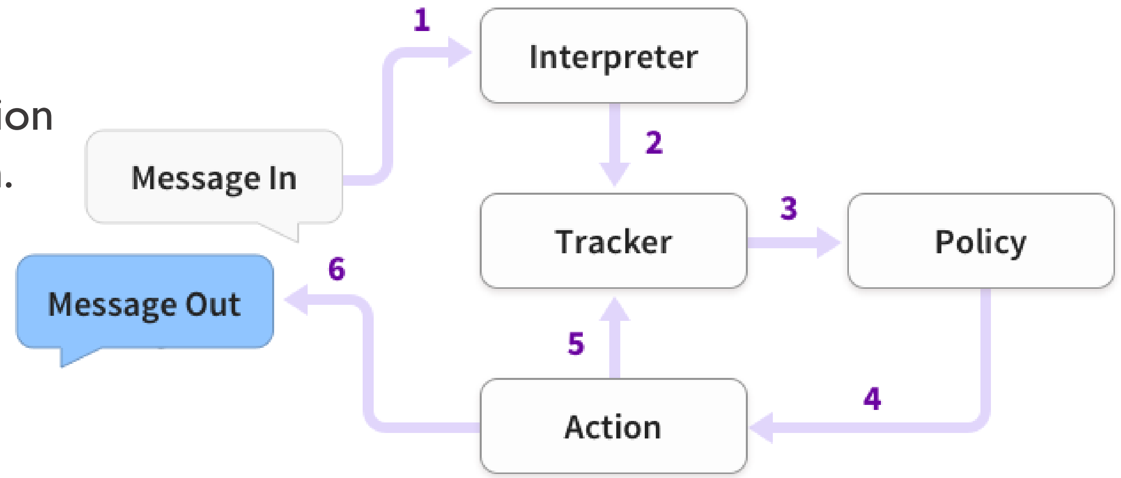
2. The Tracker is the object which keeps track of conversation state. It receives the info that a new message has come in.

3. The policy receives the current state of the tracker.

4. The policy chooses which action to take next.

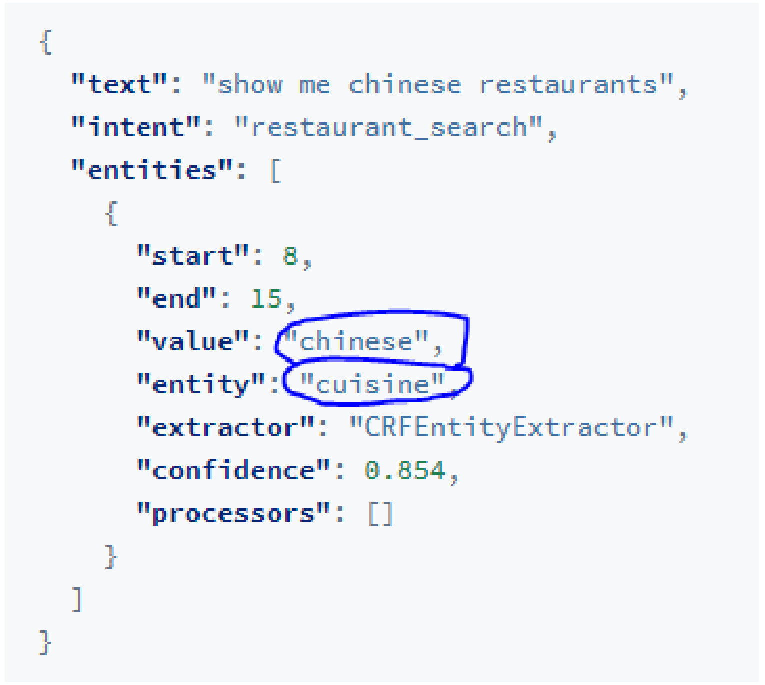
5. The chosen action is logged by the tracker.

6. A response is sent to the user.



Natural Language Understanding (NLU)

The first layer of the conversational system, **NLU**, interprets the free text provided by the user. It basically takes an unstructured text phrase or sentence, understands what the user probably **intends** to say, **extracts entities** from it and converts it into **structured data**.



```
{
  "text": "show me chinese restaurants",
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 8,
      "end": 15,
      "value": "chinese",
      "entity": "cuisine",
      "extractor": "CRFEntityExtractor",
      "confidence": 0.854,
      "processors": []
    }
  ]
}
```

Unstructured user query: 'show me Chinese restaurants'
Intent: restaurant_search; **Entities**: [cuisine=chiniese]

NLU – Training Data Format

upGrad

The training data for Rasa NLU is structured into different parts:

- common examples
- synonyms
- regex features
- lookup tables

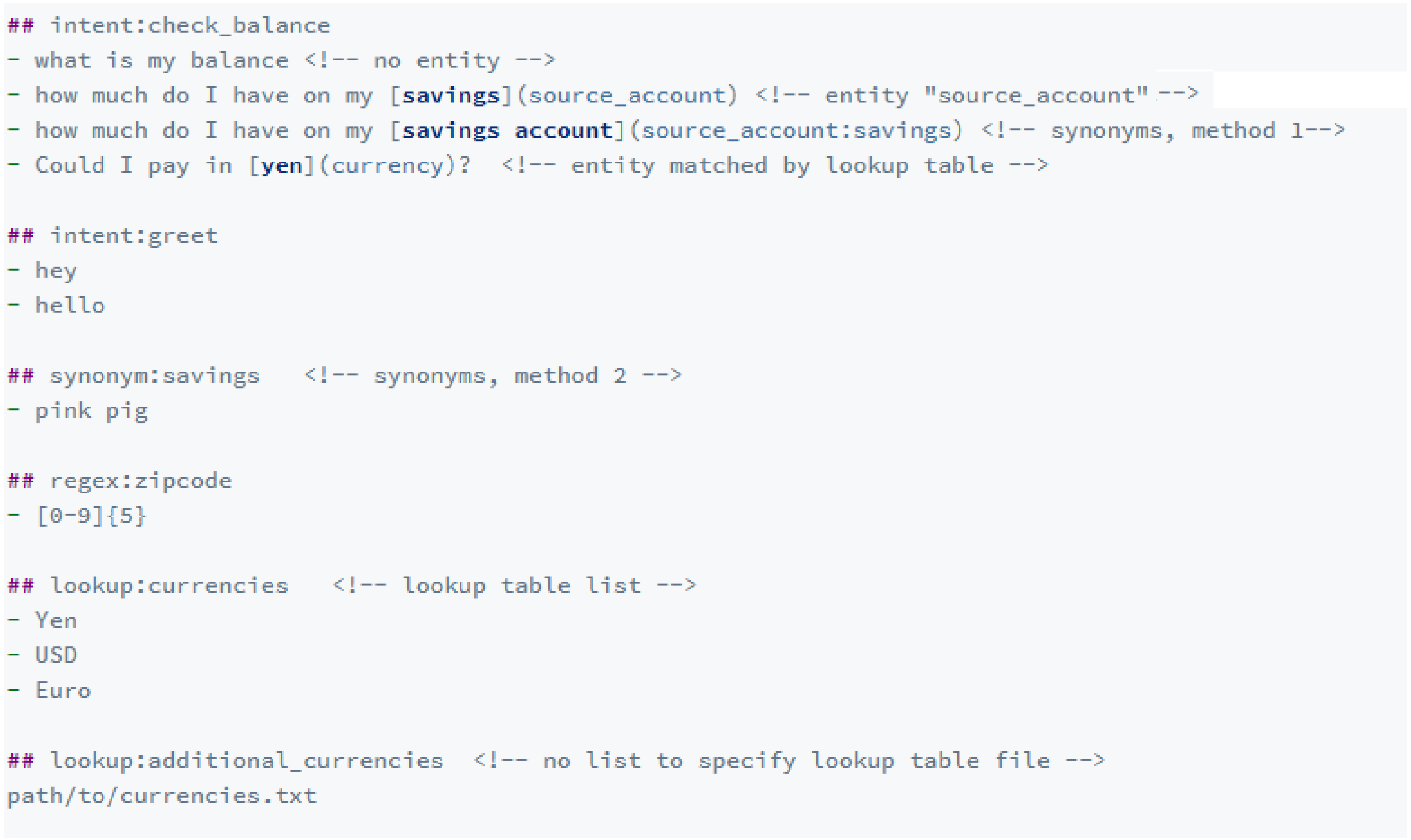
Data Formats

- Markdown Format
- JSON Format

https://rasa.com/docs/rasa/nlu/training-data-format/

Markdown Data Format

upGrad



```
## intent:check_balance
- what is my balance <!-- no entity -->
- how much do I have on my [savings](source_account) <!-- entity "source_account" -->
- how much do I have on my [savings account](source_account:savings) <!-- synonyms, method 1-->
- Could I pay in [yen](currency)?  <!-- entity matched by lookup table -->

## intent:greet
- hey
- hello

## synonym:savings   <!-- synonyms, method 2 -->
- pink pig

## regex:zipcode
- [0-9]{5}

## lookup:currencies   <!-- lookup table list -->
- Yen
- USD
- Euro

## lookup:additional_currencies  <!-- no list to specify lookup table file -->
path/to/currencies.txt
```

Training Data Format

upGrad

We can create regular expression features for both intent and entity extraction. For e.g.the following piece of code specifies regexes for extracting zip code and the entity “greet”. Which are used as feature functions of the CRF

```
## regex:zipcode
- [0-9]{5}

## regex:greet
- hey[^\\s]*
```

```
## synonym:New York City
- NYC
- nyc
- the big apple
```

Rasa provides support to map synonyms or misspellings to an entity. For example: Chinese <-> Chines.

Lookup tables in the form of external files or lists of elements may also be specified in the training data. The externally supplied lookup tables must be in a newline-separated format.

```
## lookup:plates
data/test/lookup_tables/plates.txt

## lookup:plates
- beans
- rice
- tacos
- cheese
```

Training the NLU Model

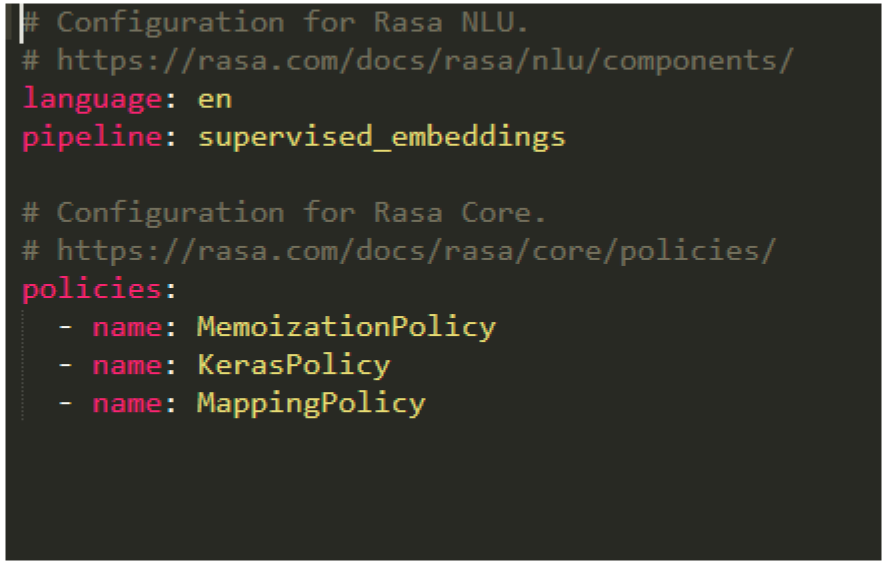
upGrad

Rasa NLU Training Pipelines

Rasa NLU comes with builtin pipelines for intent classification and entity extraction. The two most popular ones are:

- pretrained_embeddings_convert
- supervised_embeddings

./config.yml



```
# Configuration for Rasa NLU.
# https://rasa.com/docs/rasa/nlu/components/
language: en
pipeline: supervised_embeddings

# Configuration for Rasa Core.
# https://rasa.com/docs/rasa/core/policies/
policies:
  - name: MemoizationPolicy
  - name: KerasPolicy
  - name: MappingPolicy
```

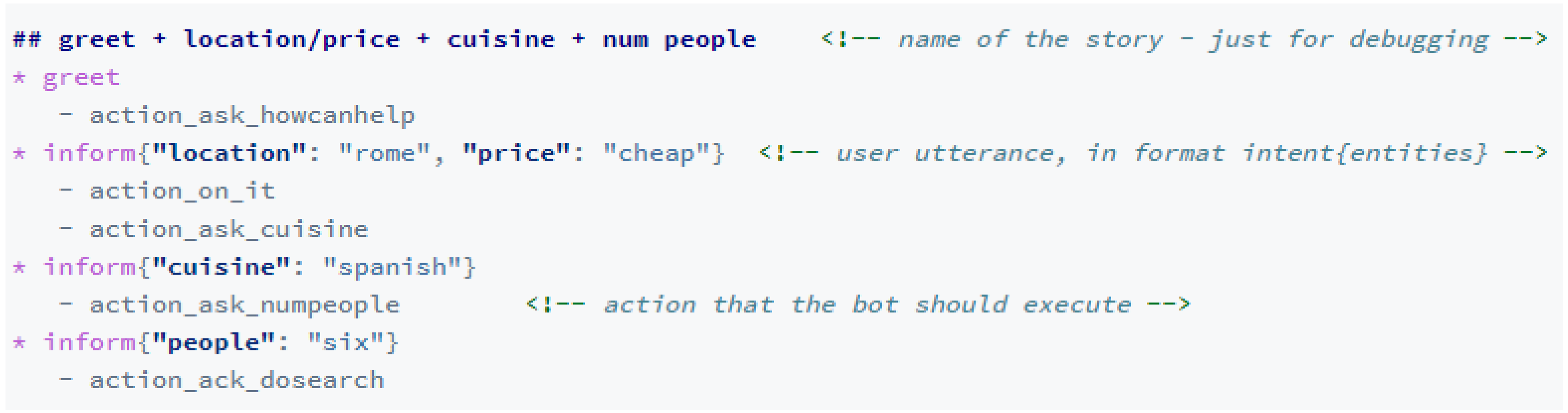
https://rasa.com/docs/rasa/nlu/choosing-a-pipeline/

Dialogue-Flow Management

upGrad

RASA Core

A dialogue management model will predict what response or action the chatbot should take based on the stage of the conversation. These actions/responses could be printing results on the screen, fetching data from a database, sending mail to the user, or simply saying "thank you!".



```
## greet + location/price + cuisine + num people    <!-- name of the story - just for debugging -->
* greet
   - action_ask_howcanhelp
* inform{"location": "rome", "price": "cheap"}  <!-- user utterance, in format intent{entities} -->
   - action_on_it
   - action_ask_cuisine
* inform{"cuisine": "spanish"}
   - action_ask_numpeople        <!-- action that the bot should execute -->
* inform{"people": "six"}
   - action_ack_dosearch
```

./stories.md

Dialogue-Flow Management

RASA Core

./domain.yml

The Domain defines the universe in which your assistant operates. It specifies the intents, entities, slots, and actions your bot should know about. Optionally, it can also include templates for the things your bot can say.

```
intents:
  - greet
  - goodbye
  - affirm
  - deny
  - mood_great
  - mood_unhappy
  - bot_challenge

actions:
- utter_greet
- utter_cheer_up
- utter_did_that_help
- utter_happy
- utter_goodbye
- utter_iamabot

templates:
  utter_greet:
  - text: "Hey! How are you?"

  utter_cheer_up:
  - text: "Here is something to cheer you up:"
    image: "https://i.imgur.com/nGF1K8f.jpg"

  utter_did_that_help:
  - text: "Did that help you?"

  utter_happy:
  - text: "Great, carry on!"

  utter_goodbye:
  - text: "Bye"

  utter_iamabot:
  - text: "I am a bot, powered by Rasa."
```

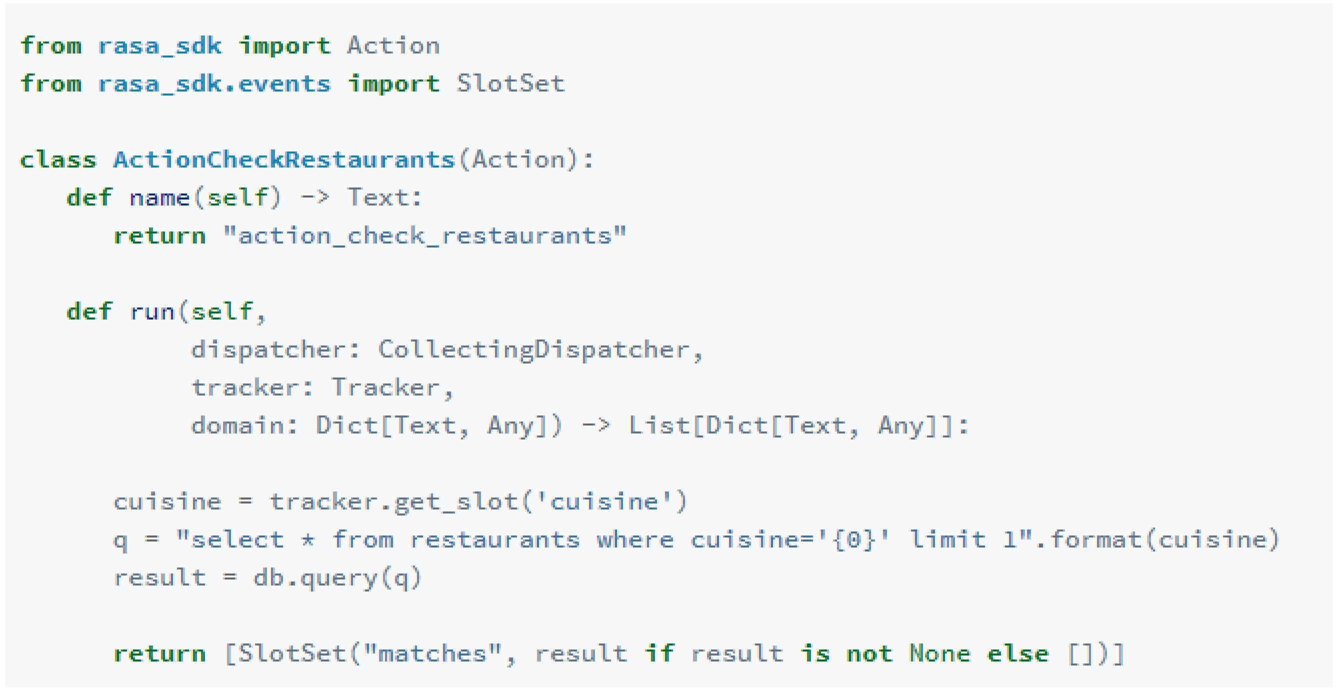
Dialogue-Flow Management

RASA Core

./actions.py

Actions are the things your bot runs in response to user input. There are four kinds of actions in Rasa:

- Utterance actions: start with utter_ and send a specific message to the user
- Retrieval actions: start with respond_ and send a message selected by a retrieval model
- Custom actions: run arbitrary code and send any number of messages (or none).
- Default actions: e.g. action_listen, action_restart, action_default_fallback



```python
from rasa_sdk import Action
from rasa_sdk.events import SlotSet

class ActionCheckRestaurants(Action):
   def name(self) -> Text:
      return "action_check_restaurants"

   def run(self,
           dispatcher: CollectingDispatcher,
           tracker: Tracker,
           domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

      cuisine = tracker.get_slot('cuisine')
      q = "select * from restaurants where cuisine='{0}' limit 1".format(cuisine)
      result = db.query(q)

      return [SlotSet("matches", result if result is not None else [])]
```

Problem Statement

upGrad

The main purpose of the bot is to help users discover restaurants quickly and efficiently and to provide a good restaurant discovery experience.

- Foodie works **only in Tier-1 and Tier-2 cities**. You can use the current HRA classification of the cities from here. Under the section 'current classification' on this page, the table categorizes cities as X, Y and Z. Consider 'X ' cities as tier-1 and 'Y' as tier-2.

- The bot should be able to **identify common synonyms** of city names, such as Bangalore/Bengaluru, Mumbai/Bombay etc.

Your chatbot should provide results for tier-1 and tier-2 cities only, while for tier-3 cities, it should reply back with something like "We do not operate in that area yet".

Problem Statement

upGrad

1.NLU training:You can use rasa-nlu-trainer to create more training examples for entities and intents. Try using regex features and synonyms for extracting entities.

2.Build actions for the bot. Read through the Zomato API documentation to extract the features such as the average price for two people and restaurant's user rating. You also need to build an 'action' for sending emails from Python.

3.Creating more stories: Use train_online.py file to create more stories. Refer to the sample conversational stories provided above. Your bot will be evaluated on something similar to the test stories shared.

4.Deployment (Optional): Deploy the model on Slack. You can create a new workspace or deploy it on an existing workspace (if you already use Slack).

Thank You !