Introduction to Chatbot with Rasa
Yogesh Kulkarni

Chatbot with Custom Actions

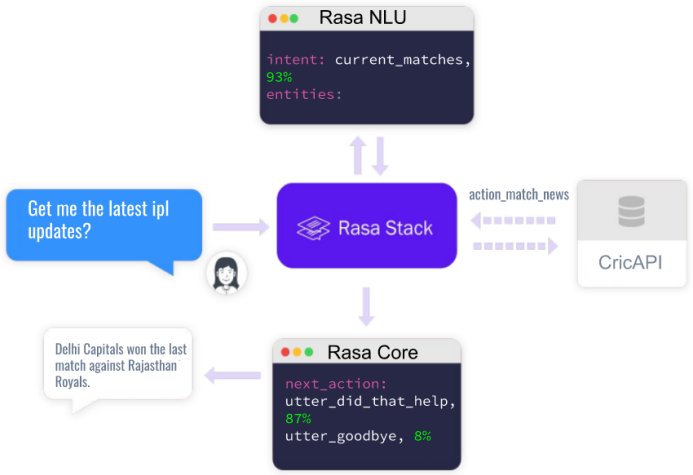
IPL Chatbot

(Ref:"Learn how to Build and Deploy a Chatbot in Minutes using Rasa (IPL Case Study!)" - Mohd Sanad Zaki Rizvi, Analytics Vidhya)

Objective

To build a chatbot capable of fetching latest info about the ongoing IPL (Indian Premier League) matches from cricapi.com site.

Recap: Architecture of Rasa chatbot



Setup

Installations

Official Documentation: https://rasa.com/docs/core/installation/

- Python
- Rasa Stack
- Spacy Language Model

Python Environment

- By installing conda, you get base or the root environment, which is the default.
- Practical tip: DO NOT install any packages in the root. ALWAYS create and env and install inside the new env.
- Env is needed especially for fragile packages like Python (its treated as a package) and rasa.
- So, `conda create -n rasa python=3.6`

(Note: Env management is again a sour point. It creates complete copy (deeeep) of python 3.6 and all other packages inside "envs" folder. Goes to 1.5 GB!! Can someone optimize it?)

IPL bot code (for reference)

- Clone repo from https://github.com/mohdsanadzakirizvi/iplbot.git
- "Complete Version" gives latest running application (Some modifications are needed, mentioned below)
- But, WE WILL BE CODING IT FROM SCRATCH
- Run following commands:

```
mkdir code; cd code;mkdir data
activate rasa
pip install -r iplbot_requirements.txt
```

Great for getting started: $spaCy + sklearn$

The spacy_sklearn pipeline combines a few different libraries and is a popular option.

```
python -m spacy download en_core_web_md # This will show
    that linking is done, but it may not have happened
    actually. So, do next step anyway

python -m spacy link en_core_web_md en --force
```

Others

- Install nb_conda_kernels to let Jupyter Notebook see the new env
- Create account on cricapi.com (free, can login using Google account). Note down API Key and set it as CRICINFOAPI env variable

Getting started . . .

Importing the Libraries

```
import rasa_nlu
import rasa_core
import spacy
```

Preparing the NLU Training Data

Training data for extracting the user intent. As you can see, the format of training data for 'intent' is quite simple in Rasa. You just have to:

- Start the line with "## intent:intent_name"
- Supply all the examples in the following lines

NLU training file

data/nlu_data.md

```
## intent:greet
- Hi
- Hey
- Hi bot
- Hey bot
- Hello
:
## intent:current_matches
- what are the current matches
- can you list the matches in ipl 2019
- which cricket match is happening right now
- which ipl match is next
- which teams are playing next in ipl
- which team will play next in ipl
:
```

Preparing the NLU Training Data

- You can include as many examples as you want for each intent.
- In fact, make sure to include slangs and short forms that you use while texting.
- The idea is to make the chatbot understand the way we type text.
- Feel free to refer to the complete version where I have given plenty of examples for each intent type.

Defining the NLU Model Configuration

This file lets us create a text processing pipeline in Rasa. Luckily for us, Rasa comes with two default settings based on the amount of training data we have:

- "spacy_sklearn" pipeline if you have less than 1000 training examples
- "tensorflow_embedding" if you have a large amount of data

config/nlu_config.yml

```
language: "en"
pipeline: spacy_sklearn
```

Training the NLU Classifier Model

On command line you can run following command:

```
python -m rasa_nlu.train -c nlu_config.yml --data
    data/nlu_data.md -o models --fixed_model_name nlu
    --project current --verbose
```

Training the NLU Classifier Model

Or programmatically you can write code

```
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

# loading the nlu training samples
training_data = load_data("data/nlu_data.md")

# trainer to educate our pipeline
trainer = Trainer(config.load("config/nlu_config.yml"))

# train the model!
interpreter = trainer.train(training_data)

# store it for future use
model_directory = trainer.persist("./models/nlu",
    fixed_model_name="current")
```

Evaluating the NLU model on a random text (first way)

Let's test how good our model is performing by giving it a sample text that it hasn't been trained on for extracting intent.

```
# A helper function for prettier output

def pprint(o):
    print(json.dumps(o, indent=2))

pprint(interpreter.parse("what is happening in the cricket
    world these days?"))
```

Evaluating the NLU model on a random text (first way)


```
{
  "intent": {
    "name": "current_matches",
    "confidence": 0.5124748469776419
  },
  "entities": [],
  "intent_ranking": [
    {
      "name": "current_matches",
      "confidence": 0.5124748469776419
    },
  :
    {
      "name": "thanks",
      "confidence": 0.03056775368880194
    }
  ],
  "text": "what is happening in the cricket world these
     days?"
}
```


Not only does our NLU model perform well on intent extraction, but it also ranks the other intents based on their confidence scores. This is a nifty little feature that can be really useful when the classifier is confused between multiple intents.

Evaluating the NLU model on a random text (2nd way)

Let's test how good our model is performing by giving it a sample text that it hasn't been trained on for extracting intent. You can open an iPython/Python shell and follow the following steps:

```
from rasa_nlu.model import Interpreter
nlu_model = Interpreter.load('./models/nlu/default/current')
nlu_model.parse('what is happening in the cricket world
    these days?')
```

Evaluating the NLU model on a random text (2nd way)


```
{'intent': {'name': 'current_matches', 'confidence':
    0.5124748469776419},
 'entities': [],
 'intent_ranking': [{'name': 'current_matches',
   'confidence': 0.5124748469776419},
  {'name': 'greet', 'confidence': 0.26316772065557126},
  {'name': 'goodbye', 'confidence': 0.09389099823148171},
  {'name': 'affirm', 'confidence': 0.06686974145943789},
  {'name': 'deny', 'confidence': 0.03302893898706549},
  {'name': 'thanks', 'confidence': 0.03056775368880194}],
 'text': 'what is happening in the cricket world these
    days?'}
```


Evaluating the NLU model on a test data

Let us now evaluate it on a test data set. However, for our purpose let's evaluate it on the data at hand i.e nlu_data.md

```
from rasa_nlu.evaluate import run_evaluation
run_evaluation("data/nlu_data.md", model_directory)
```

We get an Intent Confusion matrix with the with various evaluation results.


```
{'intent_evaluation': {'predictions': [{'text': 'Bye',
    'intent': 'goodbye',
    'predicted': 'goodbye',
    'confidence': 0.9627578046567274},
   {'text': 'Goodbye',
    'intent': 'goodbye',
    'predicted': 'goodbye',
    'confidence': 0.8796549909963912},
  :
   'precision': 1.0,
   'f1_score': 1.0,
   'accuracy': 1.0}}}
```

Making Interactive Conversations

- One of the most important aspects of a chatbot application is its ability to be interactive.
- Think of the simplest conversation our chatbot can have with a user. What would be the flow of such a conversation?

```
Me: Hi

Iplbot: Hey! How may I help you?

Me: What was the result of the last match?

Iplbot: Here are some IPL quick info: 1.The match between
     Rajasthan Royals and Delhi Capitals was recently held
     and Delhi Capitals won. 2.The next match is Warriors vs
     Titans on 22 April 2019

Iplbot: Did that help you?

Me: yes, thank you!

Iplbot: Glad that I could help!
```

Making Interactive Conversations

Let's see how we can teach a simple conversation like that to Rasa:

```
## news path 1
* greet
  - utter_greet
* current_matches
  - action_match_news
  - utter_did_that_help
* affirm or thanks
  - utter_gratitude
* goodbye
  - utter_goodbye
```

The general format is:

```
## news path 1           <--- story name for debugging
     purposes
* greet                  <--- intent detected from the user
  - utter_greet          <--- what action the bot should take
* current_matches        <--- the following intent in the
     conversation
```

This is called a user story path. I have provided a few stories in the data/stories.md file for your reference. This is the training data for Rasa Core.

Making Interactive Conversations

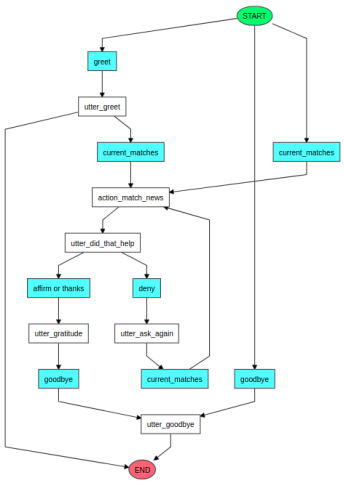
- We will teach chatbot to make responses by training a dialogue management model using Rasa Core.
- For dialog training, Rasa has 4 main components
  - Domain(config/domain.yml)
  - Stories (data/stories.md)
  - Policies (config/policy.yml)
  - Custom Actions (actions.py)

Writing Stories

The way it works is:

- Give some examples of sample story paths that the user is expected to follow
- Rasa Core combines them randomly to create more complex user paths
- It then builds a probabilistic model out of that. This model is used to predict the next action Rasa should take

Writing Stories



Writing Stories

The illustration might look complicated, but it's simply listing out various possible user stories that I have taught Rasa. Here are a few things to note from the above graph:

- Except for the START and END boxes, all the colored boxes indicate user intent
- All the white boxes are actions that the chatbot performs
- Arrows indicate the flow of the conversation
- action_match_news is where we hit the CricAPI to get IPL information

Writing Stories

data/stories.md

```
## news path 1
* greet
  - utter_greet
* current_matches
  - action_match_news
  - utter_did_that_help
* affirm or thanks
  - utter_gratitude
* goodbye
  - utter_goodbye
:
```

Now, generate a similar graph for your stories using the following command:

```
python -m rasa_core.visualize -d domain.yml -s
     data/stories.md -o graph.html
```

This is very helpful when debugging the conversational flow of the chatbot.

Defining the Domain

The domain is the world of your chatbot. It contains everything the chatbot should know, including:

- All the actions it is capable of doing
- The intents it should understand
- The template of all the utterances it should tell the user, and much more

Defining the Domain

config/domain.yml

```
actions:
- utter_greet
- utter_did_that_help
- utter_goodbye
- action_match_news
- utter_default
- utter_gratitude
- utter_ask_again

intents:
- goodbye
- greet
- thanks
- current_matches
- affirm
- deny

:
```

Defining the Domain

config/domain.yml

```
:

templates:
  utter_greet:
  - text: "Hey! What can I do for you?"
  utter_did_that_help:
  - text: "Did that help you?"
  - text: "I hope that solved your query"
  utter_goodbye:
  - text: "Bye"
  utter_default:
  - text: "I am sorry, I didn't get that. Could you please
     repeat your query?"
  - text: "I am not sure what you are aiming for."
  utter_gratitude:
  - text: "Glad that I could be of help to you!\nBye"
  utter_ask_again:
  - text: "Okay! Let's start again, please tell me what do
     you need?"
  - text: "No issues! Let's try this again.\n Please repeat
     your query?"
```

Setting Policies

- Rasa Core generates the training data for the conversational part using the stories we provide.
- It also lets you define a set of policies to use when deciding the next action of the chatbot.
- These policies are defined in the policies.yml file.
- Else these params can be passed, while training, to respective Policy constructors

config/policies.yml

```
policies:
  - name: KerasPolicy
    epochs: 100
    max_history: 5
  - name: FallbackPolicy
    fallback_action_name: 'action_default_fallback'
  - name: MemoizationPolicy
    max_history: 5
```

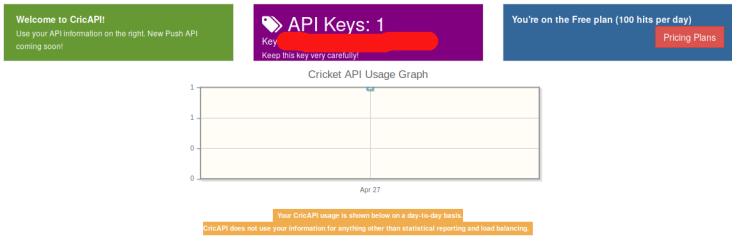
Setting Policies

- KerasPolicy uses a neural network implemented in Keras to select the next action. The default architecture is based on an LSTM (Long Short Term Memory) model
- MemoizationPolicy memorizes the conversations in your training data. It predicts the next action with confidence 1.0 if this exact conversation exists in the training data, otherwise, it predicts 'None' with confidence 0.0
- FallbackPolicy invokes a fallback action if the intent recognition has confidence below nlu_threshold or if none of the dialogue policies predict action with confidence higher than core_threshold
- One important hyperparameter for Rasa Core policies is the max_history. This controls how much dialogue history the model looks at to decide which action to take next

Custom Actions

- Using CricAPI for fetching IPL related news. It is free for 100 requests per day, which (I hope) is more than enough to satiate that cricket crazy passion you have.
- You need to first signup on the website to get access to their API: https://www.cricapi.com/
- You should be able to see your API Key once you are logged in:

Custom Actions



Modifications to original code:

- Instead of showing API key here it has been stored in ENV variable and fetched here
- Key "toss_winner_team" was subsituted into depreceated key

Custom Actions

actions.py

```
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
from datetime import datetime

import logging
import requests
import json
import os
from rasa_core_sdk import Action

logger = logging.getLogger(__name__)

API_URL = "https://cricapi.com/api/"
API_KEY = os.environ.get('CRICINFOAPI')
```

Custom Actions

actions.py

```
class ApiAction(Action):
    def name(self):
        return "action_match_news"

    def run(self, dispatcher, tracker, domain):
        print(API_URL + "matches" + "?apikey=" + API_KEY)
        res = requests.get(API_URL + "matches" + "?apikey="
    + API_KEY) #, verify=False
        if res.status_code == 200:
            data = res.json()["matches"]
            recent_match = data[0]
            upcoming_match = data[1]
```

```
            upcoming_match["date"] =
    datetime.strptime(upcoming_match["date"],
    "%Y-%m-%dT%H:%M:%S.%fZ")
            next_date = upcoming_match["date"].strftime("%d
    %B %Y")
```

Custom Actions

actions.py

```
class ApiAction(Action):

    def run(self, dispatcher, tracker, domain):
      :

            out_message = "Here some IPL quick info: 1.The
    match between {} and {} was recently held and {} won the
    toss.".format(recent_match["team-1"],
    recent_match["team-2"], recent_match["toss_winner_team"])

            dispatcher.utter_message(out_message)

            out_message = "2.The next match is {} vs {} on
    {}".format(upcoming_match["team-1"],
    upcoming_match["team-2"], next_date)

            dispatcher.utter_message(out_message)

            return []
```

Custom Actions

- Need endpoints yml to execute the actions server.
- Note: If you have external API call, like REST, need to have "webhook" word at the end, else nothing.
- My own query on this topic: https://forum.rasa.com/t/rasa-core-sdk-not-working/9228

endpoints.yml

```
action_endpoint:
  url: http://localhost:5055/webhook

core_endpoint:
  url: http://localhost:5005
```

Custom Actions

In a separate shell (cmd for Windows):

- `activate rasa`
- Come to directory where actions.py is and then run
- `python -m rasa_core_sdk.endpoint --actions actions`

This way, custom action server starts . . .

Visualizing the Training Data

```
from IPython.display import Image, display
from rasa_core.agent import Agent
%matplotlib inline

agent = Agent('config/domain.yml')
agent.visualize("data/stories.md", "story_graph.png",
    max_history=2)
i = Image(filename="story_graph.png")
display(i)
```

Training a Dialogue Model

- You can train the model using the following command:
- `python -m rasa_core.train -d domain.yml -s data/stories.md -o models/current/dialogue -c policies.yml`

Training a Dialog Model

Or do it programmatically as:

```
from rasa_core.policies import FallbackPolicy, KerasPolicy,
    MemoizationPolicy
from rasa_core.agent import Agent

# this will catch predictions the model isn't very certain
    about
# there is a threshold for the NLU predictions as well as
    the action predictions
fallback =
    FallbackPolicy(fallback_action_name="utter_unclear",
                           core_threshold=0.2,
                           nlu_threshold=0.1)
```

Training a Dialog Model

```
agent = Agent('domain.yml', policies =
    [MemoizationPolicy(max_history=2), KerasPolicy(epochs =
    200,validation_split = 0.2)])
# loading our neatly defined training dialogues
training_data = agent.load_data('data/stories.md')

agent.train(training_data)
# FOLOWING WAY has been deprecated, move these params to
    KerasPolicy
# agent.train(
#     training_data,
#     validation_split=0.2,
#     epochs=200
# )

agent.persist('models/current/dialogue')
```

Running: Talk to your Bot

Talk to your Bot

So we have the chatbot ready. It's time to chat.
One way is to run following command in shell (windows cmd)

- activate the environment,
- Come to directory where actions.py is and then run
- `python -m rasa_core.run -d models/current/dialogue -u models/current/nlu --endpoints endpoints.yml`

Talk to your Bot

Or do it programmatically as:
Both approaches expect rasa core sdk server running in a separate window, else `python -m rasa_core_sdk.endpoint --actions actions`

```
import IPython
from IPython.display import clear_output
from rasa_core.agent import Agent
from rasa_core.interpreter import NaturalLanguageInterpreter
from rasa_core.utils import EndpointConfig

def load_assistant():
    messages = ["Hi! Chat here. Type 'stop' to end"]
    interpreter =
     NaturalLanguageInterpreter.create(model_directory)
    endpoint =
     EndpointConfig('http://localhost:5055/webhook')
    agent = Agent.load('models/current/dialogue',
     interpreter=interpreter, action_endpoint = endpoint)
    print("Your bot is ready to talk! Type your messages
     here or send 'stop'")
    while True:
        a = input()
        if a == 'stop':
            break
        responses = agent.handle_text(a)
```

```
        for response in responses:
            print(response["text"])
```

Talk to your Bot

```
load_assistant()

Your bot is ready to talk! Type your messages here or send
    'stop'
hi
Hey! What can I do for you?
What are recent matches?
Here some IPL quick info: 1.The match between Mumbai Indians
    and Sunrisers Hyderabad was recently held and Mumbai
    Indians won the toss.
2.The next match is Kings XI Punjab vs Kolkata Knight Riders
    on 03 May 2019
Did that help you?
stop
```

Conclusion and next

- We utilized the capabilities of Rasa NLU and Rasa Core to create a bot with minimum training data.
- Try to use different pipelines in Rasa Core, explore more Policies, fine-tune those models,
- Check out what other features CricAPI provides, etc.
- Other APIs
- Slot filling
- Different languages (Hindi bot?)

The End

Take aways

3 take home thoughts:

- Conversational AI is a big part of the future
- ML techniques help advance state-of-the-art NLU and conversational AI
- Open source is strategically important for enterprises implementing AI

Steps for building Chatbot

- Decide domain (better if smaller)
- Design conversations (list all possible questions, answers)
- List intents (verbs), entities (nouns), actions (call-backs), response (query results)
- Train AI/ML engine
- Write backend Db code
- Create and update knowledge-base (offline, with new info)
- Test scenarios and improve

Comparison

- Rule Based (AIML)
  - Decision tree, with small samples ok
  - Pre-defined responses, so predictable
- ML Based (Rasa)
  - Large Samples a must
  - More natural responses, but initially unpredictable

Start with AIML, once data is received go with ML based

Conclusions

- RASA Is an Open Sourced Python implementation for NLP Engine / Intent Extraction / Dialogue → in which all of the above run on your machine / On premise → NO CLOUD!
- RASA can be integrated with different front ends like Slack, Facebook Messenger, or your own web app.

What Next?

- Try it out, tutorials . . .
- Subscribe to Rasa Newsletter (Rasa X has arrived!! Rasa NLU and Core got merged into Rasa 1.0 . . .)
- Build your own chatbot . . . (1 day workshop in the pipeline . . .)

References

Many publicly available resources have been refereed for making this presentation. Some of the notable ones are:

- RASA-NLU setup, installation, https://github.com/RASAHQ/rasa_nlu
- Chatbots 101 - Architecture & Terminologies - Bhavani Ravi and the event-bot code
- Building chatbots using Python/Django - Youtube video.
- GST FAQ http://www.cbec.gov.in/resources//htdocs-cbec/deptt_offcr/faq-on-gst.pdf
- "Building a Conversational Chatbot for Slack using Rasa and Python" - Parul Pandey
- "The next generation of AI assistants in enterprise" - Alan Nichol
- Top 8 Healthcare Predictions for 2019 - FROST & SULLIVAN/ Reenita Das
- How Artificial Intelligence is Changing the Healthcare Industry - Sumi menon
- Can Healthcare Chatbots Improve the Patient Experience? - Intakeq
- Pydata Berlin talk by Tom Bocklisch
- Conversational AI: Design & Build a Contextual AI Assistant - Mady Mantha