**RASA Chatbot Implemenatation and Installation Flows:**

* Introductions:

RASA Chatbot demonstrating how to build AI assistants for financial services and banking. This starter pack can be used as a base for your own development or as a reference guide for implementing common banking-industry features with Rasa. It includes pre-built intents, actions, and stories for handling conversation flows.

Open source machine learning tools for developers to build, improve, and deploy text-and voice-based chatbots and assistants.

**Rasa has two main components:**

**Rasa NLU** (Natural Language Understanding): Rasa NLU is an open-source natural language processing tool for intent classification (decides what the user is asking), extraction of the entity from the bot in the form of structured data and helps the chatbot understand what user is saying.

**Rasa Core:** a chatbot framework with machine learning-based dialogue management which takes the structured input from the NLU and predicts the next best action using a probabilistic model like LSTM neural network rather than if/else statement. Underneath the hood, it also uses reinforcement learning to improve the prediction of the next best action.

In other words, the Rasa NLU’s job is to interpret the input provided by the user in the form of structured data and Rasa Core’s job is to decide the next set of actions performed by the chatbot. Rasa Core and Rasa NLU are independent of each other and can be used separately.

1. **RASA Installation:**

1.1> Build python Environment:

1. pip install virtualenv: to install virtualenv
2. virtualenv env\_name: create vitual environment
3. env\_name\Scripts\activate: to activate env
   1. Now install RASA packages and its dependencies:
4. pip install rasa==1.10.25
5. pip install tensorflow==2.1.4
6. pip install spacy==2.3.5
7. python -m spacy download en\_core\_web\_md : Large English language model
8. python -m spacy link en\_core\_web\_md en - Link the model now
9. pip install -U rasa\_nlu

Check if RASA installed or not by pasting the following command in Anaconda command prompt:

python -c "import rasa\_nlu; print(rasa\_nlu.**version**);"

1. **RASA Executions:**

2.1> usage:

rasa [-h] [--version] {init,run,shell,train,interactive,test,visualize,data,x} ...

rasa init -----for creating file

rasa shell ------for cmd chat

rasa train ------ to train rasa model

rasa test-----to tetst rasa model

note: after setup all above things and activating virtual environment

2.2> RASA generated file intro by below command:

**command1:** rasa init

The following files will be created:

**init**.py: An empty file that helps python find your actions.

actions.py: Code for your custom actions

config.yml: Configuration of your NLU and Core models

credentials.yml: Details for connecting to other services

data/nlu.md: Your NLU training data

data/stories.md: Your stories

domain.yml: Your assistant’s domain

endpoints.yml: Details for connecting to endpoint channels

models/.tar.gz: Your initial model. Timestamp is in the format of YYYYMMDD-hhmmss. NLU-only models will have nlu prefix at the front.

2.3> rasa train : to train the model

2.4> rasa shell: to test the model on terminal

Note:

If you have multiple nlu models and would like to test a specific model, use the following command instead.

rasa shell -m models/nlu-20190515-144445.tar.gz

2.5: After training it will create models folder:

after extracting model folder:

it has two file:

nlu file -----> models/default

core file ----> models/dialogue

# 3> Data preparation and format:

3.1> Basic intro on intent and entity:

If you would like to train it using your own custom data, you can prepare it in either markdown or json format. I will be using markdown in this tutorial since it is the easiest.

Open up *data/nlu.md* data and start to modify the content according to your own use case.

## **Intent**

You can specify intent with ## intent:name\_of\_intent followed by a list of questions for the intent (space between each intent):

## intent:goodbye <!--- The label of the intent -->

- Bye <!--- Training examples for intent 'bye'-->

- Bye bot

- bye

## **Entity**

You can specify the entity inside each of the question as follow [value](name of entity):

## intent:ages

- my age is [24](age)

- my age is [25](age)

- [27](age)

## intent:names

- My name is [Tatiana](name) <!--- Square brackets contain the value of entity while the text in parentheses is a a label of the entity -->

- I am [Josh](name)

- my name is [manish](name)

- [manish](name)

Note:

In this case, name is the name of the entity and manish is the value. You need to provide a lot of examples in order to capture the entity. Please be noted that **upper case** and **lower case** affects the accuracy. Manish is not the same as manish. Hence, it is advisable to train all in lower case and parse input data to lower case during evaluation.

3.2> **Formatting intro with examples**:

a> **Domain.yml:**

intents:

- greet

- goodbye

- affirm

- thanks

- deny

- names

- out\_of\_scope

- ages

- emails

- pincodes

entities:

- name

- contact

- email

- age

- pincode

- address

- aadhar

slots:

name:

type: text

email:

type: text

age:

type: text

pincode:

type: text

actions:

- utter\_name

- utter\_thanks

- utter\_greet

- utter\_goodbye

- utter\_out\_of\_scope

- utter\_ask\_email

- utter\_ask\_age

- utter\_ask\_pincode

templates:

utter\_name:

- text: "Hey there! Tell me your name."

utter\_greet:

- text: "Nice to you meet you {name}. How can I help?"

utter\_ask\_email:

- text: "please provide your email"

utter\_ask\_age:

- text: "what is your age?"

utter\_ask\_pincode:

- text: "please provide your pincode"

utter\_goodbye:

- text: "Have a nice day {name}!"

utter\_thanks:

- text: "My pleasure."

utter\_out\_of\_scope:

- text: "Sorry, I can’t deal with that request."

- text: "I'm sorry, I can't handle that request."

- text: "I can't help you with that, I'm sorry."

- text: "Even a Rasa bot is not completely perfect - it seems like I can't handle that request."

b> **NLU\_data.md:**

## intent:goodbye <!--- The label of the intent -->

- Bye <!--- Training examples for intent 'bye'-->

- Bye bot

- bye

- bye for now

- Bye bot

- bye

## intent:greet

- Hi

- Hey

- Hi bot

- Hey bot

- Hello

## intent:thanks

- Thanks

- Thank you

- Thank you so much

- Thanks

- Thanks for that

## intent:affirm

- yes

- yes sure

- absolutely

- for sure

- yes yes yes

## intent:names

- My name is [Tatiana](name) <!--- Square brackets contain the value of entity while the text in parentheses is a a label of the entity -->

- I am [Josh](name)

- i am [manish](name)

- my name is [manish](name)

- [manish](name)

- [jay](name)

## intent:emails

- [maxmeier@firma.de](email)

- [bot-fan@bots.com](email)

- [maxmeier@firma.de](email)

- [hi@rasa.com](email)

- my email is [email@rasa.com](email)

- my email is [markjobs@ibm.com](email)

## intent:ages

- my age is [24](age)

- my age is [25](age)

- my age is [26](age)

- my age is [27](age)

- my age is [28](age)

- my age is [30](age)

- [23](age) years old

- [24](age)

- [45](age)

## intent:pincodes

- [302029](pincode)

- [111111](pincode)

- [123980](pincode)

- my pincode is [302021](pincode)

## intent:out\_of\_scope

- Is Rasa really smart?

- bots are bad

- I dont like bots

- Is Rasa bot smart?

c>**stories.md:**

## story\_greet <!--- The name of the story. It is not mandatory, but useful for debugging. -->

\* greet <!--- User input expressed as intent. In this case it represents users message 'Hello'. -->

- utter\_name <!--- The response of the chatbot expressed as an action. In this case it represents chatbot's response 'Hello, how can I help?' -->

## story\_name

\* names{"name":"Sam"}

- utter\_greet

## story\_thanks

\* thanks

- utter\_thanks

## story\_goodbye

\* goodbye

- utter\_goodbye

**4> Convert data format:**

Markdown is arguably the safest choice for beginner to create the data. However, there can be cased where the training data is automated or came from other source such as LUIS data format, WIT data format, Dialogflow data format and json. Rasa also provides a way for you to convert the data format. Check out the following link to know more about it. Make sure that the virtual environment is activated and run the following command (it converts md to json):

**rasa data convert nlu --data data/nlu.md --out data/nlu.json -f json**

--data is the path to the file or directory containing Rasa NLU data.

--out is the name of the file to save training data in Rasa format.

-f is the output format the training data should be converted into. Accepts either json or md.

Once you have all the required data, move it to the data folder and remove any existing . let’s move on to the next section.

**5> Training and testing:**

5.1> Training model:

To train the model, you can just run the following command:

**rasa train**

It will look for NLU training data files in the data folder and saves a trained model in the model folder. Remember to remove any unnecessary data files from the data folder. The name of the model will be prefixed with nlu- to indicate that this is a nlu-only model. Having said that, you can specify the path using the --data parameter. The full list of parameters can be found here.

5.2> Testing model:

You can test the model by running an interactive shell mode via the following command:

**rasa shell**

note:

If you have multiple nlu models and would like to test a specific model, use the following command instead.

**rasa shell -m models/model\_20190515-144445.tar.gz**