SAARTHI.AI ASSIGNMENT BY Ravi Sharma(B170018CS)

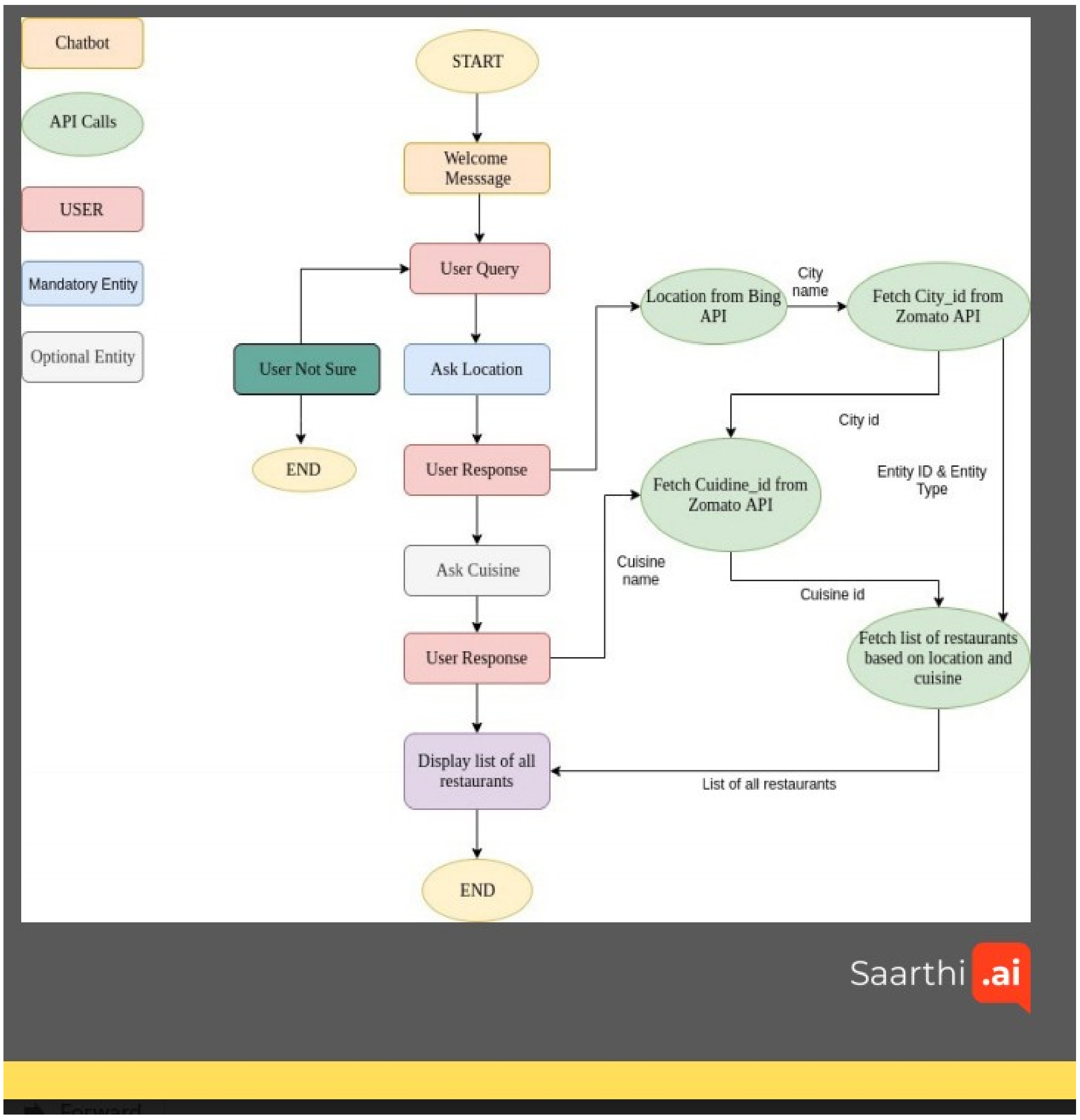
AIM-The Aim of this assignment is to develop a chatbot using rasa framework and zomato api.

Documentation on Rasa Chat Bot using Zomato Api

Abstract:
The popularity of live chat applications has been growing over the past few years. And as the AI trend keeps rising, chatbots become more a must-have rather than a nice to have part of the business. The increasing demand for chats continues to grow so to keep the customer satisfaction rate high; companies must find ways to cope with the rising volumes of inquiries coming every day to all their communication channels.

Introduction:
Rasa is a framework for developing AI powered, industrial grade chatbots. It's incredibly powerful, and is used by developers worldwide to create chatbots and contextual assistants. In this project, we are going to understand some of the most important basic aspects of the Rasa framework and chatbot development.

Architechture



Architechture:

The architechture of the bot is quiet simple.It can be explained in quiet the few steps
1.First the bot asks for the location.
2.Second the bots asks for the cuisine preference.
3.Now since both location and cuisine is available we can fetch the restaurants available using zomato api.

Introduction to Rasa?

Rasa is an open-source machine learning framework for building contextual AI assistants and chatbots.
To make complete AI-based chatbot it has to handle two things:

- *Understanding the user's message which is done through Rasa NLU.*
- *The bot should be able to carry dialogue with context which is done by Rasa Core.*

Skeleton of Rasa

Since hype was to match chatbot with humans. We will take the human analogy to understand the components of the chatbot.

Bot configuration

Firstly we will understand the body parts of the human(mostly brain, Don't worry it's not biology class) which we call "Bot configuration" in the bot world.

Primary thing we humans do is communicate. And language is the primary means of communication. So for the bot as well we need to set language. We will use the English language for the bot. But you can build a multi-lingual bot with RASA.

Now put on your apron and get ready with a scalpel to see what's in the brain . Seems like it's way complex. Chuck it, but the point is whenever we hear something we process the information through millions of neurons to understand the meaning of the sentence with its context, etc. And our brain is smart enough to generate a proper response based on a question. So are we going to build that intelligent bot?. Hold on! We can but not right now. The best way to think and start building a chatbot is like a newborn baby. It learns with experience :) Now let's understand how the brain of chatbot works. It's called NLU(Natural language understanding) unit where it's components do the job. Component includes as follows.

1. *Tokenization: We read and understand the sentence word by word, right? Similarly, tokenizer will break the sentence into words(called word tokenizer).*

2. *Featurizer: We infer meaning by words and when all words are combined in a sentence then we infer the meaning of the sentence with context, right? Similarly, tokenized words are used as features to the post components of the pipeline. These features are has meaning of the word(mathematically) which is called Embeddings. Get to know more about word embedding here. Embedding comes in below two flavors.*

Embedding:

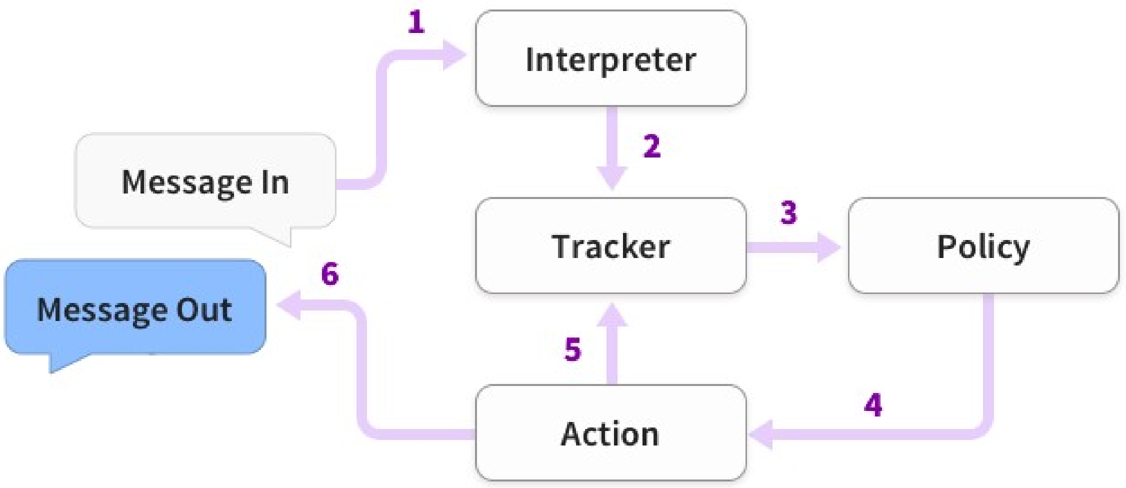
1. *Pre-trained:*
   Here word embeddings are already trained on huge text datasets with various state-of-the-art architecture. Popular word embeddings are XLNet, BERT, Glove, etc. We can use word embedding as it is in our NLP pipeline when we don't have much training data. This technique is called as transfer learning.
2. *From scratch:*
   When pre-trained does not work well because it might have trained on your domain-specific then we can train our own word embedding from scratch. It is recommended when you have sufficient training samples.
3. *Count vectorizer:*
   You can convert a sentence into features using a bag of words. Where you can have unigram, bi-gram, tri-gram.

3. *Entity extraction*
   These are a chunk of information we extract from sentences to complete the action. For example when we say I want to eat ramen. Here the entity is cuisine name.We just need to predefined cuisine names so that the bot can know the text is referring to entity cuisine.

Once the intent is identified and all entity is extracted then we can complete the action by calling the required Zomato API.

4. *Classifier*
   Now you know the meaning(features) of the sentence(words through tokenization). It's time to classify to its appropriate category. For e.g I want to travel by cab should classify to travel_cab and I want to travel by flight travel_flight. All this is done by using Machine learning or Deep learning classifier.

This diagram shows the basic steps of how an assistant built with Rasa responds to a message:



Here Interpreter is part of NLU and Tracker, policy and action are part of Core.

- *The message is passed to an Interpreter, which converts it into a dictionary including the original text, the intent, and any entities that were found.*
- *The Tracker is the object which keeps track of the conversation state. It receives the info that a new message has come in.*
- *The policy receives the current state of the tracker, chooses the next action which is logged by the tracker and response is sent to the user. There are different policies to choose from. Along with policy "slots" which act as bot's memory(context). It also influences how the dialogue progresses.*

These settings are part of config.yml (Think this file as the brain of chatbot :P)

We have gone through the psychology of the bot. Now it's time to look at the environment of the chatbot which will help it to learn. Just like a growing baby, he/she learn from whatever is experienced. Similarly will need to train a chatbot with right training data.
Which comes in the form of text utterances part of defined intent with the tagged entity for training NLU and as a story(like a conversation) to train RASA core.

As planned before we need to be very thorough with the scope of the bot. Hence we need to define its own universe in which our bot operates. It has the intents which it should classify to, entities which it should extract, slots which it should remember to maintain context, and actions which it should perform to complete the task. And response templates which bot should utter back.

Actions are the things your bot runs in response to user input.

Zomato API:

***Why Zomato API:Zomato APIs** give you access to the freshest and most exhaustive information for over 1.5 million restaurants across 10,000 cities globally. With the **Zomato APIs**, you can: Search for restaurants by name, cuisine, or location. Display detailed information including ratings, location and cuisine.*

How Zomato api is used here in the bot:

1.First the bot asks for user loaction and cuisine preference.

2.Now we get city id and cuisine id using zomato api.

3.If either of the city id and cuisine id is not present then we return Null.

4.Else We search the list of restaurants available.

Requirements:

1.Python interpreter.

2.rasa(2.1.3)

3.Zomato api key.

4.rasa-x if need to deploy.

How to install rasa and rasa-x.

1.pip3 install rasa

2.pip3 install rasa-x

How to run:

1.Download the repository.

2.Use command rasa train to train the model.

3.Start action server using rasa rasa run actions.

4.Run rasa shell to open the shell.

5.Run rasa interactive to train using user friendly ui.

6.Run rasa-x to open rasa-x.

Using Docker Image of this project directly:

1.First Install docker.

2. Now type command docker pull 78659/rasa-shell:latest

3.Now Create Docker network using-docker network create action_server

4.Now Create actions server container using

docker run -it --name action_server --net action_server -v chatbot:/app 78659/rasa-shell:latest

3.docker run -it --name shell -p 5050:5050 –net action_server -v chatbot:/app 78659/rasa-shell:latest shell

4.docker start -a action_server

5.docker start -a shell