ChatBot

- With RASA

Chatbots

- **Chatbots** are computer program that simulates human conversation through voice commands or text chats or both in natural language, understand the user's intent and send responses based on the application's business rules and data.
- They can provide responses with appropriate information or direction. It can be considered as an enhanced channel of user interaction which would move from Interactive Voice Response to Intelligent Assistant Response.

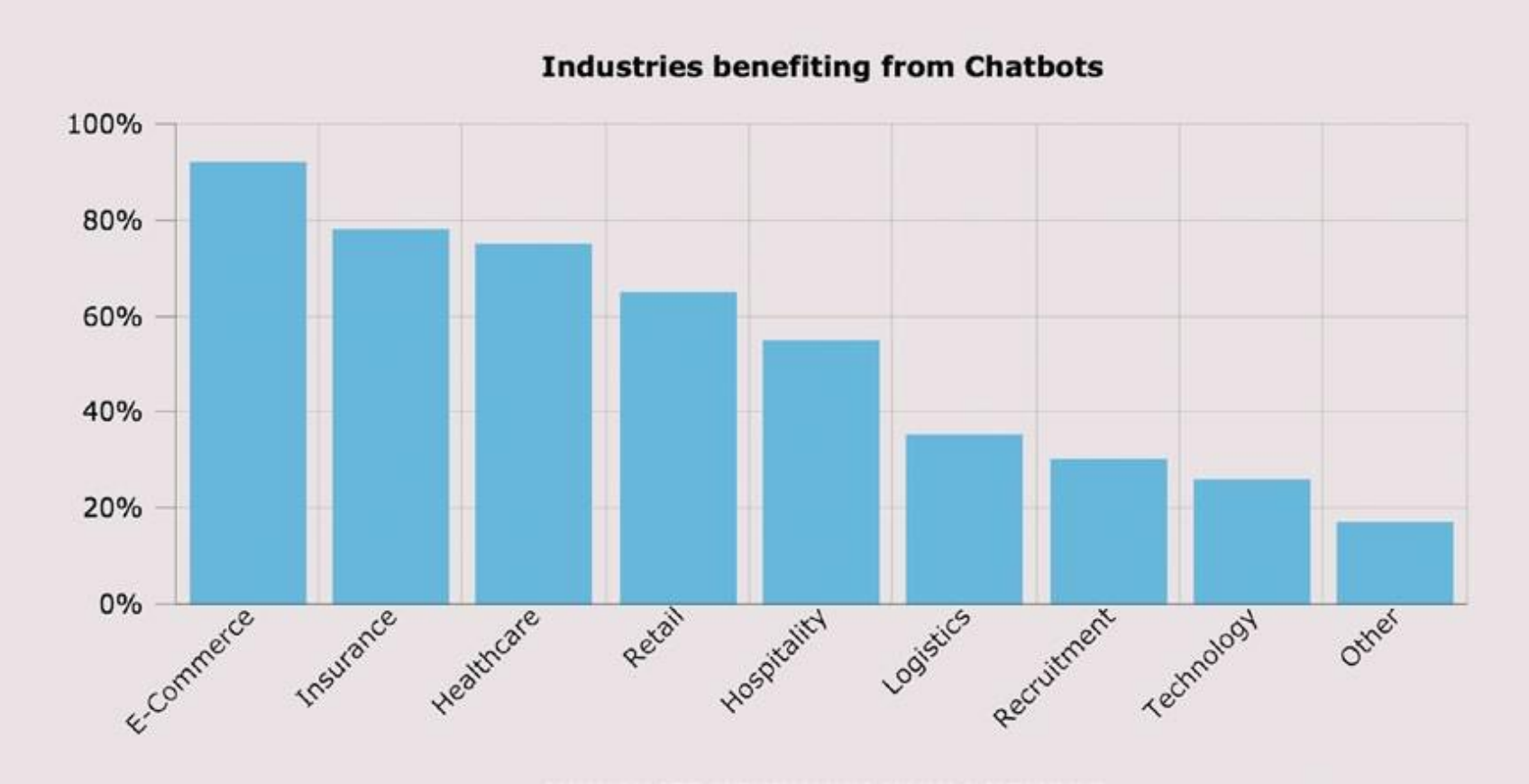
Chatbots

- You must have heard of Siri, IBM Watson, Google Allo, etc. The basic problem that these bots try to solve is to become an intermediary and help users become more productive.
- Bots tend to become more and more intelligent as they handle user data input and gain more insights from it. Chatbots are successful because they give you exactly what you want.

Chatbots


Chatbot trends over last 5 years based on web search term
Interest Over Time
100
80
60
40
20
Jan 2014
Jul 2014
Jan 2015
Jul 2015
Jan 2016
Jul 2016
Jan 2017
Jul 2017
Jan 2018
Jul 2018

Chatbots


Industries benefiting from Chatbots
100%
80%
60%
40%
20%
0%
E-Commerce
Insurance
Healthcare
Retail
Hospitality
Logistics
Recruitment
Technology
Other

RASA Framework

- https://rasa.com/
  - A framework for building conversational software
  - You can implement the actions your bot can take in Python code.
  - Rather than a bunch of if...else statements, the logic of your bot is based on a probabilistic model trained on example conversations.

Chatbots Terminologies

Action

A single step that a bot takes in a conversation (e.g. calling an API or sending a response back to the user).

Annotation

Adding labels to messages and conversations so that they can be used to train a model.

CMS

A Content Management System can be used to store bot responses externally instead of directly including it as part of the domain.

Chatbots Terminologies

Custom Action
An action written by a Rasa developer that can run arbitrary code mainly to interact with the outside world.

Default Action
A built-in action that comes with predefined functionality.

Domain
Defines the inputs and outputs of an assistant.
It includes a list of all the intents, entities, slots, actions, and forms that the assistant knows about.

Chatbots Terminologies

Entity

Structured information that can be extracted from a user message. For example a telephone number, a persons name, a location, the name of a product

Event

All conversations in Rasa are represented as a sequence of events. For instance, a UserUttered represents a user entering a message, and an ActionExecuted represents the assistant executing an action.

Chatbots Terminologies

Form

A type of custom action that asks the user for multiple pieces of information.For example, if you need a city, a cuisine, and a price range to recommend a restaurant, you can create a restaurant form to do that. You can describe any business logic inside a form.

Happy / Unhappy Paths

If your assistant asks a user for some information and the user provides it, we call that a happy path. Unhappy paths are all the possible edge cases of a bot. For example, the user refusing to give some input, changing the topic of conversation, or correcting something they said earlier.

Chatbots Terminologies

Intent
Something that a user is trying to convey or accomplish (e,g., greeting, specifying a location).

Interactive Learning
A mode of training the bot where the user provides feedback to the bot while talking to it. This is a powerful way to write complicated stories by enabling users to explore what a bot can do and easily fix any mistakes it makes.

Minimum viable assistant
A basic assistant that can handle the most important happy path stories.

Chatbots Terminologies

NLG

Natural Language Generation (NLG) is the process of generating natural language messages to send to a user. Rasa uses a simple template-based approach for NLG. Data-driven approaches (such as neural NLG) can be implemented by creating a custom NLG component.

Rasa NLU

Natural Language Understanding (NLU) deals with parsing and understanding human language into a structured format. Rasa NLU is the part of Rasa that performs intent classification and entity extraction.

Chatbots Terminologies

Pipeline

A Rasa bot’ s NLU system is defined by a pipeline, which is a list of NLU components (see “Rasa NLU Component”) in a particular order. A user input is processed by each component one by one before finally giving out the structured output.

Policy

Policies make decisions on how conversation flow should proceed. At every turn, the policy which predicts the next action with the highest confidence will be used. A Core model can have multiple policies included, and the policy whose prediction has the highest confidence decides the next action to be taken.

Chatbots Terminologies

Rasa Core
The dialogue engine that decides on what to do next in a conversation based on the context.

Rasa NLU Component
An element in the Rasa NLU pipeline.
Incoming messages are processed by a sequence of components called a pipeline. A component can perform tasks ranging from entity extraction to intent classification to pre-processing.

Slot
A key-value store that Rasa uses to track information over the course of a conversation.

Chatbots Terminologies

Story

A conversation between a user and a bot annotated with the intent / entities of the users' messages as well as the sequence of actions to be performed by the bot

Template / Response / Utterance

A message template that is used to respond to a user. This can include text, buttons, images, and other attachments.

Chatbots Terminologies

User Goal

A goal that a user wants to achieve. For example, a user may have the goal of booking a table at a restaurant. Another user may just want to make small talk. Sometimes, the user expresses their goal with a single message, e.g. “I want to book a table at a restaurant”. Other times the assistant may have to ask a few questions to understand how to help the user.

Word embedding / Word vector

A vector of floating point numbers which represent the meaning of a word. Words which have similar meanings should have vectors which point in almost the same direction. Word embeddings are often used as an input to machine learning algorithms.

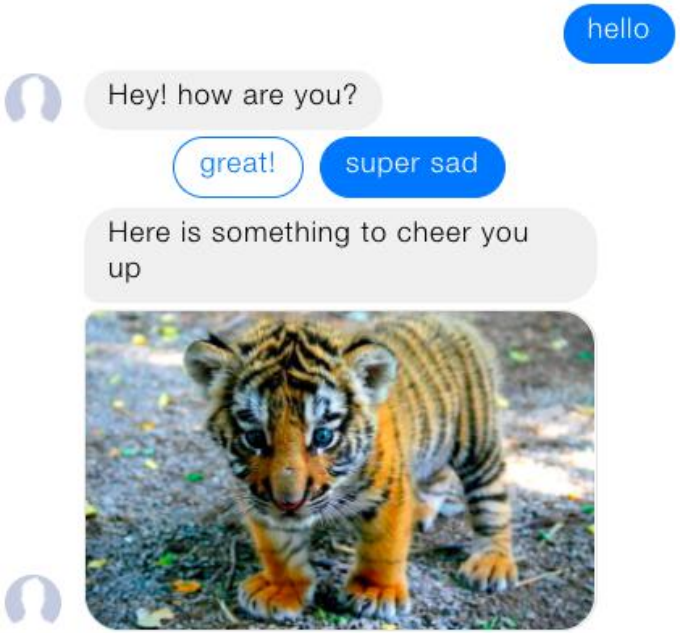
Install Rasa

To install Rasa, run the following pip command .

pip install rasa

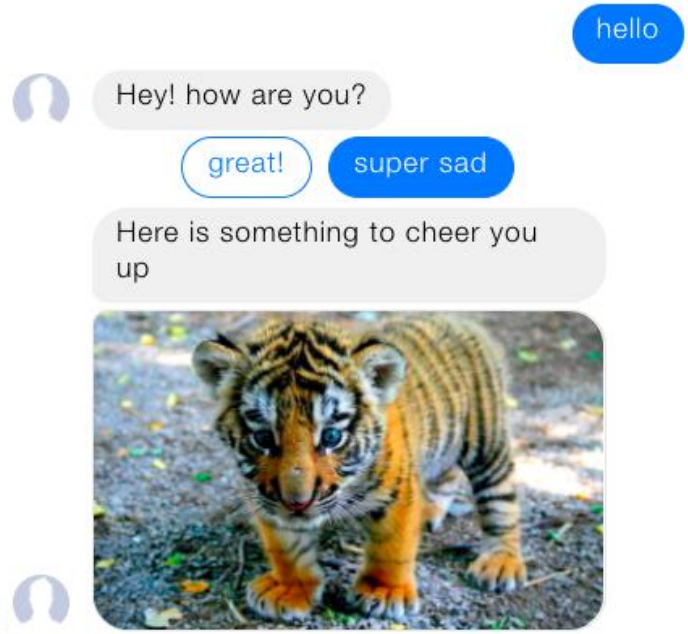
A Simple Demo

We will build a simple, friendly assistant which will ask how you're doing and send you a fun picture to cheer you up if you are sad.



A Simple Demo

1. Create a New Project
2. View Your NLU Training Data
3. Define Your Model Configuration
4. Write Your First Stories
5. Define a Domain
6. Train a Model
7. Test Your Assistant
8. Talk to Your Assistant



1. Create a New Project

The first step is to create a new Rasa project. To do this, run:

rasa init --no-prompt

The rasa init command creates all the files that a Rasa project needs and trains a simple bot on some sample data. If you leave out the --no-prompt flag you will be asked some questions about how you want your project to be set up.

1. Create a New Project

`__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 channels like fb messenger
`models/<timestamp>.tar.gz`	your initial model

2. View Your NLU Training Data

The first piece of a Rasa assistant is an NLU model, which turns user messages into structured data. To do this with Rasa, you provide training examples that show how Rasa should understand user messages, and then train a model by showing it those examples.

Run the code cell below to see the NLU training data created by the rasa init command:

$cat data/nlu.md

3. Define Your Model Configuration

The configuration file defines the NLU and Core components that your model will use. In this example, your NLU model will use the supervised_embeddings pipeline. You can learn about the different NLU pipelines here.

Let's take a look at your model configuration file.

$cat config.yml

4. Write Your First Stories

At this stage, you will teach your assistant how to respond to your messages. This is called dialogue management, and is handled by your Core model. Core models learn from real conversational data in the form of training "stories". A story is a real conversation between a user and an assistant. Lines with intents and entities reflect the user's input and action names show what the assistant should do in response.

This is how it looks as a story:

```
## story1
* greet
  - utter_greet
```

4. Write Your First Stories

Lines that start with - are actions taken by the assistant. In this tutorial, all of our actions are messages sent back to the user, like utter_greet, but in general, an action can do anything, including calling an API and interacting with the outside world.

Run the command below to view the example stories inside the file data/stories.md:

$cat data/stories.md

5. Define a Domain

The next thing we need to do is define a Domain. The domain defines the universe your assistant lives in: what user inputs it should expect to get, what actions it should be able to predict, how to respond, and what information to store. The domain for our assistant is saved in a file called domain.yml:

$cat domain.yml

`intents`	things you expect users to say
`actions`	things your assistant can do and say
`responses`	response strings for the things your assistant can say

6. Train a Model

Anytime we add new NLU or Core data, or update the domain or configuration, we need to re-train a neural network on our example stories and NLU data. To do this, run the command below. This command will call the Rasa Core and NLU train functions and store the trained model into the models/ directory. The command will automatically only retrain the different model parts if something has changed in their data or configuration.

$rasa train

7. Test Your Assistant

After you train a model, you always want to check that your assistant still behaves as you expect. In Rasa Open Source, you use end-to-end tests defined in your tests/ directory to run through test conversations that ensure both NLU and Core make correct predictions.

$rasa test

8. Talk to Your Assistant

Congratulations! 🚀 You just built an assistant powered entirely by machine learning.

The next step is to try it out! If you're following this tutorial on your local machine, start talking to your assistant by running:

$rasa shell

Tutorial: Building Assistants

Building a simple FAQ assistant

- Memoization Policy
- Response Selectors

Building a simple FAQ assistant

FAQ assistants are the simplest assistants to build and a good place to get started. These assistants allow the user to ask a simple question and get a response. We're going to build a basic FAQ assistant using features of Rasa designed specifically for this type of assistant.

In this section we're going to cover the following topics:

- Responding to simple intents with the MemoizationPolicy
- Handling FAQs using the ResponseSelector

Building a simple FAQ assistant

To prepare for this tutorial, we're going to create a new directory and start a new Rasa project.

$mkdir rasa-assistant

$rasa init

Let's remove the default content from this bot, so that the ***nlu.md***, ***stories.md*** and ***domain.yml*** files are empty.

Memoization Policy

The MemoizationPolicy remembers examples from training stories for up to a max_history of turns. we only need to pay attention to the last message the user sent, and therefore we'll set that to 1.

You can do this by editing your ***config.yml*** file as follows:

```
policies:
- name: MemoizationPolicy
  max_history: 1
- name: MappingPolicy
```

Memoization Policy

Now that we've defined our policies, we can add some stories for the goodbye, thank and greet intents to the ***stories.md*** file:

```
## greet
* greet
  - utter_greet
## thank
* thank
  - utter_noworries
## goodbye
* bye
  - utter_bye
```

Memoization Policy

We'll also need to add the intents, actions and responses to our ***domain.yml*** file in the following sections:

```
intents:
  - greet
  - bye
  - thank
```

```
responses:
  utter_noworries:
    - text: No worries!
  utter_greet:
    - text: Hi
  utter_bye:
    - text: Bye!
```

Memoization Policy

Finally, we'll copy over some NLU data from Sara into our ***nlu.md file***:

intent:greet

- Hi
- Hey
- Hi bot

intent:bye

- goodbye
- goodnight
- good bye

intent:thank

- Thanks
- Thank you
- Thank you so much
- Thanks bot
- Thanks for that
- cheers

Memoization Policy

This bot should now be able to reply to the intents we defined consistently, and in any order.

For example:

```
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input ->  bye!
Bye!
Your input ->  thanks
No worries!
Your input ->  hi!
Hi
Your input ->  goodbye
Bye!
Your input ->  thank you
No worries!
```

Memoization Policy

The file ***tests/conversation_tests.md*** contains example test conversations. Delete all the test conversations and replace them with some test conversations for your assistant so far:

```
## greet + goodbye
* greet: Hi!
  - utter_greet
* bye: Bye
  - utter_bye
```

```
## greet + thanks
* greet: Hello there
  - utter_greet
* thank: thanks a bunch
  - utter_noworries
```

```
## greet + thanks + goodbye
* greet: Hey
  - utter_greet
* thank: thank you
  - utter_noworries
* bye: bye bye
  - utter_bye
```

Memoization Policy

To test our model against the test file, run the command:

$rasa test --stories tests/conversation_tests.md

The test command will produce a directory named ***results***. It should contain a file called ***failed_stories.md***, where any test cases that failed will be printed. It will also specify whether it was an NLU or Core prediction that went wrong.

Response Selectors

The *ResponseSelector* NLU component is designed to make it easier to handle dialogue elements like Small Talk and FAQ messages in a simple manner. By using the *ResponseSelector*, you only need one story to handle all FAQs, instead of adding new stories every time you want to increase your bot's scope.

People often ask Sara different questions surrounding the Rasa products, so let's start with three intents: *ask_channels*, *ask_languages*, and *ask_rasax*. We're going to copy over some NLU data into our ***nlu.md***. It's important that these intents have an *faq/* prefix, so they're recognised as the faq intent by the *ResponseSelector*:

```
## intent: faq/ask_channels
- What channels of communication does rasa support?
- what channels do you support?
- what chat channels does rasa uses
- channels supported by Rasa
- which messaging channels does rasa support?

## intent: faq/ask_languages
- what language does rasa support?
- which language do you support?
- which languages supports rasa
- can I use rasa also for another laguage?
- languages supported

## intent: faq/ask_rasax
- I want information about rasa x
- i want to learn more about Rasa X
- what is rasa x?
- Can you tell me about rasa x?
- Tell me about rasa x
- tell me what is rasa x
```

Response Selectors

Next, we'll need to define the responses associated with these FAQs in a new file called ***responses.md*** in the *data/* directory:

```
## ask channels
* faq/ask_channels
  - We have a comprehensive list of [supported connectors](https://rasa.com/docs/core/connectors/),
    you don't see the one you're looking for, you can always create a custom connector by following
    [this guide](https://rasa.com/docs/rasa/user-guide/connectors/custom-connectors/).

## ask languages
* faq/ask_languages
  - You can use Rasa to build assistants in any language you want!

## ask rasa x
* faq/ask_rasax
 - Rasa X is a tool to learn from real conversations and improve your assistant. Read more [here](ht
```

Response Selectors

The *ResponseSelector* should already be at the end of the NLU pipeline in our ***config.yml***:

```yaml
language: en
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
```

Response Selectors

Now that we've defined the NLU side, we need to make Core aware of these changes. Open your ***domain.yml*** file and add the faq intent:

```
intents:
  - greet
  - bye
  - thank
  - faq
```

Response Selectors

We'll also need to add a retrieval action, which takes care of sending the response predicted from the ***ResponseSelector*** back to the user, to the list of actions. These actions always have to start with the *respond_* prefix:

```
actions:
  - respond_faq
```

Next we'll write a story so that Core knows which action to predict:

```
## Some question from FAQ
* faq
    - respond_faq
```

Response Selectors

After all of the changes are done, train a new model and test the modified FAQs:

$rasa train

$rasa shell

Response Selectors

Here's a minimal checklist of files we modified to build a basic FAQ assistant:

- `data/nlu.md` : Add NLU training data for `faq/` intents
- `data/responses.md` : Add responses associated with `faq/` intents
- `config.yml` : Add `ReponseSelector` in your NLU pipeline
- `domain.yml` : Add a retrieval action `respond_faq` and intent `faq`
- `data/stories.md` : Add a simple story for FAQs
- `test_stories.md` : Add E2E test stories for your FAQs