Introduction to Chatbot with Rasa
Yogesh Kulkarni

Introduction to Chatbots

Introduction

Calling the Call Center

- Calling to an IVR (Integrated Voice Response)
- A pre-recorded menu selection.
- "Please press 1 for Account Details, Please press 2 for . . . "
- Till it comes to your option.
- Else, you are given access to a person to talk to.

Boring? Annoying? But still heavily used . . . , Why?

(Ref: Deep Learning and NLP A-Z - Kirill Eremenko)

Instead, how about typing/saying your query directly and getting the answer right away?

Solution

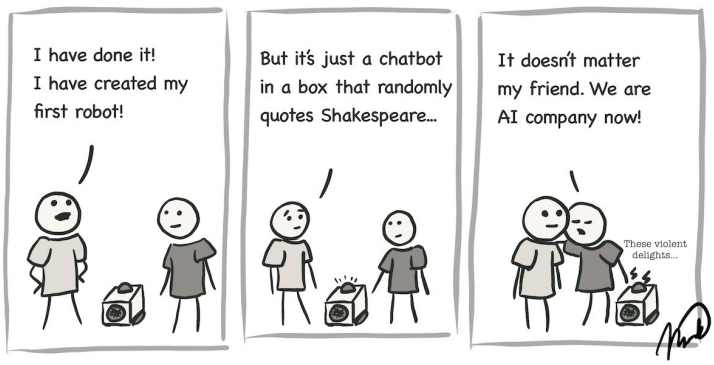
Chatbots!!!

- Which problem of IVR it is solving?
- Advantages?
- Disadvantages?
- Gaining popularity . . .
- Many platforms
- Any local chatbot companies/platforms?

Crystal Ball

- 85% Of customer interactions will be managed without a human by 2020 – Gartner prediction
- "The global chatbot market is expected to reach $1.23 billion by 2025" - Business Insider

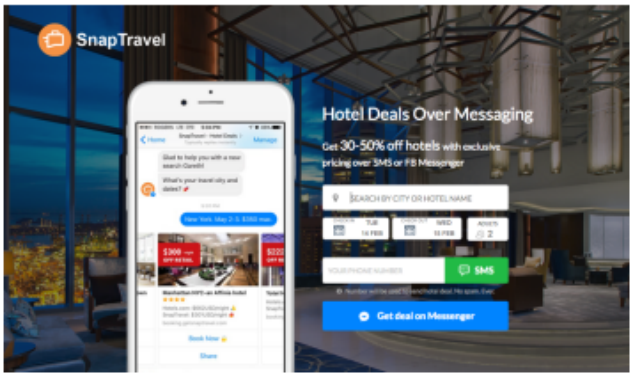
Chatbot == AI



(Ref: How to build awesome Rasa chatbot for a web - Martin Novak)

Sample Application

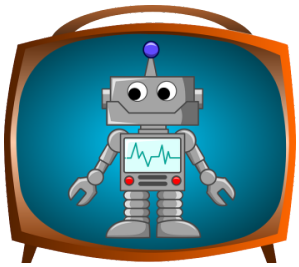
SnapTravel has processed $1 million in hotel bookings inside Messenger.



(Ref: Innovation in Health - Ritesh Ptael, et al)

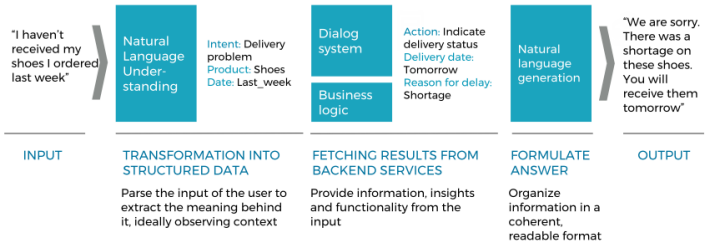
So, What is a Chatbot?

"Chatbots are a form of human-computer dialog system which operates through natural language via text or speech"- Deryugina, 2010; Sansonnet et al., 2006.



(Ref: Rasa - mdd01 course on github)

Anatomy of a Chatbot



(Ref: Chatbots and AI - botfuel)

Why so many chatbot startups?

- VCs appear excited with this new tool, more services, more opportunities, new battlegrounds for the big players (likely leading to acquisitions).
- So even without real technological breakthroughs, there is at least some money to be made investing in bot startups.
- But the real issue is : Truly 'conversational' software is a difficult problem to solve.

(Ref: We don't know how to build conversational software yet (Alan Nichol Apr 2016))

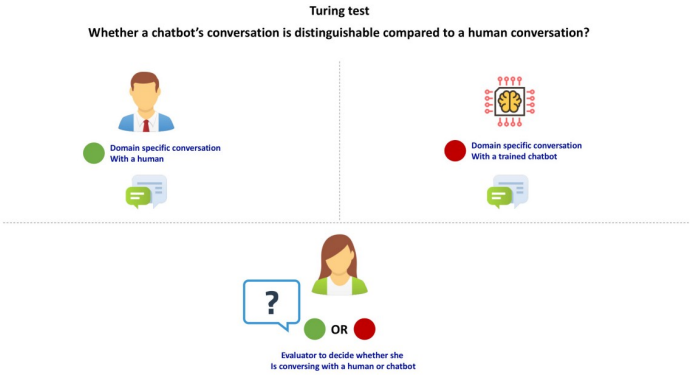
How difficult?

So what tools do developers need to do better than this?

(Ref: A New Approach to Conversational Software - Alan Nichol)

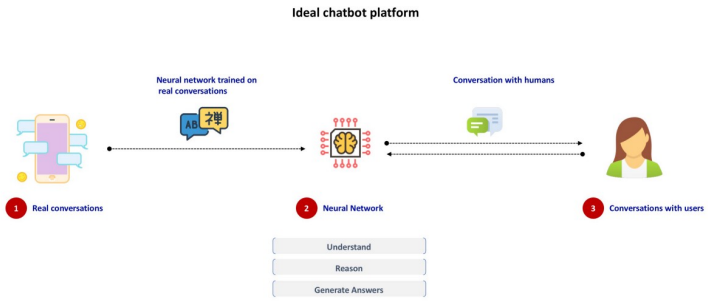
NLU is AI

Understanding Natural Language is Hallmark of Artificial Intelligence!!



(Ref: Conversational AI: Understanding the Basics and Building a Chatbot in Rasa module - Manikandan Jeeva)

NLU is AI



But the current bots are not this generic.

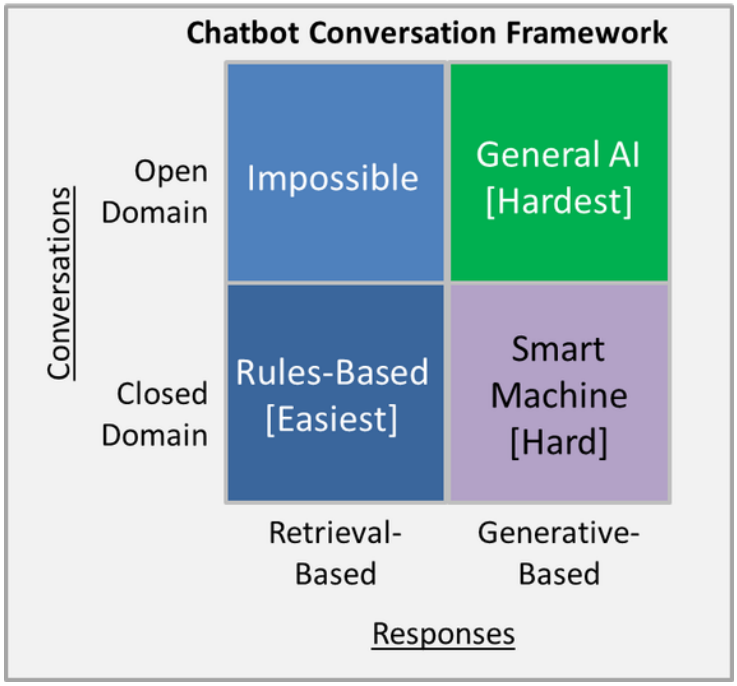
(Ref: Conversational AI: Understanding the Basics and Building a Chatbot in Rasa module - Manikandan Jeeva)

Types of Chatbots

- Command & response: Stateless bots are essentially a command line app over HTTP
- Hard-coded conversation flows: navigate a flow chart defined. http://superscriptjs.com/ allows that. Evi is an intelligent bot built with "knowledge base" technology.
- Fuzzy/continuous/fluid state: that's the goal. Human conversations don't follow a template

(Ref: We don't know how to build conversational software yet (Alan Nichol Apr 2016))

Classification of Chatbots

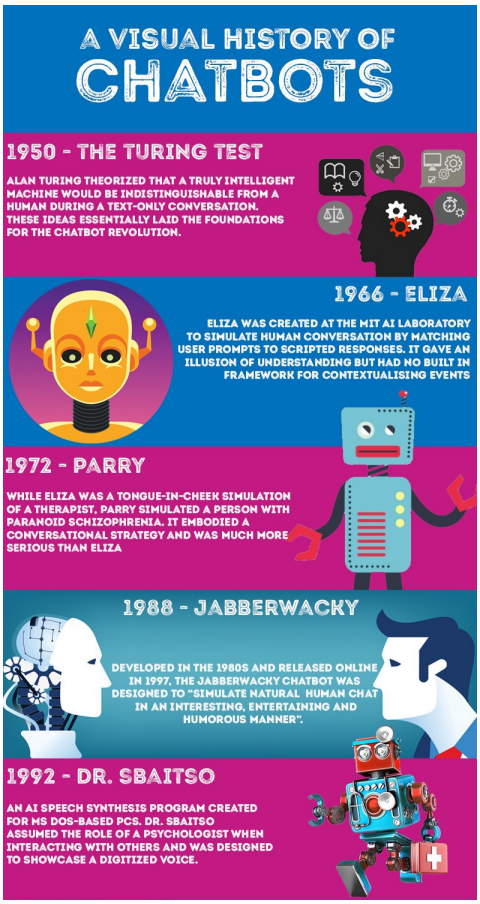


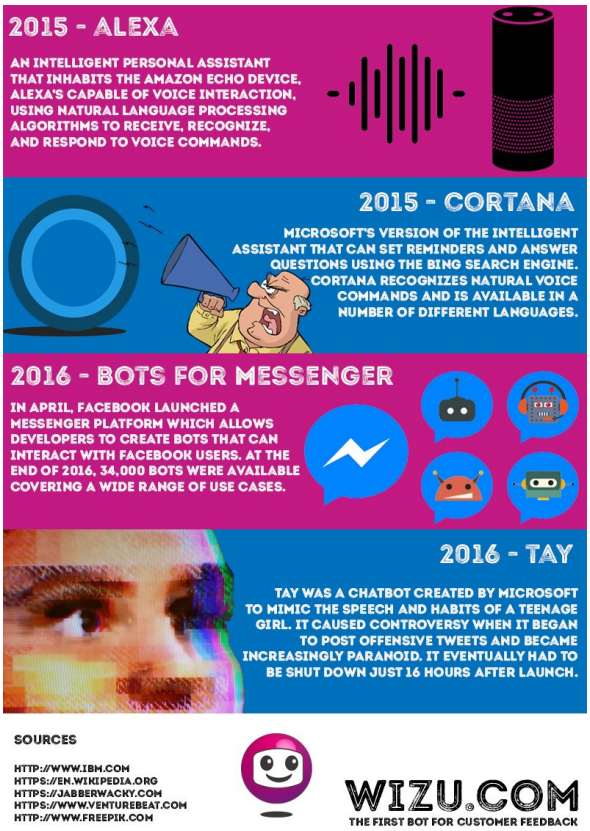
- Retrieval-based models (easier) use a repository of predefined responses and some kind of heuristic to pick an appropriate response based on the input and context.
- Generative models (harder) are based on Machine Translation techniques, but instead of translating from one language to another, we "translate" from an input to an output (response).

History

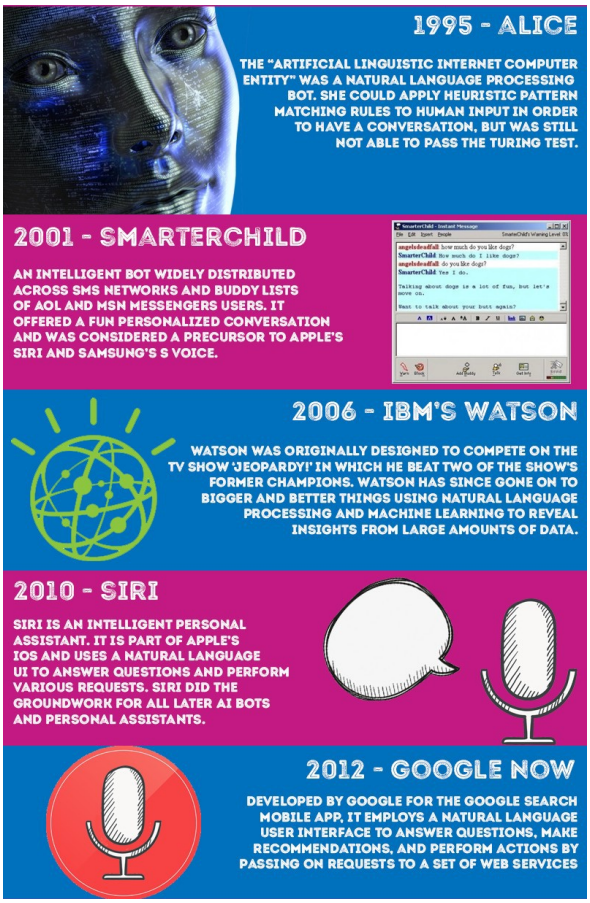
- Chatbot history: starts in 1960s.
- Eliza by MIT professor Joseph Weizenbaum: a psychotherapist, Pattern based.
- ALICE ("Artificial Linguistic Internet Computer Entity"), 1995, Richard Wallace using AIML (artificial intelligence markup language)
- Then of course, most tech giants

(Ref: Understanding AI Chatbots, Challenges, Opportunities & Beyond - Pramod Chandrayan)

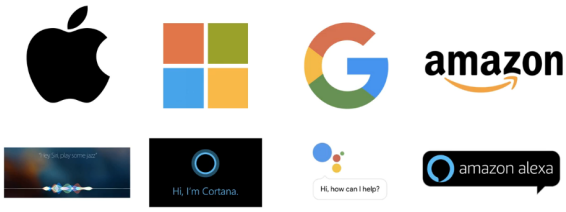




History



The Giants are at it ...



(Ref: Deep Learning and NLP A-Z - Kirill Eremenko)

If you want to develop one

- Chatbots or QA systems, predominantly voice based,
- Underlying processing is primarily Natural Language Processing (NLP).
- You can have your own chatbot, specific to you!!
- NLP is the core skill needed.

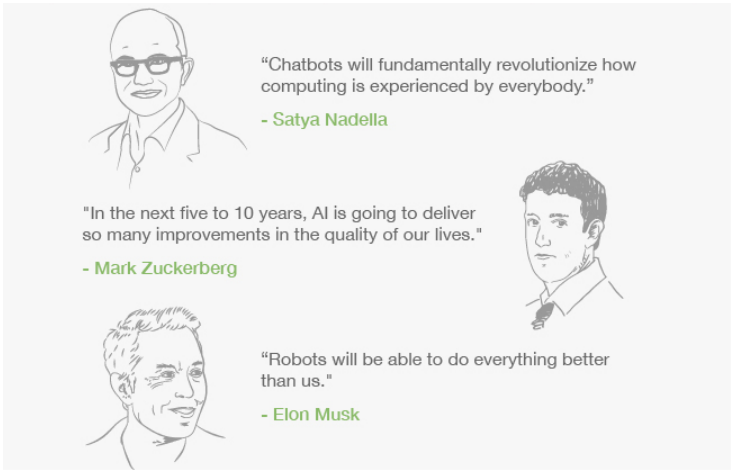
Why so much popularity?

Chatbots are:

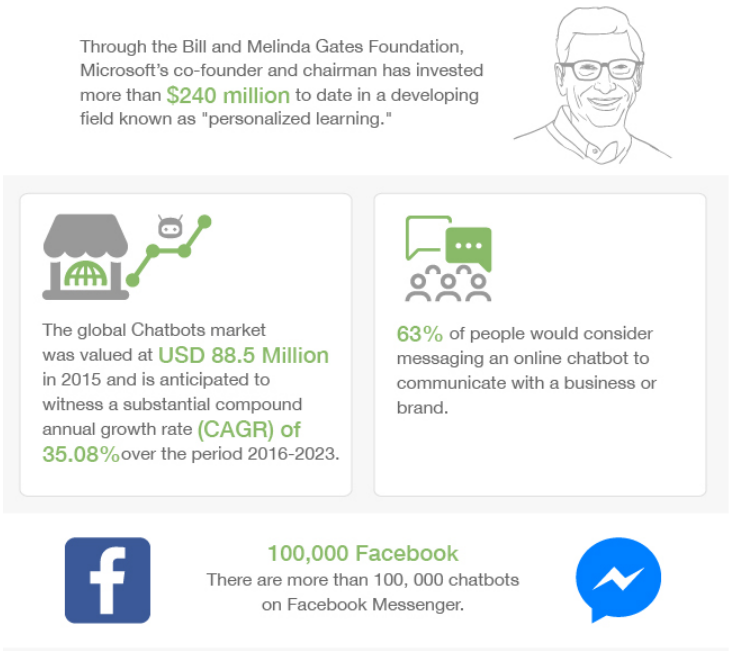
- Autonomous and Always Available
- Drive Conversation
- Able to handle millions of requests, scalable.

But to have a good Chatbot, at core, we would need expertize in NLP!!

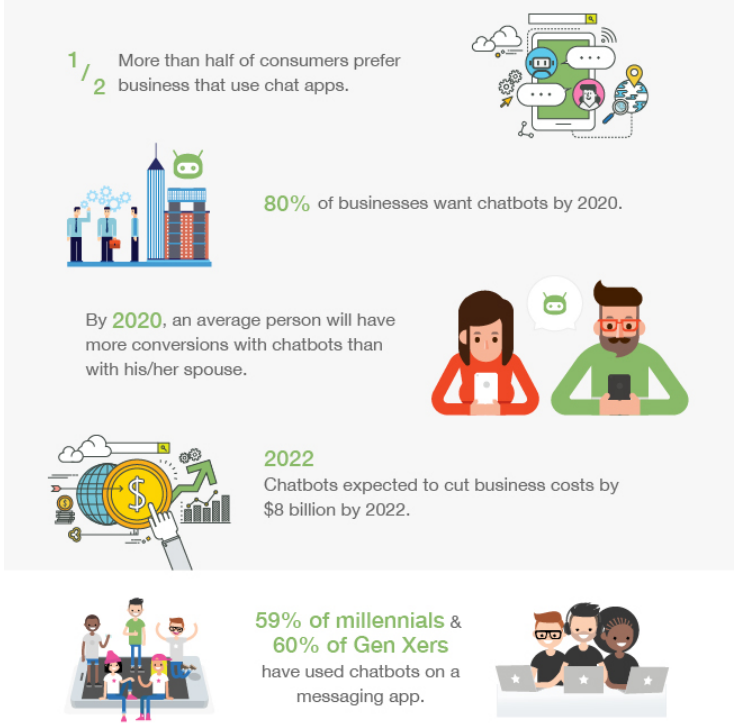
Forecasts

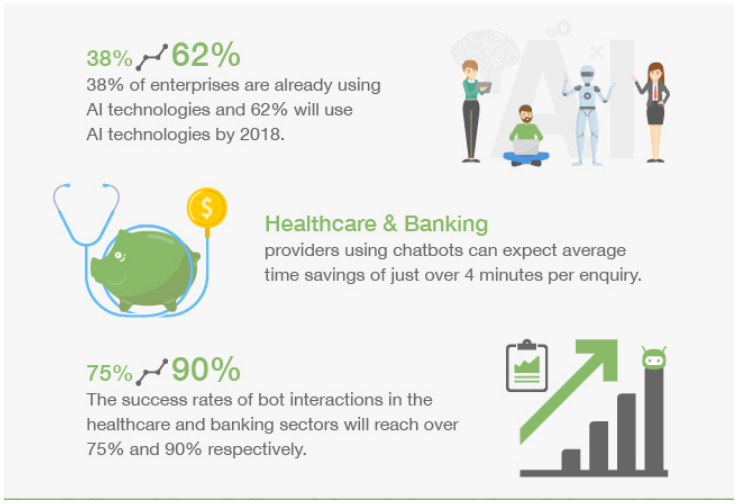


(Ref: Subro.io)



Forecasts





(Ref: Subro.io)

Challenges for Chatbot

- Security: should ensure that only relevant data is being asked and captured as an input and also is being securely transmitted over the Internet.
- Making Chatbot stick, like-able and functioning
- Language Modeling: meaning based vectorization, even for vernacular.
- etc . . .

Pros

- Anytime, day or night
- Can handle repetitive, boring tasks
- Scalable
- Consistent
- Can gather data

Cons

- NLU is hard, AI is not GENERAL yet
- Cant design for ANY interaction
- Can be Risky
- Cant trust for sensitive data

Platforms

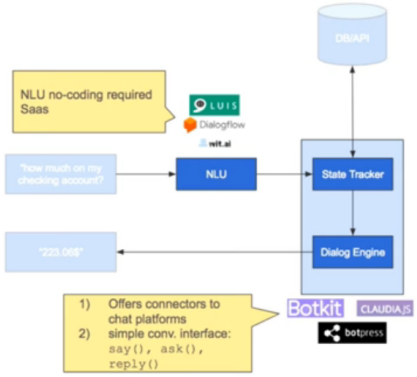
Chatbot Platforms

Bot Building Platforms

- Set of tools and architecture
- To help you design unique conversation scenarios, define corresponding actions and analyze interactions.
- Understand Natural language (NLU—Natural language understanding),
- Process the conversation text and extracts information (NLP—Natural Language Processing) and
- Respond to the user preserving the context of the conversation (NLG—Natural Language Generation).

(Ref: Chatbots 101 - Architecture & Terminologies - Bhavani Ravi)

Typical Chatbot Platform



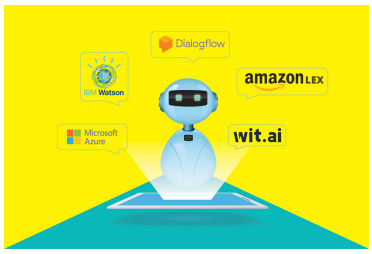
NLU is over net and Dialog management is still if-and-else.

(Ref: The talk would be about Rasa, an open-source chatbots platform - Nathan Zylbersztejn)

Conversation Platforms

Established players

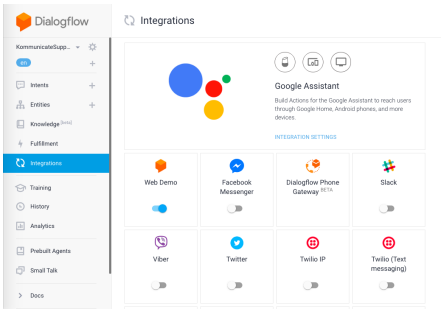
- Google DialogFlow : https://dialogflow.com
- Facebook Wit.ai : https://www.wit.ai
- IBM Watson Assistant : https://www.ibm.com/cloud/watson-assistant/
- Microsoft LUIS : https://www.luis.ai/
- Amazon Lex : https://aws.amazon.com/lex
- RASA : https://www.rasa.com/



(Ref : Dialogflow vs Lex vs Watson vs Wit vs Azure Bot — Which Chatbot Service Platform To Use?)

Google Dialogflow

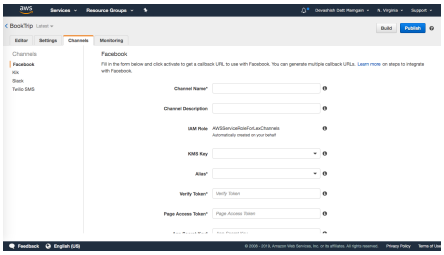
- Previous known as API.ai
- Completely closed-source product with APIs and web interface.
- Voice and text-based conversational interface
- Easy to even non-techies to create basic bots.



(Ref : Dialogflow vs Lex vs Watson vs Wit vs Azure Bot — Which Chatbot Service Platform To Use?)

Amazon Lex

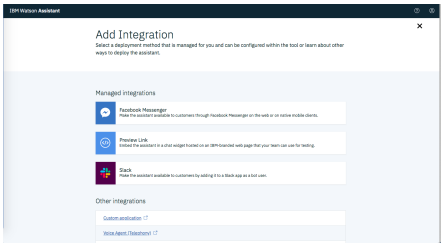
- Same deep learning technologies as Alexa
- Voice and text-based conversational interface
- Provides a web interface to create and launch bots.



(Ref : Dialogflow vs Lex vs Watson vs Wit vs Azure Bot — Which Chatbot Service Platform To Use?)

IBM Watson Assistant

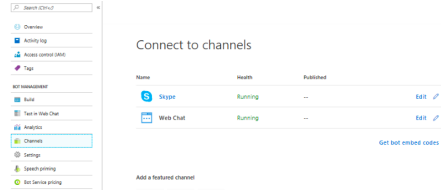
- Has support for searching for an answer from the knowledge base
- First, you need to create a Skill and then go to Assistant to integrate it with other channels.



(Ref : Dialogflow vs Lex vs Watson vs Wit vs Azure Bot — Which Chatbot Service Platform To Use?)

Microsoft LUIS, Azure Bot Service

- Web interface is available to create and publish bots which is fairly easy to understand.

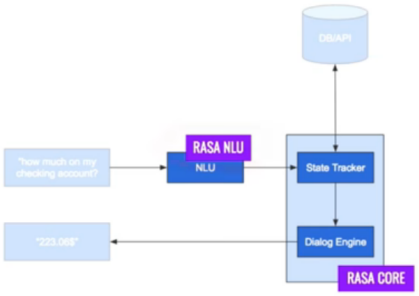


(Ref : Dialogflow vs Lex vs Watson vs Wit vs Azure Bot — Which Chatbot Service Platform To Use?)

Conversation Platforms

We will be looking more closely at Rasa Platform...

Rasa Chatbot Architecture

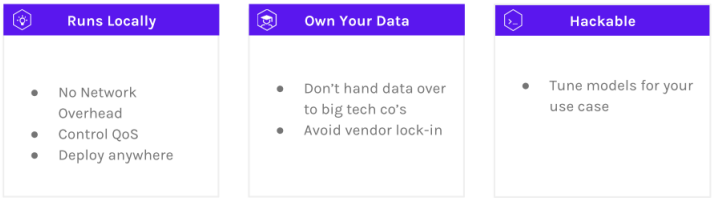


Machine Learning based and in Python.

(Ref: The talk would be about Rasa, an open-source chatbots platform - Nathan Zylbersztejn)

Why Rasa?

Rasa Stack Work-flow



(Ref: Conversational AI: Building clever chatbots - Tom Bocklisch)

Advantages of Rasa

- Some popular chatbot platforms are API.ai, Wit.ai, Facebook APIs, Microsoft LUIS, IBM Watson, etc. are HOSTED services, meaning they send data over net.
- RASA-NLU builds a local NLU (Natural Language Understanding) model for extracting intent and entities from a conversation. No data is passed over net.
- It's open source, fully local and above all, free!
- It is also compatible with wit.ai, LUIS, or api.ai, so you can migrate your chat application data into the RASA-NLU model.

Why Local? - Security

- Enterprises (and their customers) want AI assistants, not FAQ chatbots, but they're difficult to build; if you're not Google, and that is where Rasa comes in.
- Fully controlling and owning your IP and data is crucial, and can't be done using cloud APIs. In the age of GDPR, HIPAA companies run Rasa on-premise or in a private cloud.

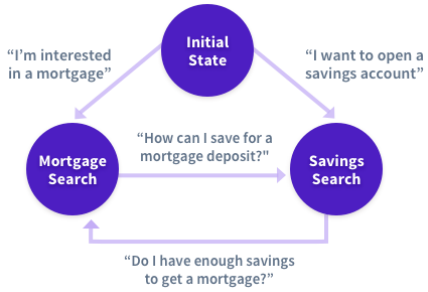
(Ref: Our Open Source Community Is Growing Faster than Ever. Weren't Chatbots Meant to Be Dead? - Rasa Community Update - Alexander Weidauer)

Control

- You don't have to hand over your data to FB/MSFT/GOOG
- You don't have to make a https call to parse every message.
- You can tune models to work well on your particular use case.

State Machines

Everyone uses state machines and state machines don't scale



But then how to handle "Straying from the happy path"?

(Ref: A New Approach to Conversational Software - Alan Nichol)

Support for Frontends

Rasa support right out of the box for:

- **Your Own Website!!!**
- Facebook Messenger
- Slack
- Telegram
- Twilio
- Microsoft Bot Framework
- Cisco Webex Teams
- RocketChat
- Mattermost
- Custom Connectors

(Ref: Messaging and Voice Channels - Rasa Docs)

Rasa Approach

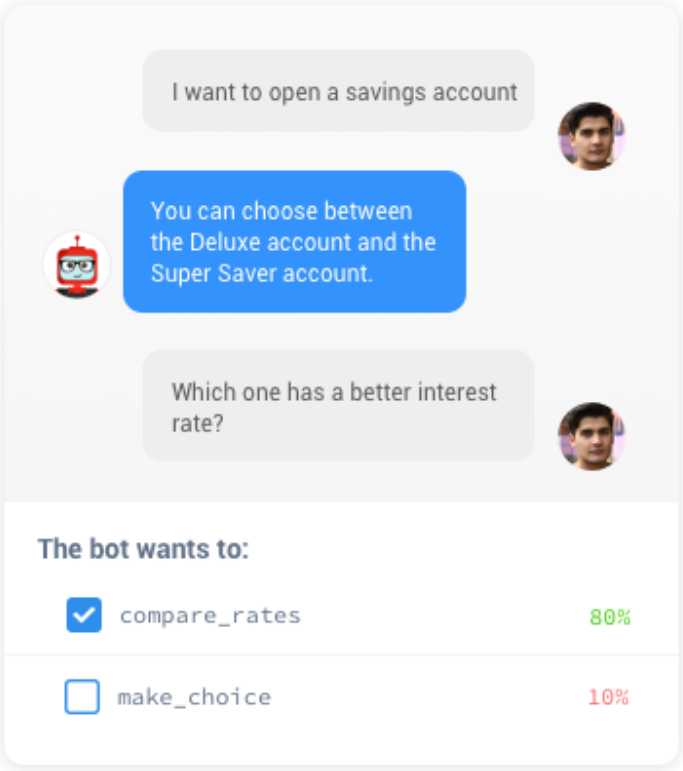
How can we use machine learning to move beyond the bag-of-rules approach?

- The Rasa Approach: flowcharts are useful for doing the initial design of a bot and describing a few of the happy paths, but you shouldn't take them literally and hard-code a bunch of rules. We've seen many times how this approach doesn't scale beyond simple conversations.
- With Rasa Core, you manually specify all of the things your bot can say and do. We call these actions. One action might be to greet the user, another might be to call an API, or query a database. Then you train a probabilistic model to predict which action to take given the history of a conversation.

(Ref: A New Approach to Conversational Software - Alan Nichol)

Rasa Approach

- In interactive learning mode, you provide step-by-step feedback on what your bot decided to do. It's kind of like reinforcement learning, but with feedback on every single step (rather than just at the end of the conversation).
- When your bot chooses the wrong action, you tell it what the right one would have been. The model updates itself immediately (so you are less likely to encounter the same mistake again) and once you finish, the conversation gets logged to a file and added to your training data.



RASA Core

(Ref: A New Approach to Conversational Software - Alan Nichol)

Rasa Approach

- You've instantly resolved an edge case, without staring at your code for ages figuring out what went wrong. And because you're providing step-by-step feedback, rather than a single reward at the end, you're teaching the system much more directly what's right and wrong.
- From experience it was found that, a couple of dozen short conversations are enough to get a first version of your system running.

(Ref: A New Approach to Conversational Software - Alan Nichol)

Other Reasons

Rasa Stack is the leading open-source machine learning toolkit for developers to extend bots beyond answering simple questions.
Following figures are of Oct, 2018:

(Ref: Building Conversational AI w Rasa Stack — Alan Nichol at PyBay2018)

Can't we develop the chatbot ourselves?

- Its not simple/mundane software development. Need people with varied backgrounds apart from developers, especially ML/DL/NLP.
- Conversational AI (meaning, the one thats more than atomic Q and A) is a hard problem. Even the biggies haven't cracked it fully (Google's duplex can do only one or two things!!).
- Softwrae patterns are not developed yet.
- Rasa helps in taking us at some level and being open source you can help it grow further.

(Ref: Building Conversational AI w Rasa Stack — Alan Nichol at PyBay2018)

Is Rasa better than others?

- Not too much, as for NLU capabilities, even compared to Google (Dialoglow), Facebook (Wit.API), etc, as per Benchmark study done by TU Munich
- Depends very much on the training data.
- But being open source and native, makes it a lot more attractive.
- No data over Internet makes it fast and secure.
- Sustainable: Not a fly-by-night operator. AT least you have full source that works!! (Apple acquired and closed init.ai)

(Ref: The talk would be about Rasa, an open-source chatbots platform - Nathan Zylbersztejn)

Limitations of Algorithms

- NLP is moving incredibly fast
- BERT and GPT-2 pre-training have much better representations
- But one obvious thing that came out was : "Algorithms alone aren't enough: you also need powerful tools for collecting and annotating training data."
- For Chatbots, eal conversations with your assistant are by far the most valuable source of data.

(Ref: Algorithms alone won't solve conversational AI - Introducing Rasa X - Alan Nichol)

Btw, Rasa-X has arrived

"With Rasa X, we're giving all developers tools for viewing and filtering conversations between humans and their Rasa assistant, for turning those conversations into training data, for managing and versioning models, and for easily giving test users access to their assistants."
- Alan Nichol

Rasa-X?

What Rasa X is:

- Rasa X is a tool that helps you build, improve, and deploy AI Assistants that are powered by the Rasa framework.
- Rasa X runs on your own computer and you can deploy it to your own server.
- None of your conversations or training data are ever sent anywhere.
- Rasa X comes in Community ($0) and Enterprise (paid) editions. The community edition is free but not open source
- Rasa X is a tool to learn from real conversations and improve your assistant.
- Using it is totally optional. If you don't want to, you can just use Rasa on its own.

What Rasa X is NOT:

- It's not a hosted service.
- It's not an all-in-one, point-and-click bot platform.

What happens to Rasa NLU and Rasa Core

- Rasa NLU and Core (aka Rasa Stack) are now part of a single open source framework: Rasa, and version 1.0 is available
- You can still use NLU and Core independently, they are just part of the same package now.
- We will use Rasa NLU and Core in a traditional (old) way to understand how chatbot is developed

Introduction to Rasa

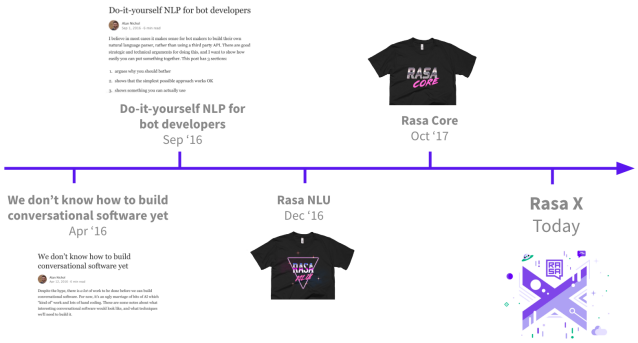
Introduction to RASA Platform

(Ref: https://www.rasa.com/docs/getting-started/overview/)

Overview

- The Rasa Stack is a pair of open source libraries (Rasa NLU and Rasa Core) that allow developers to expand chatbots and voice assistants beyond answering simple questions.
- Using state-of-the-art machine learning, your bots can hold contextual conversations with users.
- Rasa is production ready and used in large companies everywhere.

Journey



Who is it for?

- The intended audience is mainly people developing bots, starting from scratch or looking to find a a drop-in replacement for wit, LUIS, or Dialogflow.
- The setup process is designed to be as simple as possible.
- Rasa NLU is written in Python, but you can use it from any language through a HTTP API.
- If your project is written in Python you can simply import the relevant classes.
- If you're currently using wit/LUIS/Dialogflow, you just:
  - Download your app data from wit, LUIS, or Dialogflow and feed it into Rasa NLU
  - Run Rasa NLU on your machine and switch the URL of your wit/LUIS api calls to localhost:5000/parse

RASA Stack

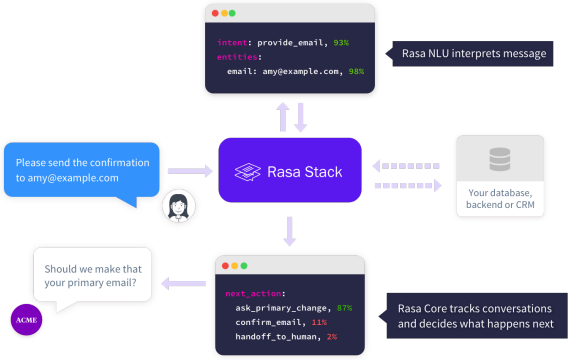
- Rasa NLU performs Natural Language Understanding, which means taking free-form text like **Please send the confirmation to amy@example.com** and turning it into structured data.
- Rasa Core performs Dialog Management, which means keeping track of a conversation, and deciding how to proceed.
- Both Rasa Core and NLU use Machine Learning to learn from real example conversations.
- Rasa NLU and Core are independent. You can use NLU without Core, and vice versa.

RASA Stack

(Ref: https://rasa.com/docs/get_started_step1/)

Summary: The Rasa Stack

Rasa NLU's job is to interpret messages, and Rasa Core's job is to decide what should happen next.



Installation

Python

- Install Anaconda for Python 3.7
- Else from Python.org and then additionally all requisite libraries
- Ubuntu :

```
sudo apt-get install build-essential python-dev git
```

- Windows Build tools: Make sure the Microsoft $VC++$ Compiler Visual Studio 2015 is installed, so python can compile any dependencies or https://visualstudio.microsoft.com/visual-cpp-build-tools/ Download the installer and select VC++ Build tools in the list.

Conda Installation

- Install the Conda(miniconda) from https://docs.conda.io/en/latest/mini as per the OS
- Check the Conda version `conda --version` conda 4.7.10
- In case need to upgrade, run below command `conda update conda`

Setup in Virtual 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_env python=3.7`
- Python 3.6 as different asyncio format, so better to do it in 3.7
- Activate the new environment to use it

```
LINUX, macOS: conda activate rasa
WINDOWS: activate rasa
```

(Note: Env management is again a sour point. It creates complete copy (deeeep) of python 3.7 and all other packages inside "envs" folder. Goes to 1.5 GB!! Can someone optimize it?)

Rasa Installation

- Install latest Rasa stack.
- Rasa NLU + Core is now in the single package. Do not install them separately like in past. Your mileage may vary.
- `pip install rasa`

If you get Microsoft Build tool error:

- Go to https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2017
- Build tools for Visual Studio 19 are fine too. Select only $C++$ Build tools. Install.
- Re-run rasa install command

Spacy Installation

Additionally, spaCy:

```
pip install rasa[spacy]
python -m spacy download en
python -m spacy download en_core_web_md
python -m spacy link en_core_web_md en
```

If you get linking permission error:

- Run cmd as administrator,
- Activate rasa_env
- Do all the above spacy commands.

Other Installations

To show conda envs in notebook `conda install nb_conda_kernels`
Apart from this, may need to

- `pip install nest_asyncio`
- Write following code to test, in ipynb put this in the first cell

```
import nest_asyncio

nest_asyncio.apply()
print("Event loop ready.")
```

Some more (optional):

- Download and install ngrok from https://ngrok.com/download
- Need to have API keys for Slack, CRICINFO, ZOMATO, etc

Exercise: Check Installation

RASA Starter Pack

- Clone the starter pack provided by Rasa from https://github.com/RasaHQ/starter-pack-rasa-stack
- It gives latest running application
- Let's take a look at the folder structure and the files that were created
- Install git, can git clone above package.
- `cd starter-pack-rasa-stack;activate rasa; pip install -r requirements.txt`

Exercise: Check Installation

RASA Starter Pack

- The "domain.yml" file describes the travel assistant's domain. It specifies the list of intents, entities, slots, and response templates that the assistant understands and operates with.
- The "data/nlu_data.md" file describes each intent with a set of examples that are then fed to Rasa NLU for training.
- The "data/stories.md" file provides Rasa with sample conversations between users and the travel assistant that it can use to train its dialog management model.
- Rasa provides a lot flexibility in terms of configuring the NLU and core components. For now, we'll use the default "nlu_config.yml" for NLU and "policies.yml" for the core model.

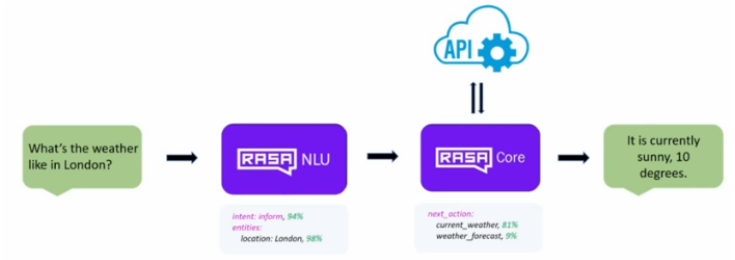
Exercise: Check Installation

RASA Starter Pack

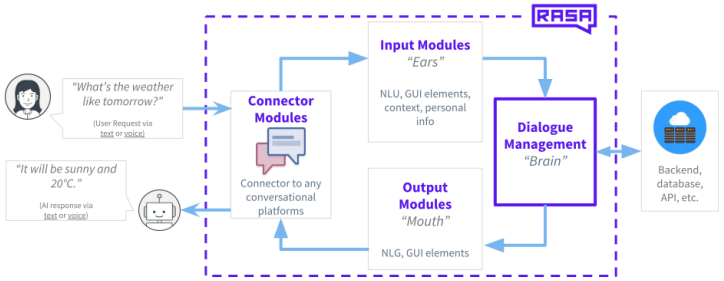
- May need to rename data files to have standrard names, else use command line flags in the following commands
- Train
- Use Rasa Shell to execute.
- Look at data files to see what interaction has been coded.
- Accordingly run the chatbot

Rasa Concepts

Summary: The Rasa Stack



Like a Human!!



(Ref: Conversational AI: Building clever chatbots - Tom Bocklisch)

Rasa NLU: Intents and Entities

RASA NLU Server

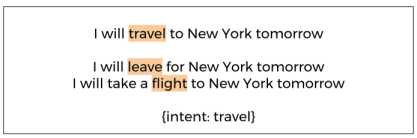
- A RASA-NLU platform needs to be trained before we start using it.
- We need to supply it with a few sentences and mention which are the intents and entities in it.

Natural Language Understanding (NLU)

Goal: take a sentence string, and extract structured information. Decide one from set of predefined intents. Extract entities: types and values.
(Ref: Building Conversational AI w Rasa Stack - Alan Nichol PyBay2018)

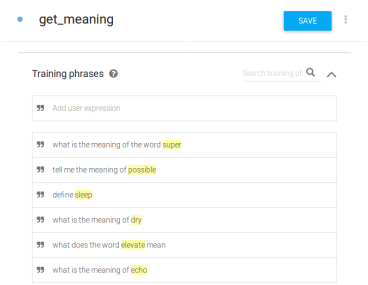
Intents

- Intents are the actions/categories of the sentences
- An Intent is a collection of expressions(what the user says) that mean the same thing but are constructed differently.
- Each intent corresponds to one action your user wants to perform.
- Consider it as the aim or target of the user input. If a user say, "Which day is today?", the intent would be finding the day of the week.
- For example, "I wish to book a flight from Mumbai to Pune on 27 March" has "flight-booking" as the intent



Intents

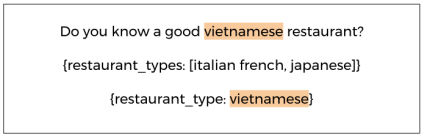
- For example, An intent “greet” will have the following expressions “hi,” hello,” hey,” all meaning the same thing as a greeting or conversation initiator.
- The following is an intent training done in dialogflow to get_meaning of a word.



(Ref: Chatbots 101 - Architecture & Terminologies - Bhavani Ravi)

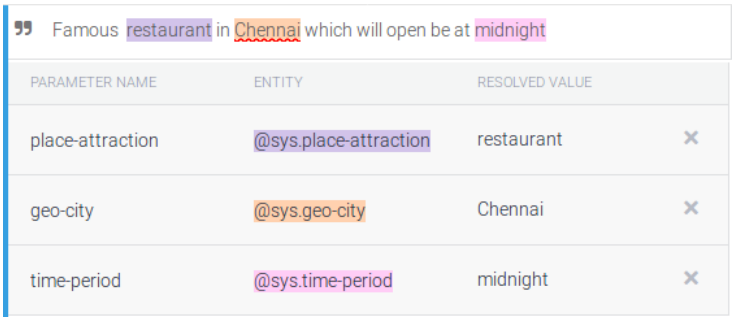
Entity

- Entities are the necessary variables needed to fulfill the actions.
- An Entity is an information extracted from what a user says. These are the details you want the bot engine to capture in order to perform the action.
- For the “flight-booking” intent, “Mumbai”,“Pune” and “27 March” as the entities.
- Consider it as the useful information from the user input that can be extracted.
- From previous example, by intent we understand the aim is to find the day of week, but of which date? If we extract “Today” as the entity, we can perform the action on today.



Entities

For example, When you look for a famous place in a city you would need to capture information like the kind of place, the city and the personal preferences like timings etc.,



(Ref: Chatbots 101 - Architecture & Terminologies - Bhavani Ravi)

NLU Training Data Formats

- There are two formats to sepcificy entities (type and value) in training sentences
- Such training file is collection of such annotated sentences.
- These annotations can be specified in either Json or Markdown format.

NLU Training Data: Markdown .md

```
## intent:greet
- hey
- hello there
- hi

## intent:mood_unhappy
- so sad. Only a picture of a [puppy](animal) could make it better.
- I am very sad. I need a [cat](animal) picture.
- Extremely sad. Only cute [doggo](animal) pics can make me feel better

## intent:mood_great
- perfect
- very good
- great

...
```

Entities in square brackets and intents in round brackets
(Ref: Building Conversational AI w Rasa Stack - Alan Nichol at PyBay2018)

NLU Training Data: Json

Sample: https://github.com/RASAHQ/rasa_nlu/blob/master/data/examples/rasa.json

```
{
        "text": "show me a mexican place in the centre",
        "intent": "restaurant_search",
        "entities": [
          {
            "start": 31,
            "end": 37,
            "value": "centre",
            "entity": "location"
          },
          {
            "start": 10,
            "end": 17,
            "value": "mexican",
            "entity": "cuisine"
          }
        ]
}
```

Entity and its value is specified using string index in the original “text”.

NLU Training Data

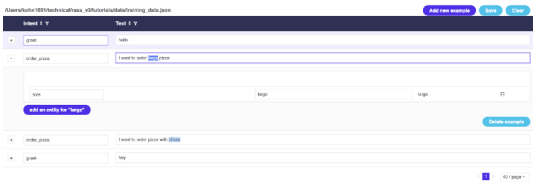
Following is the explanation of some of the fields mentioned in the json seen:

- text: the input sentence.
- intent: action or category, in this instance “restaurant_search”. This is typically the call-back function name.
- entities: array of entities. Here, there are two. One is of type ‘location’ with value as ‘centre’, whereas the other is of type ‘cuisine’ with value ‘mexican’. ‘start’ and ‘end’ specify beginning and ending indices of the word in the sentence.

You can use the below online tool as well, to generate this json file:
https://rasahq.github.io/rasa-nlu-trainer/

NLU Trainer Tool

- There is a great tool (rasa_nlu_trainer) you can use to add new examples/Intents/entities.
- To install it, run in terminal: `npm i -g rasa-nlu-trainer` `(you'll need nodejs and npm for this)` . For other platforms, need to look around.
- Run: `rasa-nlu-trainer -v <path to the training data file>`
- Here is a screenshot of the trainer:



- There is one web version too!! https://rasahq.github.io/rasa-nlu-trainer/

NLU Training

- The training part generates an ML model when you feed the training data
- This training can be either by python program or by command line

NLU Training by Command Line

- Training:

```
$ python -m rasa_nlu.train \
    --config sample_configs/config_spacy.yml \
    --data data/examples/rasa/demo-rasa.json \
    --path projects
```

- Serving:

```
$ python -m rasa_nlu.server --path projects
```

- Here "projects" is the model name

Training NLU by Python Script

```
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
from rasa_nlu.model import Metadata, Interpreter

def train (data, config_file, model_dir):
    training_data = load_data(data)
    trainer = Trainer(config.load(config_file))
    trainer.train(training_data)
    model_directory = trainer.persist(model_dir,
     fixed_model_name = 'chat')

train('data/nlu_train.md', 'nlu_config.yml', 'models/nlu'
```

This will train the nlu model and save it at 'models/nlu/'.

Testing NLU model by Python Script

Lets try and test how is the model working.

```
interpreter = Interpreter.load('./models/nlu/default/chat')

def ask_question(text):
    print(interpreter.parse(text))

ask_question("Show me a mexican place in the centre")
```

Testing NLU model

The output looks something like this,

```
{'intent': {'name': 'query_days_in_month', 'confidence':
    0.6128492334410716}, 'entities': [{'start': 17, 'end':
    24, 'value': 'january', 'entity': 'month', 'confidence':
    0.5002208121139594, 'extractor': 'ner_crf'}],
    'intent_ranking': [{'name': 'query_days_in_month',
    'confidence': 0.6128492334410716}, {'name': 'bye',
    'confidence': 0.1951941314517159}, {'name': 'greet',
    'confidence': 0.19195663510721267}], 'text': 'How many
    days in January'}

{'intent': {'name': 'query_days_in_month', 'confidence':
    0.6105606961005596}, 'entities': [{'start': 17, 'end':
    22, 'value': 'march', 'entity': 'month', 'confidence':
    0.5002208121139594, 'extractor': 'ner_crf'}],
    'intent_ranking': [{'name': 'query_days_in_month',
    'confidence': 0.6105606961005596}, {'name': 'bye',
    'confidence': 0.20686031746694789}, {'name': 'greet',
    'confidence': 0.18257898643249237}], 'text': 'How many
    days in March'
```

Testing NLU model

- Here as you can see, the model was perfectly able to tag the user question to it's intent (check 'name' section under 'intent').
- It says, that for the first question, the intent was 'query_days_in_month' and the extracted entity was 'January' (check 'value'under 'entities').
- One cool thing is in the output of second question, even though we didn't provide this in example, it was perfectly able to guess the intent and even extract the entity 'march'.

Rasa NLU: Intents and Entities: Hands-On Exercise

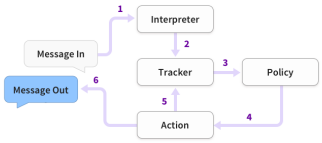
Rasa NLU: Intents and Entities: Hands-On Exercise

Over to notebook . . .

Rasa Core: Dialog Management

RASA Core: Actions and Stories

- The job of Rasa Core is to essentially generate the reply message for the chatbot.
- Stories define the sample interaction between the user and chatbot in terms of intent and action taken by the bot.
- It takes the output of Rasa NLU (intent and entities) and applies Machine Learning models to generate a reply.



- The input message is interpreted by an Interpreter to extract intent and entity.
- It is then passed to the Tracker that keeps track of the current state of the conversation.
- The Policy applies Machine Learning algorithm to determine what should be the reply and choses Action accordingly.
- Action updates the Tracker to reflect the current state.

Actions

- Actions are basically the operations performed by the bot either asking for some more details to get all the entities or integrating with some APIs or querying the database to get/save some information.
- It could be replying something in return, querying a database or any other thing possible by code.
- So, an Action is a task that you expect your bot to do.
- In most cases, An external API performs this action. Since the bot platforms do not support external API calls, An external program is used to drive that functionality.
- For example, When you ask your bot to order pizza for you, the bot extracts all the information(Entities) required to order pizza say, size, type, topping etc and sends it to an external API and gets a response whether the order is successful or not.

(Ref: Chatbots 101 - Architecture & Terminologies - Bhavani Ravi)

Actions

- Actions are either utterances, which means, the output that we want to send the user, or you can actually create your own class that inherent form Action.
- Say: "actions.ActionOrderPizza", here again Rasa has a very cool interface, you should inherent from Class Action and add your logic in the derived class according to what you need, here is an example, create the file action.py (just a stub below):

```
from rasa_core.actions import Action

class ActionOrderPizza(Action):
    def name(self):
        return 'action_order_pizza'
    def run(self):
        print('Ordering Pizza is completed! It should
     be with you soon :)')
        pass
```

Dialog Stories Data

- Rasa core's next state prediction is not flow chart based but machine learning based.
- It uses current context, intents, current state in the conversation, etc to decide next steps.
- You need to give some samples of interactions, in form of stories.
- These are a sample interaction between the user and bot, defined in terms of intents captured and actions performed.
- So developer can mention what to do if you get a use input of some intent with/without some entities.
- Like saying if user intent is to find the day of week and entity is today, find day of week of today and reply.

To make this work, Rasa need some files, which stores all the training and model information to build the bot.

Dialog Stories Data

Contains a bunch of stories to learn from. From each stories, creates a probability model of interactions.

```
## happy path
* greet
  - utter_greet
* mood_great
  - utter_happy
* mood_affirm
  - utter_happy
* mood_affirm
  - utter_goodbye

## sad path 1
* greet
  - utter_greet
* mood_unhappy
  - utter_ask_picture
* inform{"animal":"dog"}
  - action_retrieve_image
  - utter_did_that_help
* mood_affirm
  - utter_happy
```

- Simplest actions are utterances (template-ed fixed response strings)
- More flexible are the ones called "actions_" for API calls

(Ref: Building Conversational AI w Rasa Stack - Alan Nichol at PyBay2018)

Dialog Stories Data

Lets create a sample user-bot interaction.
stories.md

```
## story1
* greet
  - utter_greet
* query_days_in_month{"month":"January"}
  - utter_answer_31_days
* query_days_in_month{"month":"February"}
  - utter_answer_28_days
* query_days_in_month{"month":"April"}
  - utter_answer_30_days
* bye
  - utter_bye
```

- Here we are defining a sample interaction between the bot and user in form of a story.
- It goes like this, if user says something and its intent is 'greet' bot will perform action 'utter_greet'.
- One more example, if user's message has 'query_days_in_month' intent and 'February' then bot will perform 'utter_answer_28_days' action.

Training Dialog Data

Lets train the dialog management part,

```
def train_dialog(dialog_training_data_file, domain_file,
     path_to_model = 'models/dialogue'):
    logging.basicConfig(level='INFO')
    agent = Agent(domain_file,
              policies=[MemoizationPolicy(max_history=1)])
    training_data =
     agent.load_data(dialog_training_data_file)
    agent.train(
        training_data,
        augmentation_factor=50,
        epochs=200,
        batch_size=10,
        validation_split=0.2)
    agent.persist(path_to_model)

train_dialog('data/stories.md', 'domain.yml')
```

Training Policy

- Here, you can change the training policy in which you can define your own LSTM or RNN for dialog training.
- One important point, 'max_history' is used to define how one action is dependent on previous questions.
- If max_history is 1, then it just memorizes individual intent and its related actions. Due to lack of training data (huge load of stories), I am just making it memorize the rules now, you can create some sample stories and see the true potential of rasa dialog management.

Testing

Lets see if the bot is able to reply back to us and answers our question.

```
# Loading the Agent
rasaNLU = RasaNLUInterpreter("models/nlu/default/chat")
agent = Agent.load("models/dialogue", interpreter= rasaNLU)

# asking question
agent.handle_message('Hi')
>>> [{'recipient_id': 'default', 'text': 'Hey'}]

# once more
agent.handle_message('How many days in January')
>>> [{'recipient_id': 'default', 'text': 'There are 31 days
     in the mentioned month.'}]

# once more
agent.handle_message('How many days in April')
>>> [{'recipient_id': 'default', 'text': 'There are 30 days
     in the mentioned month.'}]

# once more
agent.handle_message('Bye')
>>> [{'recipient_id': 'default', 'text': 'Goodbye'}]
```

And it does!

Observations

- Mapping every combination of intent-entity to its action in stories is quite inefficient. Just think of individually mapping each month to its answer. This could be handled by using custom actions, where the action calls a python function with all the info like the intent and entity.
- The true potential of the rasa nlu and core shines when you give it more data to train.

Online Training the Dialog

Can stories be created by saving online interactions?
online_train.py:

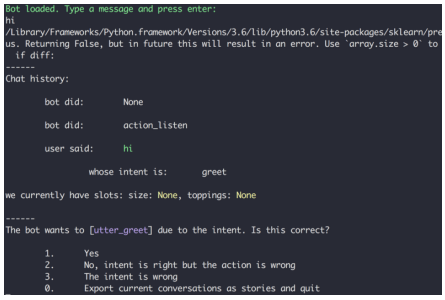
```
def run_online_trainer(input_channel,
                       interpreter,
                       domain_def_file='chat_domain.yml',

    training_data_file='./data/stories.md'):
    agent = Agent(  domain_def_file,
                    policies=[KerasPolicy(),
    MemoizationPolicy()],
                    interpreter=interpreter)
    agent.train_online( training_data_file,
                        input_channel=input_channel,
                        max_history=2,
                        batch_size=500,
                        epochs=200,
                        max_training_samples=300)
    return agent
if __name__ == '__main__':
    logging.basicConfig(level='INFO')
    interpreter =
     RasaNLUInterpreter('./models/nlu/default/chat')
    run_online_trainer(ConsoleInputChannel(), interpreter)
```

Run by `python online_train.py`

Online Training the Dialog

You should get an interactive session in which you should follow the expected flow, for example:



- In the first green line it says that the bot has been loaded and is waiting for the user input.
- Next we type: "hi"
- Now you can see that the system prints the "Chat history":
  - we can see that the flow is correct:
  - bot did: action_listen
  - user said: hi
  - whose intent is: greet

Online Training the Dialog

- Now the system is waiting for your feedback, in this case the flow is correct, so we should type: 2.
- You can continue doing this as long as you want, when you finish type "0" to export the new stories.md file (give it new name, such: "stories_v1.md").
- Now replace the previous stories file (back it up first!) or concatenate them in the same stories file, that way you should have more and more training data!

Rasa Core: Dialog Management: Hands-On Exercise

Rasa Core: Dialog Management: Hands-On Exercise

Over to notebook . . .

Config Files

Config file

The training configuration is defined as below . It contains two main info language of your bot and the NLP library to use. From Rasa 1 onwards there is no different policies.yml. That info is merged into config file itself nlu_config.yml

```
# Configuration for Rasa NLU.
language: "en"
### pipeline: supervised_embeddings or
    pretrained_embeddings_spacy
pipeline:
- name: "nlp_spacy"
- name: "tokenizer_spacy"
- name: "intent_entity_featurizer_regex"
- name: "intent_featurizer_spacy"
- name: "ner_spacy"
- name: "ner_crf"
- name: "ner_synonyms"
- name: "intent_classifier_sklearn"

# Configuration for Rasa Core.
policies:
  - name: MemoizationPolicy
  - name: KerasPolicy
  - name: MappingPolicy
```

Pipeline can be specified as combo or explicit.

Choosing a Pipeline

- If you have less than 1000 total training examples, and there is a spaCy model for your language, use the pretrained_embeddings_spacy pipeline else use the supervised_embeddings pipeline.
- pretrained_embeddings_spacy pipeline: uses pre-trained word vectors from either GloVe or fastText.
- supervised_embeddings pipeline: fits these specifically for your dataset.
- The advantage of the pretrained_embeddings_spacy pipeline that word similarity is already in place.
- The advantage of the supervised_embeddings pipeline is that your word vectors will be customised for your domain and not generic similarity.

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

Explicit Pipeline

supervised_embeddings is equivalent to

```
language: "en"

pipeline:
- name: "WhitespaceTokenizer"
- name: "RegexFeaturizer"
- name: "CRFEntityExtractor"
- name: "EntitySynonymMapper"
- name: "CountVectorsFeaturizer"
- name: "CountVectorsFeaturizer"
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: "EmbeddingIntentClassifier"
```

You can customize, only care to be taken is that order is important. NER can not come before tokernizer!!

Domain file

- Here you list all of the intents, entities, actions and similar information.
- You can also add sample bot reply templates and use them as actions.

```
intents:
- greet
- goodbye
- mood_great
- mood_unhappy

entities:
- animal

actions:
- utter_greet
- utter_did_that_help
- utter_happy
- utter_goodbye
- mymodule.RetrieveImage
```

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

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

  utter_happy:
  - text: "Great carry on!"

  utter_goodbye:
  - text: "Bye"
```

Custom action shown above is to be written in a separate file (elaborated later)

(Ref: Building Conversational AI w Rasa Stack - Alan Nichol at PyBay2018)

Domain file

Define domain file which contains sample templates to reply back to user.
domain.yml

```
intents:
# place your intents
 - greet
 - query_days_in_month
 - bye
entities:
# place your entities
 - month
templates:
# sample replies
 utter_greet:
  - "Hey, how can I help you?"
 utter_answer_31_days:
  - "There are 31 days in the mentioned month."
 utter_answer_30_days:
  - "There are 30 days in the mentioned month."
 utter_answer_28_days:
  - "There are 28 days in the mentioned month."
 utter_bye:
  - "Goodbye"
```

Domain file

domain.yml (continued)

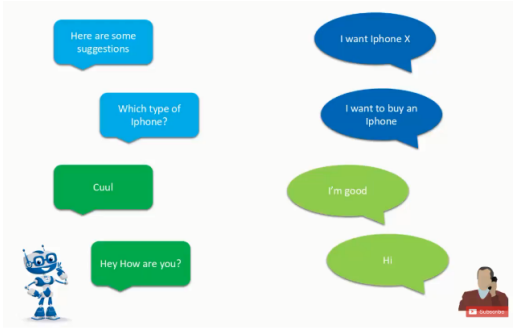
```
actions:
# templates (as they are reply actions),
# also custom actions if any
 - utter_greet
 - utter_answer_31_days
 - utter_answer_30_days
 - utter_answer_28_days
 - utter_bye
```

- Here we have listed down the intents and entities (all present in nlu training file), along with some templates and actions.
- Templates contains replies that you want the bot to make, in this case I want the bot to greet, say goodbye and also answer the user asked question of number of days in the month.
- And right now bot will only answer these three types of month answers (bot is agnostic of leap years, stupid bot)

Sample Conversation flow

Chatbot Workflow

Say, you want interaction like below, (go from bottom to top, alternate, right, left)



How do you build this interaction flow?

Stories, Paths

Happy path and Sad path

```
stories_md = """
## happy path                <!-- name of the story - just for debugging -->
* greet
  - utter_greet
* mood_great                <!-- user utterance, in format intent[entities] -->
  - utter_happy
* mood_affirm
  - utter_happy
* mood_affirm
  - utter_goodbye


## sad path 1                <!-- this is already the start of the next story -->
* greet
  - utter_greet             <!-- action the bot should execute -->
* mood_unhappy
  - utter_cheer_up
  - utter_did_that_help
* mood_affirm
  - utter_happy
```

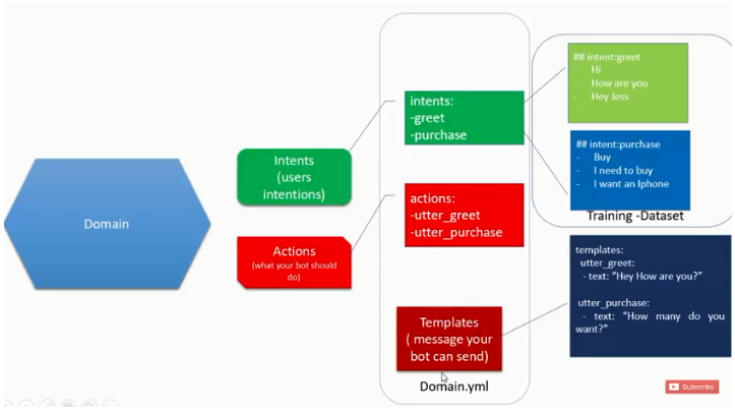
Stories, Paths

Purchase path

```
## good purchase path              <!-- name of the story - for making purchase
* greet
  - utter_greet
* mood_great                <!-- user utterance, in format intent[entities] -->
  - utter_happy
* mood_affirm
  - utter_happy
* purchase                  <!-- user intention to purchase things --->
  - utter_purchase
  - utter_working_on_it
* purchase_affirm           <!-- user utterance, in format intent[entities] -->
  - utter_purchase
* purchase_affirm
  - utter_price
  - utter_working_on_it
* mood_affirm
  - utter_goodbye
```

Domain

Domain contains everything



Exercises

Hands-On : https://github.com/RasaHQ/workshop-rasax/blob/master/exercis

What's Next?
How these modules work?
Can dig deep, to find that these are all Machine Learning techniques!!
Can you guess a few?

Want to give it a try ...
Hands-On: Chatbot in 3 files

```
mkdir firstbot
cd firstbot
activate rasa_env
rasa init  --no-prompt
```

- Creates project structure and dummy files.
- Good to run and execute
- You can run as is or add your data to nlu.md, stories.md and domain.yml; retrain and run.
- If you get any warnings replace "rasa" with "python -W ignore -m rasa "

(Ref: How to build awesome Rasa chatbot for a web - Martin Novak)

NLU

- Your chatbot needs to understand what your users are saying,
- Give alternative examples to learn from that identify your users intent.
- For example, you teach your bot that "Hi" and "Hello" represent an intent greeting.
- Do this in data/nlu.md

```
## intent:greeting
- hi
- hello
```

Dialog

- Also need to teach your bot how to respond to these intents in varying sequences that in Rasa are called stories
- Store that in data/stories.md
- Stars represent intents identified by Natural Language Understanding based on your data/nlu.md
- Dashes represent responses (called utterances) provided by your chatbot and defined as templates in domain.yml

```
## first story
* greeting
  - utter_greet_back
* farewell
  - utter_farewell
```

Domain
domain.yml is everything that you need for a basic chatbot.

```
intents:
- greeting
- farewell
templates:
  utter_greet_back
  - text: Hi there!
  - text: Hello, it is a pleasure to meet you.
  utter_farewell
  - text: Good bye, my friend.
actions:
- utter_greet_back
- utter_farewell
```

Re-train

```
rasa train
```

Run

```
rasa shell --quiet --enable-api --log-file out.log --cors *
```

Ask: "Are you a bot?"

Sample Chat

```
Bot loaded. Type a message and press enter (use '/stop' to
    exit):

Your input ->  hi

Hey! How are you?

Your input ->  perfect

Great, carry on!

Your input ->  are you a bot?

I am a bot, powered by Rasa.

Your input ->  bye

Bye

Your input ->  /stop
```

The End

Conclusions

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 . . .

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