Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 7

Dialog Engines

Nicola Capuano and Antonio Greco

DIEM – University of Salerno

Outline

- Task Oriented Dialogue Systems
- Introduction to Rasa
- Building a Chatbot with Rasa
- Custom Actions

Task Oriented Dialogue Systems

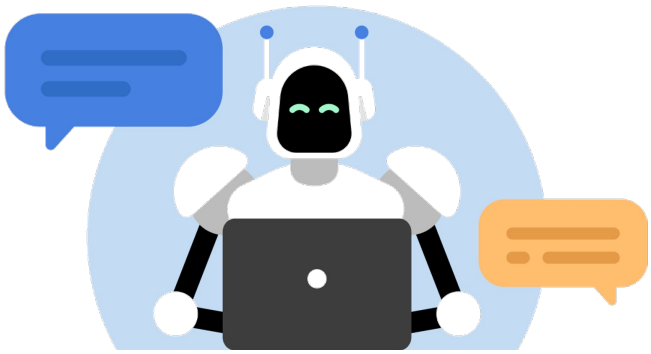
Types of Conversational AI

Chit-Chat

- No specific goal
- Focus on generating natural responses
- The more conversational turns the better

Task-Oriented Dialogue Systems (TOD)

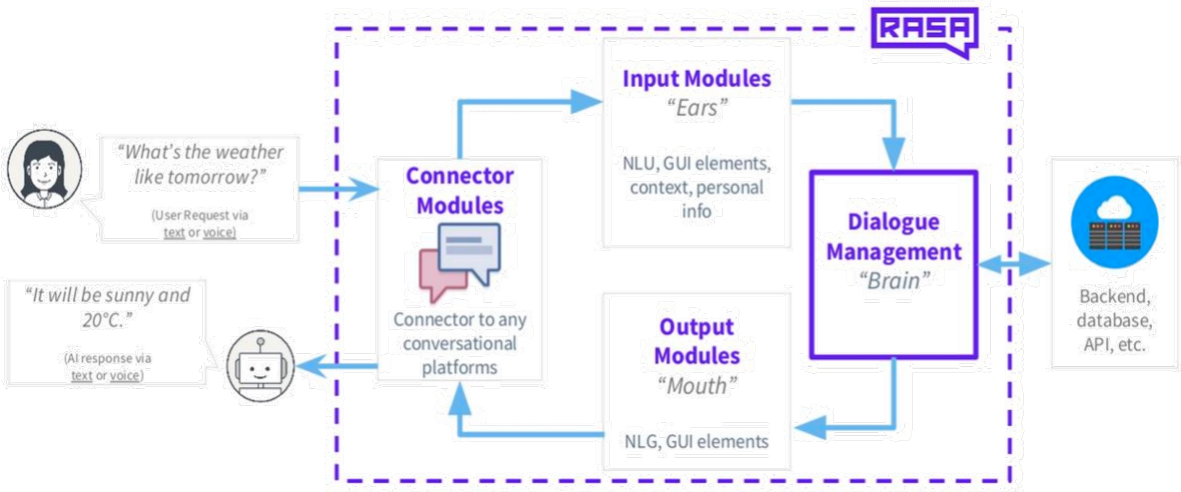
- Help users achieve their goal
- Focus on understanding users, tracking **states**, and generating next actions
- The less conversational turns the better



Task Oriented Dialogue

- **I have a question**
  - *Which room is the dialogue tutorial in?*
  - *When is the IJCNLP 2017 conference?*
- **I need to get this done**
  - *Book me the flight from Seattle to Taipei*
  - *Schedule a meeting with Bill at 10:00 tomorrow*
- **I need a recommendation**
  - *Can you suggest me a restaurant?*
  - *Can you suggest me something to see near me?*

TOD System Architecture



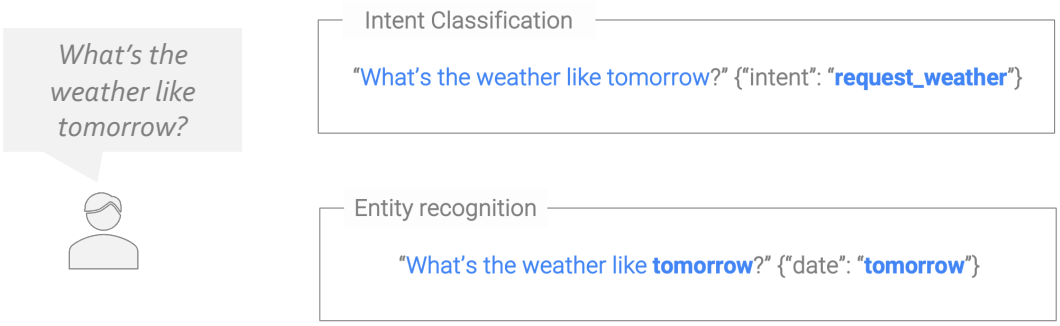
Rasa is a popular framework for building **TOD systems**

https://rasa.com/

Natural Language Understanding

Two main tasks:

- **Intent Classification**
  - Approached as a **multi-label sentence classification task**
- **Entity recognition**
  - Approached with **NER (rule-based or ML-based)**



Conversation design

Planning the **types of conversations** your assistant will be able to have

- Asking who your **users** are
- Understanding the **assistant's purpose**
- Documenting the **most typical conversations** users will have with the assistant

It's **difficult to anticipate everything** users might ask

- You should only rely on **hypothetical conversations** in the early stages of development
- Then train your assistant on **real conversations** as soon as possible

Introduction to

Rasa Intro

Open-Source **Conversational Framework**

- Launched in **2016**
- Used globally to create **thousands of bots** in various languages

Rasa Basic Units

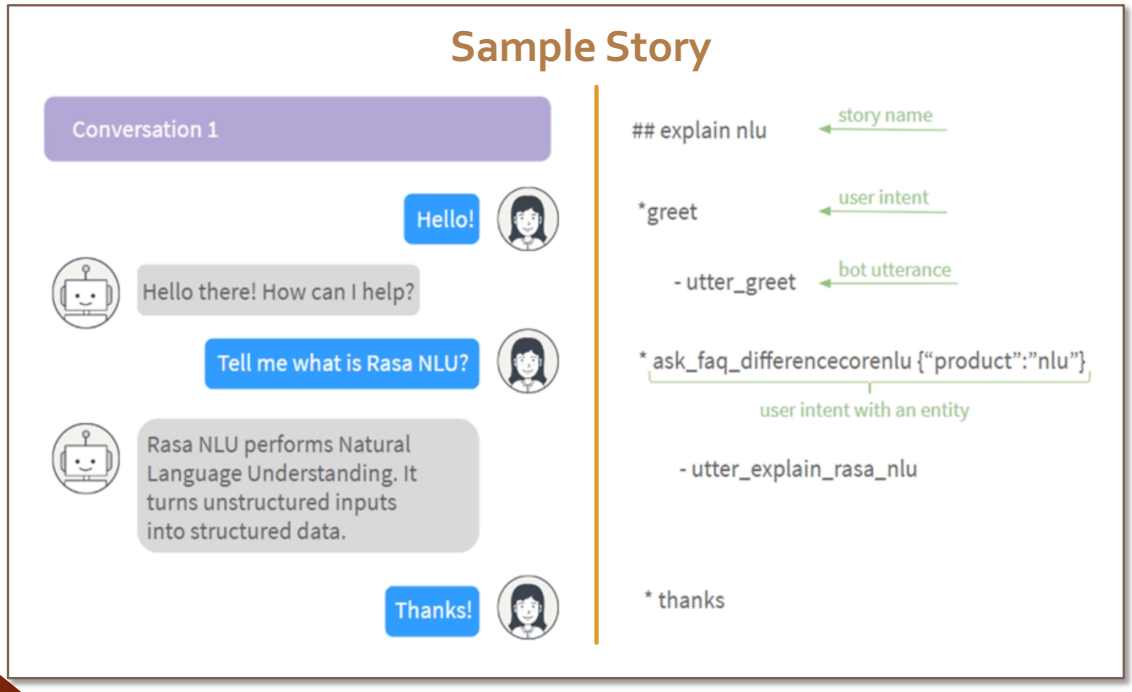
- **Intents:** what the user **wants to achieve**
- **Entities:** **terms or objects** that are relevant/necessary for the intent

Rasa Intro

Rasa Basic Units (continued)

- **Actions:** what should the bot do in **response to intents**
  - **Responses:** predefined **utterances**
  - **Complex Actions:** custom **Python code** interacting with other systems (e.g., databases, Web APIs)
- **Slots:** **variables** used to store information extracted from user inputs during a conversation
- **Forms:** sets of slots used to collect **multiple pieces of information** from the user in a structured manner
- **Stories:** sequences of user intents and bot actions used to pre-program **dialog scenarios**

Rasa Intro



Installing Rasa

Create and activate a new **virtual environment**:

- `python -m venv rasa.env`
- `source rasa.env/bin/activate`

Install Rasa

- `pip install rasa`

Rasa Project

- Most information is stored in **YAML files**
- **Python code** needed to program complex actions

Create a New Project

rasa init

...

```
Welcome to Rasa! 🤖

To get started quickly, an initial project will be created.
If you need some help, check out the documentation at https://rasa.com/docs/rasa.
Now let's start! 👇🟫

? Please enter a path where the project will be created [default: current directory] myBot
? Path 'myBot' does not exist 🧐. Create path? Yes
```

...

```
Created project directory at '/Users/niccap/Sorgenti/VSCode/NLP/Rasa/myBot'.
Finished creating project structure.
? ? Do you want to train an initial model? 💪🟫 Yes
Training an initial model...
```

...

```
Your Rasa model is trained and saved at 'models/20240808-163752-ornery-notch.tar.gz'.
? Do you want to speak to the trained assistant on the command line? 🤖 Yes
```

Create a New Project

rasa init 

```
...

Welcome to Rasa! 🤖

To get started quickly, an initial project will be created.
If you need some help, check out the documentation at https://rasa.com/docs/rasa.
Now let's start! 👇

? Please enter a path where the project will be created [default: current directory] myBot
? Path 'myBot' does not exist 🤔. Create path? Yes

...

Created project
Finished creat
? ? Do you want
Training an in

...

Your Rasa mode
? Do you want
```

```
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input ->  hello
Hey! How are you?
Your input ->  I'm very sad today
Here is something to cheer you up:
Image: https://i.imgur.com/nGF1K8f.jpg
Did that help you?
Your input ->  yes, thanks
Great, carry on!
Your input ->  bye
Bye
Your input ->  /stop
```

Directory Structure

- `actions` — Includes Python code for **custom actions**
- `data`
  - `nlu.yml` — Defines **intents** and **entities**
  - `rules.yml` — Defines **short conversation paths** that should always be followed
  - `stories.yml` — Defines general **stories** to train the model
- `models` — Includes the **trained models**
- `tests` — Includes Bot **test cases**
- `config.yml` — Defines **pipelines**, **policies**, and **components**
- `credentials.yml` — Credentials for **external platforms**
- `domain.yml` — **The main file**: lists all **intents**, **entities**, **slots**, **responses**, **forms**, and **actions**
- `endpoints.yml` — Lists the **endpoints** your Bot can use

domain.yml

```yaml
intents:
  - greet
  - goodbye
  - affirm
  - deny
  - mood_great
  - mood_unhappy
  - bot_challenge

responses:
  utter_greet:
  - text: "Hey! How are you?"

  utter_cheer_up:
  - text: "Here is something to cheer you up:"
    image: "https://i.imgur.com/nGF1K8f.jpg"

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

  utter_happy:
  - text: "Great, carry on!"

  utter_goodbye:
  - text: "Bye"

  utter_iamabot:
  - text: "I am a bot, powered by Rasa."

session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: true
```

minutes

If the user starts a new interaction after the previous session expires, the **data** from the previous session **must be transferred** to the new one

nlu.yml

```yaml
- intent: greet
  examples: |
    - hey
    - hello
    - hi
    - hello there
    - good morning
    - good evening
    - moin
    - hey there
    - let's go
    - hey dude
    - goodmorning
    - goodevening
    - good afternoon

- intent: goodbye
  examples: |
    - cu
    - good by
    - cee you later
    - good night
    - bye
    - goodbye
    - have a nice day
    - see you around
    - bye bye
    - see you later

- intent: bot_challenge
  examples: |
    - are you a bot?
    - are you a human?
    - am I talking to a bot?
    - am I talking to a human?

- intent: affirm
  examples: |
    - yes
    - y
    - indeed
    - of course
    - that sounds good
    - correct

- intent: deny
  examples: |
    - no
    - n
    - never
    - I don't think so
    - don't like that
    - no way
    - not really
```

...

nlu.yml

```yaml
- intent: mood_great
  examples: |
    - perfect
    - great
    - amazing
    - feeling like a king
    - wonderful
    - I am feeling very good
    - I am great
    - I am amazing
    - I am going to save the world
    - super stoked
    - extremely good
    - so so perfect
    - so good
    - so perfect
```

```yaml
- intent: mood_unhappy
  examples: |
    - my day was horrible
    - I am sad
    - I don't feel very well
    - I am disappointed
    - super sad
    - I'm so sad
    - sad
    - very sad
    - unhappy
    - not good
    - not very good
    - extremly sad
    - so saad
    - so sad
```

Note: To train the model to recognize intents, RASA needs at least **7-10 utterances per intent**

stories.yml

```yaml
- story: sad path 1
  steps:
  - intent: greet
  - action: utter_greet
  - intent: mood_unhappy
  - action: utter_cheer_up
  - action: utter_did_that_help
  - intent: affirm
  - action: utter_happy
```

```yaml
- story: sad path 2
  steps:
  - intent: greet
  - action: utter_greet
  - intent: mood_unhappy
  - action: utter_cheer_up
  - action: utter_did_that_help
  - intent: deny
  - action: utter_goodbye
```

```yaml
- story: happy path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: mood_great
  - action: utter_happy
```

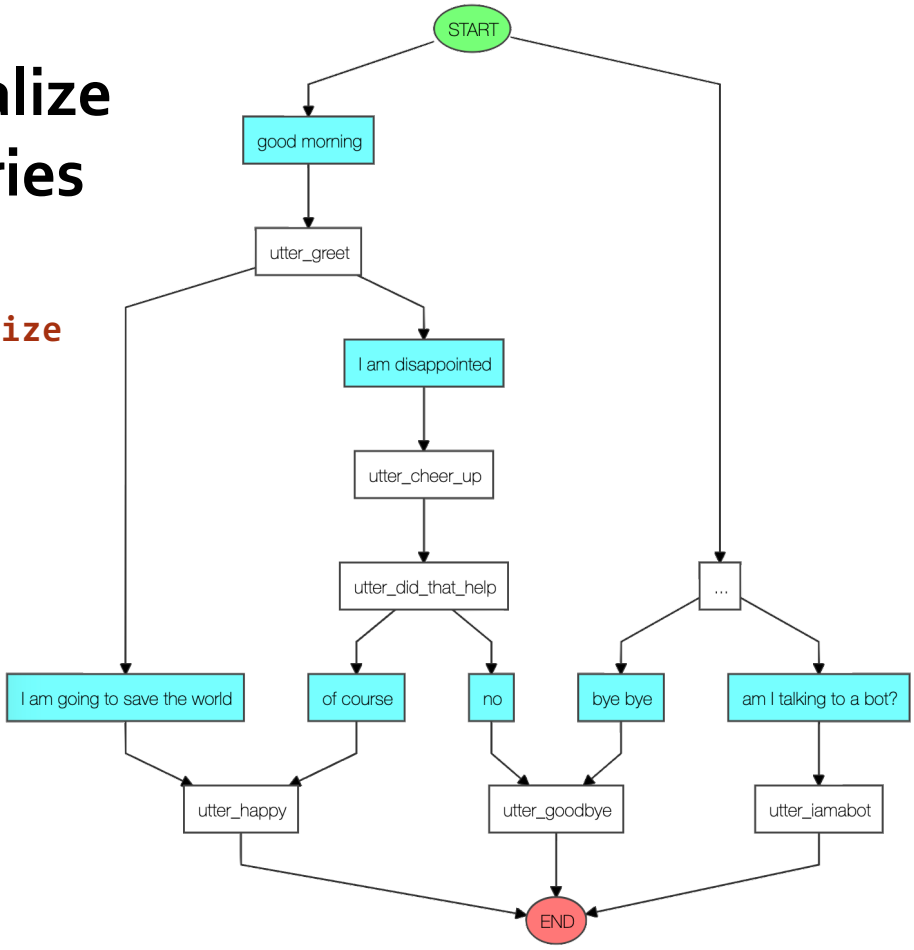
rules.yml

```yaml
- rule: Say goodbye anytime the user says goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye

- rule: Say 'I am a bot' anytime the user challenges
  steps:
  - intent: bot_challenge
  - action: utter_iamabot
```

Visualize Stories

`rasa visualize`



Other Commands

- `rasa train`

  Trains a model using your NLU data and stories, saves trained model in **./models**

- `rasa shell`

  Loads your trained model and lets you **talk to your assistant on the command line**

- `rasa run`

  Starts a server with your trained model

  ```
  INFO     root  - Starting Rasa server on http://0.0.0.0:5005
  ```

  For **cross origin calls** use: `rasa run --cors "*"`

- `rasa -h` (help)

Other Commands

- **`rasa train`**

```
usage: rasa [-h] [--version] {init,run,shell,train,interactive,telemetry,test,visualize,data,export,x,evaluate} ...

Rasa command line interface. Rasa allows you to build your own conversational assistants 🤖. The 'rasa' command allows you
to easily run most
common commands like creating a new bot, training or evaluating models.

positional arguments:
  {init,run,shell,train,interactive,telemetry,test,visualize,data,export,x,evaluate}
                        Rasa commands
    init                Creates a new project, with example training data, actions, and config files.
    run                 Starts a Rasa server with your trained model.
    shell               Loads your trained model and lets you talk to your assistant on the command line.
    train               Trains a Rasa model using your NLU data and stories.
    interactive         Starts an interactive learning session to create new training data for a Rasa model by chatting.
    telemetry           Configuration of Rasa Open Source telemetry reporting.
    test                Tests Rasa models using your test NLU data and stories.
    visualize           Visualize stories.
    data                Utils for the Rasa training files.
    export              Export conversations using an event broker.
    x                   Run a Rasa server in a mode that enables connecting to Rasa Enterprise as the config endpoint.
    evaluate            Tools for evaluating models.

options:
  -h, --help            show this help message and exit
  --version             Print installed Rasa version
```

- **`rasa -h`** (help)

Rasa REST API

Rasa will provide you with a **REST endpoint**

- You can post messages and receive the Bot's responses from **external systems** (e.g., Web apps)
- **Add the REST channel** to your **`credentials.yml`**:

```
rest:
#  # you don't need to provide anything here - this channel doesn't
#  # require any credentials
```

- After restarting your Rasa server, you can reach the bot at: **http://<host>:<port>/webhooks/rest/webhook**
- Documentation: **https://rasa.com/docs/rasa/connectors/your-own-website/**

Request and Response Format

You can **POST JSON requests** with the following format:

```
{
  "sender": "test_user", // sender ID
  "message": "I'm sad!"
}
```

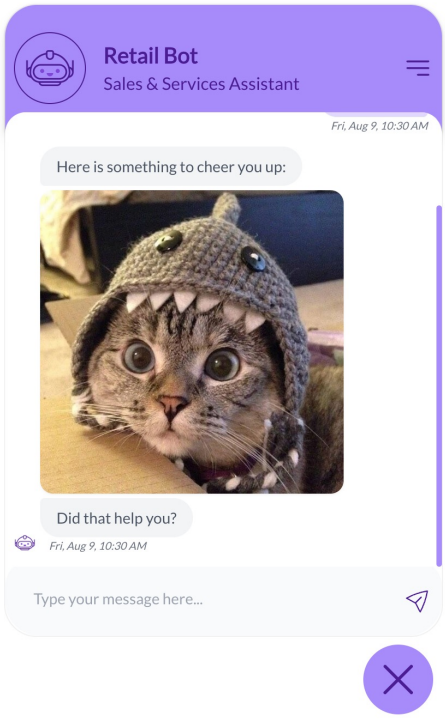
The **response** from Rasa will be a **JSON body of bot responses**, for example:

```
[
    {
        "recipient_id": "test_user",
        "text": "Here is something to cheer you up:"
    },
    {
        "recipient_id": "test_user",
        "image": "https://i.imgur.com/nGF1K8f.jpg"
    },
    {
        "recipient_id": "test_user",
        "text": "Did that help you?"
    }
]
```

Web-based Frontends

You can integrate a Rasa bot into **your website**

- **Custom Implementation**
  - Build your own frontend using HTML/CSS/JavaScript
- **Use a pre-built solution**
  - **Rasa Widget** (React-based)
  - Clone from: https://github.com/JiteshGaikwad/Chatbot-Widget/tree/Widget2.0
  - Copy the `./dist` files to your web project to use the widget



More on Connectors

- You can enable **authentication**
- You can use **web-socket** for real-time interaction
- Rasa provides **built-in connectors** for:
  - Facebook Messenger
  - Slack
  - Telegram
  - Twilio
  - Microsoft Bot Framework
  - Cisco Webex Teams
  - RocketChat
  - Mattermost
  - Google Hangouts Chat

https://rasa.com/docs/rasa/messaging-and-voice-channels/

Building a Chatbot with

Domain File

Defines **everything your assistant knows:**

- **Responses:** These are the things the assistant can say to users
- **Intents:** These are categories of things users say
- **Entities:** These are pieces of information extracted from incoming text
- **Slots:** These are variables remembered over the course of a conversation
- **Actions:** These add application logic and extend what your assistant can do

Domain File

Basic responses:

```
responses:
 utter_greet:
   - text: "Hey there!"
 utter_goodbye:
   - text: "Goodbye :("
 utter_default:
   - text: "Sorry, I didn't get that, can you rephrase?"
 utter_youarewelcome:
   - text: "You're very welcome."
 utter_iamabot:
   - text: "I am a bot, powered by Rasa."
```

Domain File

Multiple responses: when triggered, one of this responses will be randomly selected

```
responses:
 utter_greet:
 - text: "Hey, {name}. How are you?"
 - text: "Hey, {name}. How is your day going?"
```

Slots: `{name}` will be filled with the value of the **name** slot ("None" until it's filled)

Domain File

Responses: buttons and images

```
responses:
 utter_greet:
 - text: "Hey! How are you?"
   buttons:
   - title: "great"
     payload: "/mood_great"
   - title: "super sad"
     payload: "/mood_sad"
 utter_cheer_up:
 - text: "Here is something to cheer you up:"
   image: "https://i.imgur.com/nGF1K8f.jpg"
```

Buttons and corresponding intents

Text + image

Domain File

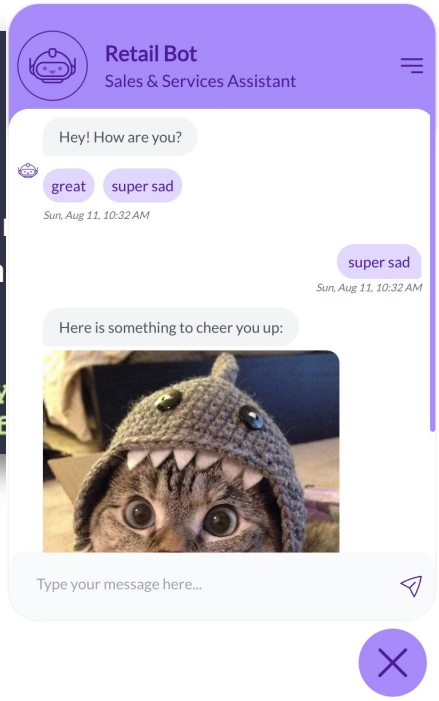
Responses: buttons and images

```
responses:
 utter_greet:
 - text: "Hey! How are you?"
   buttons:
   - title: "great"
     payload: "/mood_great"
   - title: "super sad"
     payload: "/mood_sad"
 utter_cheer_up:
 - text: "Here is something to cheer y
   image: "https://i.imgur.com/nGF1K8f
```

Buttons ar
correspon

Text + image

Rendering depends on the chosen **channel**



Domain File

List of intents

```
intents:
 - affirm
 - deny
 - greet
 - thankyou
 - goodbye
 - search_concerts
 - search_venues
 - compare_reviews
 - bot_challenge
 - how_to_get_started
```

- Must correspond to the intents on the **NLU file**
- **Tip:** Start with the fewest intents possible

  You can **add or change intents at any time**

Domain File

List of entities

- Can be **numbers**, **dates**, **country names**, **product names**, ...
- **Standard entities** can be extracted with **pre-built models**

```
entities:
   - PERSON           # entity extracted by SpacyEntityExtractor
   - time             # entity extracted by DucklingEntityExtractor
   - membership_type  # custom entity extracted by DIETClassifier
   - priority         # custom entity extracted by DIETClassifier
```

  - Specific **modules** must be included in the **config file**

- **Custom entities** can be extracted with **regular expressions**, **lookup tables** or **machine learning**
  - The **NLU file** will specify how

NLU File

Aimed at training the system to **extract structured information** from user messages

- This includes the **intents** and involved **entities**

Training data

- **Example user utterances categorized by intent**

Extra information

- **Regular Expressions:** patterns to capture specific types of entities
- **Lookup Tables:** comprehensive lists of possible values for entities
- **Synonyms:** Define synonyms for common terms to ensure that variations in user input are understood correctly

NLU File

Sample **lists of utterances** divided by **intent**

```
nlu:
  - intent: book_flight
    examples: |
      - I want to book a flight to [New York](city)
      - Book a flight from [Los Angeles](city) to [Chicago](city)
      - Can you help me book a flight to [San Francisco](city)?
      - I need a flight ticket to [Boston](city)
      - I'd like to fly from [Houston](city) to [Atlanta](city)

  - intent: check_flight_status
    examples: |
      - What is the status of flight [AA123](flight_number)?
      - Can you tell me if flight [UA456](flight_number) is delayed?
      - I want to check the status of flight number [DL789](flight_number)
      - Is flight [AA123](flight_number) on time?
```

```
entity: city
value: New York
```

Involved **entities: city** and **flight_number**

- must correspond to those listed in the **domain file**

NLU File

Sample **extra information**

```
- lookup_table: city
  examples: |
    - New York
    - Los Angeles
    - Chicago
    - San Francisco
    - Boston
    - Houston
    - Atlanta
```

```
- synonym:
    - synonym: flight
      examples: |
        - flight
        - flight ticket
        - plane ticket
        - air ticket
```

```
- regex:
    - name: flight_number
      pattern: "\\b[A-Z0-9]{2,5}\\b"  # Regex for flight numbers
```

Without lookup tables and regex, custom **entities** are simply recognized based on **machine learning**

NLU File

Entity Roles allow you to add more details to your entities

Example:

I am looking for a flight from New York to Boston.

entity:	location	entity:	location
value:	New York	value:	Boston
role:	origin	role:	destination

```
nlu:
- intent: book_a_flight
  examples: |
    - I am looking for a flight from [New York]{"entity":"location", "role":"origin"} to
[Boston]{"entity":"location", "role":"destination"}.
```

NLU File

Good practices:

- Start with the **smallest possible number of intents**
  - Most people want to do the **same thing**
  - Start with the **most common**
- Additional intents will come from **user data**
- Don't use intents to **store information** (use **entities** instead)

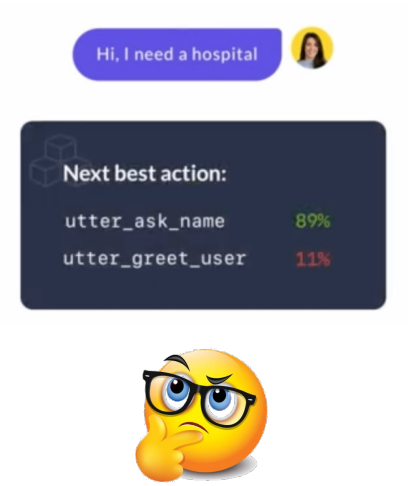
book_train:
- One train ticket
- Need to book a train ride
- A rail journey please

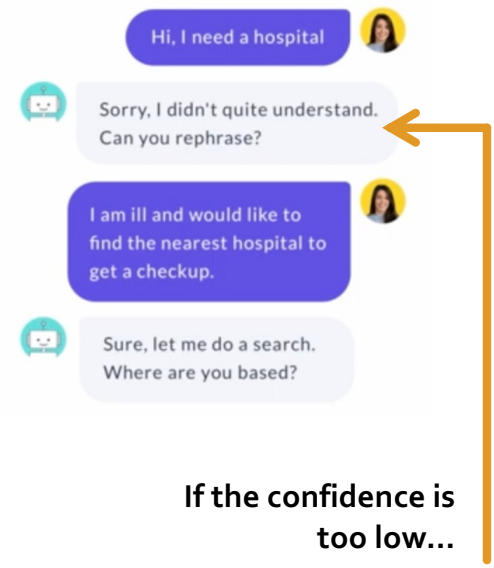
book_plane:
- One plane ticket
- Need to book a plane ride

make_booking:
- One train ticket
- Need to book a train ride
- A [rail](train) journey please
- One [plane](air) ticket
- Need to book a [plane](air) ride
- i'd like to book a trip
- Need a vacation

Stories File

Stories are **training data** to teach your assistant **what it should do next**





If the confidence is too low…

Stories File

```
stories:
   - story: happy path
     steps:
     - intent: greet
     - action: utter_greet
     - intent: moood_great
     - action: utter_happy
```

- Start with **common flows (happy paths)**
- Add common **errors/digressions**
- Use **interactive learning** to improve stories
- Once your model is in production add more data from **real user conversations**

Stories File

```
stories:
- story: newsletter signup with OR
  steps:
  - intent: signup_newsletter
  - action: utter_ask_confirm_signup
  - or:
    - intent: affirm
    - intent: thanks
  - action: action_signup_newsletter
```

OR Statements:

Same action for **different intents**

```
stories:
- story: beginning of conversation
  steps:
  - intent: greet
  - action: utter_greet
  - intent: goodbye
  - action: utter_goodbye
  - checkpoint: ask_feedback
```

Checkpoints:

Link to **other stories**

Rules File

Rules are a way to describe **short pieces of conversations** that **always go the same way**

- Not suitable for **multi-turn interactions**
- Not used to train ML algorithms but **applied as is**

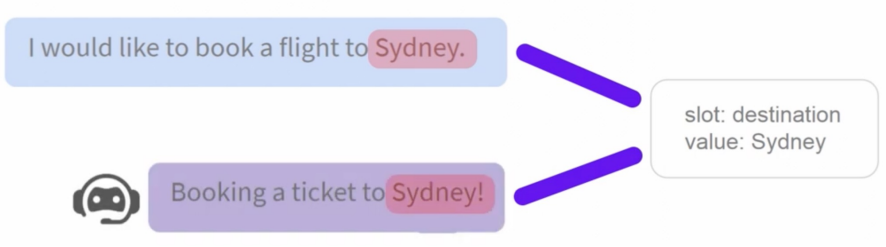
Rules file:

```
rules:
   - rule: Greeting Rule
     steps:
     - intent: greet
     - action: utter_greet
```

Slots

Slots are your **assistant's memory**

- Enable your assistant to **store important details** and **later use** them in a specific context



- Can be configured to **influence the flow of the conversation**

Slots and Entities

Slots are defined in the **domain file** and are usually connected to **entities**

Example:

```
entities:
- destination

slots:
  destination:
    type: text
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: destination
```

Type can be **text**, **boolean**, **categorical** **float**, **list**, **any**

This slot will **not influence** the conversation flow

This slot is filled with the value of the **destination entity** (if set)

Slot Mappings

Allow you to define **how each slot will be filled in**

- Are applied **after each user message**

Example:

```
entities:
- entity_name
slots:
  amount_of_money:
    type: any
    mappings:
    - type: from_entity
      entity: number
      intent: make_transaction
      not_intent: check_transaction
```

Send $200 to Ben.
entity: number

Intent: make_transaction
slot is set

entity: number
Did I receive the $1000 that Alice sent me yesterday?

Intent: check_transaction
slot is not set

Slot Mappings

Allow you to define **how each slot will be filled in**

- Are applied **after each user message**

Example:

```
entities:
- entity_name
slots:
  amount_of_money:
    type: any
    mappings:
    - type: from_entity
      entity: number
      intent: make_transaction
      not_intent: check_transaction
```

Mapping parameters:

- **intent:** only applies the mapping when this intent is predicted
- **not_intent:** does not apply the mapping when this intent is predicted
- **role:** only applies the mapping if the extracted entity has this role

Use Slots in Responses

You can create more **dynamic responses** by including slots in the responses

```
slots:
  name:
    type: any


responses:
  utter_greet:
    - text: “Hello {name}! How are you?”
    - text: “Hello there :)”
    - text: “Hi. How can I help you today?”
```

If **name** is not set, then its value will be **None**

Pipeline Configuration

The **config.yml** file defines the **NLU pipeline**, and the **dialog policies** used by Rasa

- **Language:** defines the **language** of the bot (e.g., en, fr, it).
- **Pipeline:** specifies the steps to process user messages (NLU pipeline) to **extract intents and entities**
- **Policies:** defines how the bot should handle dialogue and **predict next actions**.

```
language: en

pipeline: null
# The default pipeline is used to train your model.

policies: null
# The default policies are used to train your model.
```

NLU Pipeline

The pipeline defines the sequence of components that process user messages:

- **Tokenizers:** Break down the text into tokens (words, subwords)
- **Featurizers:** Convert tokens into numerical features that models can use
- **Classifiers:** Determine the user's intent
- **Entity Extractors:** Identify named entities (e.g., names, dates)

```
pipeline:
  - name: WhitespaceTokenizer
  - name: CountVectorsFeaturizer
  - name: DIETClassifier
    epochs: 150
  - name: EntitySynonymMapper
```

NLU Pipeline

Tokenizers:

- **WhitespaceTokenizer:** Splits text into tokens based on whitespace
- **SpacyTokenizer:** Leverages SpaCy's tokenization
- ...

Featurizers:

- **CountVectorsFeaturizer:** Converts text into a bag-of-words
- **ConveRTFeaturizer:** Uses pre-trained ConveRT embeddings specialized for conversational data
- **SpacyFeaturizer:** Leverages SpaCy's pre-trained word embeddings
- ...

NLU Pipeline

Classifiers:

- **DIETClassifier:** A multi-task transformer-based classifier for both intent classification and entity extraction
- **SklearnIntentClassifier:** Uses scikit-learn algorithms (e.g., SVM or logistic regression) for intent classification
- …

Entity extractors:

- **RegexEntityExtractor:** Extracts entities using regular expressions for pattern matching
- **SpacyEntityExtractor:** Uses SpaCy's pre-trained models to extract named entities based on SpaCy's NER system.
- …

Training Policies

Training policies are techniques your assistant uses to **decide on how to respond back to the user**

Policy priority defines how assistant makes decisions when multiple policies predict the next actin with the same accuracy

config.yml

```yaml
recipe: default.v1
language:  # your language
pipeline:
  # - <pipeline components>

policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
    max_history: 5
    epochs: 200
  - name: RulePolicy
```

Training Policies

- **Rule Policy:** assistant makes the decision on how to respond based on rules defined in **rules.yml**

```
rules:
- rule: Chitchat
  steps:
  - intent: chitchat
  - action: utter_chitchat
```

Training Policies

- **Memoization Policy:** assistant makes the decision on how to respond matching the real interaction with stories from **stories.yml**

Hello

Hey! How can I help you?

Can I check the balance of my card?


```
stories:
- story: check account balance
  steps:
  - intent: greet
  - action: utter_greet
  - intent: check_balance
  - action: utter_ask_userid
  - intent: inform
    entities:
      -  userid: “1234”
  - action: check_balance
  - intent: thanks
  - action: utter_goodbye
```

Training Policies

- **TED Policy**: assistant makes the decision on how to respond by **learning from the data** defined in **stories.yml**
  - **Transformer Embedding Dialogue (TED)** is a neural network architecture for next action prediction
  - **Max History:** how many conversational steps your assistant keeps in the memory when making the prediction
  - **Epochs:** the number of epochs used to train the model

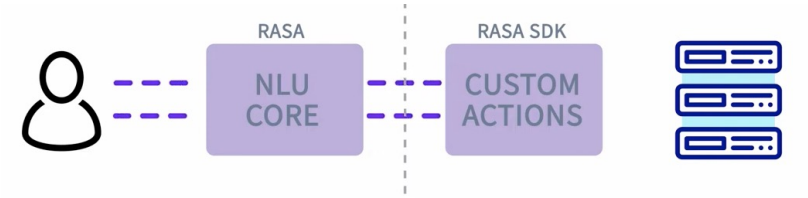
```
- name: TEDPolicy
  max_history: 5
  epochs: 200
```

Custom Actions

Custom Actions

What do we want virtual assistants to do?

- Send back appropriate message
- Send an email
- Make a calendar appointment
- Fetch relevant information from a database
- Check information from an API
- Calculate something specific



Example

nlu.yml

```yaml
- intent: inquire_time
  examples: |
    - what time is it?
    - what time is it in [Amsterdam](place)?
    - what time is it in [London](place)?
    - tell me the time in [Lisbon](place)
    - what is the current time in [Berlin](place)
    - what time is it in [amsterdam](place)[amsterdam](place)
- lookup: place
  examples: |
    - brussels
    - zagreb
    - london
    - lisbon
    - amsterdam
    - seattle
```

rules.yml

```yaml
- rule: Tell the time
  steps:
  - intent: inquire_time
  - action: action_tell_time
```

domain.yml

```yaml
intents:
- inquire_time

entities:
- place

actions:
- action_tell_time
```

config.yml

```yaml
language: en

pipeline:
  - name: WhitespaceTokenizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    entity_recognition: False
    epochs: 100
  - name: RegexEntityExtractor
    use_lookup_tables: True

policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
    max_history: 5
    epochs: 100
  - name: RulePolicy
```

Enables the use of lookup tables for entity recognition

The model must be then trained with…

rasa train

actions.py

```python
from typing import Any, Text, Dict, List

import arrow      # pip install arrow
import dateparser # pip install dateparser
from rasa_sdk import Action, Tracker
from rasa_sdk.events import SlotSet
from rasa_sdk.executor import CollectingDispatcher

city_db = {
    'brussels': 'Europe/Brussels',
    'zagreb': 'Europe/Zagreb',
    'london': 'Europe/Dublin',
    'lisbon': 'Europe/Lisbon',
    'amsterdam': 'Europe/Amsterdam',
    'rome': 'Europe/Rome',
    'seattle': 'US/Pacific'
}
```

actions.py

```python
class ActionTellTime(Action):

    def name(self) -> Text:
        return "action_tell_time" # returns the name of the action

    def run(self, dispatcher: CollectingDispatcher, # allows you to send messages back to the user
            tracker: Tracker, # provides the state of the conversation
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        current_place = next(tracker.get_latest_entity_values("place"), None)
        utc = arrow.utcnow()

        if not current_place:
            msg = f"It's {utc.format('HH:mm')} utc now. You can also give me a place."
            dispatcher.utter_message(text=msg) # sends a message back to the user
            return []

        tz_string = city_db.get(current_place, None)
        if not tz_string:
            msg = f"It's I didn't recognize {current_place}. Is it spelled correctly?"
            dispatcher.utter_message(text=msg) # sends a message back to the user
            return []

        msg = f"It's {utc.to(city_db[current_place]).format('HH:mm')} in {current_place} now."
        dispatcher.utter_message(text=msg) # sends a message back to the user

        return []
```

Action Server

To enable custom actions the action server endpoint must be set in **endpoints.yml**...

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

- Then the action server must be run: **rasa run actions**
- We can then try the model: **rasa shell**

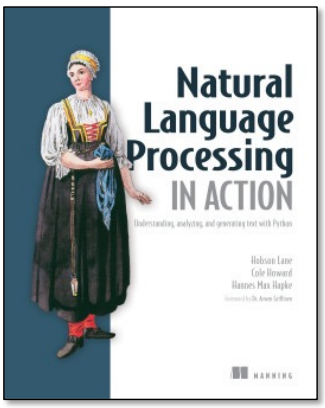
```
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input ->  hello
Hey! How are you?
Your input ->  what time is it?
It's 18:54 utc now. You can also give me a place.
Your input ->  what time is it in amsterdam?
It's 20:55 in amsterdam now.
Your input ->  what time is it in rome?
It's 20:55 in rome now.
```

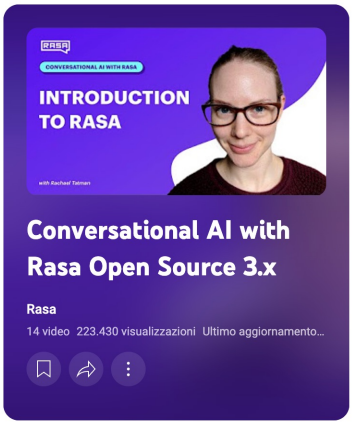
References

Natural Language Processing IN ACTION

Understanding, analyzing, and generating text with Python

Chapter **12**





14 Video Tutorial

https://www.youtube.com/playlist?list=PL75e0qA87dlEjGAc9j9v3a5h1mxI2Z9fi

Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 7

Dialog Engines

Nicola Capuano and Antonio Greco

DIEM – University of Salerno