Language Understanding Systems

The Rasa Dialogue Engine Tutorial

Evgeny A. Stepanov

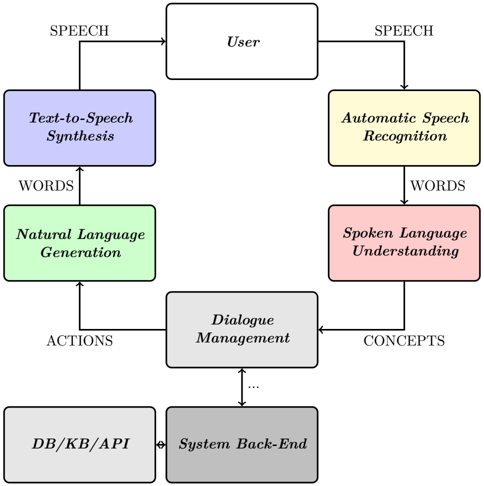
VUI, Inc. & SISL, DISI, UniTN
evgeny.stepanov@unitn.it

Outline

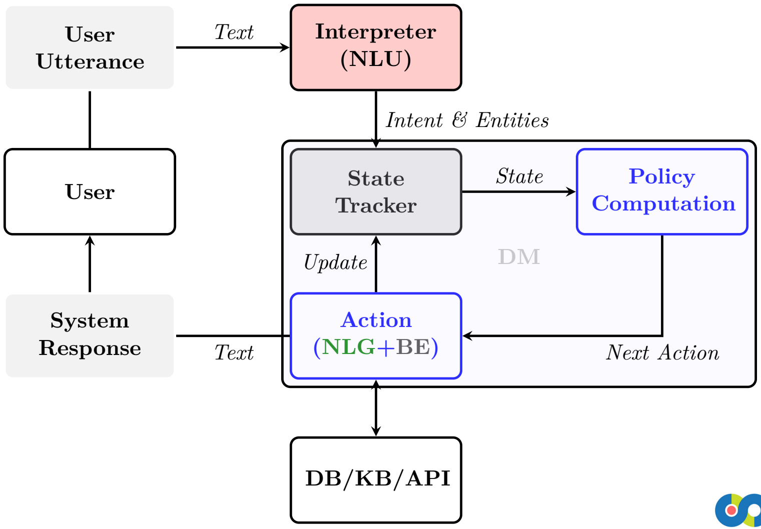
Section 1

General Overview

Spoken Dialogue System



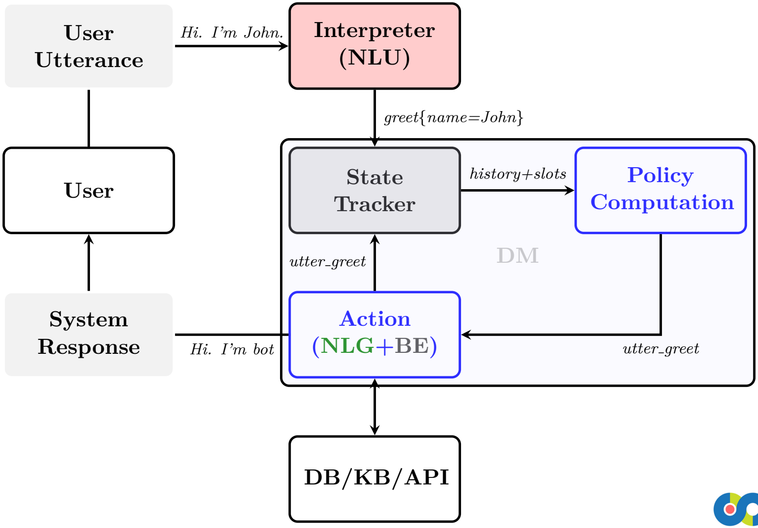
Rasa Overview



Rasa Overview: *Pipeline Steps*

1. The **message** is received and passed to an `Interpreter`, which converts it into a dictionary including the original **text**, the **intent**, and any **entities** that were found.
2. The `Tracker` is the object which keeps track of conversation **state**. It receives the info that a new message has come in.
3. The *policy* receives the current **state** of the tracker.
4. The *policy* chooses which **action** to take next.
5. The chosen *action* is logged by the tracker.
6. A **response** is sent to the user.

Rasa Overview



Section 2

Simple Bot

Rasa Packages

rasa_core	chatbot framework
rasa_nlu	NLU library
rasa_core_sdk	custom action sdk for rasa_core

Starter Pack Folder Contents

- data
  - stories.md
  - nlu_data.md
- actions.py
- domain.yml
- policies.yml
- endpoints.yml
- nlu_config.yml

Starter Pack Folder Contents

data	training data
stories.md	policy training data
nlu_data.md	NLU training data
actions.py	application action definitions
domain.yml	application domain definition
policies.yml	policy ensemble configuration
endpoints.yml	action endpoint configuration
nlu_config.yml	NLU pipeline configuration

Conversational Agent Building Steps

1. define an application domain
2. write stories for policy training
3. write utterances for NLU training
4. define system actions (if not default)
5. configure & train policy ensemble
6. configure & train NLU pipeline
7. configure action endpoints

Conversational Agent Building Steps

1. define an application domain
2. write stories for policy training
3. write utterances for NLU training
4. define system actions (if not default)
5. configure & train policy ensemble
6. configure & train NLU pipeline
7. configure action endpoints
8. start action server
9. start the agent

Conversational Agent Building Steps

1. define an application domain	domain.yml
2. write stories for policy training	stories.md
3. write utterances for NLU training	nlu_data.md
4. define system actions (if not default)	actions.py
5. configure & train policy ensemble	policies.yml
6. configure & train NLU pipeline	nlu_config.yml
7. configure action endpoints	endpoints.yml
8. start action server	
9. start the agent	

Building the Agent: Training Policy Ensemble

```
python -m rasa_core.train -d domain.yml \
                          -s data/stories.md \
                          -c policies.yml \
                          -o models/dialogue
```

Building the Agent: Training NLU Model(s)

```
python -m rasa_nlu.train -d data/nlu_data.md \
                         -c nlu_config.yml
                         -o models/nlu
```

Running the Agent: Starting Action Server

```
python -m rasa_core_sdk.endpoint --actions actions
```

Running the Agent: Starting the Agent

```
python -m rasa_core.run -d models/dialogue/ \
                        -u models/nlu/default/model_?/ \
                        --endpoints endpoints.yml
```

Running the Agent: Starting the Agent

```
python -m rasa_core.run -d models/dialogue/ \
                        -u models/nlu/default/model_?/ \
                        --endpoints endpoints.yml
```

Exercise

See the effects of starting the agent:

1. with/without NLU model
2. with/without action server

Running the Agent: Effects

- Running without NLU model
  - Runs the agent with default `RegexInterpreter`
    - Input format:
      `/intent{"entity_name": "entity_value", ...}`
    - Intents (from `domain.yml`):
      `/greet, /goodbye, /thanks, /deny, /joke, /name{"name": "some name"}`

Signals & Interactive Systems

Running the Agent: Effects

- Running without NLU model
  - Runs the agent with default `RegexInterpreter`
    - Input format:

      `/intent{"entity_name": "entity_value", ...}`
    - Intents (from `domain.yml`):

      `/greet, /goodbye, /thanks, /deny, /joke, /name{"name": "some name"}`
- Running without action server – agent won't be able to execute actions defined in `action.py`
  - `/joke`

Signals & Interactive Systems

Section 3

Application Domain Definition

Domain Definition

- *intents*: things you expect users to say
- *entities*: pieces of info you want to extract from messages
- *slots*: information to keep track of during a conversation
- *actions*: things agent can do and say
- *templates*: template strings for the things agent can say

Domain Definition

- *intents*: things you expect users to say
- *entities*: pieces of info you want to extract from messages
- *slots*: information to keep track of during a conversation
- *actions*: things agent can do and say
- *templates*: template strings for the things agent can say
- open `domain.yml`

Application Domain Definition: *Intents*

```
intents:
- greet
- goodbye
- thanks
- deny
- joke
- name
```

- List of labels user utterance will be classified into by Interpreter
  - used in `nlu_data.md`
  - used in `stories.md`
  - used in `actions.py`
- Warning if stories contain intents not defined in `domain.yml`

Application Domain Definition: *Entities*

```
entities:
- name
```

- List of labels words in user utterance will be labeled into by Interpreter (i.e. for Concept Tagging)
  - used in `nlu_data.md`
  - used in `stories.md`
  - used in `actions.py`

Application Domain Definition: *Slots*

```
slots:
  name:
    type: text
```

- Most slots influence the prediction of the next action.
- For the prediction, the slots value is not used directly, but rather it is featurized.
- Possible slot types
  - unfeaturized
  - text
  - list
  - bool
  - float
  - categorical
  - custom

Application Domain Definition: *Slots vs. Entities*

```
slots:
  name:
    type: text
```

```
entities:
- name
```

Application Domain Definition: *Slots vs. Entities*

```
slots:
  name:
    type: text
```

```
entities:
- name
```

- the set of slots and the set of entities usually overlap
- entities without slot (unused, doesn't make sense)
- extra slots to keep track of additional information

Application Domain Definition: *Actions*

```
actions:
- utter_name
- utter_thanks
- utter_greet
- utter_goodbye
- action_joke
```

- Several types of actions
  - **default actions**:
    `action_listen`
    `action_restart`
    `action_default_fallback`
  - **utter actions**: `utter_*`; sends a message to the user
  - **custom actions**: run arbitrary code
    - `action_joke`
    - form actions (later)

Application Domain Definition: *Templates*

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

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

  utter_goodbye:
  - text: "Talk to you later!"

  utter_thanks:
  - text: "My pleasure."
```

- `"Nice to you meet you {name}. How can I help?"`
- `{name}` allows to use slot value in system response

Application Domain Definition: Custom Actions

- open `action.py`

Application Domain Definition: Custom Actions

- open `action.py`

```
class ActionJoke(Action):
    def name(self):
        # define the name of the action which can then be
        # included in training stories
        return "action_joke"

    def run(self, dispatcher, tracker, domain):
        # what your action should do --> make an api call
        request = json.loads(requests.get(
                    'https://api.chucknorris.io/jokes/random').text)
        # extract a joke from returned json response
        joke = request['value']
        # send the message back to the user
        dispatcher.utter_message(joke)
        return []
```

Writing Stories

- open `data/stories.md`

Writing Stories

- open `data/stories.md`

```
## story_greet
* greet
 - utter_name

## story_joke_02
* greet
 - utter_name
* name{"name":"Lucy"}
 - utter_greet
* joke
 - action_joke
* thanks
 - utter_thanks
* goodbye
 - utter_goodbye
```

`##` the name of the story (useful for debugging)

`*` user input expressed as intent

- `* name{"name":"Lucy"}` user response with an entity

`-` system response as an action

Writing NLU Utterances

- open `data/nlu_data.md`

Writing NLU Utterances

- open `data/nlu_data.md`

```
## intent:greet
- Hi
- Hey

## intent:name
- My name is [John](name)
- I am [Josh](name)

## intent:joke
- Can you tell me a joke?
- Tell me a joke
```

`##` the label of the intent

`-` training examples for the intent

- `[John]` the value of the entity
- `(name)` the label of the entity

Exercise 1

- define new intent (e.g. ask age)
- define utter action for it
- write stories for it
- write NLU utterances for it
- re-train models

Exercise 2

- define custom action for age intent (e.g. check if user is not a minor)

Section 4

Configuration Files

Action Endpoints: `endpoints.yml`

- open `endpoints.yml`

Action Endpoints: `endpoints.yml`

- open `endpoints.yml`

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

- core will call an endpoint you can specify, when a custom action is predicted.
- the endpoint should be a webserver that reacts to this call, runs the code and optionally returns information to modify the dialogue state.
- action server is specified using the `endpoints.yml`
- invoked by `rasa_core.run` as `--endpoints endpoints.yml`

Policy Configuration: `policies.yml`

- open `policies.yml`

Policy Configuration: `policies.yml`

- open `policies.yml`

```
policies:
  - name: KerasPolicy
    epochs: 200
    max_history: 3
  - name: MemoizationPolicy
    max_history: 3
```

- Defines a policy ensemble of Keras (NN) and Memoization policies
- Several policies are provided by rasa (consult documentation)
  - maxent
  - embedding
  - FallbackPolicy
  - others

Policy Configuration: `FallbackPolicy`

- Required to fallback
  - if user message is not understood (i.e. the intent recognition has a confidence below **`nlu_threshold`** – min confidence needed to accept an NLU prediction)
  - if none of the dialogue policies predict an action with confidence higher than **`core_threshold`** – min confidence needed to accept an action prediction
- the thresholds, as well as fallback action can be defined via command-line (if `FallbackPolicy` is in the ensemble) as:

```
rasa_core.train --nlu_threshold 0.1 \
                --core_threshold 0.1 \
                --fallback_action action_default_fallback
```

Exercise 3

- define `utter_default` template
- add `FallbackPolicy` to the ensemble (if not present)
- re-train models
- vary thresholds to see the effects

NLU Configuration: `nlu_config.yml`

- open `nlu_config.yml`

NLU Configuration: nlu_config.yml

- open nlu_config.yml

```
language: "en"

pipeline: spacy_sklearn

"pretrained_embeddings_spacy": [
    "SpacyNLP",
    "SpacyTokenizer",
    "SpacyFeaturizer",
    "RegexFeaturizer",
    "CRFEntityExtractor",
    "EntitySynonymMapper",
    "SklearnIntentClassifier",
],
```

- uses a spacy_sklearn (pretrained_embeddings_spacy) pipeline defined in Rasa NLU
- consult Rasa NLU documentation for available pipelines and properties

Section 5

Training & Evaluating Policies

Training & Evaluating Policies

- Rasa Core provides script for training the policies in several modes
  - batch mode (default)
  - interactive mode
  - comparing policies (haven't tried)
- it also provides script for the evaluation of policies (on test data like `stories.md`)
  - standard evaluation

    ```
    python -m rasa_core.test --core models/dialogue \
                             -s test_stories.md \
                             -o results
    ```
  - end-to-end evaluation

    ```
    python -m rasa_core.test --core models/dialogue \
                             --nlu models/nlu/current \
                             --stories e2e_stories.md \
                             --e2e
    ```

End-to-End Evaluation Requirements

- end-to-end evaluation requires stories to contain annotated actual user utterances

```
## end-to-end story
* greet: hello
 - utter_name
* name: my name is [John](name)
 - utter_greet
* joke
 - action_joke
```

Signals & Interactive Systems

Exercise 4

- write ‘normal’ stories (≈ 5)
- write end-to-end stories (‘verbalize’ the ‘normal’ ones)
- evaluate policies on those

Section 6

Slot Filling

Slot Filling & Form Actions

- **slot filling** is one of the most common conversation patterns – collect information from a user in order to do something
- information is usually collected in a row (until all required pieces are collected)
- `FormAction` is an action that contains the logic to loop over the required slots and ask the user for this information.

Slot Filling & Form Actions: domain.yml

example from ‘formbot’ example

```
forms:
  - restaurant_form
```

Slot Filling & Form Actions: actions.py

```
def name(self):
    return "restaurant_form"

def required_slots(tracker: Tracker) -> List[Text]:
    return ["cuisine", "num_people", "outdoor_seating", "preferences", "feedback"]

def slot_mappings(self):
    return {"cuisine": self.from_entity(entity="cuisine", not_intent="chitchat"),
            "num_people": [self.from_entity(entity="num_people",
                            intent=["inform", "request_restaurant"]),
                            self.from_entity(entity="number")],
            "outdoor_seating": [self.from_entity(entity="seating"),
                                 self.from_intent(intent='affirm', value=True),
                                 self.from_intent(intent='deny', value=False)],
            "preferences": [self.from_intent(intent='deny',
                                             value="no additional preferences"),
                             self.from_text(not_intent="affirm")],
            "feedback": [self.from_entity(entity="feedback"), self.from_text()]}

def submit(self, dispatcher: CollectingDispatcher,
           tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict]:
    dispatcher.utter_template('utter_submit', tracker)
    return []
```

Slot Filling & Form Actions: stories.md

```
## happy path
* request_restaurant
    - restaurant_form
    - form{"name": "restaurant_form"}
    - form{"name": null}
```

Exercise 5: Homework

- analyze formbot example in Rasa Core (clone the repo)
- define a form action (will all the requirements)
- test the agent