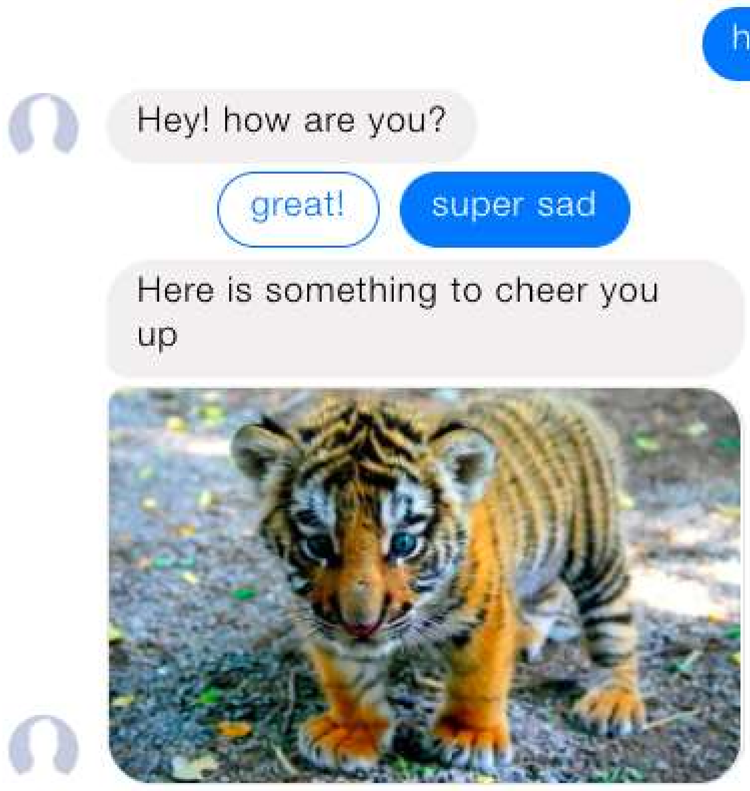
Building a Simple Bot

Note

This tutorial will show you the different parts needed to build a bot. Be aware that this is a small example to get started quickly. It doesn't include a lot of training data, so there is some room for improvement of the final bot performance.

Example Code on GitHub

Here we show how to create your first bot, adding all the pieces of a Rasa application. This might be easier to follow if you also look at Plumbing - How it all fits together.


hello

Hey! how are you?

great! super sad

Here is something to cheer you up

Goal

We will create a very simple bot that checks our current mood and tries to cheer us up if we are feeling sad. It will query our mood and based on our reply will respond with a funny image or a message.

Let’s start by creating a project folder:

```
mkdir moodbot && cd moodbot
```

We need to create two data files (dialogue stories and NLU examples), as well as two configuration files (dialogue domain and NLU config). The final structure should look like this:

```
moodbot/
├── data/
│   ├── stories.md            # dialogue training data
│   └── nlu.md                # nlu training data
├── domain.yml                # dialogue configuration
└── nlu_model_config.json     # nlu configuration
```

Let’s go through each of them!

1. Define a Domain

The first thing we need is a `Domain`. The domain defines the universe your bot lives in.

Here is an example domain for our moodbot, `domain.yml`:

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

actions:
- utter_greet
- utter_cheer_up
- utter_did_that_help
- utter_happy
- utter_goodbye

templates:
  utter_greet:
  - text: "Hey! How are you?"
    buttons:
    - title: "great"
      payload: "great"
    - title: "super sad"
      payload: "super sad"

  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"
```

So what do the different parts mean?

`intents`	things you expect users to say. See Rasa NLU for details.
`entities`	pieces of info you want to extract from messages. See Rasa NLU for details.
`actions`	things your bot can do and say
`slots`	information to keep track of during a conversation (e.g. a users age)
`templates`	template strings for the things your bot can say

In our simple example we don't need `slots` and `entities`, so these sections don't appear in our definition.

How does this fit together? Rasa takes the `intent`, `entities`, and the internal state of the dialogue, and selects one of the `actions` that should be executed next. If the action is just to say something to the user, Rasa will look for a matching template in the domain (action name equals the utter template, as for `utter_greet` in the above example), fill in any variables, and respond. For actions which do more than just send a message, you can define them as python classes and reference them in the domain by their module path. See Defining Custom Actions for more information about custom actions.

Note

There is one additional special action, `ActionListen`, which means to stop taking further actions until the user says something else. It is not specified in the `domain.yml`

2. Define an interpreter

An interpreter is responsible for parsing messages. It performs the Natural Language Understanding (NLU) and transforms the message into structured output. In this example we are going to use Rasa NLU for this purpose.

In Rasa NLU, we need to define the user messages our bot should be able to handle in the Rasa NLU training data format. In this tutorial we are going to use Markdown Format for NLU training data. Let's create some intent examples in `data/nlu.md`:

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

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

## intent:mood_affirm
- yes
- indeed
- of course
- that sounds good
- correct

## intent:mood_deny
- no
- never
- I don't think so
- don't like that
- no way

## intent:mood_great
- perfect
- very good
- 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
- extremely good
- so so perfect
- so good
- so perfect

## intent:mood_unhappy
- 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 so good
- not very good
- extremly sad
- so saad
- so sad
```

Furthermore, we need a configuration file, `nlu_model_config.json`, for the NLU model:

```
{
  "pipeline": "spacy_sklearn",
  "path" : "./models/nlu",
  "data" : "./data/nlu.md"
}
```

We can now train an NLU model using our examples (make sure to install Rasa NLU first, as well as spaCy).

Let's run

```
python -m rasa_nlu.train -c nlu_model_config.json --fixed_model_name current
```

to train our NLU model. A new directory `models/nlu/default/current` should have been created containing the NLU model. Note that `default` stands for project name, since we did not specify it explicitly in `nlu_model_config.json`.

Note

To gather more insights about the above configuration and Rasa NLU features head over to the Rasa NLU documentation.

3. Define stories

So far, we've got an NLU model, a domain defining the actions our bot can take, and inputs it should handle (intents & entities). We are still missing the central piece, **stories to tell our bot what to do at which point in the dialogue**.

A **story** is a training data sample for the dialogue system. There are two different ways to create stories (and you can mix them):

- create the stories by hand, writing them directly to a file
- create stories using interactive learning (see Interactive Learning).

For this example, we are going to create the stories by writing them directly to `stories.md`. Stories begin with `##` and a string as an identifier. User actions start with an asterisk, and bot actions are specified by lines beginning with a dash. The end of a story is denoted by a newline. See Stories - The Training Data for more information about the data format.

Enough talking, let's head over to our stories:

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

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

## sad path 2
* greet
  - utter_greet
* mood_unhappy
  - utter_cheer_up
  - utter_did_that_help
* mood_deny
  - utter_goodbye

## say goodbye
* goodbye
  - utter_goodbye
```

Be aware, although it is a bit faster to write stories directly by hand instead of using interactive learning, special care needs to be taken when using slots, as they need to be properly set in the stories.

4. Put the pieces together

There are two things we still need to do: train the dialogue model and run it.

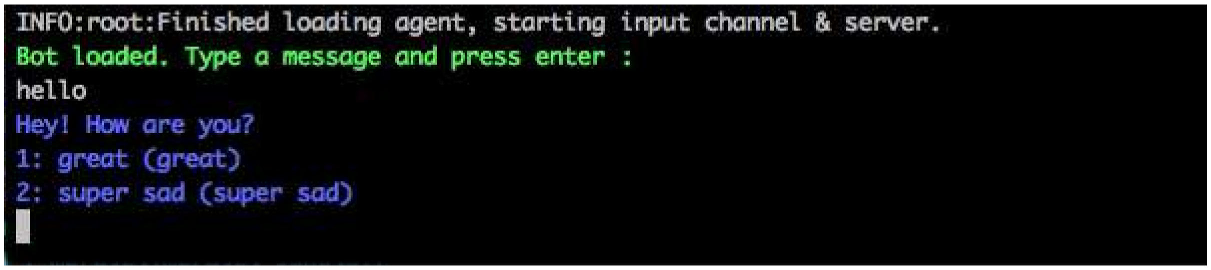
To train the dialogue model, run:

```
python -m rasa_core.train -s data/stories.md -d domain.yml -o models/dialogue --epochs 300
```

This will train the dialogue model for `300` epochs and store it into `models/dialogue`. Now we can use that trained dialogue model and the previously created NLU model to run our bot. Here we'll just talk to the bot on the command line:

```
python -m rasa_core.run -d models/dialogue -u models/nlu/default/current
```

And there we have it! A minimal bot containing all the important pieces of Rasa Core.



```
INFO:root:Finished loading agent, starting input channel & server.
Bot loaded. Type a message and press enter :
hello
Hey! How are you?
1: great (great)
2: super sad (super sad)
```

Note

Button emulation does not work in console output, you need to type words like “great” or “sad” instead of numbers 1 or 2.

Bonus: Handle messages from Facebook

If you want to handle input from Facebook instead of the command line, you can specify that as part of the run command, after creating a credentials file containing the information to connect to facebook. Let’s put that into `fb_credentials.yml`:

```
verify: "rasa-bot"
secret: "3e34709d01ea89032asdebfe5a74518"
page-access-token: "EAAbHPa7H9rEBAAuFk4Q3gPKbDedQnx4djJJ1JmQ7CAqO4iJKrQcNT0wtD"
```

If you are new to Facebook Messenger bots, head over to Facebook Messenger Setup for an explanation of the different values.

After setting that up, we can now run the bot using

```
python -m rasa_core.run -d models/dialogue -u models/nlu/current \
   --port 5002 --connector facebook --credentials fb_credentials.yml
```

and it will now handle messages users send to the Facebook page.

How was the tutorial? Click to Vote

Great! Didn't Work Didn't Finish