RASA STACK

A FRAMEWORK FOR BUILDING CHATBOTS

PROOF OF CONCEPT

FOR

BUILDING A CHATBOT USING RASA

Why Rasa?

There are plenty of easy-to-use bot building frameworks developed by big companies like Google and Microsoft. For instance, **DialogueFlow** which is developed by Google and **Bot Framework** developed by Microsoft. Both of them have custom language understanding modes. These frameworks seem to be great and useful when we don’t have any existing data to train the bot. However, In some situations or circumstances we may want to build a Chatbot whose user input must not be processed on servers that are owned by Google and Microsoft. For example, we may want a ChatBot for a business organization in which a bot receives a potentially sensitive and confidential information from users. In such case, we may feel comfortable to maintain all the components of Chatbot at our home.

This is where RASA platform comes in and makes things easier. It is an open source bot building framework. It doesn’t have any components on the server that we can call using an API. This means it will take more work to get it running. However, being incomplete control pf all the components of your chatbot is totally worth the time investment.

RASA STACK

Rasa consists of two components Rasa **NLU** and Rasa **CORE**. Rasa NLU is responsible for *natural language understanding* of the chatbot. Its main purpose is, given an input sentence, It has to predict the *intent* of that sentence, extract the *entities* from the sentence such as age, date of birth. Intent dictates how the chatbot should respond to the input from the user. Entities are used to make responses more customized.

The second component, Rasa **CORE**, the next component in Rasa Stack. It takes the structured input from Rasa **NLU** or it takes the output produced by Rasa NLU and chooses which action bot should take using a probabilistic model like LSTM neural network.

The coolest thing about Rasa is every part of the stack is fully customizable and interchangeable. It is possible to use Rasa **CORE** and Rasa **NLU** separately and independently.

Development Procedure

For developing this chatbot we are using python 3.x version

Setting up Rasa NLU

The recommended way to install Rasa NLU is using pip:

command: pip3 install rasa_nlu

Note: If you want to install packages of version 3.x use pip3 in the pip.

Installing Pipeline Dependencies

command:

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

This will install Rasa NLU as well as its language model for English language.

Getting started with Rasa NLU

We are developing a simple restaurant search chatbot here.

1. Prepare your NLU Training Data

This data is just a list of messages that you expect to receive from user. They are annotated with intents and entities that NLU should learn to extract.

Make a training data and save it in a .md file. Let us say nlu.md here.

```
## intent:greet
- hey
- hello
- hi
- good morning
- good evening
- good afternoon
- hey there
- hi there
- hii

## intent:restaurant_search
- i'm looking for a place to eat
```

- I want to grab lunch
- I am searching for a dinner spot
- food [near me](location)
- [Japanese](cuisine) restaurants [near me](location:here)
- restaurants [near me](location)
- place [near me](location) to eat
- [russian](cuisine) near me
- [italian](cuisine) restaurants near me
- restaurants in the [midtown](location)
- i'm looking for a [chines](cuisine:chinese) restaurant in the [north](location) of town
- show me [chinese](cuisine) restaurants
- show me a [mexican](cuisine) place in this area
- i am looking for an [indian](cuisine) spot
- i am looking for an [Desi](cuisine:indian) restaurants
- want to eat [spicy](cuisine:indian) food
- I would like a [punjabi](cuisine:indian) food
- search for restaurants in the [center](location)
- anywhere in the [west](location)
- anywhere near [18328](zipcode)
- I am looking for [asian fusion](cuisine) food
- I am looking a restaurant in [29432](zipcode)
- restaurants [here](location)
- looking for finest food [here](location)

## synonym:center
- central

```
- midtown
- middle

## regex:zipcode
- [0-9]{5}

## regex:greet
- hey[\s]*.*
- hi+[\s]*.*

## regex:restaurant_search
- [a-zA-Z0-9\s]*restaurants
- restaurants[a-zA-Z0-9\s]*
- [a-zA-Z0-9\s]*food
- food[a-zA-Z0-9\s]*

## intent:thankyou
- thanks!
- thank you
- thx
- thanks very much
- thanks
- thankyou very much
- thanks a lot
- thanks. Have a nice day
- thanks. have a good one.
- have a good one

## intent:mood_great
- I am doing great
- I am absolutely alright
- I am fine
- I am happy. How about you
- fine
- alright
- good
- going good
- doing good
- happy
- superb
- All good. How are you?
- great
- doing great
- I am good
- absolutely fine

## regex:mood_great
- doing[\s]*(good|great|awesome)

## intent:mood_unhappy
```

```
- I am not doing good
- I am doing bad
- I am unhappy
- I am so sad
- I am not happy
- I am sick

## intent:mood_deny
- No it didn't help
- I am not satisfied
- not okay
```

2. Define your Machine learning Model

Rasa NLU has a number of different components, which together makes a pipeline. We have to create a configuration file and specify which pipeline we want to use. Here we are using pre-defined ***tensorflow_embedding*** pipeline and save the file as **"nlu_config.yml"**.

nlu_config.yml

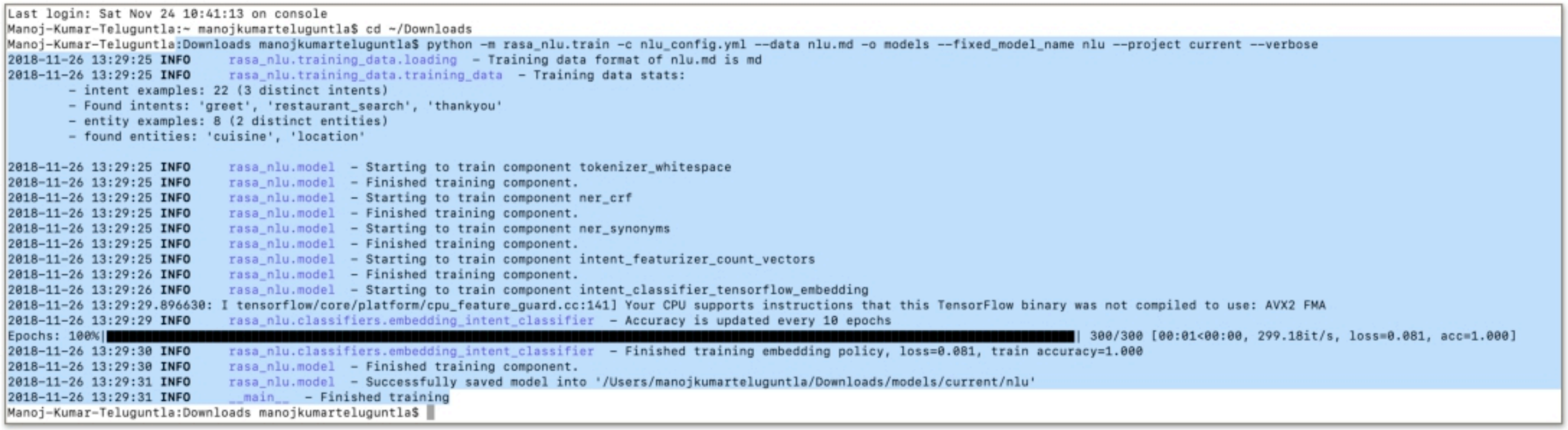
```
language: en
pipeline: tensorflow_embedding
```

3. Train your Machine Learning NLU Model

To train a model, start the rasa_nlu.train command, and tell it where to find the configuration file and training data.

Command:

python -m rasa_nlu.train -c nlu_config.yml --data nlu.md -o models --fixed_model_name nlu --project current —verbose



```
Last login: Sat Nov 24 10:41:13 on console
Manoj-Kumar-Teluguntla:~ manojkumarteluguntla$ cd ~/Downloads
Manoj-Kumar-Teluguntla:Downloads manojkumarteluguntla$ python -m rasa_nlu.train -c nlu_config.yml --data nlu.md -o models --fixed_model_name nlu --project current --verbose
2018-11-26 13:29:25 INFO     rasa_nlu.training_data.loading  - Training data format of nlu.md is md
2018-11-26 13:29:25 INFO     rasa_nlu.training_data.training_data  - Training data stats:
        - intent examples: 22 (3 distinct intents)
        - Found intents: 'greet', 'restaurant_search', 'thankyou'
        - entity examples: 8 (2 distinct entities)
        - found entities: 'cuisine', 'location'

2018-11-26 13:29:25 INFO     rasa_nlu.model  - Starting to train component tokenizer_whitespace
2018-11-26 13:29:25 INFO     rasa_nlu.model  - Finished training component.
2018-11-26 13:29:25 INFO     rasa_nlu.model  - Starting to train component ner_crf
2018-11-26 13:29:25 INFO     rasa_nlu.model  - Finished training component.
2018-11-26 13:29:25 INFO     rasa_nlu.model  - Starting to train component ner_synonyms
2018-11-26 13:29:25 INFO     rasa_nlu.model  - Finished training component.
2018-11-26 13:29:25 INFO     rasa_nlu.model  - Starting to train component intent_featurizer_count_vectors
2018-11-26 13:29:26 INFO     rasa_nlu.model  - Finished training component.
2018-11-26 13:29:26 INFO     rasa_nlu.model  - Starting to train component intent_classifier_tensorflow_embedding
2018-11-26 13:29:29.896630: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2018-11-26 13:29:29 INFO     rasa_nlu.classifiers.embedding_intent_classifier  - Accuracy is updated every 10 epochs
Epochs: 100%|████████████████████████████████████████████████████████████████████████████████████████████████████| 300/300 [00:01<00:00, 299.18it/s, loss=0.081, acc=1.000]
2018-11-26 13:29:30 INFO     rasa_nlu.classifiers.embedding_intent_classifier  - Finished training embedding policy, loss=0.081, train accuracy=1.000
2018-11-26 13:29:30 INFO     rasa_nlu.model  - Finished training component.
2018-11-26 13:29:31 INFO     rasa_nlu.model  - Successfully saved model into '/Users/manojkumarteluguntla/Downloads/models/current/nlu'
2018-11-26 13:29:31 INFO     __main__  - Finished training
Manoj-Kumar-Teluguntla:Downloads manojkumarteluguntla$ 
```

If everything goes fine, you will see a message saying Finished training and successfully saved in to specified folder.

Running the Trained Model

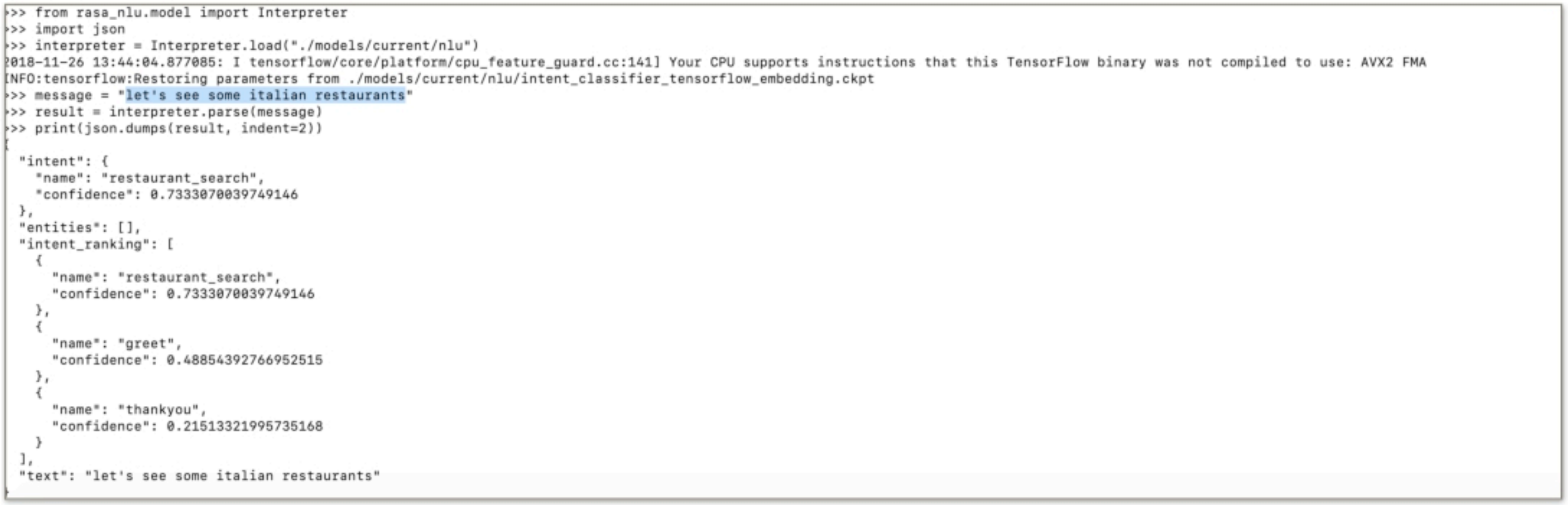
There are two ways that we can run our model, directly from python, or by starting a http server. To use your new model in python, create an **Interpreter** object and pass message to its **parse()** method.

Test Cases

Testing our model with different test cases.

1. Lets test our trained model with message: **“Lets see some Italian restaurants”**

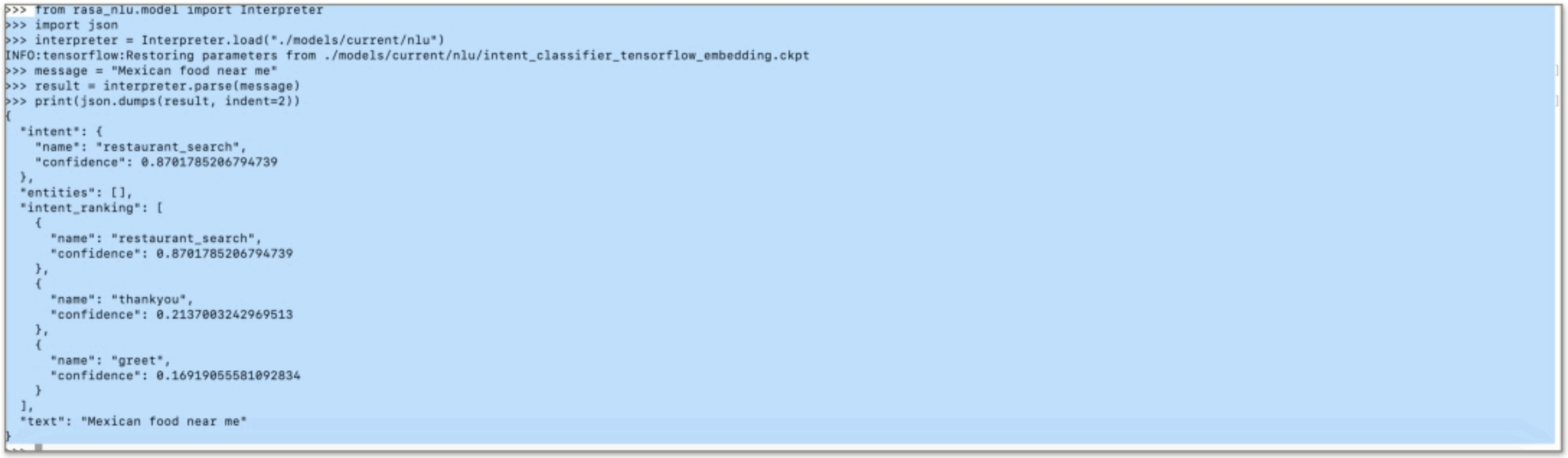
Result:



```
>> from rasa_nlu.model import Interpreter
>> import json
>> interpreter = Interpreter.load("./models/current/nlu")
018-11-26 13:44:04.877085: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
NFO:tensorflow:Restoring parameters from ./models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
>> message = "let's see some italian restaurants"
>> result = interpreter.parse(message)
>> print(json.dumps(result, indent=2))

 "intent": {
   "name": "restaurant_search",
   "confidence": 0.7333070039749146
 },
 "entities": [],
 "intent_ranking": [
   {
     "name": "restaurant_search",
     "confidence": 0.7333070039749146
   },
   {
     "name": "greet",
     "confidence": 0.48854392766952515
   },
   {
     "name": "thankyou",
     "confidence": 0.21513321995735168
   }
 ],
 "text": "let's see some italian restaurants"
```

2. message: **“Mexican food near me”**

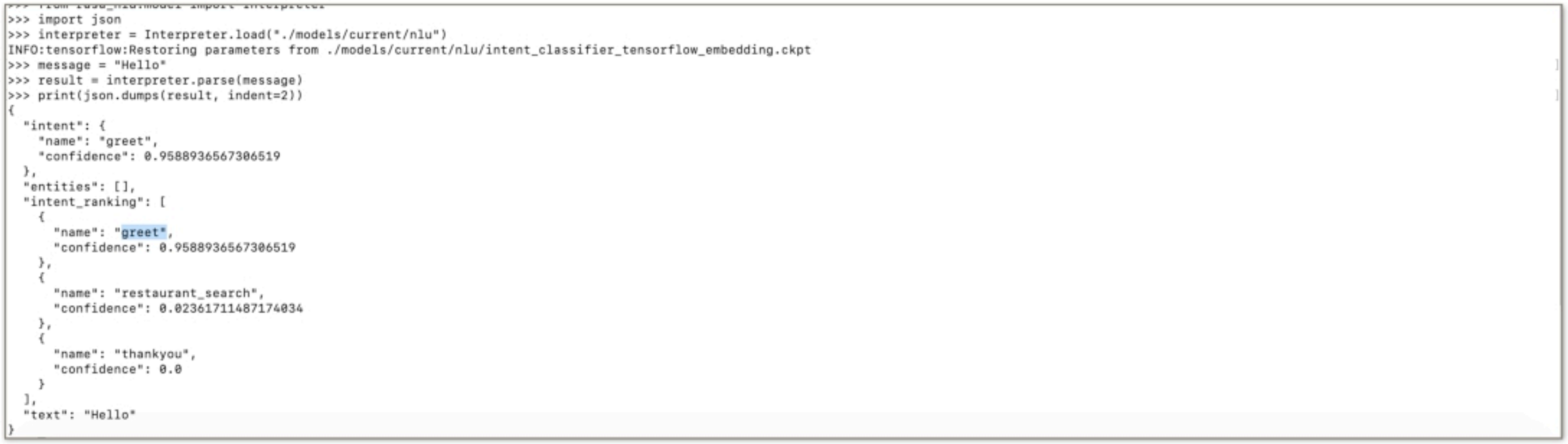
Result:



```
>>> from rasa_nlu.model import Interpreter
>>> import json
>>> interpreter = Interpreter.load("./models/current/nlu")
INFO:tensorflow:Restoring parameters from ./models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
>>> message = "Mexican food near me"
>>> result = interpreter.parse(message)
>>> print(json.dumps(result, indent=2))
{
  "intent": {
    "name": "restaurant_search",
    "confidence": 0.8701785206794739
  },
  "entities": [],
  "intent_ranking": [
    {
      "name": "restaurant_search",
      "confidence": 0.8701785206794739
    },
    {
      "name": "thankyou",
      "confidence": 0.2137003242969513
    },
    {
      "name": "greet",
      "confidence": 0.16919055581092834
    }
  ],
  "text": "Mexican food near me"
}
```

3. message: **“Hello”**

Result:



```
>>> import json
>>> interpreter = Interpreter.load("./models/current/nlu")
INFO:tensorflow:Restoring parameters from ./models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
>>> message = "Hello"
>>> result = interpreter.parse(message)
>>> print(json.dumps(result, indent=2))
{
  "intent": {
    "name": "greet",
    "confidence": 0.9588936567306519
  },
  "entities": [],
  "intent_ranking": [
    {
      "name": "greet",
      "confidence": 0.9588936567306519
    },
    {
      "name": "restaurant_search",
      "confidence": 0.02361711487174034
    },
    {
      "name": "thankyou",
      "confidence": 0.0
    }
  ],
  "text": "Hello"
}
```

RASA CORE

Installation:

Recommended way to install Rasa Core is using **pip**. It installs the latest stable version.

Command:

```
pip3 install rasa_core
```

1. Writing Stories

Rasa core learns from different examples conversations. Each different conversation is called a story. A story have a unique name and unique path it flows. We have to create stories and train Rasa Core Accordingly.

A story starts with ## and after **name** (optional).

Small story looks like this:

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

In the above story, user greets chatbot and it greets back the user. In the realtime, There is response can be handled in many ways. It can be direct response that Chatbot trained with or there may be API calls to get the response that to be sent to user.

A typical **stories** file looks like this:

```
## happy path
* greet
- utter_greet
* mood_great
- utter_goodtohear
* restaurant_search
- utter_typeofcuisine
* restaurant_search{"cuisine": "chinese"}
- utter_whichlocation
* restaurant_search{"location": "north"}
- utter_restaurants
* thankyou
- utter_thankyou
- action_restart

## happy path with cuisine
* greet
- utter_greet
* mood_great
- utter_goodtohear
* restaurant_search{"cuisine": "chinese"}
- utter_whichlocation
* restaurant_search{"location": "north"}
- utter_restaurants
* thankyou
- utter_thankyou
```

```
## happy path with location
* greet
- utter_greet
* mood_great
- utter_goodtohear
* restaurant_search{"location": "west"}
- utter_typeofcuisine
* restaurant_search{"cuisine": "mexican"}
- utter_restaurants
* thankyou
- utter_thankyou

## greet with restaurant search
* greet
- utter_greet
* restaurant_search
- utter_typeofcuisine
* restaurant_search{"cuisine": "indian"}
- utter_whichlocation
* restaurant_search{"location": "29432"}
- utter_restaurants
* thankyou
- utter_thankyou

## greet bad mood restaurant search
* greet
- utter_greet
```

```
* mood_unhappy
- utter_cheer_up
* restaurant_search
- utter_typeofcuisine
* restaurant_search{"cuisine": "indian"}
- utter_whichlocation
* restaurant_search{"location": "west"}
- utter_restaurants
* thankyou
- utter_thankyou

## happy path with location cuisine
* greet
- utter_greet
* mood_great
- utter_goodtohear
* restaurant_search{"cuisine": "indian","location": "north"}
- utter_restaurants
* thankyou
- utter_thankyou
```

We have to **save** the file with **.md** extension.

2. Define a Domain

The next thing we need to do is defining a domain. Domain is the place where our Chatbot lives in.

Domain file consists of different parts.

intents	things we expect user to say (its Rasa NLU for Rasa Core).
actions	things our bot can do and say.
templates	template strings that our bot can say.
entities	piece of info that we want to extract from the messages.
slots	information to keep track during a conversation.

Our simple Example here do not have slots and entities.

```
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?"

  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"
```

save the content in the file with extension **.yml**

3. Train a Dialogue Model

Next step is to train a neural network on example stories that we created. Training is done using following command.

Command:

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

4. Talking to Our bot

We have added both Rasa NLU and Rasa CORE models and trained the models. Now we can talk to our bot and observe the responses it gives to us. Input has to match our trained data and output will be according to it.

Command:

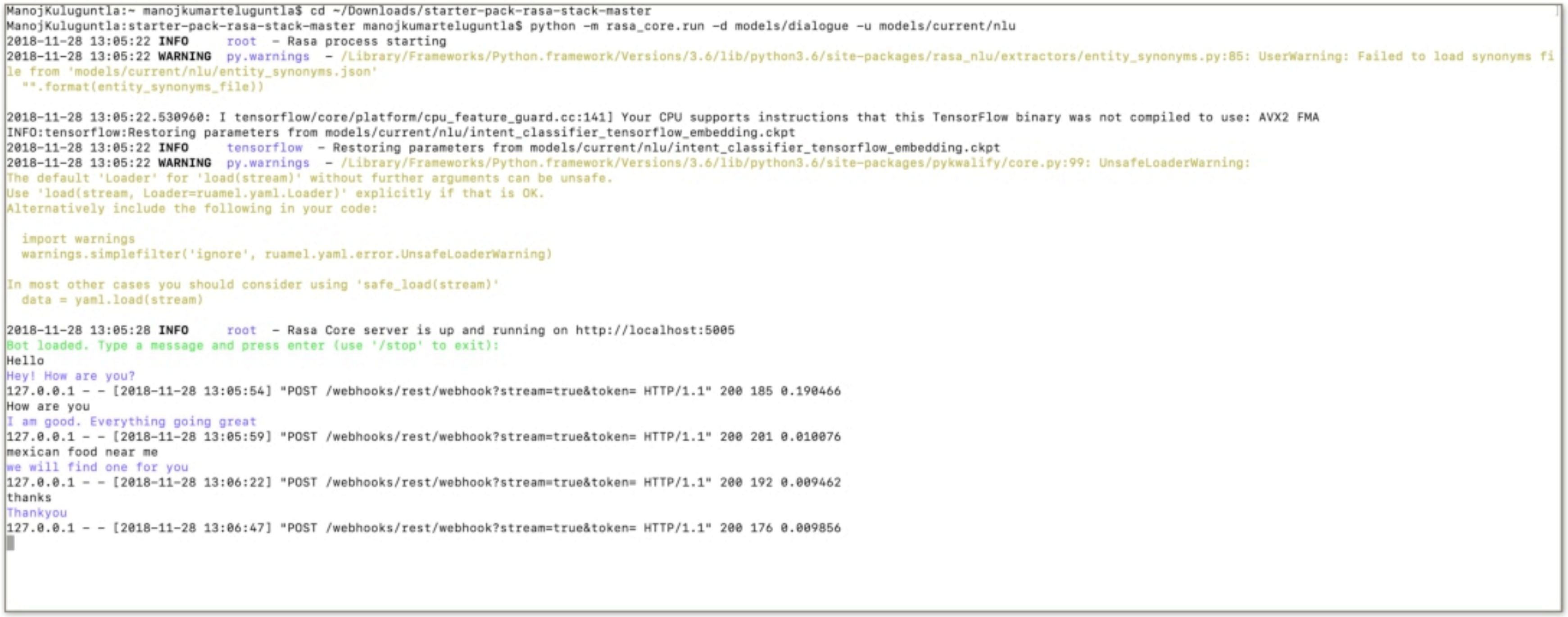
```
python -m rasa_core.run -d models/dialogue -u models/current/nlu
```

Rasa core server will be up and running on port **5005**

Here are some results of ChatBot responding to user messages.

Rasa core executes the stories according to user input and corresponding template has been said out.

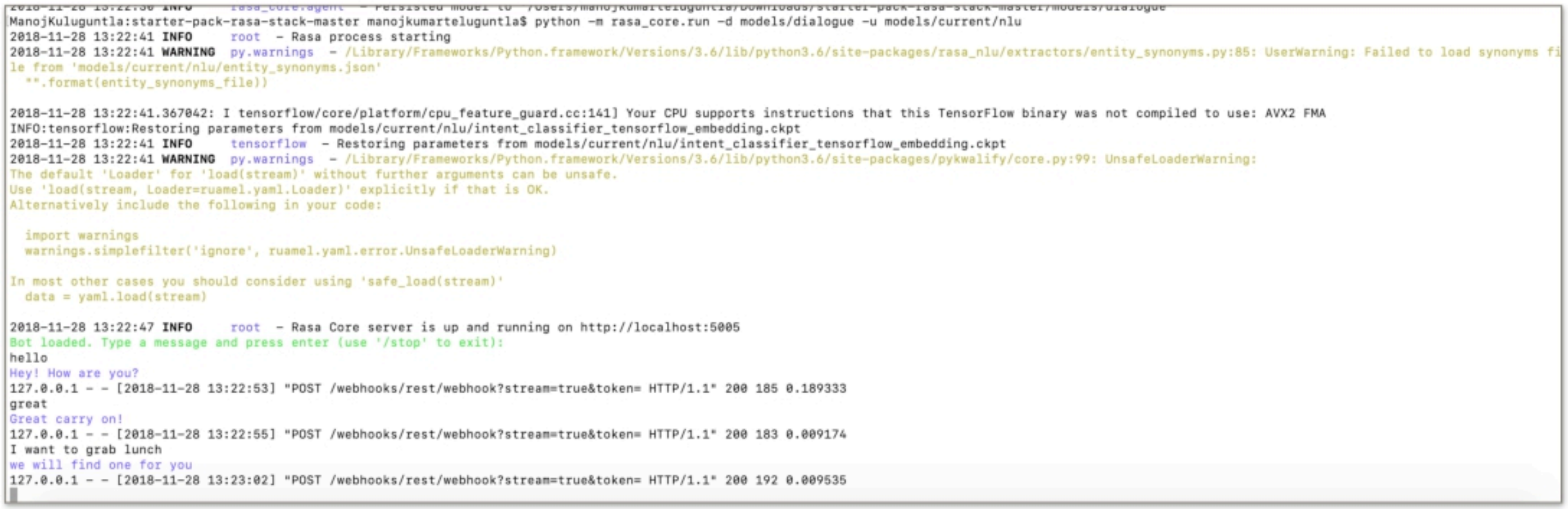
Result 1:



```
ManojKuluguntla:~ manojkumarteluguntla$ cd ~/Downloads/starter-pack-rasa-stack-master
ManojKuluguntla:starter-pack-rasa-stack-master manojkumarteluguntla$ python -m rasa_core.run -d models/dialogue -u models/current/nlu
2018-11-28 13:05:22 INFO     root  - Rasa process starting
2018-11-28 13:05:22 WARNING  py.warnings  - /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/rasa_nlu/extractors/entity_synonyms.py:85: UserWarning: Failed to load synonyms fi
le from 'models/current/nlu/entity_synonyms.json'
  "".format(entity_synonyms_file))

2018-11-28 13:05:22.530960: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
INFO:tensorflow:Restoring parameters from models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
2018-11-28 13:05:22 INFO     tensorflow  - Restoring parameters from models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
2018-11-28 13:05:22 WARNING  py.warnings  - /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/pykwalify/core.py:99: UnsafeLoaderWarning:
The default 'Loader' for 'load(stream)' without further arguments can be unsafe.
Use 'load(stream, Loader=ruamel.yaml.Loader)' explicitly if that is OK.
Alternatively include the following in your code:

  import warnings
  warnings.simplefilter('ignore', ruamel.yaml.error.UnsafeLoaderWarning)

In most other cases you should consider using 'safe_load(stream)'
  data = yaml.load(stream)

2018-11-28 13:05:28 INFO     root  - Rasa Core server is up and running on http://localhost:5005
Bot loaded. Type a message and press enter (use '/stop' to exit):
Hello
Hey! How are you?
127.0.0.1 - - [2018-11-28 13:05:54] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 185 0.190466
How are you
I am good. Everything going great
127.0.0.1 - - [2018-11-28 13:05:59] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 201 0.010076
mexican food near me
we will find one for you
127.0.0.1 - - [2018-11-28 13:06:22] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 192 0.009462
thanks
Thankyou
127.0.0.1 - - [2018-11-28 13:06:47] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 176 0.009856
```

Result 2:

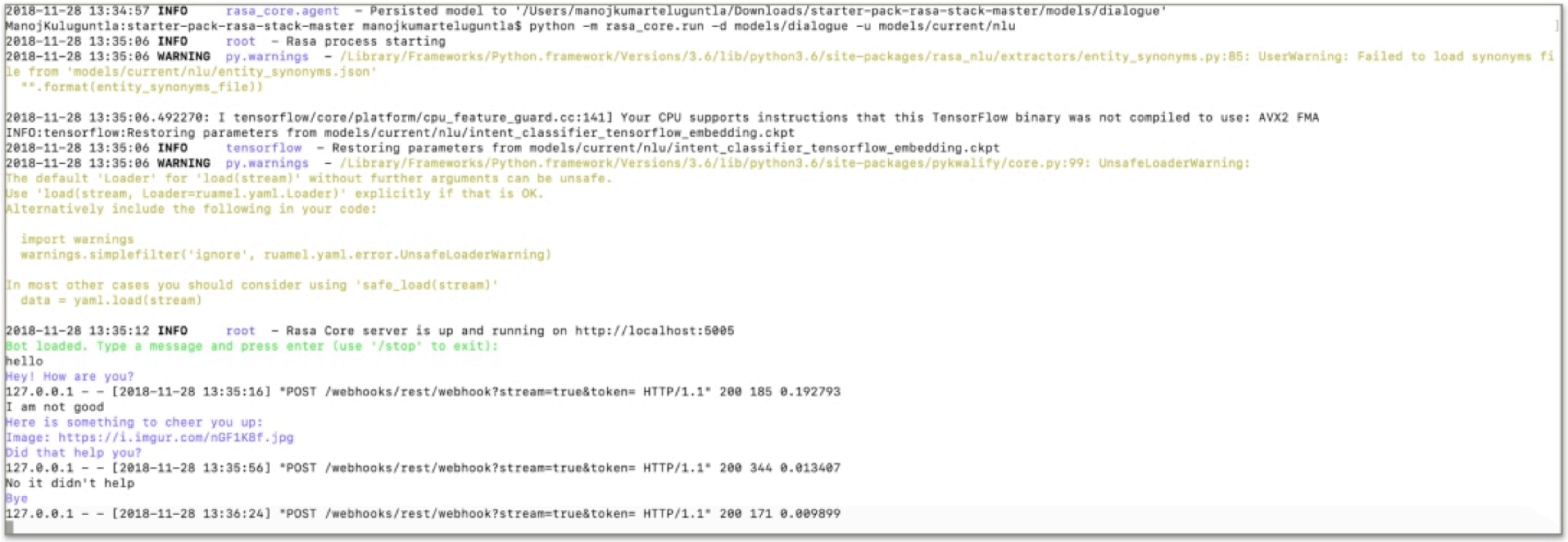
In this case Rasa Core selected story **happy path** and executed it.



```
2018-11-28 13:22:36 INFO     rasa_core.agent  - Persisted model to '/Users/manojkumarteluguntla/Downloads/starter-pack-rasa-stack-master/models/dialogue
ManojKuluguntla:starter-pack-rasa-stack-master manojkumarteluguntla$ python -m rasa_core.run -d models/dialogue -u models/current/nlu
2018-11-28 13:22:41 INFO     root  - Rasa process starting
2018-11-28 13:22:41 WARNING  py.warnings  - /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/rasa_nlu/extractors/entity_synonyms.py:85: UserWarning: Failed to load synonyms fi
le from 'models/current/nlu/entity_synonyms.json'
  "".format(entity_synonyms_file))

2018-11-28 13:22:41.367042: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
INFO:tensorflow:Restoring parameters from models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
2018-11-28 13:22:41 INFO     tensorflow  - Restoring parameters from models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
2018-11-28 13:22:41 WARNING  py.warnings  - /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/pykwalify/core.py:99: UnsafeLoaderWarning:
The default 'Loader' for 'load(stream)' without further arguments can be unsafe.
Use 'load(stream, Loader=ruamel.yaml.Loader)' explicitly if that is OK.
Alternatively include the following in your code:

  import warnings
  warnings.simplefilter('ignore', ruamel.yaml.error.UnsafeLoaderWarning)

In most other cases you should consider using 'safe_load(stream)'
  data = yaml.load(stream)

2018-11-28 13:22:47 INFO     root  - Rasa Core server is up and running on http://localhost:5005
Bot loaded. Type a message and press enter (use '/stop' to exit):
hello
Hey! How are you?
127.0.0.1 - - [2018-11-28 13:22:53] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 185 0.189333
great
Great carry on!
127.0.0.1 - - [2018-11-28 13:22:55] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 183 0.009174
I want to grab lunch
we will find one for you
127.0.0.1 - - [2018-11-28 13:23:02] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 192 0.009535
```

Result 3:

In this scenario, story **sad path** has got executed.



```
2018-11-28 13:34:57 INFO     rasa_core.agent  - Persisted model to '/Users/manojkumarteluguntla/Downloads/starter-pack-rasa-stack-master/models/dialogue'
ManojKuluguntla:starter-pack-rasa-stack-master manojkumarteluguntla$ python -m rasa_core.run -d models/dialogue -u models/current/nlu
2018-11-28 13:35:06 INFO     root  - Rasa process starting
2018-11-28 13:35:06 WARNING  py.warnings  - /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/rasa_nlu/extractors/entity_synonyms.py:85: UserWarning: Failed to load synonyms fi
le from 'models/current/nlu/entity_synonyms.json'
  "".format(entity_synonyms_file))

2018-11-28 13:35:06.492270: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
INFO:tensorflow:Restoring parameters from models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
2018-11-28 13:35:06 INFO     tensorflow  - Restoring parameters from models/current/nlu/intent_classifier_tensorflow_embedding.ckpt
2018-11-28 13:35:06 WARNING  py.warnings  - /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/pykwalify/core.py:99: UnsafeLoaderWarning:
The default 'Loader' for 'load(stream)' without further arguments can be unsafe.
Use 'load(stream, Loader=ruamel.yaml.Loader)' explicitly if that is OK.
Alternatively include the following in your code:

  import warnings
  warnings.simplefilter('ignore', ruamel.yaml.error.UnsafeLoaderWarning)

In most other cases you should consider using 'safe_load(stream)'
  data = yaml.load(stream)

2018-11-28 13:35:12 INFO     root  - Rasa Core server is up and running on http://localhost:5005
Bot loaded. Type a message and press enter (use '/stop' to exit):
hello
Hey! How are you?
127.0.0.1 - - [2018-11-28 13:35:16] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 185 0.192793
I am not good
Here is something to cheer you up:
Image: https://i.imgur.com/nGF1K8f.jpg
Did that help you?
127.0.0.1 - - [2018-11-28 13:35:56] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 344 0.013407
No it didn't help
Bye
127.0.0.1 - - [2018-11-28 13:36:24] "POST /webhooks/rest/webhook?stream=true&token= HTTP/1.1" 200 171 0.009899
```

Fallback Actions

We sometimes want to server wants to fall back to a fallback action like saying ***“Sorry, I did n’t understand what you said”***. Then we have to add an fall back action which will be called when intent recognition in NLU is below some value(generally **nlu_threshold** value) or if none of the dialogue policies predict an action with confidence higher than some value(usually **core_threshold**).

we can set our own values to nlu_threshold and core_threshold.

rasa_core.train scripts provides parameters to adjust these thresholds:

- - nlu_threshold	Minimum confidence needed to accept an NLU prediction
- - core_threshold	Minimum confidence needed to accept an action from rasa core
- - fallback_action	Name of the action to be called when if the confidence of intent and action to be called are below threshold

If we want to run this from python, we have to use like this:

```
from rasa_core.policies.fallback import FallbackPolicy
from rasa_core.policies.keras_policy import KerasPolicy
from rasa_core.agent import Agent

fallback = FallbackPolicy(fallback_action_name="action_default_fallback",
                          core_threshold=0.3,
                          nlu_threshold=0.3)
agent = Agent("domain.yml", policies=[KerasPolicy(), fallback])
```

action_default_fallback is default fallback action in Rasa Core, which will send the utter_default template message to the rasa core. We have to make sure to specify this template in a domain.yml file.

It will revert back to the state of the conversation that before the user message that caused the fallback, so that will not influence the prediction of future actions.

Eg:

```
  templates:
    utter_default:
    -text: “Sorry, I can’t understand that”
```

Things to be noted:

Rasa core will not be able to understand and take the entity from intents if intents are not declared in a story path with the entities.

For example, If the story is defined like this

```
## greet bad mood restaurant search
* greet
- utter_greet
* mood_unhappy
- utter_cheer_up
*  restaurant_search
-  utter_typeofcuisine
```

If we consider **restaurant_search** intent in the story above, It is declared in the story path without entities being declared in it. Then Rasa core will not be able to recognize the entities that is being sent by **rasa_nlu.**

If we want rasa to be able to recognize the entity in a intent and match the story according to that, we have to declare intent with entities.

Example:

```
* restaurant_search{“cuisine": "indian"}
- utter_whichlocation
```

In the above example, we declared an intent **restaurant_search** with entities then, rasa_core server will be able to recognize the entities in the intent and match the story according to it.

Training Data Format

We can provide training data as markdown or as json, as a single file or directory containing multiple files in it.

Note: markdown is usually easier to work with.

Markdown format

It is the easiest RASA NLU training format for humans to read and write. We have already seen an example file containing training data in markdown format.

A sample example of training data in markdown format

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

JSON Format

The JSON format consists of a top-level object called rasa_nlu_data which has keys common_examples, entity_synonyms and regex_features.

Here is the example of training data in JSON format

```
{
    "rasa_nlu_data":
        { "common_examples":
        [], "regex_features" :
        [], "lookup_tables"  :
        [], "entity_synonyms":
        []
    }
}
```

The common_examples are used to train our model. We should put all our training data in common_examples

Entity Synonyms

If we define entities as having same value then they will be treated as synonyms. Here is an example for that

1st Method:

```
## intent:check_balance
- what is my balance <!-- no entity -->
- how much do I have on my [savings](source_account) <!-- entity "source_account" has value "savings" -->
- how much do I have on my [savings account](source_account:savings) <!-- synonyms, method 1-->
- Could I pay in [yen](currency)?  <!-- entity matched by lookup table -->
```

In the above example, entity **"source_account"** has a value **"savings"**. In the next line source_account entity also has a value **"savings account"** which is a synonym of **"savings"**. We can define synonyms in this way, This is the one way of defining synonyms.

2nd Method:

In second method, we make a list of synonyms for a particular entity value. So that classifier will learn to map these entities to the same entity. Here is the example for that.

```
## synonym:savings   <!-- synonyms, method 2 -->
- pink pig
- Checking account
- Savings account
```

In the above example, we are defining a list of synonyms that has same value of savings. NLU will learn to match these values to the savings value. However, this happens only after entities have been extracted, so we have to provide training data on these values so Rasa will learn to pick them up.

Regular Expressions Features (regex)

Regular expressions can be used to **intent** classification and **Entity** extraction. If our intent or entity has a particular structure, we can then define a regular expression and train the model for ease detection of that entity or intent. For zip code example it might look like this:

```
## regex:zipcode
- [0-9]{5}
```

Here the name zip code does not define any name for an **entity** nor an **intent**. Model will learn to consider all 5 digit numbers and classify them as zipcode intent or extract it as entity.

Above Example says how it should be declared when training data is in markdown format.

Fasttext model

Need for text correction:

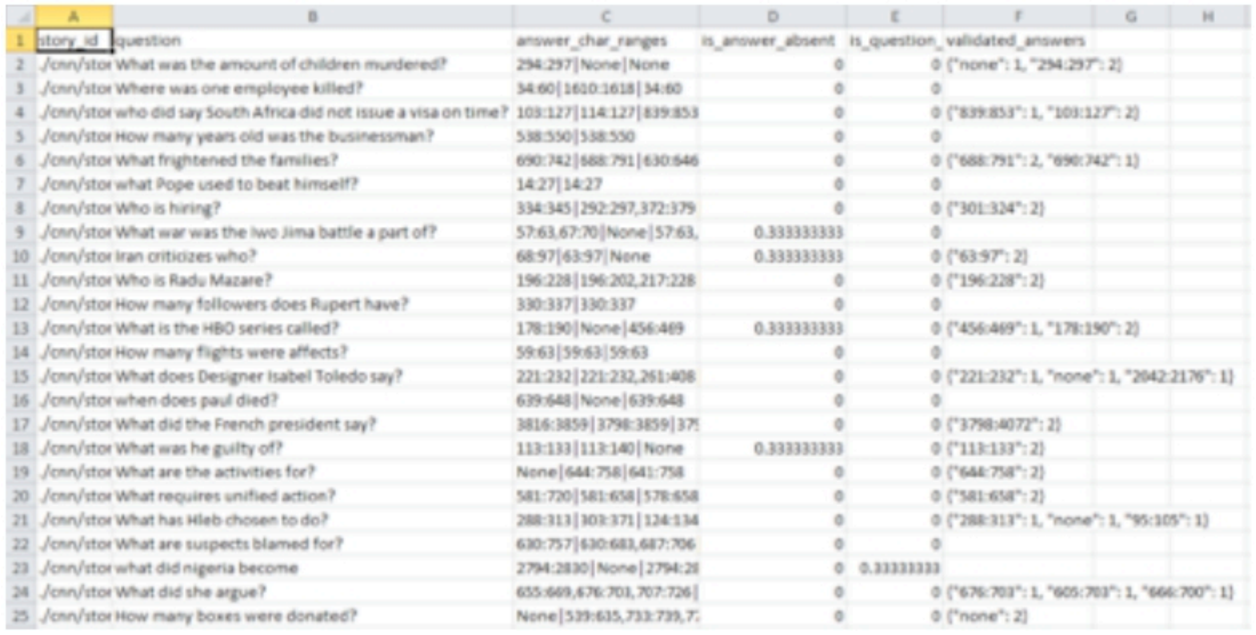
The chatbot that we are making can take user's input, process it and give back the desired output. This works fine in an ideal environment, but in reality, people make mistakes while typing. So there is a strong need for a model that can try to correct a typing error and recognize the input.

Fasttext model can be imported from Gensim library and can be used to process data, and correct it. We train this model so that the predictions will be accurate enough.

Training with Fasttext:

The dataset that we use for this model should be a regular chat between two users. This might be similar to conversations with chatbot.

We only take the second column in this dataset shown above and use it to build and train our model.



	A	B	C	D	E	F	G	H
1	story_id	question	answer_char_ranges	is_answer_absent	is_question_	validated_answers		
2	./cnn/stor	What was the amount of children murdered?	294:297|None|None	0	0	{"none": 1, "294:297": 2}		
3	./cnn/stor	Where was one employee killed?	34:60|1610:1618|34:60	0	0			
4	./cnn/stor	who did say South Africa did not issue a visa on time?	103:127|114:127|839:853	0	0	{"839:853": 1, "103:127": 2}		
5	./cnn/stor	How many years old was the businessman?	538:550|538:550	0	0			
6	./cnn/stor	What frightened the families?	690:742|688:791|630:646	0	0	{"688:791": 2, "690:742": 1}		
7	./cnn/stor	what Pope used to beat himself?	14:27|14:27	0	0			
8	./cnn/stor	Who is hiring?	334:345|292:297,372:379	0	0	{"301:324": 2}		
9	./cnn/stor	What war was the Iwo Jima battle a part of?	57:63,67:70|None|57:63,	0.333333333	0			
10	./cnn/stor	Iran criticizes who?	68:97|63:97|None	0.333333333	0	{"63:97": 2}		
11	./cnn/stor	Who is Radu Mazare?	196:228|196:202,217:228	0	0	{"196:228": 2}		
12	./cnn/stor	How many followers does Rupert have?	330:337|330:337	0	0			
13	./cnn/stor	What is the HBO series called?	178:190|None|456:469	0.333333333	0	{"456:469": 1, "178:190": 2}		
14	./cnn/stor	How many flights were affects?	59:63|59:63|59:63	0	0			
15	./cnn/stor	What does Designer Isabel Toledo say?	221:232|221:232,261:408	0	0	{"221:232": 1, "none": 1, "2042:2176": 1}		
16	./cnn/stor	when does paul died?	639:648|None|639:648	0	0			
17	./cnn/stor	What did the French president say?	3816:3859|3798:3859|37	0	0	{"3798:4072": 2}		
18	./cnn/stor	What was he guilty of?	113:133|113:140|None	0.333333333	0	{"113:133": 2}		
19	./cnn/stor	What are the activities for?	None|644:758|641:758	0	0	{"644:758": 2}		
20	./cnn/stor	What requires unified action?	581:720|581:658|578:658	0	0	{"581:658": 2}		
21	./cnn/stor	What has Hleb chosen to do?	288:313|303:371|124:134	0	0	{"288:313": 1, "none": 1, "95:105": 1}		
22	./cnn/stor	What are suspects blamed for?	630:757|630:683,687:706	0	0			
23	./cnn/stor	what did nigeria become	2794:2830|None|2794:28	0	0.33333333			
24	./cnn/stor	What did she argue?	655:669,676:701,707:726|	0	0	{"676:703": 1, "605:703": 1, "666:700": 1}		
25	./cnn/stor	How many boxes were donated?	None|539:635,733:739,7	0	0	{"none": 2}		

```
import gensim
from gensim.models.fasttext import FastText
import pandas as pd

#reading the csv file
col=['col1','col2','col3','col4','col5']
data= pd.read_csv("rdany.csv",encoding = "ISO-8859-1", header=None,
names=col)
data.drop(['col1','col3','col4','col5'],axis=1,inplace=True)
```

```
#building the data
k=[]
for sent in data['col2']:
    k.append(sent)
import re
dat=[]
for sen in data['col2']:
    wordList = str(sen).split()
    dat.append(wordList)

#building the model
model = FastText(dat, size=100, window=5, min_count=1, workers=4)
model.build_vocab(dat, update=True)

#training the model
model.train(dat, total_examples=len(dat), epochs=100)

#saving the model
model.save("fasttext2.model")
```

Integrating this model with RASA CHATBOT

The model that we saved above will be saved as “fasttext2.model”. Now we load this model in this in the “RasaCoreRun.py” file. Now that the model is already trained, it will not affect the performance by much.

Approach:

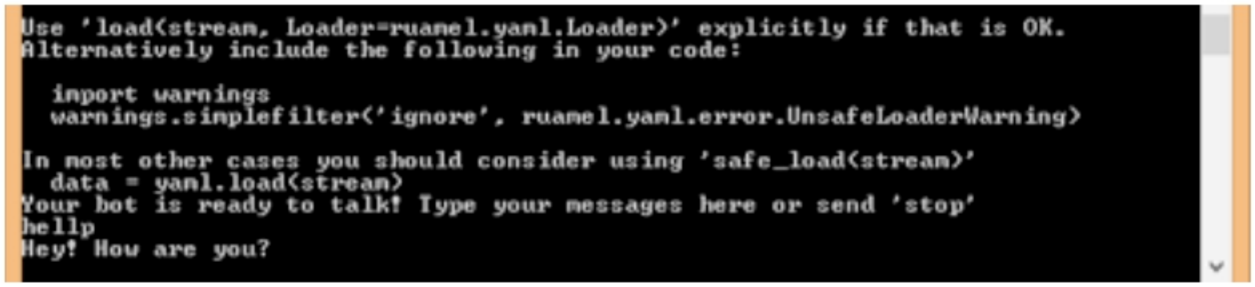
The input that we take from the chatbot is taken into a variable and is compared to the data in “nlu.md” file. So we take this data, and process each word, correct it and put it back as a sentence again.

This correction is done by the fasttext model.

```
def proc(text):
    st=""
    for word in range(len(text)):
        corr=model.wv.most_similar(text[word], topn=5)
        temp=0
        for i in range(len(corr)):
            for j in range(len(data_words)):
                if corr[i][0]==data_words[j]:
                    text[word]=corr[i][0]
                    temp=1
                    break
            if temp==1:
                break
        if word!=len(text)-1:
            st=st+text[word]+" "
        else:
            st=st+text[word]
    return st
```

This function checks the corrected words from corr=model.wv.most_similar(text[word], topn=5) where corr= [('hello?', 0.8630349636077881), ('help', 0.8182423710823059), ('hello', 0.7792395353317261), ('help,', 0.7560293078422546), ('hello,', 0.7384251356124878)] when text[word]= “hello”

Output when the user types “hellp”. It took it as “hello”, which is greet intent and replied the respective output.



```
Use 'load(stream, Loader=ruamel.yaml.Loader)' explicitly if that is OK.
Alternatively include the following in your code:

  import warnings
  warnings.simplefilter('ignore', ruamel.yaml.error.UnsafeLoaderWarning)

In most other cases you should consider using 'safe_load(stream)'
  data = yaml.load(stream)
Your bot is ready to talk! Type your messages here or send 'stop'
hellp
Hey! How are you?
```

REST API Integration using JIRA

Download JIRA software as per your system specifications

Create your account on JIRA and login into it to get localhost link

Create project in JIRA using same login credentials and you will get a unique project key for a specific project

Import JSON using below command in Anaconda prompt

```
conda install -c conda-forge jira
```

Write a python code in RasaCoreRun.py to call the API and get ticket details from JIRA.

Create JIRACLASS and initialize it localhost url, username and password implement getIssue(), getProjects(), createIssue(), deleteIssue methods which will call internal JIRA inbuilt functions to execute corresponding functions.

In **domain.yml** define intents, actions, templates as follows:

```
intents:
 -api
actions:
 -utter_api
templates:
 utter_api:
 -Here is your ticket
```

In **nlu.md** define the user messages our bot should be able to handle

```
## intent:api
- ticket
- I want a ticket
- jira ticket
- api
```

BOT will ask for project key, description and project type from user if he respond by sending a ‘0’ for above attributes the default of each attribute will be displayed, otherwise information given by user for each attribute will be displayed.

```
import JSON
```

Using object of JIRACLASS we will call createIssue() function which will return a JSON object. The JSON object needs to be parsed into python dictionary. Using dumps() function the object will be converted into String and then loads() function will convert that String into python dictionary.0