**BOFA Hackathon**

**CHATBOT-UTTERANCE GENERATOR**

**by Team PHOENIX**

**Team Members :**

* **Debparna Biswas (1580878)**
* **Arya Sharma (1580197)**
* **Geeta Sharma (1580096)**
* **Aviral Gupta (1674709)**
* **Manvi Gupta (1711899)**

**Abstract**

A novel scheme is proposed here to create a Chatbot- Utterance Generator. Chat bots are intelligent systems that understand user’s natural language queries and respond accordingly in a conversation. In banks, at customer care centers and enquiry desks, human is insufficient and usually takes long time to process the single request which results in wastage of time and also reduce quality of customer service. The primary goal of this chat-bot is, customer can interact with mentioning their queries in plain English and the chat-bot can resolve their queries with appropriate response in return. Developing a chat bot will provide a smart solution to solve these queries, provide information as and when required, improve service and increase number of customers. Customers can mention their queries in natural language and the chat-bot can respond to them with correct answer.

**Generating Utterances**

Utterance generator creates sample utterances that we can use within intents for our Chatbot. We are using *Alexa Utterance Generator* to generate utterances. We have also referred different banking websites and collected FAQs to obtain utterances.

The response of a chatbot depends on the intent of the question the customer is asking. So the chatbot must be able to classify the question according to its intent and respond accordingly. So we are sorting out every possible category of intents and generating utterances of each intent.

**Preparing the Dataset**

We have prepared our data set as questions that people usually ask to bank employees, at customer care centers or inquiry desks and categorize the questions according to their intents. Training data can be provided as Markdown or as JSON, as a single file or as a directory containing multiple files. We are using JSON format here.

The training data is structured into four different parts which is compatible with the specific ML model we have used (Rasa NLU) : 1. Common examples 2. Synonyms 3. Regex features 4. Lookup tables

**JSON format** :

{

" rasa\_nlu\_data": {

"common\_examples": [],

"regex\_features" : [],

"lookup\_tables" : [],

"entity\_synonyms": []

}

}

**1. Common examples** : Common examples have three components: *text, intent*and*entities*. The first two are strings while the last one is an array.

* The **text** is the user message [required]
* The **intent** is the intent that should be associated with the text [optional]
* The **entities** are specific parts of the text which need to be identified [optional]

Entities are specified with a start and an end value, which together make a python style range to apply to the string,

For example: In the example below, with text=" turn on my visa card ",

then text [11 : 15] == 'visa'.

{

"rasa\_nlu\_data": {

"common\_examples": [

{

"text": "turn on my visa card",

"intent": " Activate\_Card",

"entities": [

{

"start": 11,

"end": 15,

"value": "visa",

} ]

},

]

}

}

**2. Regex features :** Regular expressions can be used to support the intent classification and entity extraction. For example, if your entity has a deterministic structure (like a Bank Card number or an email address), you can use a regular expression to ease detection of that entity. For the Card-number example it might look like this:

{

"rasa\_nlu\_data": {

"regex\_features": [

{

"name": " Card\_Number",

"pattern": "[0-9]{16}"

}

],

}

}

**3. Entity Synonyms :** If you define entities as having the same value they will be treated as synonyms.

Here is an example below. Here all the words "email", "email address", "email id", "google mail id" etc mean the same while training the data. So they have been given the value "email".

{

"rasa\_nlu\_data": {

"entity\_synonyms": [

{

"value": "email",

"synonyms": [

"email",

"email address",

"email id",

"google mail id"

]

} ]

}

}

**4. Lookup Tables :** The supplied lookup table files must be in a newline-delimited format. For example,  data/test/lookup\_tables/plates.txt may contain:

*tacos*

*beef*

*mapo tofu*

*burrito*

*lettuce wrap*

And can be loaded as:

{

"rasa\_nlu\_data": {

"lookup\_tables": [

{

"name": "plates",

"elements": "data/test/lookup\_tables/plates.txt"

}

]

}

}

When lookup tables are supplied in training data, the contents are combined into a large, case-insensitive regex pattern that looks for exact matches in the training examples.  These regexes are processed identically to the regular regex patterns directly specified in the training data.

So we are using JSON format for the training data which will be fed to the machine learning model (Rasa NLU). The common examples is the main focus here.

We are trying to collect significant amount of examples of customer conversations. Since this is a classification problem the data must be diverse enough to allow ample number of categories (or intents) to be made.

**The Machine Learning Model**

We are using the RasaNLU of RASA Stack to create our ML Model. RASAstack is an open-source AI tool and being an opensource framework, it is easy to customize. Rasa NLU is an open-source natural language processing tool for intent classification, response retrieval and entity extraction in chatbots.

For example, taking a sentence like

"I am looking for a Mexican restaurant in the center of town"

and returning structured data like

{

"intent": "search\_restaurant",

"entities": {

"cuisine" : "Mexican",

"location" : "center"

}

}

Rasa NLU consists of two entity points- Train and Server.

The *training part* generates an ML model when you feed the training data.

The *server* part is where the generated ML model is served via an API.

Rasa NLU uses NLP pipeline which gets converted into an ML model for training the loaded Training Data.

A processing pipeline is the main building block of the Rasa NLU model. It defines what processing stages the incoming user messages will have to go through until the model output is produced.

* It uses SpacyNLP, SpacyTokenizer, SpacyFeaturizer and RegexFeaturizer in the *Preprocessing* stage.
* NER\_CRF and EntitySynonymMapper is used for *Entity Extraction*.
* It uses [SklearnIntentClassifier](https://github.com/bhavaniravi/rasa_nlu/blob/1bb7c545398101fb3b38d64c3e930738f95b7d98/rasa_nlu/classifiers/sklearn_intent_classifier.py#L48) for *Intent Classification*.

After the training process Rasa stores the trained model This model is now ready to predict the intents of the utterances.

**Conclusion**

The proposed solution is expected to be a stepping stone in having in place an intelligent query handling program thereby replicate the customer service experience with one difference that the customer would be interacting with a bot instead of a real person and yet get the queries attended and resolved. Developing a trained chatbot from proper utterances will provide a smart solution to solve queries, provide information as and when required, improve service and increase number of customers. It removes human factors included in organization and can give 24/7 hours service to increase productivity.