Chatbot

Weizenbaum created the first chatbot in the MIT Artificial Intelligence Lab and called it ELIZA. ELIZA imitated the "active listening" strategies of a touchy-feely 1960s Rogerian therapist. A chatbot or chatterbot can be defined as a computer based program that imitates human (text-to-voice/text) conversation. There is a wide variety of chatbots in use today. They may be used for entertainment, marketing, education and a lists of various chatbots can be found at chatbot.org. These bots can range anywhere from talking to William Shakespeare to speaking with ANNA at IKEA Inc. While some are designed so they are able to compete in a number of different chatbot competitions like the Loebner Prize others are used as a tool for entertainment or for information retrieval. For example, chatbot Sofia [12] can assist in teaching mathematics, VPbot imitates a patient that medical students can interview. Some chatbots are used in e-business and as a way to communicate to customers. Rita (Real time Internet Technical Assistant) is used in the ABN AMRO Bank to assist customers in doing financial tasks, such as a wire money transfer. In a company called GetAbby, Abby is used to administer customer relationships. The chatbot uses voice recognition techniques during real time phone calls to track and save information from phone calls including customer name, address, and conversations. This allows the company to track customer calls in a cheaper and more efficient way. A chatbot named Anna is used to interact with customers on Ikea's website. It guides and facilitates users in navigation of Ikea's site by allowing them to enter specific questions and then directing them to the appropriate place. Chatbot Sofia was part of an experiment conducted at the Harvard mathematics department in 2003. The experiment investigated the process of teaching and learning mathematics with the help of a chatbot. Chatbot Sofia has an encyclopedic glossary of mathematics definitions and a general knowledge background, and is able to solve simple mathematical problems. The tool saves all the conversations which can later be analyzed in order to get more information on how Two Case Studies in Using Chatbots for Security Training 267 students are learning, what questions they ask, and what mistakes they frequently make. It was concluded that Sofia contributes to a variety of teaching methods and builds an open source knowledge database for different parts of mathematics. In addition, by teaching Sofia and watching how it learns, students may gain a better understanding regarding the process of teaching mathematics. Computer Simulator in Educational Communication (CSIEC) is a web-based human-computer communication system that uses natural language. This chatbot may be used as a chatting partner for learning the English language. It imitates

human emotions and personalities. Moreover, conversations are not limited to any specific subject. The ideal user's input should be acoustic and then converted into text but so far only keyboard inputs are used since the speech recognition program still needs improvement. It is important to note that when constructing such various chatbots, the goal of what the chatbot should do and the audience to whom the chatbot is for, should be considered and analyzed thoroughly. Tailoring the chatbot for the users and also addressing the appropriate questions/responses of a typical user will produce a more human-like effect. Allowing the chatbot to achieve such an effect may help to improve its overall intention as well as the user's experience of using the chatbot

A chatbot is an intelligent piece of software that is capable of communicating and performing actions similar to a human. Chatbots are used a lot in customer interaction, marketing on social network sites and instantly messaging the client. There are two basic types of chatbot models based on how they are built; Retrieval based and Generative based models.

1. Retrieval based Chatbots

A retrieval-based chatbot uses predefined input patterns and responses. It then uses some type of heuristic approach to select the appropriate response. It is widely used in the industry to make goal-oriented chatbots where we can customize the tone and flow of the chatbot to drive our customers with the best experience.

2. Generative based Chatbots

Generative models are not based on some predefined responses.

They are based on seq 2 seq neural networks. It is the same idea as machine translation. In machine translation, we translate the source code from one language to another language but here, we are going to transform input into an output. It needs a large amount of data and it is based on Deep Neural networks.

About the Python Project – Chatbot

In this Python project with source code, we are going to build a chatbot using deep learning techniques. The chatbot will be trained on the dataset which contains categories (intents), pattern and responses.

We use a special recurrent neural network (LSTM) to classify which category the user's message belongs to and then we will give a random response from the list of responses.

Let's create a retrieval based chatbot using NLTK, Keras, Python, etc.

The Dataset

The dataset we will be using is 'intents.json'. This is a JSON file that contains the patterns we need to find and the responses we want to return to the user.

Prerequisites

The project requires you to have good knowledge of Python, Keras, and Natural language processing (NLTK).

Types of files

Intents.json – The data file which has predefined patterns and responses.

train_chatbot.py – In this Python file, we wrote a script to build the model and train our chatbot.

Words.pkl – This is a pickle file in which we store the words Python object that contains a list of our vocabulary.

Classes.pkl – The classes pickle file contains the list of categories.

Chatbot_model.h5 – This is the trained model that contains information about the model and has weights of the neurons.

Chatgui.py – This is the Python script in which we implemented GUI for our chatbot. Users can easily interact with the bot.

STEPS:

Import and load the data file

Preprocess data

Create training and testing data

Build the model

Predict the response

1. Import and load the data file

First, make a file name as train_chatbot.py. We import the necessary packages for our chatbot and initialize the variables we will use in our Python project.

The data file is in JSON format so we used the json package to parse the JSON file into Python.

```
import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

import json

import pickle


import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout

from keras.optimizers import SGD

import random


words=[]

classes = []

documents = []

ignore_words = ['?', '!']

data_file = open('intents.json').read()
```

```
intents = json.loads(data_file)
```

2. Preprocess data

When working with text data, we need to perform various preprocessing on the data before we make a machine learning or a deep learning model. Tokenizing is the most basic and first thing you can do on text data. Tokenizing is the process of breaking the whole text into small parts like words.

Here we iterate through the patterns and tokenize the sentence using nltk.word_tokenize() function and append each word in the words list. We also create a list of classes for our tags.

```
for intent in intents['intents']:

for pattern in intent['patterns']:


#tokenize each word

w = nltk.word_tokenize(pattern)

words.extend(w)

#add documents in the corpus

documents.append((w, intent['tag']))


# add to our classes list

if intent['tag'] not in classes:

classes.append(intent['tag'])
```

Now we will lemmatize each word and remove duplicate words from the list. Lemmatizing is the process of converting a word into its lemma form and then creating a pickle file to store the Python objects which we will use while predicting.

```
# lemmatize, lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(documents), "documents")
# classes = intents
print (len(classes), "classes", classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)


pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```

3. Create training and testing data

Now, we will create the training data in which we will provide the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn’t understand text so we will convert text into numbers

```
# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
```

```
# training set, bag of words for each sentence
for doc in documents:
# initialize our bag of words
bag = []
# list of tokenized words for the pattern
pattern_words = doc[0]
# lemmatize each word - create base word, in attempt to represent related words
pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
# create our bag of words array with 1, if word match found in current pattern
for w in words:
bag.append(1) if w in pattern_words else bag.append(0)

# output is a '0' for each tag and '1' for current tag (for each pattern)
output_row = list(output_empty)
output_row[classes.index(doc[1])] = 1

training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")
```

4. Build the model

We have our training data ready, now we will build a deep neural network that has 3 layers. We use the Keras sequential API for this. After training the model for 200 epochs, we achieved 100% accuracy on our model. Let us save the model as ‘chatbot_model.h5’.

```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons

# equal to number of intents to predict output intent with softmax

model = Sequential()

model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(train_y[0]), activation='softmax'))


# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])


#fitting and saving the model

hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)

model.save('chatbot_model.h5', hist)


print("model created")
```

5. Predict the response (Graphical User Interface)

Now to predict the sentences and get a response from the user to let us create a new file ‘chatapp.py’.

We will load the trained model and then use a graphical user interface that will predict the response from the bot. The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.

Again we import the necessary packages and load the ‘words.pkl’ and ‘classes.pkl’ pickle files which we have created when we trained our model:

```
import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

import pickle

import numpy as np


from keras.models import load_model

model = load_model('chatbot_model.h5')

import json

import random

intents = json.loads(open('intents.json').read())

words = pickle.load(open('words.pkl','rb'))

classes = pickle.load(open('classes.pkl','rb'))
```

To predict the class, we will need to provide input in the same way as we did while training. So we will create some functions that will perform text preprocessing and then predict the class.

```
def clean_up_sentence(sentence):

# tokenize the pattern - split words into array

sentence_words = nltk.word_tokenize(sentence)

# stem each word - create short form for word

sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]

return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence


def bow(sentence, words, show_details=True):

# tokenize the pattern

sentence_words = clean_up_sentence(sentence)

# bag of words - matrix of N words, vocabulary matrix

bag = [0]*len(words)

for s in sentence_words:

for i,w in enumerate(words):

if w == s:

# assign 1 if current word is in the vocabulary position

bag[i] = 1

if show_details:

print ("found in bag: %s" % w)

return(np.array(bag))
```

```
def predict_class(sentence, model):
# filter out predictions below a threshold
p = bow(sentence, words,show_details=False)
res = model.predict(np.array([p]))[0]
ERROR_THRESHOLD = 0.25
results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
# sort by strength of probability
results.sort(key=lambda x: x[1], reverse=True)
return_list = []
for r in results:
return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
return return_list
```

After predicting the class, we will get a random response from the list of intents.

```
def getResponse(ints, intents_json):
tag = ints[0]['intent']
list_of_intents = intents_json['intents']
for i in list_of_intents:
if(i['tag']== tag):
result = random.choice(i['responses'])
break
return result


def chatbot_response(text):
ints = predict_class(text, model)
```

```
res = getResponse(ints, intents)

return res
```

Now we will code a graphical user interface. For this, we use the Tkinter library which already comes in python. We will take the input message from the user and then use the helper functions we have created to get the response from the bot and display it on the GUI. Here is the full source code for the GUI.

```
#Creating GUI with tkinter

import tkinter

from tkinter import *



def send():

msg = EntryBox.get("1.0",'end-1c').strip()

EntryBox.delete("0.0",END)


if msg != '':

ChatLog.config(state=NORMAL)

ChatLog.insert(END, "You: " + msg + '\n\n')

ChatLog.config(foreground="#442265", font=("Verdana", 12 ))


res = chatbot_response(msg)

ChatLog.insert(END, "Bot: " + res + '\n\n')


ChatLog.config(state=DISABLED)

ChatLog.yview(END)
```

```
base = Tk()
base.title("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set

#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12",
height=5,
bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)
```

#Place all components on the screen

scrollbar.place(x=376,y=6, height=386)

ChatLog.place(x=6,y=6, height=386, width=370)

EntryBox.place(x=128, y=401, height=90, width=265)

SendButton.place(x=6, y=401, height=90)

base.mainloop()

Installing of Rasa NLU It's time to install Rasa and some of its libraries pip install rasa_nlu We have successfully installed Rasa NLU, let's install its libraries pip install rasa_nlu[spacy] python -m spacy download en python -m spacy download en_core_web_md python -m spacy link en_core_web_md en pip install rasa_nlu[tensorflow] pip install rasa_core_sdk

Training the Data These are some libraries which we use them later Let's start training our Chatbot, before starting, let's create some folder, mkdir Careerbot cd Careerbot mkdir fold cd folder mkdir data cd data Now go to the data folder manually in your windows, Now create nlu.md file nlu.md We are using markdown documentation for our bot, We can even use .json but I am using here markdown, which I think it as easy.

Here you can train your data using intent, entity and its examples. To make this bot simple, I am not using any entity. nlu.md(https://github.com/shreyas6652/Chatbot RasaNLU/blob/master/CareerBot/wall Let's create a file as nlu_config.yml in (https://github.com/shreyas6652/Chatbot e/nlu_config.yml). In this file, we will create a pipeline for our data Now go command prompt, and train the data python -m rasa_nlu.train -c nlu_config.yml nlu --project current --verbose

In this you can see, we are using 1780 intent example and 8 distinct name, as well entity. After successfully training the data. Our work is to train the bot using python. We train the data in nlu_model.py (https://github.com/shreyas6652/Chatbot RasaNLU/blob/master/CareerBot/wall Save this file in the fold folder.

Here you can train your data using intent, entity and its examples. To make this bot simple, I am not using any entity. Get the https://github.com/shreyas6652/Chatbot-using- RasaNLU/blob/master/CareerBot/wall-e/data/nlu.md) nlu_config.yml in fold

folder https://github.com/shreyas6652/Chatbot-using-RasaNLU/blob/master/CareerBot/wall

ipeline for our data Now go command prompt, and train the data c nlu_config.yml --data/nlu.md -o models

e are using 1780 intent example and 8 distinct intent, even the intent

After successfully training the data. Our work is to train the bot using python. We train the https://github.com/shreyas6652/Chatbot-using RasaNLU/blob/master/CareerBot/wall-e/nlu_model.py)

RasaNLU/blob/master/CareerBot/wall-

o models --fixed_model_name

intent, even the intent

After successfully training the data. Our work is to train the bot using python. We train the using-

After training, we need to run it python nlu_model.py

Training the Chatbot After training the model, We need to create the files in the data folder. Use notepad to create (better use Notepad++) domain.yml stories.md policy.yml endpoints.yml (For local server) action.py (To combine all file and to launch bot) You can get this entire file in (https://github.com/shreyas6652/Chatbot-using- RasaNLU/tree/master/CareerBot/wall-e)

Lets us open the prompt to connect the server python -m rasa_core_sdk.endpoint --actions actions. Do not close the server, Open new anaconda prompt

conda activate botenv cd CareerBot cd fold python -m rasa_core.train -d domain.yml -s data/stories.md -o models/dialogue -c policy.yml After training now run the bot python -m rasa_core.run -d models/dialogue -u models/current/nlu --endpoints endpoints.yml

Uses of the bot Medical applications

Health bots are designed to help with health the user health-related information. If we tell the symptoms, they give the related diagnosis. We can also set up an appointment and set the remainder for the doctor consultation. We c headache etc. We

can also search for the availability of medicines in the nearby medical shops instead of visiting or calling medical shops and wasting time.

Applications in transportation Tourism Chatbot help people to search, select and book tickets. We can also ask about the good hotels and lodge facilities in the place of interest. It also gives us information about the nearby visiting places, so that we can visit those places too. We can also book the f advance. It can also provide information about the cab facilities in the tourist place. Building this kind of information about a particular place.

Health bots are designed to help with health-related issues. They give related information. If we tell the symptoms, they give the related diagnosis. We can also set up an appointment and set the remainder for the doctor consultation. We can also get the suggestions for cold, cough, headache etc. We can also search for the availability of medicines in the nearby medical shops instead of visiting or calling medical shops and

Applications in transportation people to search, select and book tickets. We can also ask about the good hotels and lodge facilities in the place of interest. It also gives us information about the nearby visiting places, so that we can visit those places too. We can also book the flight tickets in advance. It can also provide information about the cab facilities in the tourist place. Building this kind of Chatbot helps us to get all the information about a particular place. They give related information. If we tell the symptoms, they give the related diagnosis. We can also set up an appointment and set the remainder an also get the suggestions for cold, cough, headache etc. We can also search for the availability of medicines in the nearby medical shops instead of visiting or calling medical shops and people to search, select and book tickets. We can also ask about the good hotels and lodge facilities in the place of interest. It also gives us information about the nearby visiting places, so light tickets in advance. It can also provide information about the cab facilities in the helps us to get all the

Applications in the E-commerce field Mobile messengers connected with Chatbot and the e-commerce industry business can open a new channel for selling the products online. These Chatbot interact with the users and help them to either buy or sell their products efficiently online. We can also ask the information about the product based on its cost, color, size(in case of clothes) etc. They dramatically advance shopping experience and improve sales online. We can also get the configuration related information for the electronic gadgets which help the customer to search efficiently.

They make our work easy and save time. Information about the nearby multi-specialty hospitals can be asked and we can also know about the specialists for the particular disease.

Conclusion

By creating the Chatbot, It reduces the manpower. We get any information fast. A person may get tired of saying the same thing many times but Chatbot can say the same thing many times. A person may commit a mistake in giving information but Chatbot never does mistake (if the person has entered the data correctly in bot). We need to pay for a person every month, but Chatbot is a one-time investment.

Advantages

- Gives information rapidly
- Makes less error
- Does Not require human beings ● Available anytime
- User-friendly
- Bias information Disadvantages
- Makes the same kind of mistakes always
- Needs to update data now and then.
- People are not comfortable with a Chatbot
- Trusting a Chatbot is not easy
- Chatbot may get hack, So need to provide proper security
- Old people can't adjust to the new system easily.