RAGHU INSTITUTE OF TECHNOLOGY

AUTONOMOUS

*Dakamarri (v), Bheemunipatnam (M),VisakhapatnamDist, Andhra Pradesh, PIN-531162*

*(Approved by AICTE, Accredited by NBA and NACC –‘A’ Grade, and permanently affiliated to JNTU Kakinada, Andhrapradesh)*

******

2020 – 2021 II B. Tech., CSE II - Semester

PROJECT NAME

COMMAND-LINE INTERFACE PERSONAL ASSISTANT

**Prepared By**

P. MANOHAR (193J1A05F2)

R. JAGADESH (193J1A05F9)

**DEPARTMENT OFCOMPUTER SCIENCE AND ENGINEERING**

RAGHU INSTITUTE OF TECHNOLOGY

AUTONOMOUS

*Dakamarri (v), Bheemunipatnam (M)*

*Visakhapatnam Dist, Andhra Pradesh, PIN-531162*

*(Approved by AICTE, Accredited by NBA and NACC –‘A’ Grade, and permanently affiliated to JNTU Kakinada, Andhrapradesh)*



**CERTIFICATE**

Name of the Laboratory : Mini Project

Name of the Students : P. MANOHAR (193J1A05F2)

R. JAGADESH (193J1A05F9)

Department : CSE

Program : B.TECH

Year : 2020 – 2021

Semester : 4

**ABSTRACT**

Intelligent personal assistants are an important achievement, which have become an indispensable part of the ubiquitous digitalization process.

These virtual assistants can be found in all gadgets such as smartphones, tablets and also smart watches now. The increasing competition in this area has led to many improvements. Big companies like Amazon, Google, Microsoft and Apple offer a complete digital infrastructure that can be controlled by voice assistants.

From digital marketing tasks, scheduling appointments and managing events to personal errands. You can make a virtual assistant do almost anything. Nowadays mobility is a reality. The small size and the rising computational power of personal mobiles devices enable the access and the exchange of an increasingly greater volume of data and information, anywhere and anytime. Multimodal interfaces, combining.

When a user asks a personal assistant to perform a task, the natural language audio signal is converted into digital data that can be analyzed by the software. Then this data is compared with a database of the software using an innovative algorithm to find a suitable answer. This database is located on distributed servers in cloud networks. For this reason, most personal assistants cannot work without a reliable Internet connection.

**INTRODUCTION**

A virtual assistant is a technology based on artificial intelligence. The software uses a device’s microphone to receive voice requests while the voice output takes place at the speaker. But the most exciting thing happens between these two actions.

It is a combination of several different technologies: voice analysis and language processing.

Virtual Assistants are the helping hands to the people who engage in redundant tasks daily. Anyone can simply automate their personal tasks to be done by assistants. Here it is the basic virtual assistant that helps students.

**Present system :**

The present project system is virtual Assistant. In this project it can perform operations like a basic chat, conducting mcq test on desired topic, gathering a quick definition from wikipedia. It is simple to understand and can be used by anyone who is not even familiar with simple building personal assistants. It is user friendly. It is fast.

Virtual Assistants are the helping hands to the people who engage in redundant tasks daily. Anyone can simply automate their personal tasks to be done by assistants. Here it is the basic virtual assistant that helps students.

**DRAWBACKS:**

This programme contains some drawbacks like it cannot store the data permanently into a database. Storing queries in database makes the assistant to self run by implementing a artificial intelligence algorithm,. These are some drawbacks in source code.

Apart from basic chat as it completely relies on scraping web constant internet connection is required.

These drawbacks in source code can be resolved by maintaining a database and make assistant to work on the data stored and enchance itself using artificial intelligence.

**FILES INCLUDED**

1. Chatbot,ipynb
2. Scraper.py
3. Chatbot\_trainer.ipynb
4. Intents.json
5. Voice.py

**Source code of Scraper.py :**

import re

import urllib

import requests

import wikipedia

from helium import \*

from bs4 import BeautifulSoup

import urllib.request as urllib2

class Scraper:

def \_\_init\_\_(self,text):

self.text = text

def definition(self):

definition = wikipedia.summary(self.text,sentences=3)

return definition

def resources(self):

self.text = urllib.parse.quote\_plus(self.text) # avl+trees

url = 'https://google.com/search?q='+self.text

response = requests.get(url)

soup = BeautifulSoup(response.text,'lxml')

sources = dict()

for a in soup.find\_all('a'):

href = a.get('href')

if '/url' in href:

source = href[7:]

source = source.partition('&')[0]

domain = ''.join(re.findall(r'(?<=\.)([^.]+)(?:\.(?:co\.uk|ac\.us|[^.]+(?:$|\n)))',source))

if('/' not in domain and domain!='google'):

try:

sources[domain].append(source)

except KeyError:

sources[domain] = [source]

return sources

def mcqs(self):

self.text = 'sandford mcqs '+self.text

text = urllib.parse.quote\_plus(self.text)

url = 'https://google.com/search?q='+text

response = requests.get(url)

soup = BeautifulSoup(response.text,'lxml')

for a in soup.find\_all('a'):

href = a.get('href')

if '/url' in href:

sanfoundry = href[7:]

sanfoundry = sanfoundry.partition('&')[0]

break

opener = urllib2.build\_opener()

opener.addheaders = [('User-agent', 'Mozilla/5.0')]

resource = opener.open(sanfoundry)

data = resource.read()

soup = BeautifulSoup(data,'lxml')

text=soup.text

answers\_list = re.findall(r'(?<=: )(.?)(?:$|\n)',text,flags=re.IGNORECASE)

explaination\_list = re.findall(r'Explanation: [^\n]\*',text,flags=re.IGNORECASE)

COUNTER = 0

tag = soup.find\_all('p')

mcqs = dict()

for p in tag[1:len(tag)-3]:

value = p.text

if(re.match(r'\d+\.',value)):

values = value.split('\n')

try:

values.remove('View Answer')

values.append(answers\_list[COUNTER])

values.append(explaination\_list[COUNTER])

except:

pass

if(len(values)>1):

if(len(values)==7):

mcqs[values[0]] = {'options':[values[1],values[2],values[3],values[4]], 'answer':values[5], 'explain':values[6]}

elif(len(values)==5):

mcqs[values[0]] = {'options':[values[1],values[2]], 'answer':values[3], 'explain':values[4]}

COUNTER+=1

else:

COUNTER+=1

# get google forms link and send it to user

return mcqs

**Source code of chatbot\_trainer.ipynb:**

import nltk

nltk.download('punkt')

nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

import json

import pickle

import numpy as np

import random

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout

from tensorflow.keras.optimizers import SGD

words = []

classes = []

documents = []

ignore\_words = ['?', '!']

data\_file = open('intents.json').read()

intents = json.loads(data\_file)

for intent in intents['intents']:

for pattern in intent['patterns']:

# take each word and tokenize it

w = nltk.word\_tokenize(pattern)

words.extend(w)

# adding documets

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

# adding classes to our class

if intent['tag'] not in classes:

classes.append(intent['tag'])

# print(words,documents,classes,sep='\n')

words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore\_words]

words = sorted(list(set(words)))

classes = sorted(list(set(classes)))

print(len(documents),"documents")

print(len(classes),"classes")

print(len(words),"words")

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

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

# initializing training data

training = []

output\_empty = [0] \* len(classes)

for doc in documents:

# initializing bag of words

bag = []

# list of tokenized words for the pattern

pattern\_words = doc[0]

# lemmetize each word - create base word, in attempt to represented 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 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")

# create model

# 3 layers. 1st: 128 neurons 2nd: 64 neurons 3rd: output layer 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. Using Stochastic gradient descent with Nesterov accelerated gradient gives good result

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

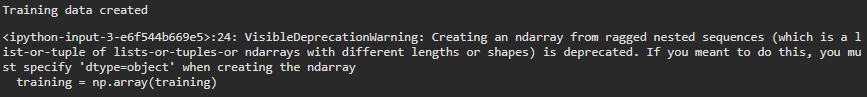
hist = model.fit(np.array(train\_x), np.array(train\_y), epochs=200, batch\_size=5, verbose=1)

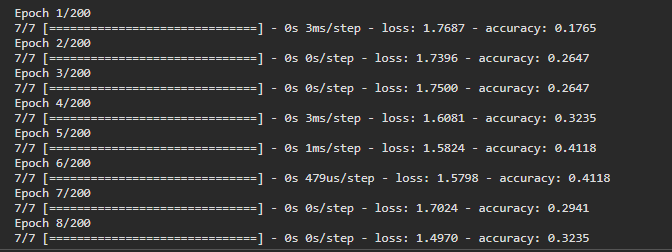
model.save('chatbot\_model.h5', hist)

print("model created")

**Output** :







**Source code of chatbot.ipynb:**

import time

import json

import pickle

import numpy as np

import random

import nltk

from nltk.stem import WordNetLemmatizer

from tensorflow.keras.models import load\_model

import sys

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'

import tensorflow

import logging

logging.getLogger('tensorflow').disabled = True

from scraper import Scraper

import pyttsx3

engine = pyttsx3.init()

def print\_and\_speak(text):

print(text)

engine.say(text)

engine.runAndWait()

lemmatizer = WordNetLemmatizer()

model = load\_model('chatbot\_model.h5')

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

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

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

# clean sentence to words

def clean\_up\_sentence(sentence):

sentence\_words = nltk.word\_tokenize(sentence)

'''print(sentence\_words)'''

sentence\_words = [lemmatizer.lemmatize(word.lower()) for word in sentence\_words]

'''print(sentence\_words)'''

return sentence\_words

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

'''print("bag: ",bag)'''

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 threashold

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

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(msg):

ints = predict\_class(msg, model)

res = getResponse(ints, intents)

return (ints,res)

class finalRes:

def \_\_init\_\_(self,input\_,ints,res):

self.tag = ints[0]['intent']

self.res = res

if self.tag == 'search':

self.tagSearch()

elif self.tag == 'test':

self.tagTest()

def tagSearch(self):

print\_and\_speak(self.res+' ')

queryTerm = input()

o = Scraper(queryTerm)

try:

print\_and\_speak(o.definition())

print('='\*100,'Here are some resources','='\*100,sep='\n')

for key, value in o.resources().items():

print(key+' : '+ ' , '.join(value))

print('='\*100)

except Exception as e:

#print(e,'='\*100,end='\n')

print\_and\_speak("Sorry, I can't get you try asking something else.")

main()

def tagTest(self):

marks = 0

print\_and\_speak(self.res+' ')

queryTerm = input()

o = Scraper(queryTerm)

try:

mcqs = o.mcqs()

for q, op in mcqs.items():

print(q, '\noptions :-', '\n'.join(mcqs[q]['options']), sep='\n', end='\n')

print('\nAnswer - ', end = ' ')

res = input().lower()

if res == mcqs[q]['answer']:

marks+=1

print\_and\_speak('[✓] Right answer\n')

print('='\*30, mcqs[q]['explain'], '='\*30, sep='\n')

print()

time.sleep(2)

else:

print\_and\_speak('[✕] Wrong answer\n')

print('Answer: ', mcqs[q]['answer'],'\n')

print('='\*30, mcqs[q]['explain'], '='\*30, sep='\n')

print()

time.sleep(2)

print\_and\_speak("Score: "+ str(marks)+ '/'+ str(len(mcqs)))

except Exception as e:

#print(e,'='\*100,end='\n')

print\_and\_speak("Sorry, I can't get you try asking something else.")

main()

def main():

input\_ = input()

if input\_.lower() == 'exit':

print\_and\_speak('Glad to see you again. Bye!')

sys.exit(0)

elif not input\_:

print\_and\_speak("Don't nullify! I'm here to help you")

main()

else:

ints, res = chatbot\_response(input\_)

#print(ints, res)

tag = ints[0]['intent']

if tag not in ['greeting','goodbye','thanks','options']:

finalRes(input\_,ints, res)

else:

print\_and\_speak(res)

if tag == 'goodbye':

sys.exit(0)

if \_\_name\_\_ == '\_\_main\_\_':

while True:

try:

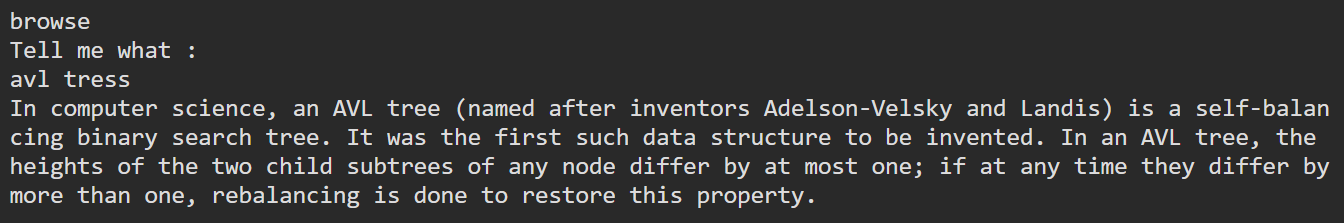
main()

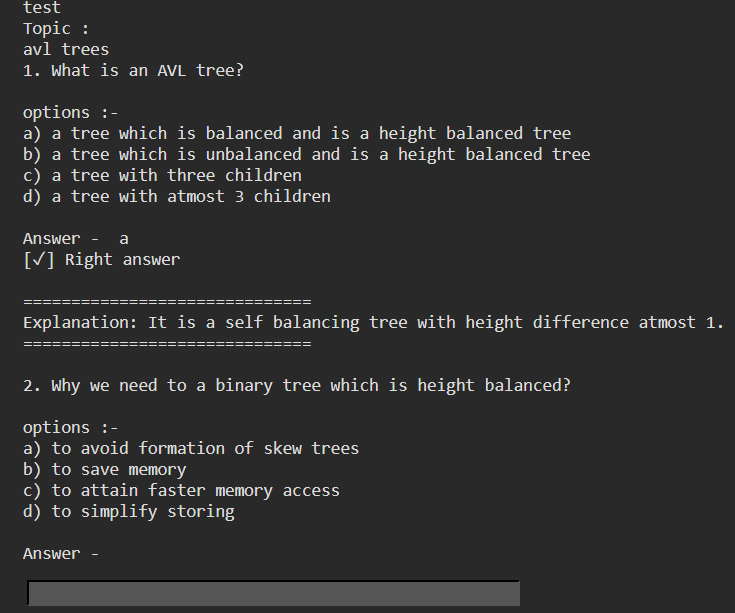
except KeyboardInterrupt:

print\_and\_speak('Exiting.. Have a nice day!')

sys.exit(0)

**Output** :





**Source code of intents.json:**

{"intents": [

{"tag": "greeting",

"patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],

"responses": ["Hello, thanks for asking", "Good to see you again", "Hi there, how can I help?"],

"context": [""]

},

{"tag": "goodbye",

"patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],

"responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],

"context": [""]

},

{"tag": "thanks",

"patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],

"responses": ["Happy to help!", "Any time!", "My pleasure"],

"context": [""]

},

{"tag": "noanswer",

"patterns": [],

"responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand"],

"context": [""]

},

{"tag": "options",

"patterns": ["How you could help me?", "What you can do?", "What help you provide?", "How you can be helpful?", "What support is offered"],

"responses": ["I can help you in your studies by quickly gathering a definition of topic you want to know and I can also conduct a mcq test on desired topic"],

"context": [""]

},

{"tag": "search",

"patterns": ["Get", "Show results of ", "Browse for", "I want to know about", "Help me in understanding", "Google", "Google for", "Show details of", "Tell me about"],

"responses": ["What do you want to search ? : ", "Tell me what : "],

"context": [""]

},

{"tag": "test",

"patterns": ["Take a test", "test time", "test me"],

"responses": ["Topic : ", "Choose topic : "],

"context": [""]

}

]

}

**Reference :**

1. <https://towardsdatascience.com/how-to-create-a-chatbot-with-python-deep-learning-in-less-than-an-hour-56a063bdfc44>

2. <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

3. <https://github.com/mherrmann/selenium-python-helium/blob/master/docs/cheatsheet.md>

4. <https://www.youtube.com/watch?v=XVv6mJpFOb0>