**QUT Chatbot Guide**

All imports used in this project are:

from re import search

from nltk.stem.lancaster import LancasterStemmer

import pickle

import json

import random

import tensorflow

import tflearn

import numpy

import nltk

import requests

from bs4 import BeautifulSoup

You will need to pip install any missing modules to run the chatbot.

First time running the code will train the data. In the code, the training process is set here.

#

# Run training data through neural networks and save model

#

net = tflearn.input\_data(shape=[None, len(training[0])])

net = tflearn.fully\_connected(net, 8)

net = tflearn.fully\_connected(net, 8)

net = tflearn.fully\_connected(net, len(output[0]), activation="softmax")

net = tflearn.regression(net, optimizer='adam', loss='categorical\_crossentropy', learning\_rate=0.01)

# TensorBoard is used to visualise the accuracy and error rate of the model

# tensorboard\_dir can be set to a different directory if needed. Just make sure to change it in both lines below

model = tflearn.DNN(net, tensorboard\_verbose=3, tensorboard\_dir='/tmp/tflearn\_logs/')

model.save('/tmp/tflearn\_logs/')

# If model needs to be trained, un-comment below block of code to train the model

# Can save the model with a name to load on TensorBoard here

model.fit(training, output, n\_epoch=80, batch\_size=8,

          show\_metric=True, run\_id='NAME\_OF\_TEST\_HERE')

model.save("model.tflearn")

After running the training, you will know it worked if you see the below in the console.

Text

Description automatically generated

After training the data, you can adjust the code so that the saved model can be re-used. This can be done by altering the commented out here, highlighted below.

# If model needs to be trained, un-comment below block of code to train the model

# Can save the model with a name to load on TensorBoard here

model.fit(training, output, n\_epoch=80, batch\_size=8,

          show\_metric=True, run\_id='NAME\_OF\_TEST\_HERE')

model.save("model.tflearn")

# If model is already trained, comment above block of code and un-comment the line below

model.load("model.tflearn")

# If model needs to be trained, un-comment below block of code to train the model

# Can save the model with a name to load on TensorBoard here

# model.fit(training, output, n\_epoch=80, batch\_size=8,

#           show\_metric=True, run\_id='NAME\_OF\_TEST\_HERE')

# model.save("model.tflearn")

# If model is already trained, comment above block of code and un-comment the line below

model.load("model.tflearn")

This will cause the saved model to be used and will save time by not re-running the training process each time the chatbot is used. When changes are made to the intent file, the training process will need to be re-done. To do this, uncomment the above code and delete the trained data files (data.pickle and all model.tflearn files) from the folder, as shown below.

Graphical user interface, text

Description automatically generated

The parameters that can be adjusted (and may need changes depending on adjustments made to intents file) are the **optimizer**, **learning\_rate**, and **n\_epoch**. You will likely only need to change the number of epochs from these parameters, most likely increasing the number of epochs as the intent file grows. To test the results, use the TensorBoard module. You can change the directory for saving the files, just ensure that you also change it in the terminal code before running. You should also comment out the last line of the main.py file ( chat() ) to prevent the chatbot from running to focus on testing.

tensorboard --logdir=/tmp/tflearn\_logs/ --host localhost

Following the local host link from running this code will bring up TensorBoard locally which is a site used for viewing graphs that show accuracy/loss for trained data.

In our chat() function, this is where all conversation takes place initially. If the user types quit, the chatbot will end. Within this function, we call our other functions that follow up with a series of predetermined questions. More functions/functionality can be added by following a similar pattern.

Text

Description automatically generatedShown below are how the TRA threats questions and industry threats questions are called. The tag is identified from the user input by matching the pattern and can be used to call a function.

To do similar, add to the intent file using the same structure that we have in place. You can set up different questions with patterns/tags to identify

    {

      "tag": "threats",

      "patterns": [

        "What are the most serious cyber security threats faced by my business currently?",

        "Most serious cyber security threats for my business?",

        "What cyber security threats do I face?",

        "What are some cyber security threats my business face?"

      ],

      "responses": [

        "First I will need to know more about your current security. What security systems are currently in place?"

      ]

    },

The “tag” is what the object is labelled as, in a way. The “patterns” are different phrases that can be matched to this object. This is an array and can/should be multiple strings. The responses are what the chatbot will say back to the user. Generally, 1 response will be fine, but it depends on how much you want to personalise the output.

Our threat objects look like this currently.

    {

      "tag": "TLS communication",

      "patterns": [

        "TLS communication"

      ],

      "responses":

        ["Update all connections to TLS 1.3 or 1.2 with ACSC recommended Cyphers."

      ],

      "cost": 2000,

      "priority": "High"

    },

The idea behind this structure, is the tag is the threat name, the pattern will be what the user will say that could be identified as that threat, the response is the mitigation for the threat, then the cost and priority are base line values for the threat. We were unsure how you determined threats based on user input so the patterns will need updating, we just re-used the threat names to keep functionality.

For the audio version of the chatbot you able to speak to the chatbot and for it to reply to you as well using these two imports.

import pyttsx3

import speech\_recognition as sr

# Wit.ai keys are 32-character uppercase alphanumeric strings

WIT\_AI\_KEY = "YOUR\_API\_KEY\_HERE"

Replace this line of code with your API key here.

# obtain audio from the microphone

r = sr.Recognizer()

with sr.Microphone() as source:

print("Listening... ")

audio = r.listen(source)

try:

resposne = r.recognize\_wit(audio, key=WIT\_AI\_KEY)

print(f"You: {resposne}")

inp = resposne

except sr.UnknownValueError:

inp = "Wit.ai could not understand audio"

except sr.RequestError as e:

raise TypeError(

"Could not request results from Wit.ai service; {0}".format(e))

For speech recognition, the chatbot will say ‘Listen’ and you speak.

**NOTE**: Best done in a quite setting as system can pick up background noise easily.

The Engine used for this chatbot is WIT.ai. the engine can be changed to another service of your choosing. The link below is a guide for how it can be changed.

<https://www.pragnakalp.com/speech-recognition-speech-to-text-python-using-google-api-wit-ai-ibm-cmusphinx/>

print(

"Ashwin: Thank you for that. Now please tell me what are the TRA details of your supporting people?")

bot\_answer.say("Thank you for that. Now please tell me what are the T R A details of your supporting people?")

bot\_answer.runAndWait()

For the chatbot to speak back to the user it uses ***pyttsx3.*** Simply copy over your text with and place it in the function. **NOTE:** Notice TRA must be spelled out to be read aloud correctly. Follow this practice with other abbreviations as well.

voices = bot\_answer.getProperty('voices') #getting details of current voice

bot\_answer.setProperty('voice', voices[0].id)

To change, the voice of the chatbot the following can be added. Full guide over at https://pypi.org/project/pyttsx3/