# chatbot

April 22, 2023

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
import json
with open('/content/drive/MyDrive/intents.json', 'r') as file:
    kb = json.load(file)
```

[ ]:
```
print(type(kb))
```

```
<class 'dict'>
```

[ ]:
```
%cd "/content/drive/MyDrive/Colab Notebooks"
!pip install import-ipynb
import import_ipynb
```

```
/content/drive/MyDrive/Colab Notebooks
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: import-ipynb in /usr/local/lib/python3.9/dist-
packages (0.1.4)
Requirement already satisfied: IPython in /usr/local/lib/python3.9/dist-packages
(from import-ipynb) (7.34.0)
Requirement already satisfied: nbformat in /usr/local/lib/python3.9/dist-
packages (from import-ipynb) (5.8.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.9/dist-
packages (from IPython->import-ipynb) (4.4.2)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.9/dist-
packages (from IPython->import-ipynb) (0.18.2)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.9/dist-packages (from IPython->import-ipynb) (67.6.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.9/dist-
packages (from IPython->import-ipynb) (0.2.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.9/dist-packages (from IPython->import-ipynb) (0.1.6)
Requirement already satisfied: pygments in /usr/local/lib/python3.9/dist-
```

```
packages (from IPython->import-ipynb) (2.14.0)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.9/dist-
packages (from IPython->import-ipynb) (5.7.1)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.9/dist-
packages (from IPython->import-ipynb) (4.8.0)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from IPython->import-ipynb) (3.0.38)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.9/dist-
packages (from IPython->import-ipynb) (0.7.5)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.9/dist-
packages (from nbformat->import-ipynb) (5.3.0)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.9/dist-
packages (from nbformat->import-ipynb) (2.16.3)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.9/dist-
packages (from nbformat->import-ipynb) (4.3.3)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
/usr/local/lib/python3.9/dist-packages (from jedi>=0.16->IPython->import-ipynb)
(0.8.3)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
/usr/local/lib/python3.9/dist-packages (from jsonschema>=2.6->nbformat->import-
ipynb) (0.19.3)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.9/dist-
packages (from jsonschema>=2.6->nbformat->import-ipynb) (22.2.0)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.9/dist-
packages (from pexpect>4.3->IPython->import-ipynb) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.9/dist-packages
(from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->IPython->import-ipynb)
(0.2.6)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.9/dist-packages (from jupyter-core->nbformat->import-
ipynb) (3.2.0)
```

[ ]:
```
!pip install tflearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: tflearn in /usr/local/lib/python3.9/dist-packages
(0.5.0)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages
(from tflearn) (1.16.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages
(from tflearn) (1.22.4)
Requirement already satisfied: Pillow in /usr/local/lib/python3.9/dist-packages
(from tflearn) (8.4.0)
```

[ ]:
```
import spacy
import pickle
```

```
from random import randint

with open("/content/drive/MyDrive/data.pickle", "rb") as f:
    words, labels, training, output = pickle.load(f)
```

[ ]:
```
import tflearn
import tensorflow as tf

tf.compat.v1.reset_default_graph()

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)

model = tflearn.DNN(net)

model.load("/content/drive/MyDrive/model.tflearn")
```

[ ]:
```
import nltk
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

import numpy
import random
import sqlite3
```

[ ]:
```
conn = sqlite3.connect('chatbot_users.db')

# create a table to store user information
conn.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        age INTEGER,
        gender TEXT,
        location TEXT
    )
''')

conn.execute('''
    CREATE TABLE IF NOT EXISTS likes (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        userid INTEGER,
        interest TEXT NOT NULL,
        FOREIGN KEY(userid) REFERENCES users(id)
```

```
    )
''')

conn.execute('''
    CREATE TABLE IF NOT EXISTS dislikes (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        userid INTEGER,
        uninterest TEXT NOT NULL,
        FOREIGN KEY(userid) REFERENCES users(id)
    )
''')

# To get the list of anyone's likes, you'd use a query like:

# select users.name, likes.interest
# from users
# join likes on users.id=likes.userid

# close the connection
conn.commit()
conn.close()
```

[ ]:
```
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[ ]:
```
True
```

[ ]:
```
nlp = spacy.load('en_core_web_sm')

class chatBot:
    def __init__(self, knowledgebase):
      self.kb = knowledgebase
      self.user = 1

    def get_user_input(self):
        user_in = input("%: ")
        return user_in

    def process_user_input(self, s):
        # turn into bag of words
```

```
    bag = [0 for _ in range(len(words))]

    s_words = nltk.word_tokenize(s)
    s_words = [stemmer.stem(word.lower()) for word in s_words]

    for se in s_words:
      for i, w in enumerate(words):
        if w == se:
          bag[i] = 1
    return numpy.array(bag)

  def determine_response(self, input):
    response = ""
    # get response from ML model
    prediction = model.predict(numpy.expand_dims(input, axis=0))
    pred_index = numpy.argmax(prediction)
    tag = labels[pred_index]
    print("tag is:" + tag)

    for tg in self.kb["intents"]:
      if tg['tag'] == tag:
        print("THE TAG WAS IN INTENTS")
        responses = tg['responses']
        print(tg['responses'])
        response = random.choice(responses)
    if response != "":
      return response
    else:
      return "Sorry, I didn't get that. Please rephrase your request"

  def isUserInfo(self, input):
    # if a user key word is in input
    keys = ["name", "like", "dislike", "age", "location", "gender"]

    for key in keys:
      if key in input:
        return True

    return False

  def processUserInfo(self, input, parsed_input):
    pos_tags = nltk.pos_tag(input.split())
    propernouns = [word for word,pos in pos_tags if pos == 'NNP']
    verbs = [word for word,pos in pos_tags if pos == 'VB' or pos == 'VBG']
    nouns = [word for word,pos in pos_tags if pos == 'NN' or pos == 'NNS']
    nums = [word for word,pos in pos_tags if pos == 'CD']
    print(propernouns)
```

```
    print(verbs)
    print(nouns)

    if "name" in input:
      try:
        #grab the proper noun from the set of tagged words
        print(pos_tags)
      except:
        print("I didn't detect a name")
      try:
        #add to db
        print("intent was name")
        conn = sqlite3.connect('chatbot_users.db')
        conn.execute("INSERT INTO users (name) VALUES (?)", (propernouns[0],))
        conn.commit()
        cursor = conn.execute("SELECT id FROM users WHERE name = '" +␣
↪propernouns[0] +"'")
        c = cursor.fetchall()
        print(type(c))
        print(c)
        self.user = c[0][0]
        print(type(self.user))
        print(self.user)
        conn.commit()
        conn.close()
        print(input)
      except BaseException as e:
        print("error adding name to database" + str(e))


    elif "age" in input:
      print(pos_tags)
      #add to db
      print("intent was age")
      conn = sqlite3.connect('chatbot_users.db')
      conn.execute("UPDATE users SET age = " + nums[0] + " WHERE id = " +␣
↪str(self.user) + "")
      conn.commit()
      conn.close()
      print(input)

    elif "location" in input:
      #grab the proper noun from the set of tagged words
      print(pos_tags)
      #add to db
      print("intent was location")
      conn = sqlite3.connect('chatbot_users.db')
```

```
      conn.execute("UPDATE users SET location = '" + propernouns[0] + "' ␣
↪WHERE id = " + str(self.user) + "")
      conn.commit()
      conn.close()
      print(input)

    elif "gender" in input:
      #grab the proper noun from the set of tagged words
      print(pos_tags)
      #add to db
      print("intent was gender")
      conn = sqlite3.connect('chatbot_users.db')
      conn.execute("UPDATE users SET gender = '" + nouns[1] + "' WHERE id = " ␣
↪+ str(self.user) + "")
      conn.commit()
      conn.close()
      print(input)

    elif "dislike" in input:
      print("intent was dislikes")
      conn = sqlite3.connect('chatbot_users.db')
      if len(verbs) > 0:
        try:
          conn.execute("INSERT INTO dislikes (uninterest, userid) VALUES (?, ?
↪)", (verbs[0],str(self.user)))
          conn.commit()
        except BaseException as e:
          print("error adding dislike to database" + str(e))
        conn.close()
        print(input)
      elif len(nouns) > 0:
        try:
          conn.execute("INSERT INTO dislikes (uninterest, userid) VALUES (?, ?
↪)", (nouns[0],str(self.user)))
          conn.commit()
        except BaseException as e:
          print("error adding like to database" + str(e))
        conn.close()
        print(input)
      else:
        print("I didn't know how to process that dislike")

    elif "like" in input:
        print("intent was likes")
        conn = sqlite3.connect('chatbot_users.db')
        if len(verbs) > 0:
          try:
```

```
            conn.execute("INSERT INTO likes (interest, userid) VALUES (?, ?
↪)", (verbs[0],str(self.user)))
            conn.commit()
          except BaseException as e:
            print("error adding like to database" + str(e))
          conn.close()
          print(input)
        elif len(nouns) > 0:
          try:
            conn.execute("INSERT INTO likes (interest, userid) VALUES (?, ?
↪)", (nouns[1],str(self.user)))
            conn.commit()
          except BaseException as e:
            print("error adding like to database" + str(e))
          conn.close()
          print(input)
        else:
          print("I didn't know how to process that like")




  def chat(self):
      self.user_input = ''
      print("Welcome to GB ChatBot!")
      print("When you are done using GB ChatBot, please type exit() to end␣
↪the program.")
      print("Please type something you'd like to know about gameboy␣
↪development: ")

      while self.user_input != 'exit()':
          # get user input
          self.user_input = self.get_user_input()
          if self.user_input != 'exit()':
              # parse/process user input
              # TODO: ---- IMPORTANT: somewhere in here, save user model to␣
↪DB.
              # pick the best response

              parsed_input = self.process_user_input(self.user_input)

              if(self.isUserInfo(self.user_input)):
                response = self.processUserInfo(self.user_input, parsed_input)
                pass
              else:
```

```
                    response = self.determine_response(parsed_input)
                # send the response
                print(response)
            else:
                print("Thanks for using GB ChatBot. Have a Nice Day!")
```

[ ]:
```
cb = chatBot(kb)
cb.chat()
```

```
Welcome to GB ChatBot!
When you are done using GB ChatBot, please type exit() to end the program.
Please type something you'd like to know about gameboy development:
%: name is Ross
['Ross']
[]
['name']
[('name', 'NN'), ('is', 'VBZ'), ('Ross', 'NNP')]
intent was name
<class 'list'>
[(5,)]
<class 'int'>
5
name is Ross
None
%: likes jumping
[]
['jumping']
['likes']
intent was likes
likes jumping
None
%: dislikes swimming
[]
['swimming']
['dislikes']
intent was dislikes
dislikes swimming
None
%: exit()
Thanks for using GB ChatBot. Have a Nice Day!
```

[ ]:
```
conn = sqlite3.connect('chatbot_users.db')
cursor = conn.execute("SELECT * FROM users")
items = cursor.fetchall()
print(items)
conn.close()
```

```
[(1, 'Bob', None, None, None), (2, 'Mary', 44, None, 'Texas'), (3, 'Louis', 34,
'man', 'Utah'), (4, 'Dylan', 33, None, 'Texas'), (5, 'Ross', None, None, None)]
```

[ ]:
```
conn = sqlite3.connect('chatbot_users.db')
cursor = conn.execute("SELECT * FROM likes")
items = cursor.fetchall()
print(items)
conn.close()
```

```
[(1, 1, 'coffee'), (2, 3, 'computers'), (3, 3, 'flying'), (4, 5, 'jumping')]
```

[ ]:
```
conn = sqlite3.connect('chatbot_users.db')
cursor = conn.execute("SELECT * FROM dislikes")
items = cursor.fetchall()
print(items)
conn.close()
```

```
[(1, 3, 'swimming'), (2, 3, 'dislikes'), (3, 5, 'swimming')]
```

[ ]: