# WORKING CHATBOT IN PYTHON

***PROJECT REPORT***

INTRODUCTION TO MACHINE LEARNING

*By*

PARAS WAHI **E18ECE026**

*Submitted to*

**Dr. Manoj Sharma**

# INDEX

# INTRODUCTION TO CHATBOT AND THE TECHNOLOGY BEHIND IT

Chatbots utilize the technology colloquially known as natural
language processing, or NLP. NLP can be classified into two basic approaches. These approaches are rule based NLP and Statistical NLP, or Deep Learning based NLP.
Rule based NLP involves manually engineering a system of grammar and syntax rules.The rule based methods are becoming less popular today.
There are certain advantages. For instance, sentence level tasks, especially in the limited
but well defined domain, can be excellent tools for parsing
and extraction.This makes this approach ideal for query analysis or simple specific casts.
The reason rule based techniques will continue to be used in combination with other techniques is because of its ability to capture specific language, phenomenon and code.Those rules directly into the syntax during parsing rule based NLP are precise.
One historical example of rule based NLP is the old style text adventure games from the 19 eighties, such as Zork.These primitive systems inferred intention based on type descriptions of actions that the player wanted the character to take. Some disadvantages to rule based NLP include the requirement for manual rule development, slow person speed due to branching code and the inability to cover the full scope of human language when it comes to chatting back. Specifically, they're often referred to as transactional chatbots in general, and generally I represent the list of options to the customer or simply detect predetermined intentions within well defined limitations. The other method involves using machine learning techniques to train a machine learning models, such as a neural network,
on a large data set of relevant text data. There are many advantages to this approach, and those advantages become more obvious with larger data sets and models with larger feature sets.
This becomes one of the main drawbacks of the statistical approach.
If you want it done right, you need to trade a very large model on a very large amount of data.
However, when done right, the results could be amazing. The famous GPT 3 natural language model from Open A I, for instance, is based on machine learning approach, which is utilizing a type of model called the Transformer a couple disadvantages of this method, or that it can be difficult
to debug since machine learning models are black box entities, meaning they learn on their own from the training data and interpreting how they learned is not possible without extensive probing and testing through external inputs and outputs.
Another issue is that the model has no built in understanding
of the language. It is trained on meaning that, especially with smaller data sets specific language phenomena might not be picked up by the training and lead The might lead to misinterpretation and syntactical error. These downsides come partly or mostly be mitigated first by increasing the size of the data set and tuning the model parameters.
And second, by combining methods from the rule based approach to the statistical approach, There are a number of third party libraries that are of enormous value to the process of building a chat bot.

## SETTING RULES FOR PRE-DEFINED TRANSACTIONAL RESPONSES

```
import nltk
from nltk.chat.util import Chat, reflections

from io import StringIO
import sys

set_pairs = [
    [
        r"my name is (.*)",
        ["Hello %1, How are you doing today ?",]
    ],
    [
        r"hi|hey|hello",
        ["Hello", "Hey there",]
    ],
    [
        r"what is your name?",
        ["You can call me a chatbot ?",]
    ],
    [
        r"how are you ?",
        ["I am fine, thank you! How can i help you?",]
    ],
    [
        r"I am fine, thank you",
        ["great to hear that, how can i help you?",]
    ],
```

```
        r"(.*) thank you so much, that was helpful",
        ["I am happy to help", "No problem, you're welcome",]
    ],
    [
        r"quit",
    ["Bye, take care. See you soon :) ","It was nice talking to you. See you soon :)"]
],
]



def chatbot():
        print("Hi, I'm your automated assistant")
        chat = Chat(set_pairs, reflections)
        chat.converse():
        old_stdout = sys.stdout
        sys.stdout = mystdout = StringIO()
        sys.stdout = old_stdout
        print('farts'mystdout.getvalue())

def converse():
    if

if __name__ == "__main__":
    chatbot()

```

# INTRODUCTION OF WORD EMBEDDINGS AND TF-IDF VECTORIZATION AND COSINE SIMILARITY

Importing necessary libraries

```
import nltk
import numpy as np
import random
import string # to process standard python strings
from nltk.chat.util import Chat, reflections
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

from io import StringIO
import sys
import json


import os

import requests
from requests import get
from lxml import html
from bs4 import BeautifulSoup

import web_query as wq
import combine_pdf
```

Pre-defined set pairs

```
flag = True

set_pairs = [
    [
        r"my name is (.*)",
        ["Hello %1, How are you doing today ?",]
    ],
    [
        r"hi|hey|hello",
        ["Hello", "Hey there",]
    ],
    [
        r"what is your name?",
        ["You can call me a chatbot ?",]
    ],
    [
        r"how are you ?",
        ["I am fine, thank you! How can i help you?",]
    ],
    [
        r"I am fine, thank you",
        ["great to hear that, how can i help you?",]
    ],
    [
        r"how can i help you? ",
        ["It is my function to help you.", "I just want to fulfill my purpose",]
    ],
```

Reading file and downloading databases

```
f=open(r'C:\Users\darf3\Documents\FLG Work\Rhyme Python Chore Bot\corpus.txt','r',errors = 'ignore')
##raw=json.load(f)
##raw= json.dumps(raw)
##seperator = ' '
##final_str = seperator.join(raw)

raw=f.read()

raw=raw.lower()# converts to lowercase
nltk.download('punkt') # first-time use only
nltk.download('wordnet') # first-time use only
sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences
word_tokens = nltk.word_tokenize(raw)# converts to list of words


lemmer = nltk.stem.WordNetLemmatizer()
#WordNet is a semantically-oriented dictionary of English included in NLTK.
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))


GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You are talking to me"]
```

Defining the functions

```
def greeting(sentence):

    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)


def chatbot():

        chat = Chat(set_pairs, reflections)

        user_input = quit
        try:
                user_input = input(">")
        except EOFError:
                print(user_input)
        if user_input:

                user_input = user_input[:-1]
                if chat.respond(user_input) != None:
                        print(chat.respond(user_input))
                else:
                        user_response = user_input
                        user_response=user_response.lower()
                        if(user_response!='bye'):
                                if(user_response=='thanks' or user_response=='thank you'):
                                        flag=False
                                print("Bot: You are welcome..")
                        else:
                                if(greeting(user_response)!= None):
                                        print("Bot: "+greeting(user_response))
                                else:

                                    if("python" in user_response):
                                        print("Bot: ",end="")
                                        print(response(user_response))
                                        sent_tokens.remove(user_response)
                                    elif("combine" and "file" in user_response):
                                        for i in range(100):
                                            print ("\n")
                                        print("Entering file combining mode. Please enter the exact directory we will be combining.")
                                        directory = input(">")
                                        os.chdir(directory)
                                        print("Thank you. Now please enter keyword to identify target files by title:")
                                        keyword = input(">")
                                        reg_pattern = r'(.*'+keyword+'.+\.pdf)|(^'+keyword+'_.*\.pdf)|(.*_'+keyword+'\.pdf)|(.*'+keyword+'.+\.pdf)'
                                        #  instantiate a bot object, I call it bot1 here
                                        bot1 = combine_pdf.FileBots(directory, reg_pattern)
                                        #  use the .locate method to find files of interest
                                        bot1.locate(show=True)
                                        bot1.pdf_merge_tree()
                                    else:
```

```
                                flag=False
                                print("Bot: Bye! take care..")
        #reply = chat.respond()
##        old_stdout = sys.stdout
##        sys.stdout = mystdout = StringIO()
##        sys.stdout = old_stdout
##        print('farts',mystdout.getvalue())



def response(user_response):
    robo_response=''
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize)
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx=vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if(req_tfidf==0):

        return None
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response



if __name__ == "__main__":
        start=True

        while flag == True:
                if start==True:
                        print("Hi, I'm your automated assistant")
                        start=False
                chatbot()
```

# MAJOR LANGUAGE MODELS AND THEIR USES

Generative pre-trained Transformer 3 or G. P. T. 3 is an auto progressive language model that uses deep learning to produce human-like text.It is the third generation language prediction model in the GPT series, created by Open A.I, a San Francisco based artificial intelligence research laboratory.
31 open A.I researchers and engineers presented the original May 28th 2020 paper introducing GPT 3.
The team increased the capacity of GPT 3 by over two orders of magnitude from that of its predecessor, GPT 2 making GPT 3, the largest non sparse
language model to date. GPT 3 is higher number of parameters granted a higher
level accuracy relative to previous versions with smaller
capacity. GPT 3s capacity is 10 times larger than that of Microsoft's  Turing NLG.
In his July 29th 2020 review in The New York Times, Farhad
Manjoo said that GPT 3, which can generate computer code and poetry as well as prose, is not just amazing spooky and humbling, but also more than a little terrifying.
Australian philosopher David Chalmers described GPT 3 as one of the most interesting and important AI systems ever produced,a review on Wired said that GPT 3 was provoking chills across Silicon Valley.
The National Law Review said that GPT 3 is an impressive step in the larger process, with Open A I and others finding useful applications for all this power while continuing
to work toward a more general intelligence, the G P T 3 model is huge, too large for the average person to train at home Open A. I does have a pay scale for access
to the GPT 3 AP I.

SENTIMENT ANALYSIS  AND INTENT CLASSIFICATION

```
import os
import model, sample, encoder

import json
import os
import numpy as np
import tensorflow as tf
import gpt_2_simple as gpt2

model_name = "345M" # The GPT-2 model we're using

gpt2.download_gpt2(model_name=model_name) # Download the model


sess = gpt2.start_tf_sess()
gpt2.finetune(sess,
file_name,
model_name=model_name,
steps=400)   # steps is max number of training steps

def interact_model(
    model_name,
    seed,
    nsamples,
    batch_size,
    length,
    temperature,
    top_k,
    models_dir
):
    models_dir = os.path.expanduser(os.path.expandvars(models_dir))
    if batch_size is None:
        batch_size = 1
    assert nsamples % batch_size == 0

    enc = encoder.get_encoder(model_name, models_dir)
    hparams = model.default_hparams()
    with open(os.path.join(models_dir, model_name, 'hparams.json')) as f:
        hparams.override_from_dict(json.load(f))

    if length is None:
        length = hparams.n_ctx // 2
    elif length > hparams.n_ctx:
        raise ValueError("Can't get samples longer than window size: %s" % hparams.n_ctx)

    with tf.Session(graph=tf.Graph()) as sess:
        context = tf.placeholder(tf.int32, [batch_size, None])
        np.random.seed(seed)
        tf.set_random_seed(seed)
        output = sample.sample_sequence(
            hparams=hparams, length=length,
            context=context,
            batch_size=batch_size,
            temperature=temperature, top_k=top_k
        )
```

```
            temperature=temperature, top_k=top_k
        )

        saver = tf.train.Saver()
        ckpt = tf.train.latest_checkpoint(os.path.join(models_dir, model_name))
        saver.restore(sess, ckpt)

        while True:
            raw_text = input("Model prompt >>> ")
            while not raw_text:
                print('Prompt should not be empty!')
                raw_text = input("Model prompt >>> ")
            context_tokens = enc.encode(raw_text)
            generated = 0
            for _ in range(nsamples // batch_size):
                out = sess.run(output, feed_dict={
                    context: [context_tokens for _ in range(batch_size)]
                })[:, len(context_tokens):]
                for i in range(batch_size):
                    generated += 1
                    text = enc.decode(out[i])
                    print("=" * 40 + " SAMPLE " + str(generated) + " " + "=" * 40)
                    print(text)
            print("=" * 80)

```

## SUMMARY

We can combine the above methods to create an adaptable and task oriented chatbot. It would be able to answer complex questions and also after training on a larger dataset be able to answer a variety of questions and also give out human-like responses.