Simple Chatbot

Import libraries

In [3]:

```
import io
import random
import string
import warnings
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')
```

Downloading and installing NLTK

In [4]:

```
pip install nltk
```

```
Requirement already satisfied: nltk in /Users/sitikhomsah/opt/anaconda3/lib/python3.9/site-packages (3.6.5)
Requirement already satisfied: click in /Users/sitikhomsah/opt/anaconda3/lib/python3.9/site-packages (from nltk) (8.0.3)
Requirement already satisfied: joblib in /Users/sitikhomsah/opt/anaconda3/lib/python3.9/site-packages (from nltk) (1.1.
0)
Requirement already satisfied: regex>=2021.8.3 in /Users/sitikhomsah/opt/anaconda3/lib/python3.9/site-packages (from nlt
k) (2021.8.3)
Requirement already satisfied: tqdm in /Users/sitikhomsah/opt/anaconda3/lib/python3.9/site-packages (from nltk) (4.62.3)
Note: you may need to restart the kernel to use updated packages.
```

Installing NLTK Packages

In [5]:

```
import nltk   #impor library NLTK
from nltk.stem import WordNetLemmatizer   #import library untuk lemmatization
nltk.download('popular', quiet=True) # for downloading packages
#nltk.download('punkt') # first-time use only
#nltk.download('wordnet') # first-time use only
```

Out[5]:

```
True
```

Reading in the corpus

Program ini menggunakan halaman Wikipedia sebagai corpus dari chatpotdengan cara meng-Copy konten halaman dan menyimpannya dalam file ‘chatbot.txt’.

In [6]:

```
f=open('chatbot.txt','r',errors = 'ignore')  #membuka file corpus dari wikipedia
raw=f.read()      #raw kini berisi semua data dari corpus per baris (raw)
raw = raw.lower()# converts to lowercase
```

Tokenisation

In [7]:

```
#tokenisasi adalah memilah-milah dokumen ke kalimat-kalimat,
#kemudian memilah setiap kalimat menjadi sekumpulan kata kata
sent_tokens = nltk.sent_tokenize(raw) # converts dokumen corpus ke kalimat-kalimat
word_tokens = nltk.word_tokenize(raw)# converts dokumen corpus ke kata-kata
```

Preprocessing

In [8]:

```
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)))
```

Keyword matching

In [9]:

```
#kata-kata pembuka didaftar terlebih dulu dan kemudian secara acak diberikan respon jawabannya
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"]
def greeting(sentence):

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

Generating Response

Bag of Words

After the initial preprocessing phase, we need to transform text into a meaningful vector (or array) of numbers. The bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

Why is it is called a “bag” of words? That is because any information about the order or structure of words in the document is discarded and the model is only **concerned with whether the known words occur in the document, not where they occur in the document.**

The intuition behind the Bag of Words is that documents are similar if they have similar content. Also, we can learn something about the meaning of the document from its content alone.

For example, if our dictionary contains the words {Learning, is, the, not, great}, and we want to vectorize the text “Learning is great”, we would have the following vector: (1, 1, 0, 0, 1).

TF-IDF Approach

A problem with the Bag of Words approach is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content”. Also, it will give more weight to longer documents than shorter documents.

One approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized. This approach to scoring is called Term Frequency-Inverse Document Frequency, or TF-IDF for short, where:

Term Frequency: is a scoring of the frequency of the word in the current document.

```
TF = (Number of times term t appears in a document)/(Number of terms in the document)
```

Inverse Document Frequency: is a scoring of how rare the word is across documents.

```
IDF = 1+log(N/n), where, N is the number of documents and n is the number of documents a term t has
appeared in.
```

Cosine Similarity

Tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus

```
Cosine Similarity (d1, d2) =  Dot product(d1, d2) / ||d1|| * ||d2||
```

where d1,d2 are two non zero vectors.

To generate a response from our bot for input questions, the concept of document similarity will be used. We define a function response which searches the user’s utterance for one or more known keywords and returns one of several possible responses. If it doesn’t find the input matching any of the keywords, it returns a response:” I am sorry! I don’t understand you”

In [10]:

```
#fungsi untuk menetapkan respon jawaban
def response(user_response):
    robo_response=''                            #pada tahap awal respon mesin diisi karakter kosong
    sent_tokens.append(user_response)           #pertanyaan user ditokenisasi dan ditambahkan di corpus
                                                # posisi paling bawah (yaitu posisi -1)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)  #token dari pertanyaan user di vektorisasi
    vals = cosine_similarity(tfidf[-1], tfidf)   #hitung similarity setiap token corspus dengan token pertanyaan
    idx=vals.argsort()[0][-2]                    #sort jarak similariti setiap token corpus dengan token pertanyaan
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if(req_tfidf==0):                            #jika pertanyaan dan semua token corpus jaraknya tinggi maka
                                                 #berarti pertanyaan tidak ada jawabannya
        robo_response=robo_response+"Please reply, I don't understand your questions"
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]   #jika jaraknya terrendah maka dipakai sebagai jawaban
        return robo_response
```

Finally, we will feed the lines that we want our bot to say while starting and ending a conversation depending upon user’s input.

In [ ]:

```
flag=True
print("Mesin: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!")
while(flag==True):
    user_response = input("Masukkan pertanyaan :")
    user_response=user_response.lower()
    if(user_response!='bye'):                          #jika user tidak keluar
        if(user_response=='thanks' or user_response=='thank you' ):  #jika ucapkan thanks/thankyou
            flag=False                                 #tandai proses berhenti
            print("Mesin: You are welcome..")          #balasan thank you
        else:
            if(greeting(user_response)!=None):           #jika response adalah kalimat greeting
                print("Mesin: "+greeting(user_response))  #tampilkan kalimat greeting
            else:                                        #jika bukan kalimat greeting
                print("Mesin: ",end="")
                print(response(user_response))          #memproses user answer disini
                sent_tokens.remove(user_response)       #user answer dihapus setelah dimunculkan
    else:
        flag=False
        print("Mesin: Bye! take care..")
        print("========================")
```

In [ ]:

In [ ]: