TITTLE: HAI COURSEWORK ONE REPORT.

MODULE CODE: COMP3074 UNUK

NAME: KEITH AROGO OWINO.

USER_ID: 18025669.

DATE: 1st December 1, 2020.

1. Design of the system

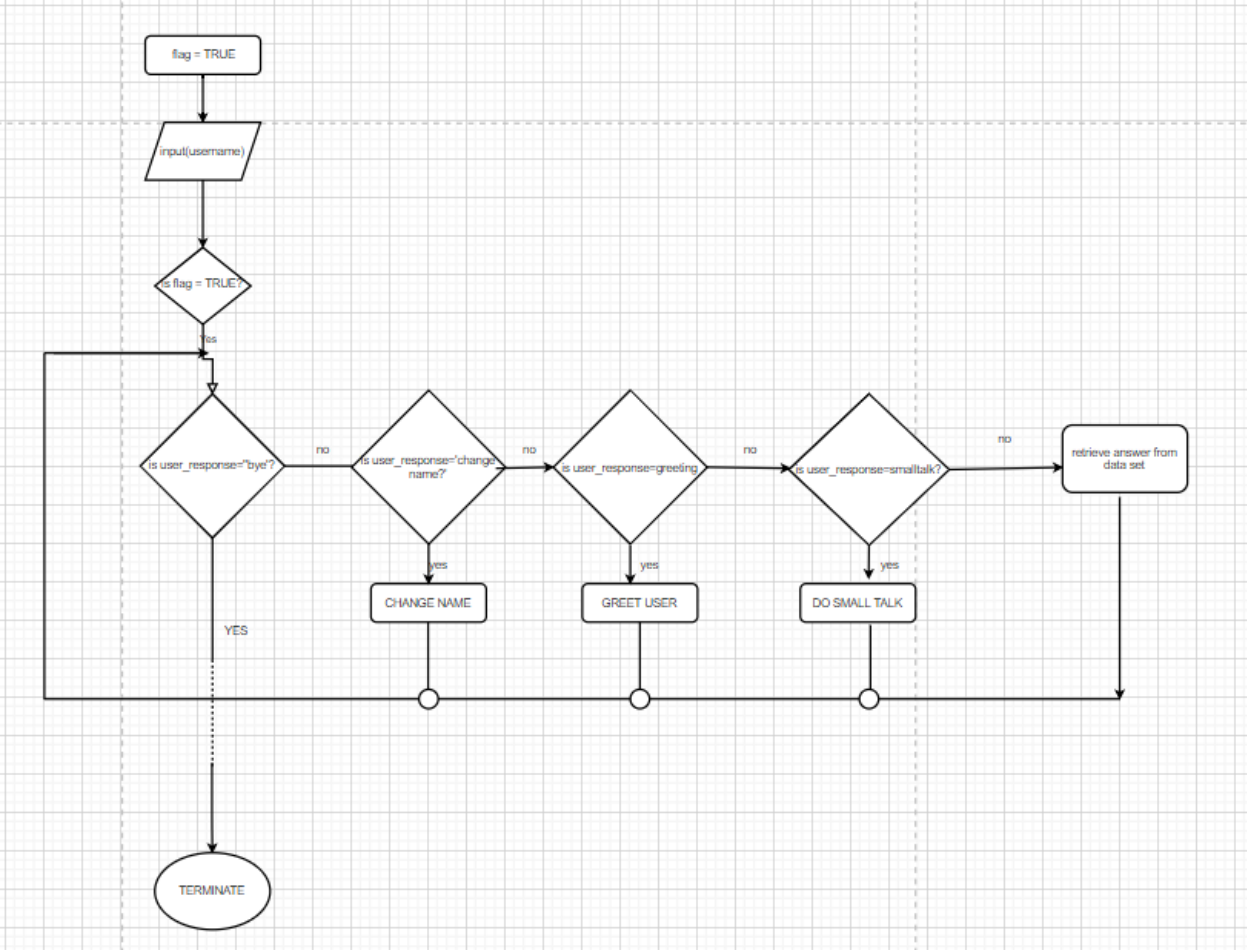

Figure 1

Here I tried to minimize the amount of compulsory variables, there is only one compulsory variable. The only compulsory variable is the user's name at the beginning of the interaction. This gave the chatbot a more natural feel during the flow of conversation, this means that the user can switch intent at any given point during the conversation with the chat bot after he/she has already input her name.

Punkt tokenizer (For English) contained within the NLTK data package;

- Removing **Noise**
- Removing the **Stop words**.
- **Stemming**
- **Lemmatization**

I used ***WordNet*** Lemmatizer Wordnet - freely and publicly available lexical database for the English language aiming to establish structured semantic relationships between words

I used **Tfidfvectorizer** to obtain the tf-idf scores.

a. Intent matching:

Intent is all about what the user wants to get out of the interaction. Intent matching detects the intent of the user. Intent is detected by observing the nature of the user's input. If the nature of the user's reply is along the lines of small talk, then a small talk response would occur for each small talk input. If the user wants to know the answer to a particular question (within the scope of the Chat bot's capabilities), the chat bot will detect a question answering intent and thus return the relevant response. It does this by recognizing entities within the user's utterance. This way the user's motive will be aligned with the chat bot's response. This was made possible by the use of if, elif, else statements.

b. Name management:

The first thing the user can input is his/her name. We call this compulsory input as it is a piece of information that the user is expected to provide and the chatbot would not function properly without it. After the user keys in their name, It is then stored. If the user wishes to change his/her name. They simply need to type in the command 'change name', 'modify name' or 'edit name' at any time. This will then ask the user for his/her preferred name in order to change it, the new name is then saved. This is handled by the ChangeName () function.

c. Small talk:

A keyword matching technique was used, I went ahead and divided 'small talk' into 'greetings' and 'small talk inputs'. As is common knowledge, some greetings do share replies with other types of greetings. For example, one can reply to a 'hey' with a 'hi', 'hey' or a 'hello' among many other examples. The function therefore chooses a random BUT SUITABLE reply and provides the user with a response. The random variation is included in order to inject a feeling of variation as would be in a real-life conversation.

d. Question answering:

If the user input is neither Small talk nor is it Name Management, it's probably a Question answering intended input. The user's input it ran against the words in the corpus and it is mapped to a potential solution based on similarity. The *TfidfVectorizer* converts a collection of raw documents into a Matrix with the TF-IDF scores. Similarity calculated by the cosine similarity measure, obtained from *'from sklearn.metrics.pairwise import cosine_similarity'*. The subsequent question's answer is then returned for the user to see, the question and answer should be of the same index for ease of retrieval.

2. Evaluation

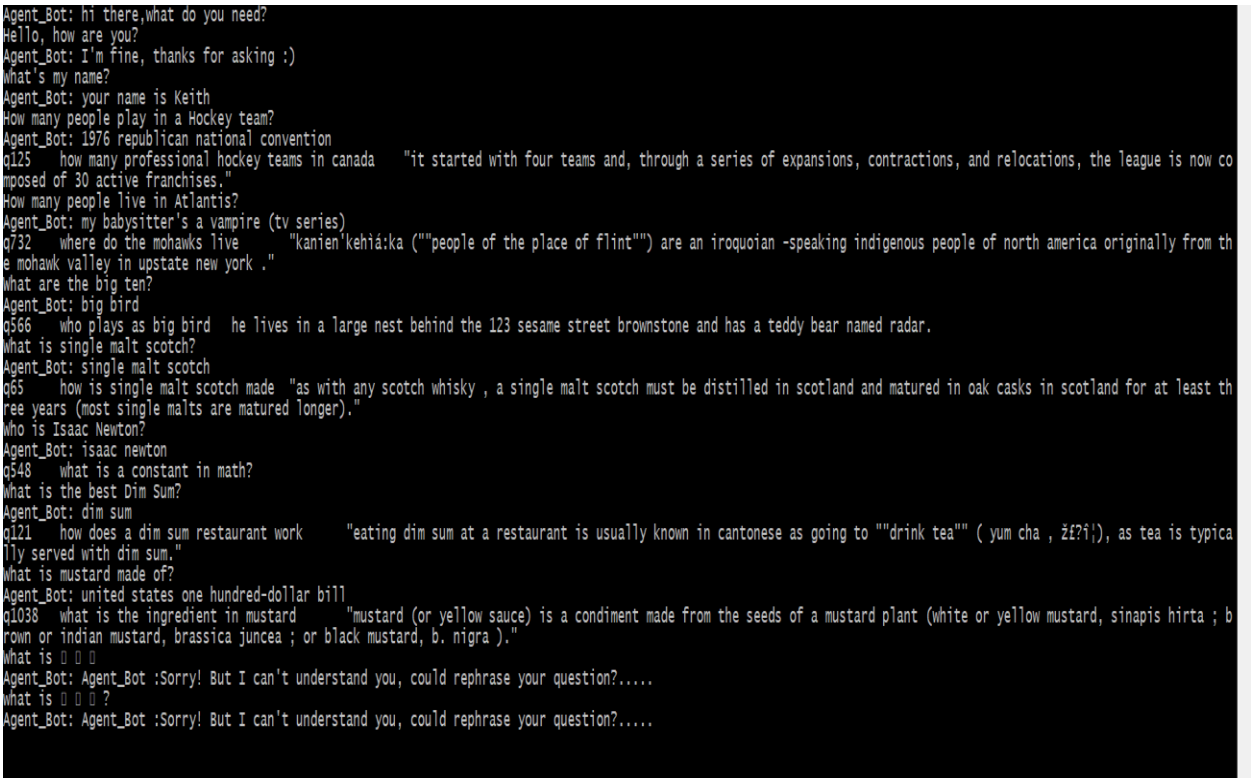

Figure 2(Test Query results)

TEST QUERY	**ANY ANSWER? YES OR NO**	**WAS THE ANSWER SATISFACTORY?**
Hello, how are you?	YES	YES
What's my name?	YES	YES
How many people play in a Hockey team?	YES	NO – But answer was Hockey related
How many people live in Atlantis?	YES	NO – Answer had nothing to do with the question
What are the big ten?	YES	NO – Answer had nothing to do with the question
What is single malt scotch?	YES	NO – But answer was Single Malt related
Who is Isaac Newton?	YES	NO – But answer was Math related
What is the best Dim Sum?	YES	NO – But answer was Dim Sum related
What is mustard made of?	YES	NO – But answer was Mustard related
What is 奶茶？	NO	NO Answer

a. Functionality:
The model functions as intended. However, the model is of limited functionality in that it can only perform basic small talk, data retrieval from a quite limited dataset and basic name change operations. It still has a limited understanding of a variety of sentence structures and grammatical implications.

b. Performance:
In terms of data retrieval from the data set, the chat bot does produce the same answers. However, should a typo or grammatical errors be introduced in terms of small talk, greeting and name changing, introducing a typo would result in the chatbot not understanding the input.

c. Affect:
Despite limited small talk ability. The small talk answers feel fairly natural. In other words, the chatbot does not feel completely natural, but an element of naturalness is felt within the slight variation of replies (for the same user input). When the user enters a greeting input for example, the chat bot returns a random response picked from a list of viable replies. Doing this (introducing a partially randomized response) makes the chatbot somewhat unpredictable and not too one dimensional. Not to mention, the user can switch intent at any given point in the conversation with the agent.

d. Short comings:
In order for the user to do a name change, they need to type in the exact command associated with it, any typo or error in grammar would mean that the chat bot would not understand what exactly the user intended/wanted.
A few question answering inputs by the user bring less suitable results.
Despite the conversations feeling somewhat natural, they are still not as natural as can be, which shows room for improvement.
While looking for a response in the data set, sometimes the model returns the last few phrases of the previous index's answers. Tried to find a solution, but no avail thus far.

e. How I would improve the system given greater resources:
First, I'd fix the name change command to be a bit more flexible. In other words, to make it possible for users using the chatbot to have a greater variety of ways of expressing themselves mimicking real life conversational tendencies and not just following one strict and rigid path.
Then, I would introduce a lot more training data for all instances small talk, name changing and question answering included. This would greatly bring it closer to real life conversation.
Incorporate the use of graphical user interface in order to boost the general appearance and experience while interacting with the conversational agent.
Incorporate the use of Sentiment analysis in order to get a firmer grip on the user experience.

STEP BY STEP GUIDE:

1. **DO THE IMPORTS FOR NECESSARY LIBRARIES**

```
#Import relevant libraries
import io
import warnings
warnings.filterwarnings('ignore')
import nltk
import numpy as np
import random
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.stem import WordNetLemmatizer
nltk.download('punkt') #first-time use only
nltk.download('wordnet') #
```

Figure 3

2. **READING THE CORPUS.**

```
#Reading the corpus:
f=open('Dataset.txt','r',errors = 'ignore')
raw=f.read()
raw=raw.lower() #convert to lowercase
```

Figure 4

3. **TOKENIZATION.**

```
sent_tokens = nltk.sent_tokenize(raw)#list of sentences
word_tokens = nltk.word_tokenize(raw)#list of words
```

Figure 5

4. **RE-PROCESSING**

Text data is usually in text format, while only numerical features are allowed.
To make the data more suitable to the task we need to convert the text to all lowercase characters for

uniformity in order for them to be treated the same.
We then go ahead and tokenize the text into a list of tokens, we tokenize both sentences and individual words.

```
#PRE-PROCESSING:

#Handles Lemmatisation of tokens
lemmer = nltk.stem.WordNetLemmatizer()
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)

#Handles Lemmatisation of text
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

Figure 6

5 **KEYWORD MATCHING –** this is for the generation of greeting and small talk responses**.**

```
#Handles the responses for any greetings (keyword matching)
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi, how can I help you?", "hey, what do you wanna know?", "What up? How can I help", "hi there,what do you need?", "hello, wha

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

#Handles the responses for any small talk *beyond greetings* (keyword matching)
def smaltalk(sentence):
    SMALLT_INPUTS = ("Hello, how are you?", "what are you doing?", "what is your name?", "what is my name?" )
    SMALT_RESPONSES = ["I'm fine, thanks for asking :)", "Trying to help you out :) ", "My name is Botty", "your name is "+user_name ]
    if sentence.lower() in SMALLT_INPUTS:
        index = SMALLT_INPUTS.index(sentence)
        return SMALT_RESPONSES[index]
```

Figure 7

6. **RESPONSE GENERATION – BAG OF WORDS, TF-IDF, COSINE SIMILARITY-**
   First, we transform text to a vector of numbers to represent occurrence.
   We then use the TF-IDF approach to prevent common and useless words and phrases from being prioritized.
   The concept of cosine similarity is used in generating the response, in that the user's utterance

is searched for any keywords.

```
#Handles the general responses by similarity
def response(user_response):
    Agent_response=''
    sent_tokens.append(user_response)
    Tfiidf_vect = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    Tfiidf = Tfiidf_vect.fit_transform(sent_tokens)
    Values = cosine_similarity(Tfiidf[-1], Tfiidf)
    Index=Values.argsort()[0][-2]
    flat = Values.flatten()
    flat.sort()
    req_Tfiidf = flat[-2]
    if(req_Tfiidf==0):
        Agent_response=Agent_response+"Agent_Bot :Sorry! But I can't understand you, could rephrase your question?....."
        return Agent_response
    else:
        Agent_response = Agent_response+sent_tokens[Index]
        return Agent_response
```

Figure 8

7. **MAIN BODY** here we define the chatbots opening and closing statements.

```
flag=True

print("Agent_Bot : My name is Agent_Bot. What is your name?. If you want to exit, type Bye!")
user_name = input()

print("Agent_Bot : Hi! "+user_name)

#Execute only while flag=true
while(flag==True):
    user_response = input()
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            print("Agent_Bot: You are welcome..")
        elif(user_response=='change name' or user_response=='modify name' or user_response== 'edit name'):
            ChangeName()
        elif(greeting(user_response)!=None):
            print("Agent_Bot: "+greeting(user_response))
        else:
            if(smaltalk(user_response)!=None):
            print("Agent_Bot: "+smaltalk(user_response))
            else:
                print("Agent_Bot: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("Agent_Bot: Farewell! take care of yourself..")
```

Figure 9