HAI Coursework 1 Report

Yee Rong Jian

Student ID: 14333177

Word count:1331

Contents

[Design of the system 2](#_Toc57217045)

[Getting Similarity 2](#_Toc57217046)

[Intent matching 2](#_Toc57217047)

[Name management 3](#_Toc57217048)

[Small talk 3](#_Toc57217049)

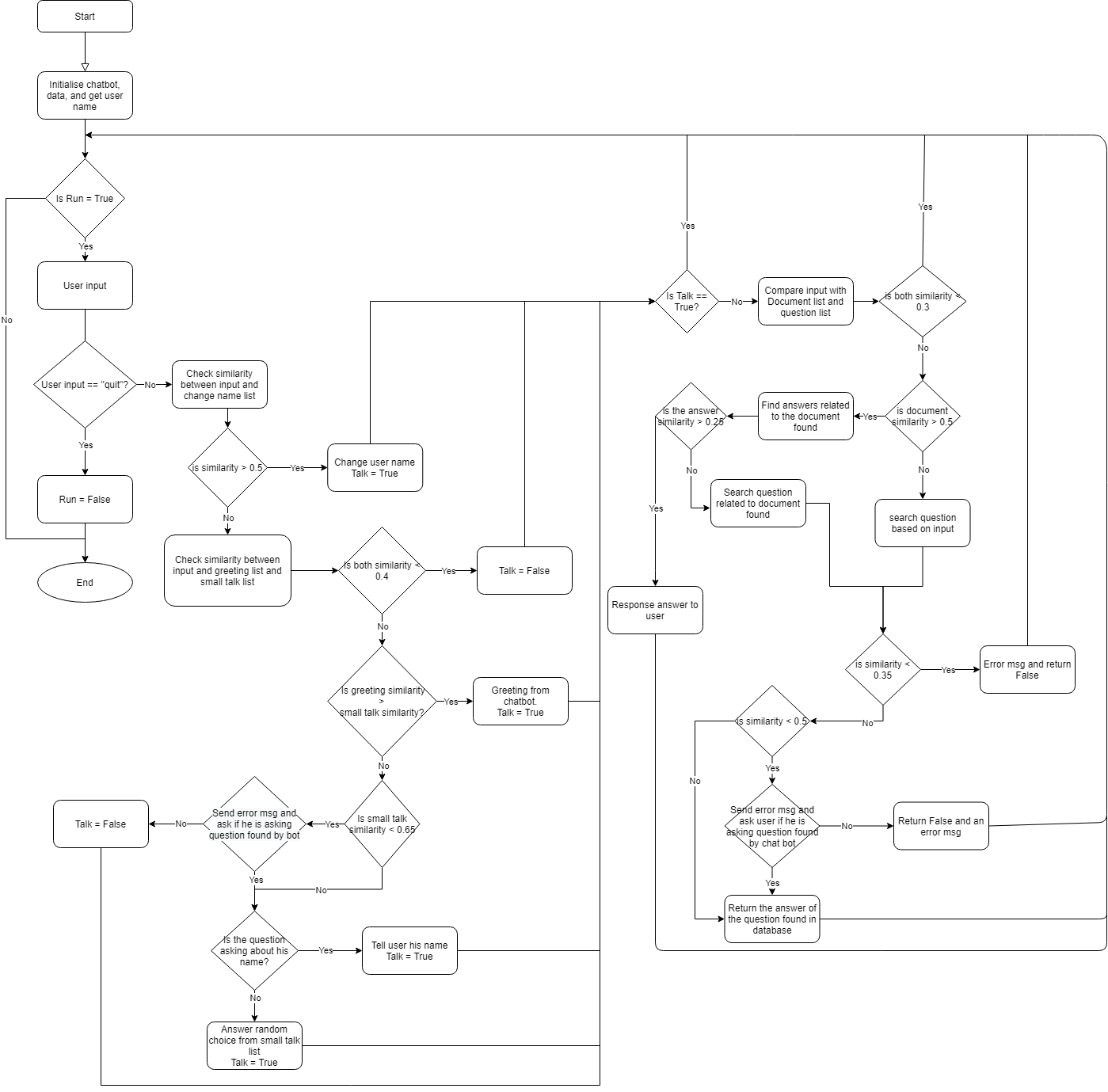
[Question answering 4](#_Toc57217050)

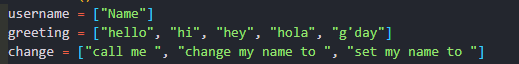
[Evaluation 4](#_Toc57217051)

[Functionality 4](#_Toc57217052)

# Design of the system

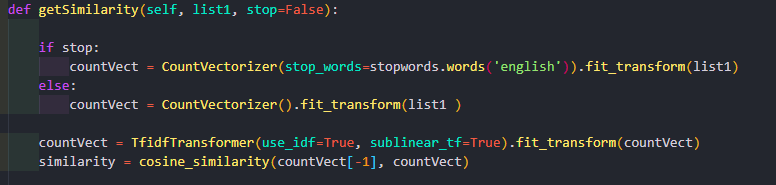
The design of this chat bot is relatively simple as the functions required and implemented are not very complicated. The following flow chart represented the basic algorithm and decision making that the program does.

Implementing and designing this chat bot, I have opted for creating multiple pre-defined list that contains related phrases that later on will be used to compare and find similarities with it.



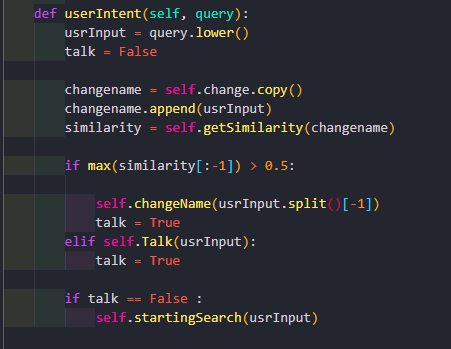
## Getting Similarity

To get similarity between the input and the list. First of all, I append the user input into the list and then feed it to the “getSimilarity” function. Within the function, count vectorizer is used to create a term document matrix. The matrix is then feed into “TfidfTransformer” to apply transformation on the matrix using their IDF weight. At last, I get the similarity by comparing the last item in the matrix which is our user input and the rest of the matrix by using cosine similarity function provided by sci-kit learn.



## Intent matching

What I chose to do with intent matching is that, after the user input their queries. The input will go through a series of comparing with a few pre-defined lists in the system. First of all, it will compare it with the “change name” list, if the query and the list is not similar it will move on to the next list. If the query is similar to the small talk list, then it will perform small talk. However, if the query fail to meet the minimum similarity of all the pre-defined list, then it will then be pass to the last intent which is the question answering part where if will perform a series of comparing that will be further discussed in the ‘Question answering’ section.

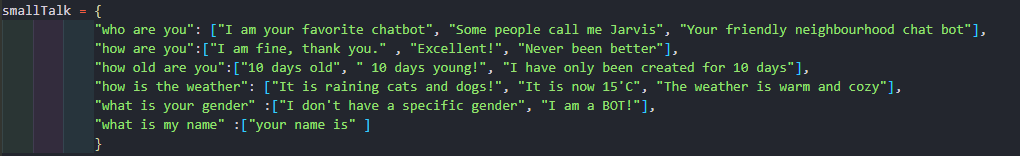


## Name management

In python string is immutable, which does not fit our purpose in the program. Therefore, I decided to use a list which only contain one element to store username. When the program starts running, the first thing that the program do is to prompt user to input their username. The program will take the last word of the input as their username. For example, “My name is Jason”. This will work as long as the user end the sentence with their desired name. To change username, the user input will be comparing to a few pre-defined phrases. If the input has a relatively high similarity rate to phrases like “change my name to”, “set my name” etc. Then it will trigger the change name function which will also take the last word of the sentence as the new username.

## Small talk

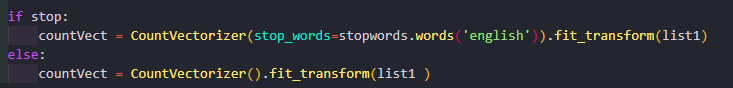
For implementing small talk intent, I create a dictionary containing a few questions as the keys and a few difference answers lists as the values shown in the following screenshot.



I have decided to include the question “what is my name” inside of small talk session instead of name management section is because asking for name is more of a bot answering situation than changing or storing value in the memory. So, when the user input has a relatively high similarity with any of the keys in the small talk dictionary, it will then respond with a random choice in the list of answers. I have set a relatively high threshold for the answering small talk because if the threshold is low, question like “what is mustard” is very likely to have a similarity value greater or equal to 0.5 as the it includes question word like “what is”, “who is” and “how are” and they are very common. Therefore, setting the threshold value as 0.65 will prevent it from answering questions are intended to be answer in the question answering section.

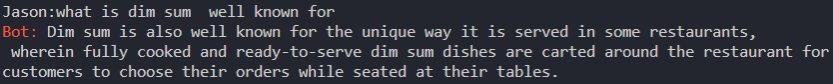
## Question answering

When all the above intent does not return true, the program will feed the user input to the search function. At the start of the function, the program will first compare the input with the document and the question in the data base. When comparing to the document, I have decided to remove all the stop words in the input so that unnecessary words that will affect the similarity score will be remove. I achieved this by passing a parameter to decide whether the function will remove stop words or not.



In other situation where “getSimilarity” is use for example, comparing input and the answer list, removing stop words is not beneficial as it will remove words that is essential to the sentence and therefore lowering the probability of finding correct answer or question.

If the similarity scores of both document and question are lower than 0.3, it will return with an error message saying that it will not be able to answer the question. If the similarity score of with the document list is greater than 0.5, the program will search answer according to the input and only response if the highest similarity of the answer is greater than 0.25 otherwise, it will then search question according to the question. The reason of doing this is because there will be cases where the answers in the data base is related to the user input but not necessarily to the question. For example, user input is ‘what is dim sum well known for’.



However, in the data base, the corresponding question for this answer is actually “how does a dim sum restaurant work”. If I do not compare similarity with the answer, it will just return an error message.

Last but not least, in all other cases, the program will just directly compare the input with the question and get the most similar question and then find answer according to the question found. When the highest similarity score found is less than 0.5 the program will return a error message following with a question found by the program that is most similar to the input and ask user if it is the one they are asking.

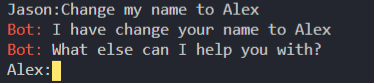
# Evaluation

The following screenshots show how the program react when asking certain question selected from the test queries.

## Functionality

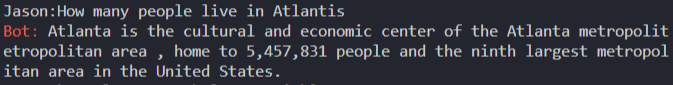
The program works as expected, answering question that can be found in the database and not answering question that are not. The name of the user is memorised properly by the system and can be changed throughout the session. The name of the user is at the front of the line when user is inputting, and to make it clear I have made the colour of the word ‘Bot’ red so that is easy to distinguish.









When the system does not understand what the input is, it will return a question for the user asking whether he meant this. This is effective when the user has grammatical mistakes and typo. However, the system can only deal with small amount of errors in the input. 

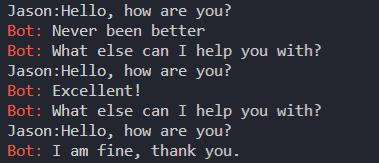
Due to the implementation of the system using similarity and not a classifier, there will be some errors when answering question. In this case, the system should not answer this question as Atlanta and Atlantis two different word, but the two words is very similar therefore the system will give an answer to it.



When the input contain characters that are not English, the system will return a error message and will let user enter again until the input does not contain any unsupported language.

## Affect

The screenshots below show an example of the small talk. It is performing very well that the system will return a random choice of answer associate with the small talk question. Due to the limited data in the small talk list, the system is only able to answer a certain number of questions in the small talk section. Therefore, the system cannot answer any question that are not in the list. Nevertheless, the small talk does feel natural.



To improve this system, I think it would be suitable to train a neural network and text classifier with a bigger dataset. So that, the performance can be maximise instead of simply using similarity to find question and answer.