RETRIEVAL-BASED BIOMEDICAL CHATBOT

A project report submitted to the Jawaharlal Nehru University in partial fulfillment of the requirements for the award of the degree of

MASTER
OF
COMPUTER APPLICATIONS

BY PRAGYANUR SAIKIA
21/10/JC/005

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

June 2023

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU
UNIVERSITY NEW DELHI
– 110067

DECLARATION

This is to certify that the project entitled “**RETRIEVAL BASED BIO-MEDICAL CHATBOT”** is being submitted to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of the degree of **Master of Computer Applications**, is a record of bonafide work carried out by me.

The matter embodied in the dissertation has not been submitted in part or full to any University or Institution for the award of any degree or diploma.

PRAGYANUR SAIKIA

Certificate

This is to certify that **Pragyanur Saikia** (21/10/JC/005) has submitted the project report titled **“Retrieval-Based Biomedical Chatbot”**. He completed the project by June 2023 under the mentorship of **Dr. Aditi Sharan**, in partial fulfillment of the requirement for the award of a degree of MCA from Jawaharlal Nehru University, New Delhi.

ASharan

Dr. Aditi Sharan

SCSS, JNU

Acknowledgement

I would like to express my sincere gratitude and appreciation to Dr. Aditi Sharan for her invaluable guidance and mentorship throughout the course of this project. Her expertise, encouragement, and insightful suggestions have significantly enriched my understanding of the subject matter and have played a crucial role in shaping the outcomes of this project and led to its successful completion.

I would also like to extend my heartfelt thanks to all the students and research scholars under Dr. Aditi Sharan's supervision. Their enthusiasm and willingness to share their knowledge and experiences have not only enhanced the quality of the work but also fostered a stimulating research environment.

I am grateful to the entire research team for their constant support and constructive discussions. Their commitment, diligence, and expertise have been instrumental in overcoming challenges and achieving the project objectives.

Finally, I would like to express my deepest gratitude to my family for their unconditional love, understanding, and unwavering support. Their belief in me has been a constant source of inspiration and motivation.

Abstract

This project presents the design and development of a retrieval-based chatbot, specifically for the medical domain. The chatbot aims to assist users in retrieving relevant information related to symptoms, causes, preventions, risk factors, complications, diagnosis-and-treatment options, epidemiology, prognosis, and the research progress in a conversation-like and user-friendly manner. The chatbot utilizes natural language processing techniques and machine learning to understand user queries and retrieves appropriate responses from a comprehensive medical knowledge base.

The chatbot employs a retrieval-based approach, which involves matching user queries to a database of predefined questions and answers. Through the use of machine learning algorithms, the chatbot is trained to recognize patterns in individual user inputs and provide accurate and contextually relevant responses. The chatbot's language model is fine-tuned on a knowledge base of medical literature and resources to ensure accuracy and up-to-date information retrieval.

Users can interact with the chatbot using the common English language, without the need for specialized medical terminology. The system is designed to handle a wide range of medical queries, including inquiries about symptoms, treatment options, causes, and the latest research findings. The chatbot's responses are carefully crafted to be informative, concise, and easily understood by users of various medical knowledge levels.

The development of the chatbot involved comprehensive requirement analysis, data collection, model training, and evaluation. User feedback and iterative improvements were incorporated to enhance the chatbot's performance and user experience. The chatbot's interface is designed to be simple, intuitive, and user-friendly, enabling users to quickly obtain the desired medical information.

The retrieval-based medical chatbot demonstrates results that are quite motivating in providing relevant medical information to users. The system has the potential to assist individuals seeking medical advice, research, or general information in a free, accessible, and convenient manner.

Table of Contents

1. INTRODUCTION

1.1. About Chatbots

Chatbots are computer programs that simulate human conversation through text or voice interactions. They are designed to automate tasks and provide information to users. Chatbots can be integrated into various platforms, such as websites, messaging apps, and mobile apps.

There are two main types of chatbots: rule-based chatbots and machine-learning (retrieval-based, generation-based, and hybrid) chatbots. Rule-based chatbots are programmed with a set of rules that define how they should respond to user input. Machine learning chatbots, on the other hand, learn to respond to user input by analyzing large amounts of data.

Chatbots are becoming increasingly popular as they offer a number of benefits to businesses and users. For businesses, chatbots can provide 24/7 customer service, reduce costs, and increase sales. For users, chatbots can provide convenience, personalization, and entertainment.

The field of chatbot theory is still in its early stages, but it is growing rapidly. As chatbots become more sophisticated, they will become increasingly important in our lives.

Here are some of the key concepts in chatbot theory:

Natural language processing (NLP) is the field of computer science that deals with the interaction between computers and human (natural) languages. NLP is essential for chatbots, as it allows them to understand and respond to human language.

Machine learning is a field of artificial intelligence that allows computers to learn without being explicitly programmed. Machine learning is used in chatbots to enable them to learn from data and improve their responses over time.

Dialogue management is the process of controlling the flow of conversation between a chatbot and a user. Dialogue management is essential for chatbots, as it ensures that the conversation is natural and productive.

1.2. Project Overview

This section gives an overview of how a retrieval-based chatbot in the biomedical domain is made and the objectives of the system. Major use cases include- immediate knowledge retrieval and clarifying confusion regarding a disease. When it comes to medical supervision, it is not always possible to get first-hand medical advice. Here is where the use of a handy bio-medical chatbot can provide great help. If we doubt that we have a disease, or need knowledge about it then we can use this chatbot to retrieve answers to our queries.

The project aims to develop a retrieval-based chatbot specifically designed for the biomedical domain. The chatbot leverages natural language processing techniques to provide users with accurate and relevant information related to various biomedical topics. The project's resources and codebase are available in the GitHub repository at https://github.com/Pragyanur/retrieval_based_biomedical_chatbot.

The retrieval-based chatbot system is built upon a comprehensive biomedical knowledge base that includes a wide range of information, including symptoms, treatments, causes, research findings, and more. The chatbot utilizes machine learning algorithms and advanced natural language processing models to understand user queries and retrieve the most appropriate responses from the knowledge base.

The main objectives of the project are as follows:

1. To study and explore the framework of chatbot-based systems.
2. To implement retrieval-based chatbot system.
3. To gather data related to causes, symptoms, preventions, treatments, complications, etc. of various diseases and medical conditions.
4. To study and explore various Natural Language Processing Techniques for preprocessing like- lemmatization, tokenization, and named entity recognition.
5. To implement machine-learning models on the above defined objectives.
6. To evaluate the models performance using different performance parameters such as loss and accuracy.

1.3. Existing systems

Retrieval-based medical chatbots use a knowledge base of medical information to answer user queries. They are typically trained on a large corpus of text data from sources, such as medical journals or patient records, and can be used to provide information on a wide range of medical topics.

Some of the **benefits** of retrieval-based medical chatbots include:

1. They can provide 24/7 access to biomedical information, which can be helpful for patients who have questions or concerns outside of regular business hours.
2. They can help patients to better understand their medical conditions and treatment options.
3. They can reduce the burden on healthcare providers by answering common questions and providing basic medical advice.

However, there are also some **limitations** to retrieval-based biomedical chatbots:

1. They are not able to provide personalized medical advice, and should not be used to replace the care of a healthcare provider.
2. They may not be able to answer all medical questions, especially those that are complex or rare.
3. They can be susceptible to errors, such as providing incorrect or outdated information.

Despite these limitations, retrieval-based medical chatbots can be a valuable tool for patients and healthcare providers. They can provide convenient and easy access to medical information and can help to improve patient education and care.

Here are some examples of existing retrieval-based medical chatbots:

1. **IBM Watson Health Assistant** is a chatbot that can provide information on a wide range of medical topics, including symptoms, diagnosis, treatment, and prevention. It was released on **February 14, 2011.**
2. **MD Chat** is a chatbot that can answer questions about a variety of medical conditions, including allergies, asthma, and diabetes. It was released on **January 14, 2019.**

1.4. Proposed system

The proposed system is a retrieval-based biomedical chatbot that can answer questions related to diseases, such as their causes, symptoms, preventions, and many more. The chatbot's architecture is designed to efficiently process user queries, retrieve accurate information from the knowledge base, and generate contextually relevant responses.

The system architecture is composed of three major components:

1. **Building and training a multiclass classifier:** This component builds and trains a multi-class intent classification model and saves it for future use. It uses bag-of-words from question patterns of corresponding intents for training.

2. **Response retrieval engine:** This component retrieves a response from the knowledge base by using pre-trained models to predict the parameters from the user input.
3. **Graphical User Interface (GUI):** This component provides an interface to interact with the chatbot application. Display the conversation by using functions to retrieve chatbot responses.

The system has the potential to be a valuable tool for patients and healthcare professionals. The system can be used to improve the quality of care for patients and to reduce the cost of healthcare. The architecture is scalable, allowing for future enhancements and integrations to meet evolving user needs and technological advancements.

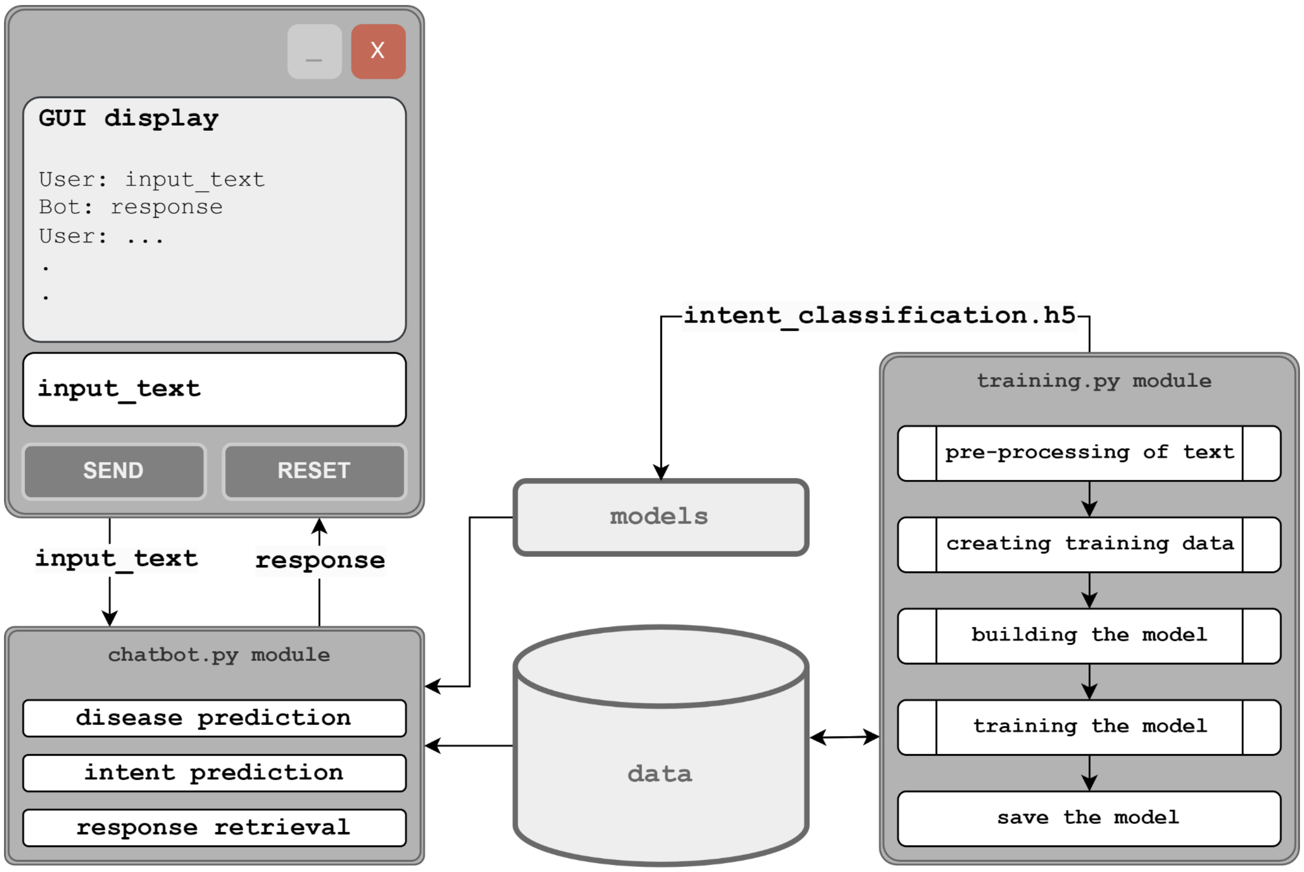


Figure: Project flowchart (brief)

2. ANALYSIS

2.1. Requirement analysis

During the requirement analysis phase, the goal is to gather, analyze, and document the specific functional and non-functional requirements of the retrieval-based biomedical chatbot. This phase involves understanding the needs and expectations of the stakeholders and translating them into clear and actionable requirements. The following aspects are considered in the requirements analysis:

2.1.1. Functional Requirements:

- **Natural Language Processing:** The chatbot should be capable of understanding user queries written in the common English language and extracting the relevant information.
- **Information Retrieval:** The chatbot should retrieve accurate and up-to-date information from the biomedical knowledge base based on user queries.
- **Query Understanding:** The chatbot should comprehend the context, intent, and semantics of user queries to provide meaningful responses.
- **Contextual Response Generation:** The chatbot should generate contextually relevant and coherent responses based on the retrieved information and user queries.
- **Personalization:** The chatbot should be capable of adapting its responses based on user preferences and historical interactions.

2.1.2. Non-functional Requirements:

- **Performance:** The chatbot should respond to user queries promptly, with minimal latency. It should be able to handle a large number of concurrent user interactions efficiently when deployed through the internet.
- **Accuracy and Reliability:** The chatbot's responses should be accurate and reliable, providing trustworthy information to users. It should acknowledge its limitations.
- **Security and Privacy:** The chatbot should handle user data and queries securely, ensuring the confidentiality and privacy of user information.
- **User Interface and User Experience:** The chatbot's interface should be user-friendly and intuitive. It should provide a seamless user experience.
- **Compatibility:** The chatbot should be compatible with desktop/laptop computers.
- **Maintainability and Extensibility:** The chatbot should be designed and implemented in a modular manner, allowing for future updates and enhancements to the system.

- **Documentation and Manual:** The chatbot should be accompanied by comprehensive documentation and a user manual to assist users in utilizing the system effectively.

2.2. Feasibility

The economic feasibility of the retrieval-based biomedical chatbot project is minimal. It can be run on any desktop/laptop computer for free. The cost of developing and deploying this chatbot is very less. The Python packages and knowledge bases that have been used come free of cost. The development environment used is Visual Studio Code, which is also free software.

Benefits:

- The chatbot can provide people with accurate and up-to-date information about diseases and medical conditions for free.
- The chatbot provides immediate information based on the query in the general English language.
- Providing this information to patients can help them make better decisions about their health care.

Technical advantages:

- A lightweight application is available which is free and open-source.
- Does not require much processing power as it uses a pre-compiled model and other techniques to reduce redundant processes.
- Its modular design facilitates updates and enhancements to the application.

2.3. Behavior and limitations

The chatbot application starts with an introduction text along with a “How to use?” manual. It provides the information that this chatbot can retrieve the following information about a set of 17 diseases (arthritis, lung cancer, HIV, breast cancer, and various other diseases):

- Cause
- Complications

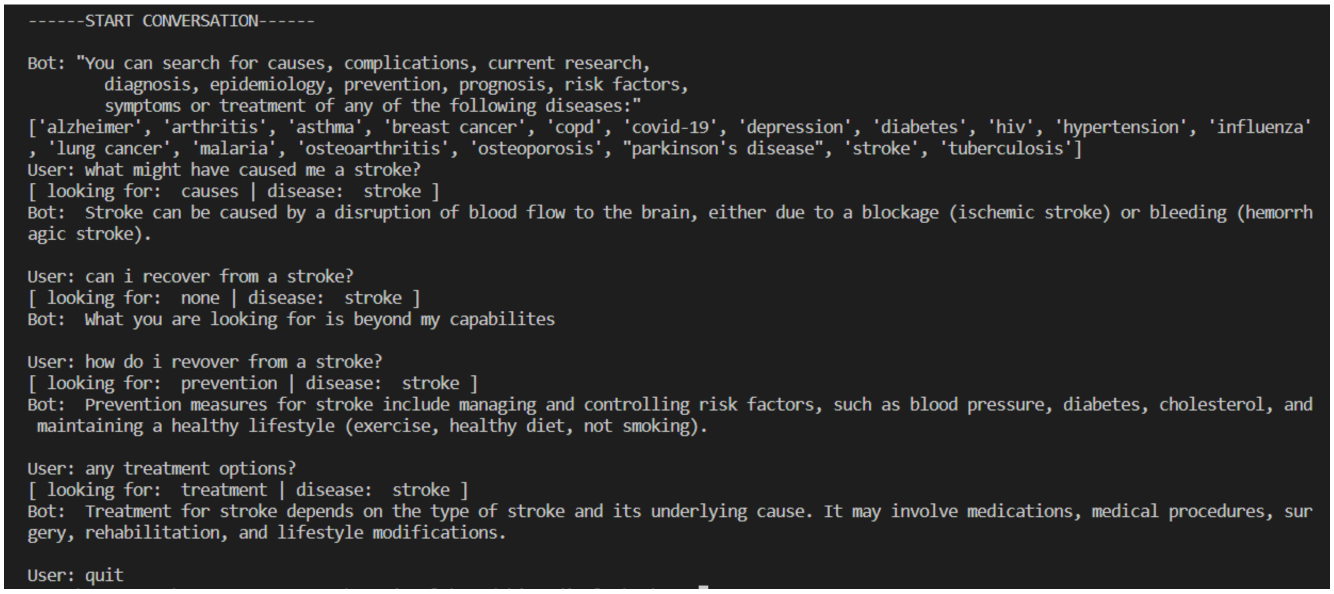
- Current research
- Diagnosis
- Epidemiology
- Prognosis
- Risk factor
- Symptoms
- Treatments
- None (from the above)

The following use-case situations are handled in the implementation of the chatbot:

- The chatbot is able to guess the context of the query by storing the most recently accessed disease in the memory of the current session/conversation.
- If the chatbot is not able to classify the input text into any meaningful intent, it matches none of them and lets' the user know that it is not capable of the users' demands.
- If data on a disease is not present and the user is trying to access its information, then the system lets the user know that the database does not contain that specific disease or the medical condition's information.
- A single conversation session can be cleared by clicking the "RESET" button. The chatbot's terminal-based application (defined as *chat()* function in the "chatbot.py") can be exited by typing "quit" or "exit" as input.

2.4. Use cases

The following output demonstrates the limitation of this chatbot and a few of the above situations:

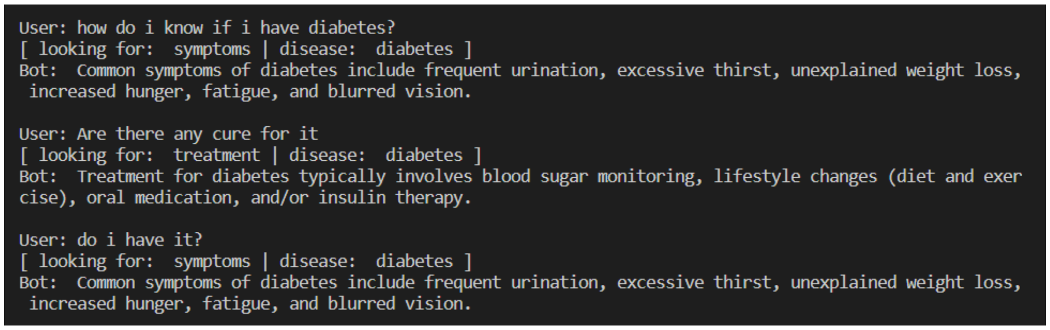


```
------START CONVERSATION------

Bot: "You can search for causes, complications, current research,
        diagnosis, epidemiology, prevention, prognosis, risk factors,
        symptoms or treatment of any of the following diseases:"
['alzheimer', 'arthritis', 'asthma', 'breast cancer', 'copd', 'covid-19', 'depression', 'diabetes', 'hiv', 'hypertension', 'influenza'
, 'lung cancer', 'malaria', 'osteoarthritis', 'osteoporosis', "parkinson's disease", 'stroke', 'tuberculosis']
User: what might have caused me a stroke?
[ looking for:  causes | disease:  stroke ]
Bot:  Stroke can be caused by a disruption of blood flow to the brain, either due to a blockage (ischemic stroke) or bleeding (hemorrh
agic stroke).

User: can i recover from a stroke?
[ looking for:  none | disease:  stroke ]
Bot:  What you are looking for is beyond my capabilites

User: how do i revover from a stroke?
[ looking for:  prevention | disease:  stroke ]
Bot:  Prevention measures for stroke include managing and controlling risk factors, such as blood pressure, diabetes, cholesterol, and
 maintaining a healthy lifestyle (exercise, healthy diet, not smoking).

User: any treatment options?
[ looking for:  treatment | disease:  stroke ]
Bot:  Treatment for stroke depends on the type of stroke and its underlying cause. It may involve medications, medical procedures, sur
gery, rehabilitation, and lifestyle modifications.

User: quit
```

Figure: Output 1

When the user is asking about “recovering from a stroke” or looking for “treatment of a stroke”, the system predicts a wrong intent class based on the text. This shows the inaccuracy of the model in predicting the correct intent (class).

The chatbot is able to understand the intent of the user for queries in standard spoken English. Following are some examples:

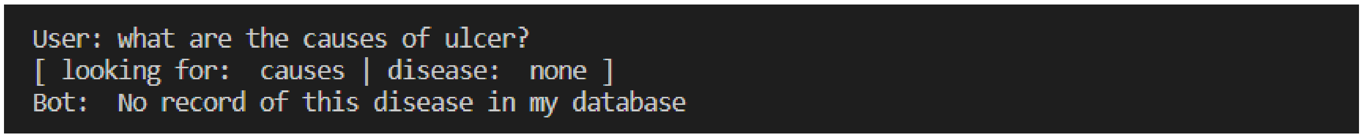


```
User: how do i know if i have diabetes?
[ looking for:  symptoms | disease:  diabetes ]
Bot:  Common symptoms of diabetes include frequent urination, excessive thirst, unexplained weight loss,
 increased hunger, fatigue, and blurred vision.

User: Are there any cure for it
[ looking for:  treatment | disease:  diabetes ]
Bot:  Treatment for diabetes typically involves blood sugar monitoring, lifestyle changes (diet and exer
cise), oral medication, and/or insulin therapy.

User: do i have it?
[ looking for:  symptoms | disease:  diabetes ]
Bot:  Common symptoms of diabetes include frequent urination, excessive thirst, unexplained weight loss,
 increased hunger, fatigue, and blurred vision.
```

Figure: Output 2

In the above segment of a conversation, we can see that the user enters their query in standard spoken English, that, which one would use while communicating with a medical professional. Yet, the chatbot is able to understand and classify the intent of the query.

If a user asks the chatbot about a disease or medical condition that has no record in the database, it responds in the following manner:



```
User: what are the causes of ulcer?
[ looking for:  causes | disease:  none ]
Bot:  No record of this disease in my database
```

Figure: Output 3

3. DESIGN

3.1. System architecture

The system architecture of the retrieval-based biomedical chatbot defines the overall structure and components of the system, illustrating how different modules and components interact with each other to provide the desired functionality. The architecture ensures the efficiency, scalability, and reliability of the chatbot system. The proposed system architecture for the retrieval-based biomedical chatbot can be described as follows:

1. **Graphical User Interface (GUI):** The GUI component provides the interface for users to interact with the chatbot system. It is a very simple Tkinter window interface that consists of the following functionality:

    - A text area for user query input
    - "SEND" button to send a query
    - Display of conversation
    - "RESET" button to reset the conversation (clears history)

    The GUI allows users to input their queries and receive responses from the chatbot almost immediately. It provides a seamless and intuitive user experience, allowing for easy communication with the system. For each response generated by the chatbot, the terminal provides us with the information- **predicted disease** and **predicted intent**.

2. **Natural Language Processing (NLP) methods:** The NLP functions are responsible for processing and understanding user queries written in the common English language. Several functions from the NLTK library have been used for processing text data with word tokenization and lemmatization. Pre-trained scispaCy models have been used to classify diseases from the user inputs.

3. **Disease Knowledge Base:** "responses.json" consists of the information about a disease's symptoms, causes, treatment methods, prevention, etc. This information is present as responses to the corresponding intent class (or "tag"). It serves as the primary source of information for the chatbot system, containing structured and indexed data that can be efficiently accessed for retrieval.

4. **Response Retrieval module:** The Retrieval Engine or the *bot_response* function is responsible for retrieving relevant information from the Disease Knowledge Base based on user queries. It uses a helper function- *predict_disease* to predict the disease from user input and to keep track of the previously accessed disease during the conversation. The intent is

also classified in this module by using the previously trained multi-class classifier model. Intent and disease help in the successful retrieval of responses by iterating through "responses.json".

5. **Intents Knowledge Base:** Consists of the most common question patterns asked by people to know about the symptoms, cause, preventions, diagnosis, prognosis, complications, risk factors, treatments, epidemiology, and current research about a disease. Each intent comprising of a good number of unique questions to train the model for common language compatibility.

6. **Training module:** The chatbot is trained on a set of the most common questions queried about diseases and medical conditions. The training module converts the questions/patterns from the "intents.json" knowledge base into a set of input and output lists to train a three-layer keras sequential model. The model is then saved as an "intent_classification.h5" file which is used to classify the intent from user input into any one of the following:

    - Symptoms
    - Causes
    - Diagnosis
    - Treatment
    - Preventions
    - Complications
    - Risk factors
    - Epidemiology
    - Prognosis
    - Current research
    - None (of the above)

7. **Integration:** The system architecture allows for integration with external APIs and services in order to improve response generation or information retrieval due to the project's modular approach.

3.2. Components and modules

The project consists of three modules that perform the tasks of:

- Training the model for classifying intents
- Retrieving a response by using two classification models
- User Interface for simplified machine-user interaction

The modules are described below:

1. **Training module:** This module does the following tasks-

    - Create lists from the database “intents.json”, such as words, classes (intents), and documents (consists of a tuple of bag-of-words and its corresponding intent tag). These are stored as pickle files for future use.
    - Convert the documents into a list of training data (training input and training output) for the neural network.
    - Create a Keras neural network model with three layers and their activation functions, optimization algorithm, and loss function to predict the intent accurately from all the classes.
    - Compile the model and save it to “intent_classification.h5”.

2. **Application module:** The following tasks are done by this module-

    - Convert the user input into a bag-of-words which is fed forward to the model for classification.
    - Create a function to predict disease using the SpaCy model for bio-medical named entity tagging. The **“en_ner_bc5cdr_md”** model is used for extracting entities with the label “DISEASE”. This module also keeps the previously accessed disease in memory (so that, not naming the disease in a follow-up query doesn’t result in incorrect responses).
    - Create a function that predicts the intent by using the model “intent_classification.h5” and uses the disease returned by predict_disease to retrieve the response from the “responses.json” database.
    - Creates a loop for a single session conversation that allows the user to enter queries and have the accurate information retrieved by the above functions, displayed through the terminal. A single session can be started by calling the *chat* function. The program can be closed by entering “quit” or “exit”.

3. **GUI module:** Incorporates the application module in providing user-machine interaction by performing the following tasks on the user’s command-

    - Display the introduction and user manual at the start.
    - The SEND button processes the input_text using previous modules to predict disease and retrieve a response.
    - Also, keeps the currently accessed disease in the record by updating the static class context with predict_disease.
    - The RESET button resets the session by clearing the conversation. It re-initializes the context to “none”.

3.3. Data Structures and Database Design

The data structure used for implementing the desired modules in the chatbot is:

1. Lists (strings, tuples, dictionaries)
2. Numpy arrays (binary data and probabilities)
3. Objects (models, files, etc.)

The database is designed in a way that reduces the time of information retrieval. Here, JSON files are used to store the data in a structured manner. The following are the structures of the database used for this project:

1. “intents.json”

```
{
    "intents": [
        {
            "tag": "symptoms",
            "patterns": ["Pattern 1", "Pattern 2","Pattern 3",...]
        },
        {
            "tag": "causes",
            "patterns": ["Pattern 1", "Pattern 2","Pattern 3",...]
        },
        …
       ]
}
```

2. “responses.json”

```
{
    "responses": [
        {
            "tag": "disease_name",
            "intent 1": ["responses"],
            "intent 2": ["responses"],
            "intent 3": ["responses"],
            "...": ["..."]
        },
        {
            "tag": "disease_name",
            "intent 1": ["responses"],
            "intent 2": ["responses"],
            "intent 3": ["responses"],
            "...": ["..."]
        }
    ]
}
```

3.3.1 Knowledge Representation: Logical Representation

Knowledge representation in the chatbot refers to the process of capturing and organizing information in a way that can be effectively utilized by the AI system. It involves selecting suitable structures and formats to represent knowledge in a manner that supports reasoning, problem-solving, and decision-making.

The dataset of responses is stored in such a manner that the retrieval of information is effective. First *disease_name* is predicted from the user input, this tells the system where to search in the knowledge base. It looks for the true value of {"tag"(of the current object in responses) == *disease_name*}. And looks for the *predicted_intent* from the input which guides it to the associated element labeled as *predicted_intent*.

The dataset intents.json forms a one-to-many relation with the dataset responses.json in the following manner:

1. Dataset "intent.json" contains the list of patterns of question for various intent (questions asked by people about diseases). Where, One intent can be called a class that represents the following type of information for one disease:

    a. Symptoms
    b. Causes
    c. Diagnosis
    d. Treatment
    e. Preventions
    f. Complications
    g. Risk factors
    h. Epidemiology
    i. Prognosis
    j. Current research
    k. None (of the above)

2. The dataset "responses.json" contains the list of responses of various intents.

3. Each object in responses contains the above intents corresponding to the disease, named as "tag": "disease_name"

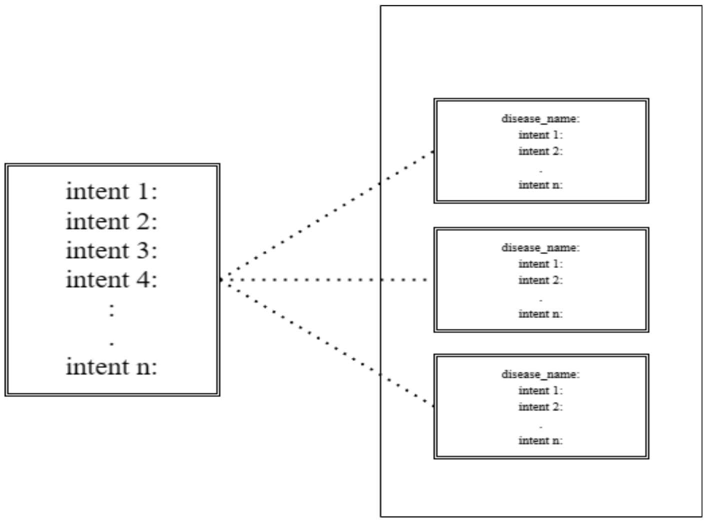


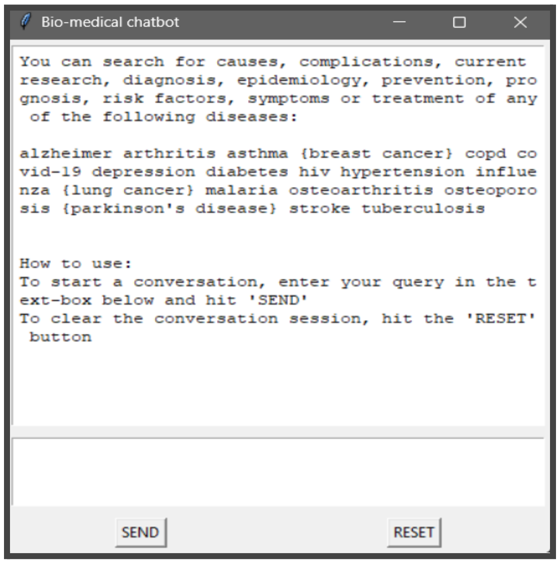
Figure: one-to-many relation between datasets

3.4. System workflow and processes

This section describes the working of the chatbot and the inherent processes that come along with it. The application works in a systematic manner where each process is followed by another towards the completion of the desired goal. Here, the processes that the system employs for the retrieval of a response based on the user's input are elaborated.

3.4.1 System Workflow

When starting the Chatbot, the display section of the GUI provides the user with a brief introduction to its capabilities and limitations. It also presents the "How to use" manual in the starting.



```
You can search for causes, complications, current
research, diagnosis, epidemiology, prevention, pro
gnosis, risk factors, symptoms or treatment of any
 of the following diseases:

alzheimer arthritis asthma {breast cancer} copd co
vid-19 depression diabetes hiv hypertension influe
nza {lung cancer} malaria osteoarthritis osteoporo
sis {parkinson's disease} stroke tuberculosis


How to use:
To start a conversation, enter your query in the t
ext-box below and hit 'SEND'
To clear the conversation session, hit the 'RESET'
 button
```

Figure: Chatbot GUI with the introduction text

The following steps briefly describe the user-system interaction workflow of this chatbot application:

I. The user enters his/her query about any disease, in the input field of the chatbot GUI. The text input is then passed to a response generation function with the click of the "SEND" button.

II. The text data is processed using NLP tools and machine learning models to extract the necessary information. These are:
   - The name of the disease that the user is trying to access the information about.
   - The intent of the type of information the user is trying to get.

III. This information is then used to retrieve responses from the knowledge base of responses. The following situations are handled:
   - If no data about the disease is present, the system responds accordingly and lets the user know about this fact.
   - If the system is not capable of performing the desired task, it lets the user know about this fact.

IV. At any moment during the conversation, the "RESET" button ends the conversation and deletes history.

The system is composed of three major modules that carry out all the processes. These interdependent modules collaboratively function to accomplish the objective of retrieving precise responses to the user's queries through the Graphical User Interface (GUI).

These modules are:

1. Building and training the model
2. Response retrieval
3. Chatbot GUI

All these modules start with import statements for the required dependencies.

3.4.2. Processes in building and training the model:

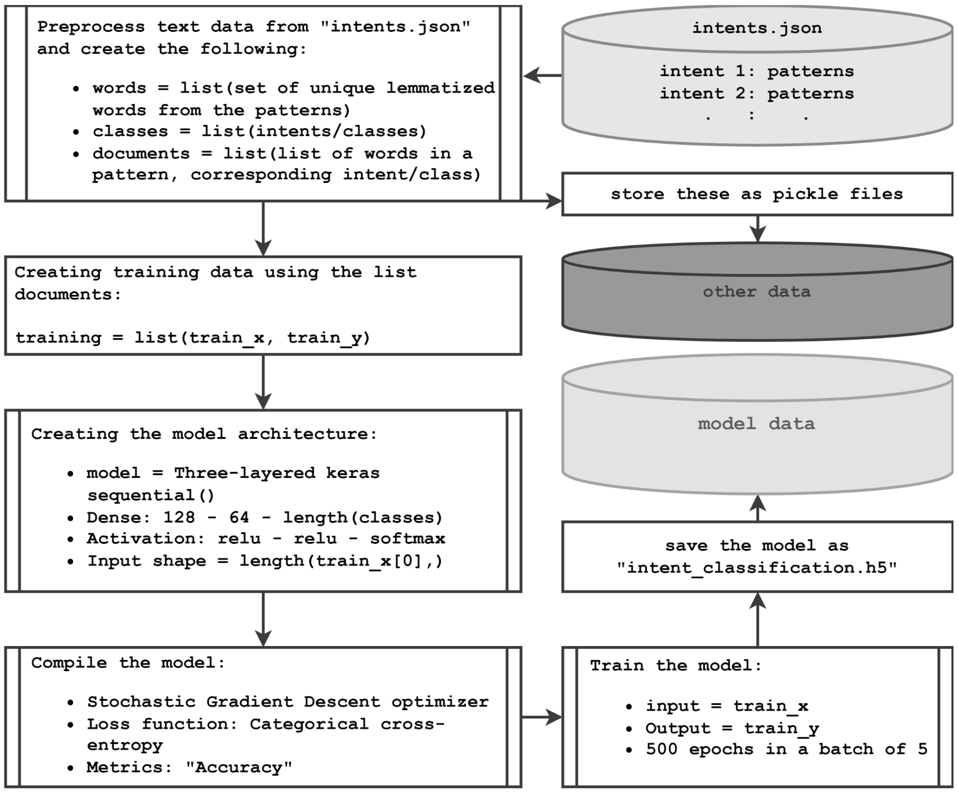

Preprocess text data from "intents.json" and create the following:
• words = list(set of unique lemmatized words from the patterns)
• classes = list(intents/classes)
• documents = list(list of words in a pattern, corresponding intent/class)
intents.json
intent 1: patterns
intent 2: patterns
store these as pickle files
other data
Creating training data using the list documents:
training = list(train_x, train_y)
model data
Creating the model architecture:
• model = Three-layered keras sequential()
• Dense: 128 - 64 - length(classes)
• Activation: relu - relu - softmax
• Input shape = length(train_x[0],)
save the model as "intent_classification.h5"
Compile the model:
• Stochastic Gradient Descent optimizer
• Loss function: Categorical cross-entropy
• Metrics: "Accuracy"
Train the model:
• input = train_x
• Output = train_y
• 500 epochs in a batch of 5

Figure: Training module workflow

I. Read and load “intents.json” as *intents*. Convert each and every *pattern* into a list of words, classes (intents), and documents (list of tuples: [list words in the pattern, intent tag]). Sort and create sets of these lists.

II. Save these lists as pickle files for easy data transfer between modules.

III. Convert the list of *documents* into training data (*train_x*, *train_y*)

Where

train_x: binary numpy array that represents the presence of words in the corresponding question pattern. Also known as the input parameters.

train_y: binary numpy array that represents the corresponding intent tag or class also known as the categorical output.

The classification equation is given by:

f(train_x) = train_y

IV. A neural network-based multi-class classifier is designed by using 3 layered keras sequential model. The first two layers use the activation function “relu” and the last layer uses “softmax” which gives the predicted probability of the respective intent/class.

V. Train and compile the model with Stochastic Gradient Descent and categorical cross-entropy as its loss function. Save the model as "intent_classification.h5".

3.4.3. Processes in response retrieval:

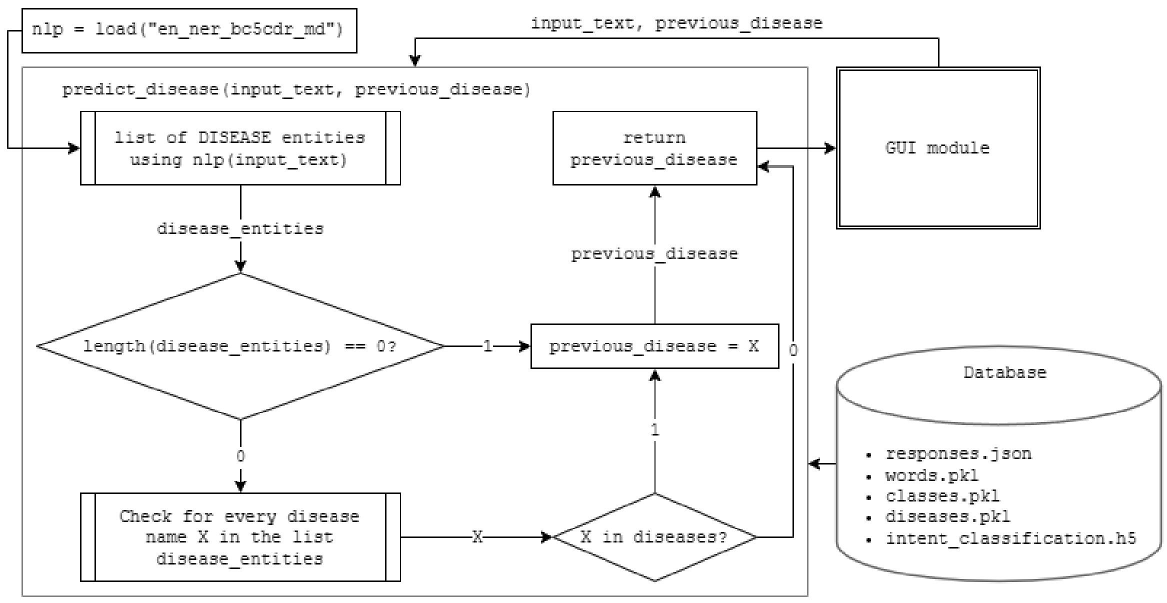

nlp = load("en_ner_bc5cdr_md")
input_text, previous_disease
predict_disease(input_text, previous_disease)
list of DISEASE entities using nlp(input_text)
return previous_disease
GUI module
disease_entities
previous_disease
length(disease_entities) == 0?
1
previous_disease = X
0
Database
responses.json
words.pkl
classes.pkl
diseases.pkl
intent_classification.h5
Check for every disease name X in the list disease_entities
X
X in diseases?

Figure: Flowchart of predict_disease function that uses SpaCy's NER model as nlp(text)

I. Read and load the file "responses.json" into *response* JSON, the pickle files into, a list of- *words, diseases,* and *classes,* and the "intent_classification.h5" into *model* respectively. Load SpaCy's biomedical NER model **"en_ner_bc5cdr_md"** as *nlp*.

II. Define function *predict_disease(input_text, previous_disease)* that returns the disease name from the *input_text* and with the help of the context of the *previous_disease*.

Use *nlp(input_text)* to create a list of entities with the labels "DISEASE" and "CHEMICAL" and store it as *disease_entities*.

If the length of *disease_entities* is 0, then return *previous_disease.* Otherwise, for each *disease* in *disease_entitites*, check if the *disease* exists in *diseases*. If it does, then update *previous_disease* with the *disease* and return it. If none of the entities exist in *diseases*, then, update *previous_disease* to "none" and return it.

III. Define a function *bot_response(input_text, disease)* that predicts the intent/class. For intent prediction, firstly, the *input_text* is converted into a binary numpy array (*bag_of_words*), which is fed into the *model*. The output array of the *model* corresponds to the probability corresponding *class* or intent. All the classes above an error_threshold are selected and sorted in decreasing order of their predicted probability. The first one is selected from the above as the intent of the user.

With the help of the **predicted disease** passed as *disease* as an argument for this function and the **predicted intent**, a response is retrieved with the help of a structured *responses* JSON object.

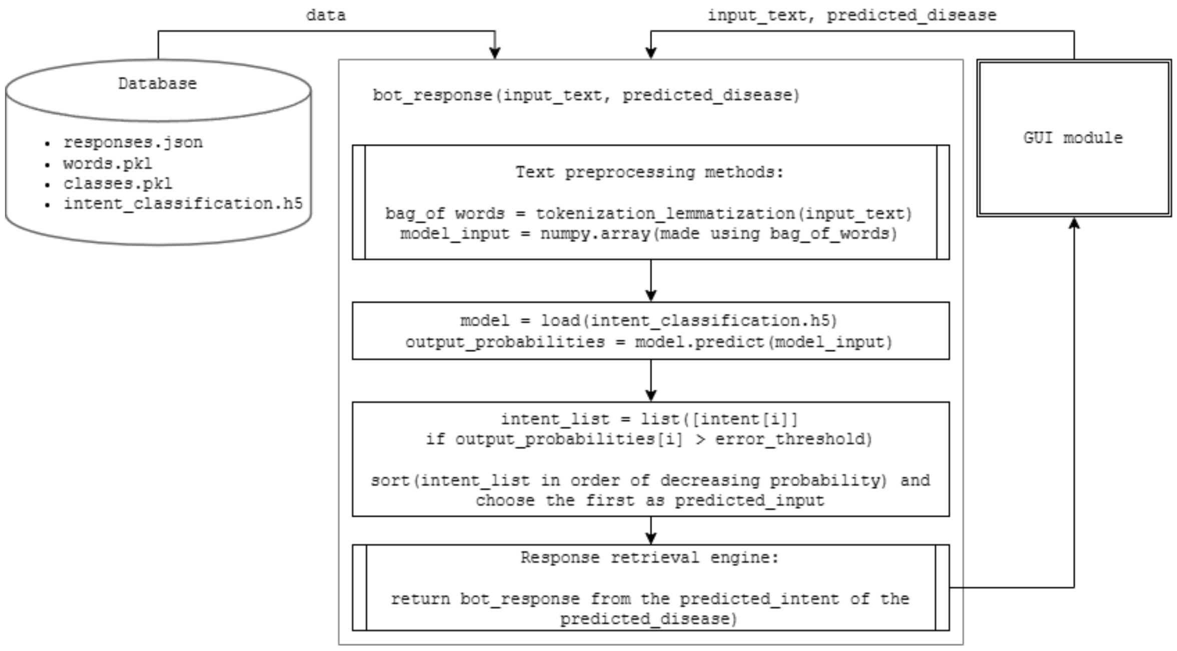

data
input_text, predicted_disease
Database
• responses.json
• words.pkl
• classes.pkl
• intent_classification.h5
bot_response(input_text, predicted_disease)
GUI module
Text preprocessing methods:
bag_of words = tokenization_lemmatization(input_text)
model_input = numpy.array(made using bag_of_words)
model = load(intent_classification.h5)
output_probabilities = model.predict(model_input)
intent_list = list([intent[i]] if output_probabilities[i] > error_threshold)
sort(intent_list in order of decreasing probability) and choose the first as predicted_input
Response retrieval engine:
return bot_response from the predicted_intent of the predicted_disease)

Figure: Flowchart for bot_response function

3.4.4. Processes in chatbot GUI:

I. The GUI is made using the Tkinter library. It imports *predict_disease* and *bot_response* functions from the chatbot application module.

II. A class context is created to keep a record of the previously accessed disease. Static methods are used to **retrieve** and **update** the disease in context.

III. When the user enters a query in the input field of the GUI and hits the "SEND" button, the *predict_disease* function is passed with the arguments *input_text* and *none* ("none" represents NULL or NO_RECORD exceptions), whose returned value is stored as the *disease*. Simultaneously the disease in context is updated.

IV. The response is generated by using the *bot_response*(*input_text*, *disease*) and displayed as the "Bot" response in the display section of the GUI.

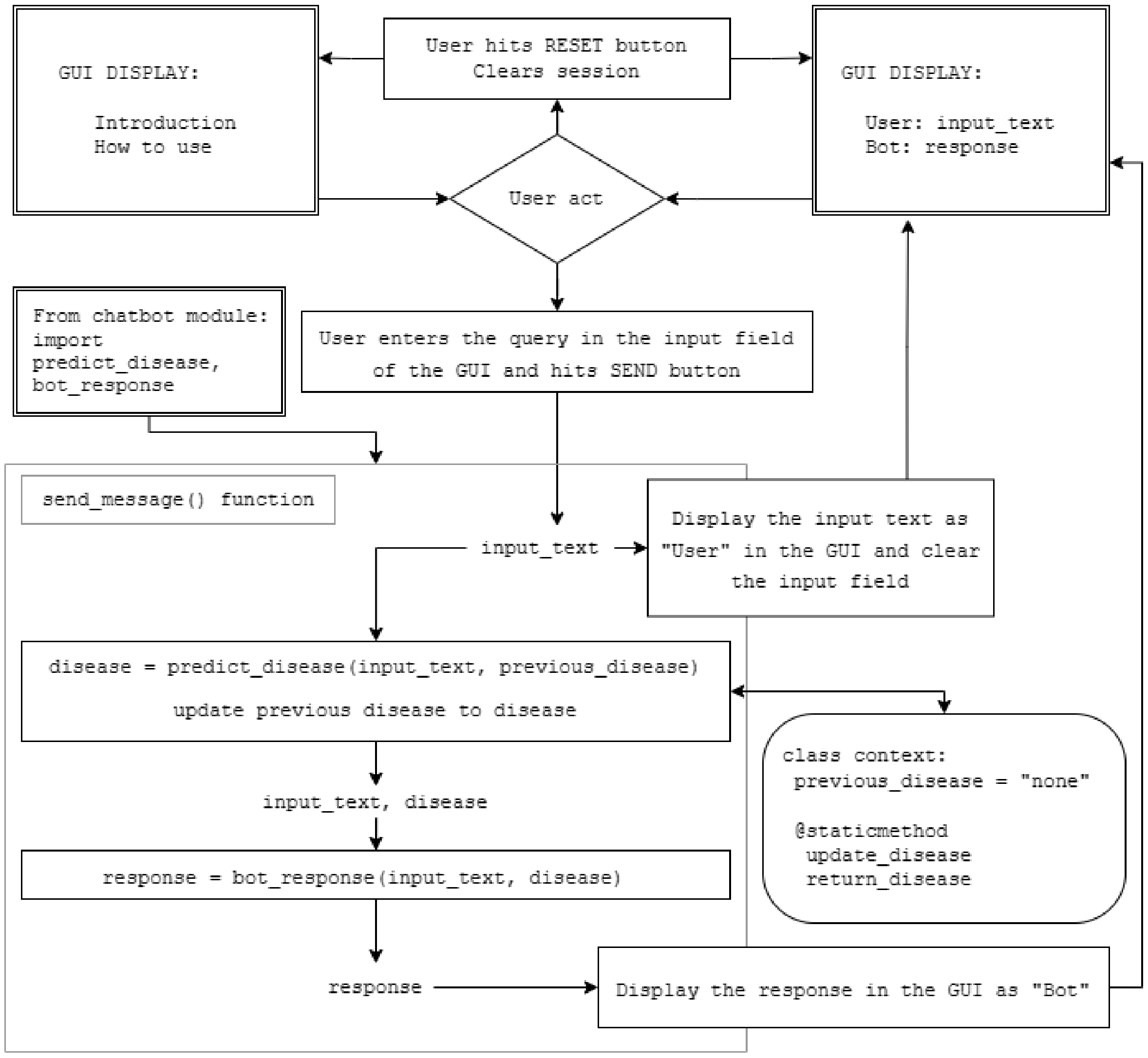

GUI DISPLAY:
Introduction
How to use
User hits RESET button
Clears session
GUI DISPLAY:
User: input_text
Bot: response
User act
From chatbot module:
import
predict_disease,
bot_response
User enters the query in the input field of the GUI and hits SEND button
send_message() function
input_text
Display the input text as "User" in the GUI and clear the input field
disease = predict_disease(input_text, previous_disease)
update previous disease to disease
class context:
previous_disease = "none"
@staticmethod
update_disease
return_disease
input_text, disease
response = bot_response(input_text, disease)
response
Display the response in the GUI as "Bot"

Figure: GUI interaction workflow and processes

3.5. Performance

The performance of this retrieval-based biomedical chatbot project can be evaluated based on several key metrics and considerations. Here are some aspects to consider when assessing the performance of the project:

1. **Accuracy:** The accuracy of the chatbot in providing correct and relevant responses to user queries is a crucial performance metric. It indicates the effectiveness of the retrieval and response generation mechanisms in retrieving the most appropriate information from the biomedical knowledge base and generating accurate responses.

    - The intent prediction model achieves an average accuracy of **94% in the classification of intent** calculated on the training data.

- The pre-trained SpaCy model **“en_ner_bc5cdr_md”** has an accuracy of **84.53%** in recognizing entities with labels “DISEASE” and “CHEMICAL”.

2. **Precision and Recall:** Each response is characterized by two parameters, which are - disease and intent. Due to which the answer is precise to its supposed query. Only 7% of the time the model fails to predict the intent correctly and reduces effectiveness.

3. **Response Coherence:** Few mechanisms implemented in this chatbot application ensure response coherence to some degree. One such mechanism is a **response based on the context of previous queries**. This significantly increases response coherence.

4. **Response Time:** The response time of the chatbot, i.e., the time taken to generate a response to a user query, is an important performance aspect. This chatbot is able to retrieve the first response in **0-2 seconds**. The responses for queries asked after this are retrieved **almost instantly**.

5. **Scalability:** As this chatbot works on a comparatively small knowledge base and only does the work of retrieving responses, it is **highly scalable**.

6. **User Satisfaction:** Due to this chatbot’s limited and **simplistic** use cases, it does its work best by retrieving valuable information in a **very intuitive and user-friendly** manner. It **understands the common English language** and provides very accurate results.

3.5.1 Optimizing Keras Model for Best Accuracy:

The retrieval-based biomedical chatbot project utilizes Keras, a popular deep-learning library, to implement machine-learning models for the task of text classification. To optimize a Keras model for the best accuracy, the following techniques are employed:

1. **Model Architecture:** Choosing an appropriate model architecture is crucial for achieving high accuracy. This involves selecting the type of neural network and designing the layers and connections within the network. The architecture should be capable of capturing the relevant patterns and relationships in the data.

   In this project, I have used a three-layer fully connected neural network architecture which uses activation functions: *“relu”* for the first two layers and *“softmax”* for the last layer to generate probabilistic data for each class.

2. **Hyperparameter Tuning:** Hyperparameters are configuration settings that control the learning process of the model. Optimizing hyperparameters, such as learning rate, batch size,

activation functions, and regularization techniques, can significantly impact the model's accuracy. After several iterations of training the most accurate model is produced.

3. **Training Data Quality:** The quality and diversity of the training data used to train the Keras model are essential for achieving high accuracy. Simple and complex query statements are generated which are fed to the model in the training phase.

4. **Regularization Techniques:** Regularization technique such as dropout is used in this model to prevent overfitting and improve the model's accuracy on unseen data. Adding dropout randomly sets a fraction of the units (neurons) in a layer to zero, effectively "dropping out" those units. This helps to prevent complex co-adaptations between neurons and reduces overfitting.

It's important to note that optimizing a Keras model for best accuracy is an iterative process that involves experimentation, evaluation, and refinement.

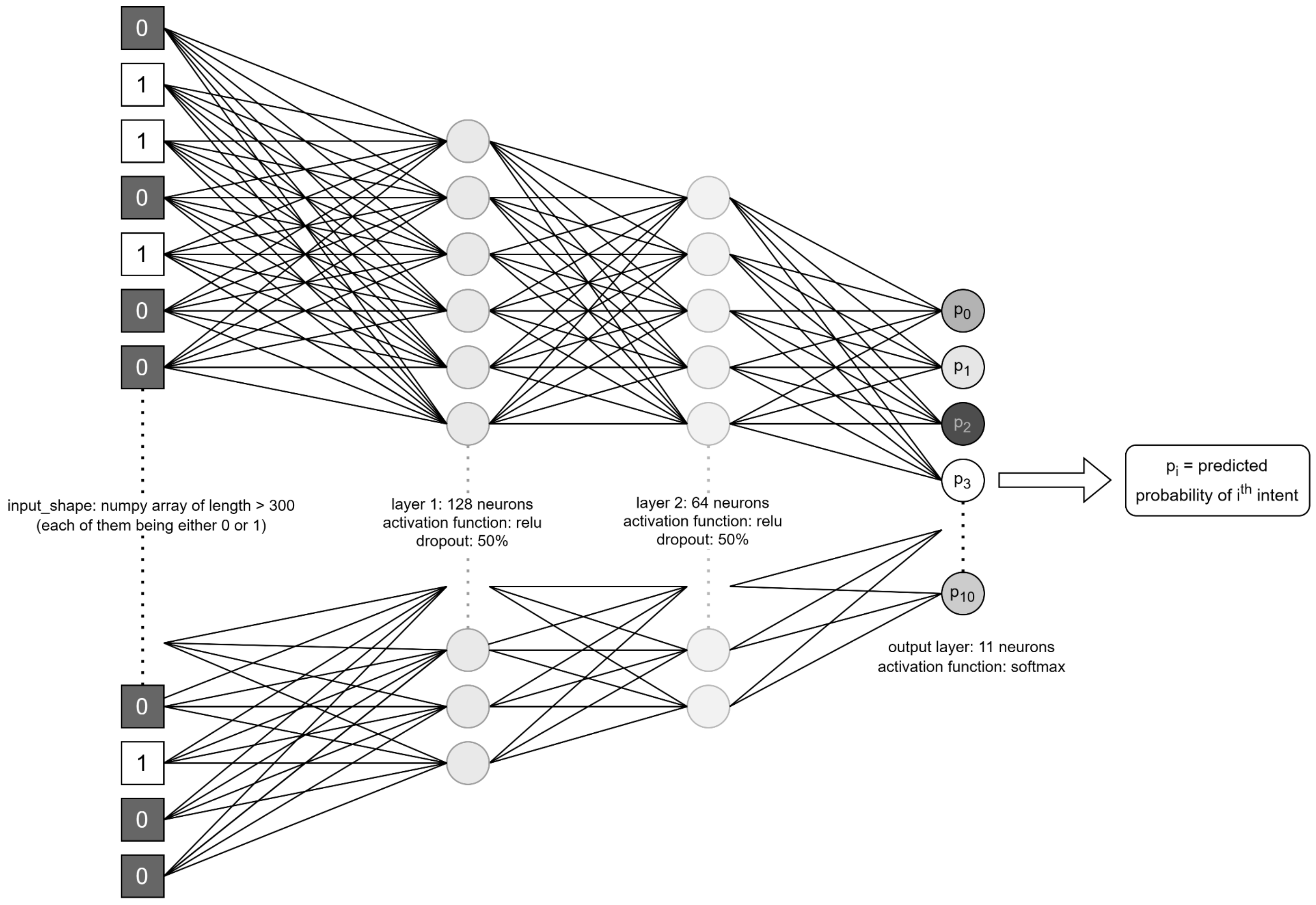

0
1
1
0
1
0
0
0
1
0
0
input_shape: numpy array of length > 300 (each of them being either 0 or 1)
layer 1: 128 neurons activation function: relu dropout: 50%
layer 2: 64 neurons activation function: relu dropout: 50%
p_0
p_1
p_2
p_3
p_{10}
output layer: 11 neurons activation function: softmax
p_i = predicted probability of i^{th} intent

Figure: Intent classifier neural network

4. IMPLEMENTATION

4.1. Technologies and programming languages

The development of the retrieval-based biomedical chatbot project involves the utilization of various technologies and programming languages to implement different components and functionalities. The following technologies and programming languages are used in the project:

1. **Python:** Python is used for developing chatbot systems due to its simplicity, readability, and vast array of machine-learning libraries. It is used for implementing the core functionalities of the chatbot. The version of Python used is 3.10.

2. **NLTK library:** It is a Natural Language Processing library, which aids in processing and understanding user queries. This project uses pre-trained **lemmatization** models and algorithms like **word tokenization** from Natural Language Tool Kit library (NLTK).

3. **ScispaCy library:** ScispaCy is a Python package containing spaCy models for processing biomedical, scientific, or clinical text. It is used to implement disease classification from query sentences. In this project, a **Named Entity Recognition** (NER) model **“en_ner_bc5cdr_md”** is used that can recognize the entities with labels “DISEASE” and “CHEMICAL”. It also considers medical condition as an entity of the label “DISEASE”.

4. **Keras library:** It provides machine learning tools that are used to implement a multi-class classifier model and train it. This project uses a keras “sequential()” model, which is built using the functions and algorithms available in this library. A three-layer fully connected neural network is designed with appropriate activation functions for each layer. The output layer uses the activation function “softmax” to generate a probability distribution for the neurons corresponding to each intent/class.

5. **Tkinter for GUI development:** Tkinter is used to develop a simple Graphical User Interface for this chatbot with the following functionalities:

    - The text area for entering a query
    - Button to send input text and execute processes for a bot response
    - Button to reset the conversation
    - Text area to display the conversation between the user and the chatbot

6. **Biomedical Knowledge Base:** The retrieval-based biomedical chatbot relies on a comprehensive biomedical knowledge base that contains structured and indexed data.

7. **Version Control and Collaboration:** Version control systems like Git and collaboration platforms like GitHub facilitate team collaboration, code management, and version tracking throughout the development lifecycle. The latest codes for the project are available in the repository.

4.2. Code structure, components, and organization

This retrieval-based biomedical chatbot project, follows a structured code organization to ensure modularity, scalability, and maintainability. The codebase is organized into several key components, each responsible for specific functionalities. Here is an overview of the code structure, components, and organization of the project:

1. **\`Data\` Directory:** This directory contains the necessary data files used by the chatbot system which include:

    - intents.json
    - responses.json
    - words.pkl
    - classes.pkl
    - diseases.pkl

2. **\`models\` Directory:** This directory contains the "intent_classification.h5" file that comprises the data of our custom-trained model for intent classification.

3. **\`utility.py\` module:** The \`utility.py\` module contains utility functions used across different components of the chatbot system. It consists of data preprocessing, text normalization, feature extraction, and other common functionalities shared among the system components. It also contains functions to check the data that is stored in the \`data\` directory and more functions for unit testing.

4. **\`training.py\` module:** Contains the complete code of training and designing the Keras model for intent classification. It saves the model as an "intent_classification.h5" file in the "models" directory.

5. **\`chatbot.py\` module:** Contains the complete code of the application which performs the tasks- prediction using pre-trained models and response retrieval. This module uses the model that was created and saved by the "training.py" module.

6. **\`chatbot_gui.py\` module:** Contains the design of the User Interface. It inherits functions from the "chatbot.py" module to get predicted responses and keeps updating the context of the previously accessed disease for each query.

The separation of components enables independent development, testing, and enhancements of each module. The codebase is well-documented with appropriate comments, making it easier for developers to understand and navigate the system.

4.3. Database and Data Handling

The database structure is chosen in such a way that it enhances the time of retrieval of information. The databases are created in an associative manner where each set of text data is given a name tag for efficient searching. This is easily done using JSON files. The database is divided into two major datasets. One ("intents.json") is used for training the intent classification model and the other ("responses.json") is used for response retrieval. These datasets are updated manually at the moment but have scope for automation.

Training is done on the "intents.json" file, which is just a categorical division of all the intents along with their question patterns. All the objects in this file follow the following structure:

- Contains the intent "tag" or class name.
- Contains the most common type of question asked by users inside the "patterns" element.

There is also another intent named "none" in the list of intents which is basically a representation of the chatbot's inability to perform other tasks.

For responses, a file structure is chosen that focuses on the quick retrieval of responses for different diseases based on the selected intent. The data is structured in the following ways:

- Each object in the JSON file contains the responses corresponding to the disease that is present as the "tag" of that object.
- Each object contains the responses for different intents.
- Each object contains a "none" named element to respond to queries that couldn't be classified into any of the meaningful intents.

The chatbot keeps the context of a conversation until the end of one session and thus does not create any privacy concerns. No personal data is stored during the conversation or after. However, to improve the model, integrating the user inputs into the training data will be beneficial.

5. TESTING

5.1. Unit testing

Through unit testing, each section of the code is adjusted to meet the desired objectives of the chatbot application. All the modules are tested separately and independent execution of each unit is carried out thoroughly. These units comprise:

- All text pre-processing units.
- Conversion of raw data to training data and converting queries into input data for the model.
- Training the model on the training data.
- Using the model to predict intent.
- Converting the output of the ANN into meaningful data.
- Response retrieval engine.
- Disease prediction unit along with the context retrieval mechanism.
- GUI design and functionalities inside it.

The training of the model is done in iterations. For each run of the training module, we check the accuracy and loss. All the parameters used for training are altered for each run until desired results are achieved.

5.2. Debugging

The following situations were encountered, and they were resolved:

1. **Dependency conflicts for the imported packages:** It is checked that all the required packages are installed and that the dependencies of one installed package do not conflict with another.

```
import random
import json
import pickle
import nltk
from nltk.stem import WordNetLemmatizer
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import SGD
```

```
...
```

2. **Bug in predict_disease function:** The function aims to predict the disease name from the list of “DISEASE” and “CHEMICAL” labeled entities, obtained from processing the text with SpaCy’s **“en_ner_bc5cdr_md”** NER model and other NLP techniques. Store the disease present at the first index in the context and update it when the queried disease is changed.

   Bugged code: The context is changed and saved respectively. After predicting the disease name from the text input and checking whether the disease name is available in the list of diseases or not. It takes reference from the context of the previously accessed disease and does not respond with relevant information.

   Resolution: Firstly, the disease name is separated and checked if it is available in the database. If not then the system returns “NO_RECORD”. If no disease name is present in the text (checked if the length(of “DISEASE” entities in a text input) is 0), the function returns the disease in the context based on the query.

   Predict_disease function algorithm after resolving.

```
def predict_disease(msg, prev_disease):
    doc = nlp(msg)
        disease_entitiies = list(ent.text for ent in doc.ents if ent.label_ ==
"DISEASE")

    if len(disease_entitiies) == 0:  # if no disease name present
        return prev_disease

    for disease in disease_entitiies:
        if disease in diseases:  # if disease name present
            prev_disease = disease
            return disease

    prev_disease = "none"  # if all the above fails
    return prev_disease
```

   Following is the context class with a record of previous_disease, which is set to “none” by default.

```
class context: # static class implemented in order to share the same data
throughout the program
    previous_disease = "none"
    @staticmethod
    def update_disease(self, x):
        self.previous_disease = x
    @staticmethod
    def return_disease(self):
        return self.previous_disease
```

Using the predict_disease, context.update_disease, and bot_response function:

```
def send_message():
    previous_disease = context.return_disease(context)
    message = input_text.get("1.0", tk.END).strip()
    if message:
        chat_display.insert(tk.END, "User: " + message + "\n\n")

        curr_d = predict_disease(message, previous_disease)
        context.update_disease(context, curr_d)
        response = bot_response(message, curr_d)

        chat_display.insert(tk.END, "Bot: " + response + "\n\n")
    input_text.delete("1.0", tk.END)
```

6. CONCLUSION AND FUTURE WORK

This project has helped me learn all the technologies and skills that were required for its complete implementation which include- learning Python for machine learning, and implementation of neural networks. It helped me gather thorough and in-depth knowledge about how various types of chatbot or dialogue systems work, and their inherent architectures. In conclusion, the retrieval-based biomedical chatbot project has successfully addressed the need for an intelligent system capable of retrieving accurate and relevant information in the medical domain. Through the utilization of advanced technologies, including natural language processing, machine learning, and a well-designed database, the chatbot demonstrates its ability to effectively respond to user queries and provide valuable insights.

Throughout the development process, significant efforts were made to ensure the system's performance, accuracy, and usability. By incorporating a carefully curated biomedical knowledge base and implementing sophisticated algorithms, the chatbot exhibits commendable performance in retrieving precise information related to symptoms, treatments, causes, and research in the medical field.

The user requirements analysis phase allowed for a comprehensive understanding of the users' needs and expectations, which guided the design and development of the chatbot. The system architecture, comprising interconnected modules, ensures efficient information retrieval and seamless interaction through the intuitive Graphical User Interface (GUI).

Economic feasibility analysis indicates that the project holds promise in terms of cost-effectiveness and potential benefits in the medical domain. The availability of open-source technologies and programming languages facilitated the development process while keeping resource requirements within manageable limits.

The project's performance evaluation demonstrates an accuracy of around 90-91%, response coherence, and minimal response time, providing users with a satisfactory experience. The scalability of the chatbot system enables it to handle increasing user loads while maintaining optimal performance.

Furthermore, the optimization of the Keras model, through careful selection of architecture and hyperparameter tuning, significantly contributes to the system's accuracy and overall performance.

As with any complex system, there is scope for future enhancements, including expanding the knowledge base, integrating additional data sources, and incorporating advanced techniques such as natural language generation for more contextually appropriate responses.

The project is complete with the code for all the modules present in the APPENDIX in chapter 7.

7. APPENDIX

7.1. Codes

Training module:

```
import random
import json
import pickle
import nltk
from nltk.stem import WordNetLemmatizer
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import SGD


lemmatizer = WordNetLemmatizer()

words = []  # set of words in the dataset intents.json
diseases = []  # diseases in responses.json
documents = []  # list(words in pattern, corresponding tag)
classes = []  # intent classes from intents.json
ignore_words = ["?", "!", "XXXX", "disease"]

file = open("data/responses.json", "r", encoding="UTF-8")
data = json.load(file)

for disease in data["responses"]:
    d = disease["tag"]
    diseases.append(d)


file = open("data/intents.json", "r", encoding="UTF-8")
intents = json.load(file)

for intent in intents["intents"]:
    for pattern in intent["patterns"]:
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        documents.append((w, intent["tag"]))

    if intent["tag"] not in classes:
        classes.append(intent["tag"])

# pre-process and organise the data
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
classes = sorted(list(set(classes)))
diseases = sorted(list(set(diseases)))


print(len(documents), " documents")

# print("Documents: ", documents)
print(len(words), "unique lemmatized words", words)
print("diseases: ", diseases)
print("classes: ", classes)
print("")

pickle.dump(words, open("data/words.pkl", "wb"))
pickle.dump(diseases, open("data/diseases.pkl", "wb"))
pickle.dump(classes, open("data/classes.pkl", "wb"))

training = []
output_empty = [0] * len(classes)

for doc in documents:
    bag = []
    pattern_words = doc[0]
    for w in words:
        if w in pattern_words:
            bag.append(1)
        else:
            bag.append(0)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1
    training.append([bag, output_row])

random.shuffle(training)
train_x = np.array([i[0] for i in training])
train_y = np.array([i[1] for i in training])

model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation="softmax"))

sgd = SGD(learning_rate=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=["accuracy"])

HIST = model.fit(
    np.array(train_x), np.array(train_y), epochs=500, batch_size=5, verbose=1
)
model.save("models/intent_classification.h5", HIST)

print("model trained and saved")
```

Chatbot application and response retrieval module:

```
import random
import json
import pickle
import spacy
import nltk
import numpy as np
from nltk.stem import WordNetLemmatizer
from keras.models import load_model
import utility as u


lemmatizer = WordNetLemmatizer()

nlp = spacy.load(
    "en_ner_bc5cdr_md"
)  # "en_core_sci_sm" model is  unable to classify the the DISEASE
model = load_model("models/intent_classification.h5", compile=False)

responses_starts = open("data/responses.json", "r", encoding="UTF-8")
```

```
responses = json.load(responses_starts)

words = pickle.load(open("data/words.pkl", "rb"))
diseases = pickle.load(open("data/diseases.pkl", "rb"))
classes = pickle.load(open("data/classes.pkl", "rb"))
# ignore_words = pickle.load(open("data/ignore_words.pkl", "rb"))
ignore_words = ["?", "!", "XXXX", "disease"]


def predict_disease(msg, prev_disease):
    doc = nlp(msg)
    disease_entitiies = list(ent.text for ent in doc.ents if ent.label_ == "DISEASE")

    if len(disease_entitiies) == 0:  # if no disease name present
        return prev_disease

    for disease in disease_entitiies:
        if disease in diseases:  # if disease name present
            prev_disease = disease
            return disease

    prev_disease = "none"  # if all the above fails
    return prev_disease


# print(diseases, predict_disease("hello lung cancer", "none"))


def bot_response(msg, disease):  # use this function to retrieve response for the
user_input
    retrieved_res = "Sorry! I don't understand"  # default if all else fails

    msg_words = nltk.word_tokenize(msg)
    msg_words = [
        lemmatizer.lemmatize(w.lower()) for w in msg_words if w not in ignore_words
    ]

    bag = []
    for word in words:
        if word in msg_words:
            bag.append(1)
        else:
            bag.append(0)

    # Using the model
    # To find the class with highest probability, sorting is done
    prediction = model.predict(np.array([bag]), verbose=False)[0]
    # >>> prediction = [a1 a2 a3 a4 a5 a6....an]----------------------------to_list-
    #                                                                              |
    error_threshold = 0.25  #                                                      v
    results = []

    results = [[i, r] for i, r in enumerate(prediction) if r > error_threshold]
    # >>> results = [[0, p0], [1, p1], ... [n, p(n-1)]]

    results.sort(key=lambda x: x[1], reverse=True)
    # "key=lambda x: x[1]" is a lambda function
    # key -> [lambda[for all tuples 'x' select it's second element]]
    # reverse the order => descending order

    return_list = []  # to store hash value pair lists

    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
        # Hash table or dictionary with key: value pairs ["intent": "tag", "probability":
pn]
    intent_tag = str(return_list[0]["intent"])
    # print("Type of: intent_tag = ", intent_tag, " ", type(intent_tag))
    # [0] means the first element in the list and
    # ["intent"] gives the value associated with the key "intent"
    # returns intent with highest probability

    all_disease = responses["responses"]

    print("[ looking for: ", intent_tag, "| disease: ", disease, "]")

    # if "tag" matches the predicted disease then choose any of the random responses
    for item in all_disease:
        if disease in item["tag"]:
            retrieved_res = random.choice(item[intent_tag])
            return retrieved_res
    return retrieved_res


def chat():
    disease = "none"  # to keep track of the accessed disease
    print("Welcome to biomedical chatbot demo:\n\n------START CONVERSATION------\n")
    print(
        'Bot: "You can search for causes, complications, current research, \n\tdiagnosis,
epidemiology, prevention, prognosis, risk factors, \n\tsymptoms or treatment of any of
the following diseases:"'
    )
    print(u.view_diseases())
    while True:
        msg = str(input("User: "))
        if msg.lower() == "exit" or msg.lower() == "quit":
            exit()
        disease = predict_disease(msg, disease)  # predict disease
        print("Bot: ", bot_response(msg, disease), "\n")  # pass it to response function
```

Chatbot GUI module:

```
import tkinter as tk
from chatbot import chat, predict_disease, bot_response
import utility as u


class context:
# static class implemented in order to share the same data throughout the program

    previous_disease = "none"

    @staticmethod
    def update_disease(self, x):
        self.previous_disease = x

    @staticmethod
    def return_disease(self):
        return self.previous_disease


def send_message():
    previous_disease = context.return_disease(context)
    message = input_text.get("1.0", tk.END).strip()
    if message:
        chat_display.insert(tk.END, "User: " + message + "\n\n")
        curr_d = predict_disease(message, previous_disease)
        context.update_disease(context, curr_d)
        response = bot_response(message, curr_d)
        chat_display.insert(tk.END, "Bot: " + response + "\n\n")
    input_text.delete("1.0", tk.END)

def clear_conversation():
    context.update_disease(context, "none")                    # update the static
variable disease to "none"
    chat_display.delete("1.0", tk.END)
    chat_display.insert(tk.END, "You can search for causes, complications, current
research, diagnosis, epidemiology, prevention, prognosis, risk factors, symptoms or
treatment of any of the following diseases:\n\n")
    diseases = list(u.view_diseases())
    chat_display.insert(tk.END, diseases)
    chat_display.insert(tk.END, "\n\n\nHow to use:\nTo start a conversation, enter your
query in the text-box below and hit 'SEND'\nTo clear the conversation session, hit the
'RESET' button\n\n\n")


root = tk.Tk()
root.title("Bio-medical chatbot")



# Chat display area
chat_display = tk.Text(root, height=20, width=50, padx=5, pady=5 )
chat_display.grid(row=0, column=0, columnspan=2, padx=5, pady=5)

# Input text area
input_text = tk.Text(root, height=3, width=50, padx=5, pady=5)
input_text.grid(row=1, column=0, columnspan=2, padx=5, pady=5)

# SEND button
send_button = tk.Button(root, text="SEND", command=send_message)
send_button.grid(row=2, column=0, padx=5, pady=5)

# RESET button
clear_button = tk.Button(root, text="RESET", command=clear_conversation)
clear_button.grid(row=2, column=1, padx=5, pady=5)

def intro(text):
    chat_display.insert(tk.END, "You can search for causes, complications, current
research, diagnosis, epidemiology, prevention, prognosis, risk factors, symptoms or
treatment of any of the following diseases:\n\n")
    diseases = list(u.view_diseases())
    chat_display.insert(tk.END, diseases)
    chat_display.insert(tk.END, "\n\n\nHow to use:\nTo start a conversation, enter your
query in the text-box below and hit 'SEND'\nTo clear the conversation session, hit the
'RESET' button\n\n\n")


root.mainloop()
```

8. REFERENCES

[1] J. Daniel and M. James H., Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall, 2007. Accessed: Apr. 10, 2023. [Online]. Available: http://113.161.98.146/jspui/handle/123456789/129

[2] Natural Language Processing - Tokenization (NLP Zero to Hero - Part 1), (Feb. 20, 2020). Accessed: Jun. 12, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=fNxaJsNG3-s

[3] “NLTK :: Natural Language Toolkit.” https://www.nltk.org/ (accessed Jun. 12, 2023).

[4] “NLTK :: nltk.tokenize.punkt.” https://www.nltk.org/_modules/nltk/tokenize/punkt.html (accessed Jun. 12, 2023).

[5] “GitHub repository of solozano0725/chatbot: Python Chatbot Project using NLTK & Keras (github.com)” https://github.com/solozano0725/chatbot (accessed May. 02, 2023).

[6] “Blog Retrieval based Chatbots — Using NLTK & Keras | by Sol M. Lozano | Medium” https://solozano0725.medium.com/retrieval-based-chatbots-using-nltk-keras-e4f86b262b17 (accessed May. 02, 2023).

[7] “sciSpaCy models scispacy | SpaCy models for biomedical text processing (allenai.github.io)” https://allenai.github.io/scispacy/ (accessed May. 12, 2023).

[8] “tkinter — Python interface to Tcl/Tk,” Python documentation. https://docs.python.org/3/library/tkinter.html (accessed Jun. 12, 2023).

[9] Python NumPy Tutorial for Beginners, (Aug. 07, 2019). Accessed: Jun. 12, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=QUT1VHiLmmI