Deployment Manual

COMP0016 2021/22 Team 20

HeartBOT

Table of Contents

Overview

- Project Title: BHF key statistics chatbot
- Team Members: Ivan Varbanov, Maheem Imran, Neil Badal

Project Introduction

- A key statistics chatbot for the British Heart Foundation (BHF) - A joint project between (University College London) UCL and BHF
- This project aims to deliver a standalone web application that provides an interactive chatbot that answers (Frequently Asked Questions) FAQs and provides key statistical information based on the BHF compendium

System Architecture & How the System Works

Terminology

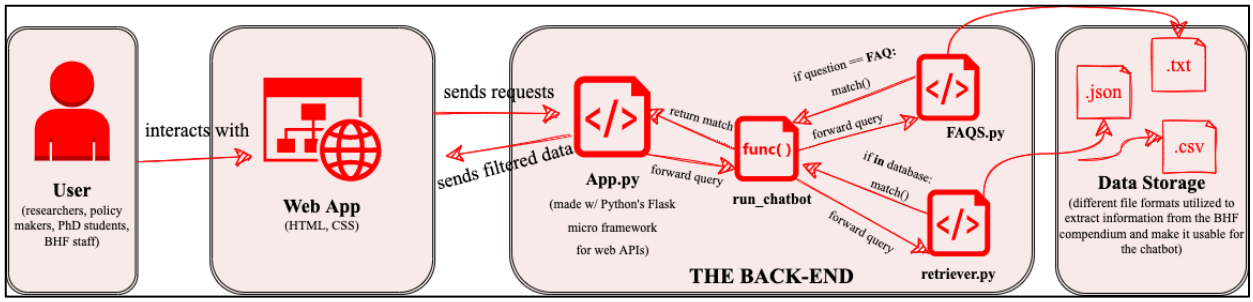
The BHF compendium is a spreadsheet. Certain words/phrases are used throughout the code and explanations below. The phrase *'table names'* refers to specific sheet names in the compendium (e.g., admissions). The phrase *'column names'* refers to the column headers in each sheet of the compendium (e.g., nation). The phrase *'column values'* refers to distinct values that each column header can take in specific sheets of the compendium (e.g., UK). *'tokens'* refers to the list of words in the user query after removing punctuation and stop words etc. *'Direct check'* refers to finding the best match for tokens in the set of table/column names. *'Synonym check'* refers to finding the best match for tokens compared to synonyms generated for each table/column name. *'N-grams'* check refers to finding the best match of sets of 'n' tokens together compared to table/column names. *'Classifier'* refers to a class which runs all the tests like Direct check, Synonym Check, N-grams check etc on the list of tokens. *'dataframe'* refers to the data structure which displays the data from different files as a table with rows and columns.

The System

The HeartBOT system consists of many components. Currently, users can interact with HeartBOT using the web application (web app) implemented using the Flask framework. The backend of the web app handles requests (queries or questions) entered by the user and provides a response based on analysis of the request. There are two main parts of the system:

1. A FAQ section that determines if a question is an FAQ or greeting and provides a relevant response if this is the case
2. A data retrieval section that determines if a question is based on data retrieval from the BHF compendium and provides relevant data if this is the case

Below is the system architecture diagram for HeartBOT. It describes the main parts of the system:



The system begins by passing the request (users questions/query) from the web app to the backend. The backend first checks if the request is a FAQ or greeting. If this is the case, then the relevant response is provided; else the system tries to extract key information from the request. This is then used to extract relevant data from the BHF compendium. If both methods fail to provide a relevant and meaningful response, a friendly error message is returned instead. This aims to guide the user on how they can rephrase their question.

The system uses a custom-made best match solution to try and find the most relevant response for any given request. The main principle of this is again broken into three parts:

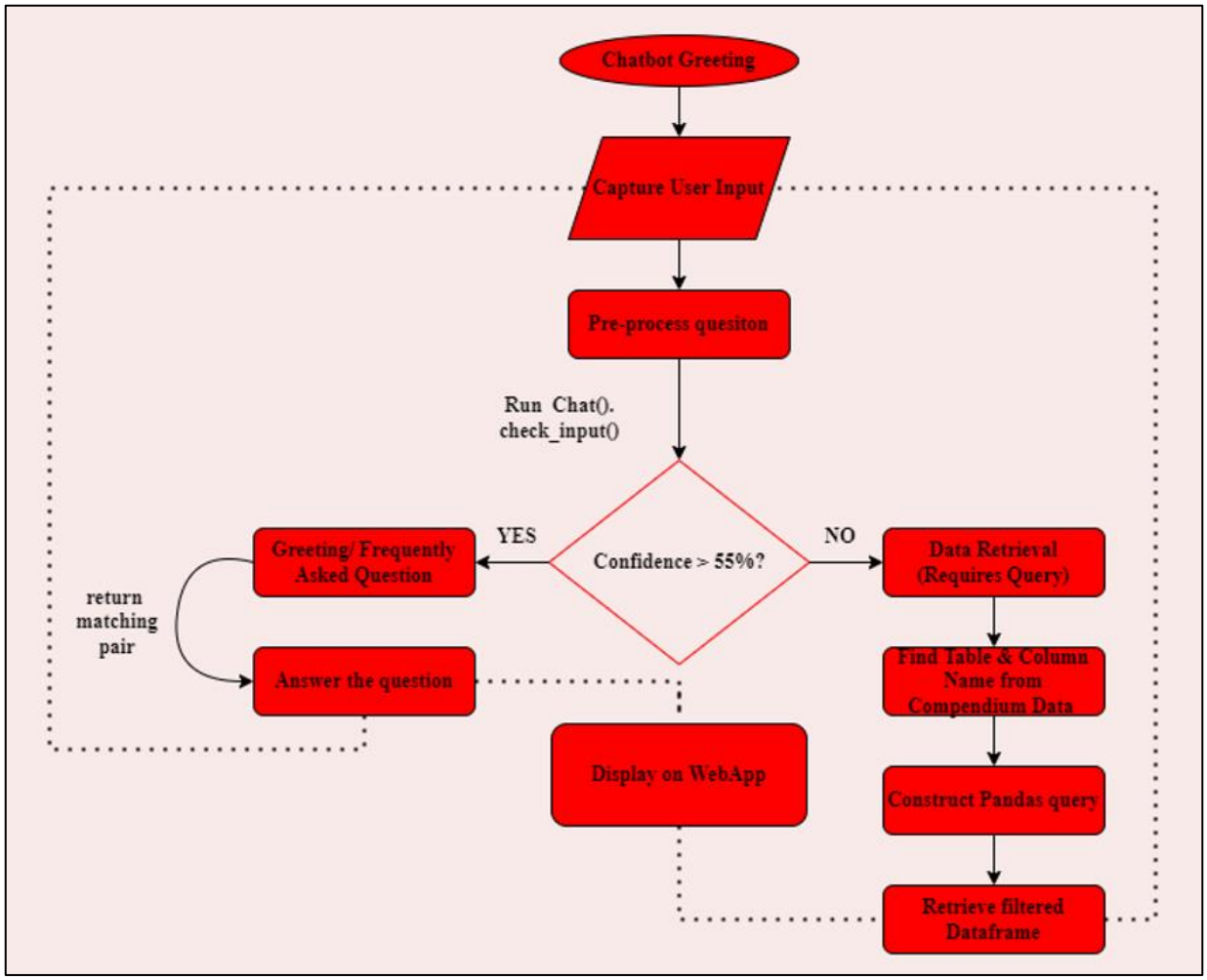
1. The best match for greetings and basic conversations works by using a regex match
2. The best match for FAQs works by analysing the words (tokens) that make up the request. These words are compared to words of a list of stored FAQs. Based on comparison, a percentage (confidence) is determined. If this percentage passes a certain threshold, the question is assumed to be an FAQ. For this best matching FAQ, the corresponding answer is returned
3. The best match for data retrieval questions works by searching the request for specific words. These words are table names, column names and column values. These are key values that help to filter and categorize the BHF compendium data. The search for these words is implemented using a classifier. It uses direct, synonym and n-gram checks to try and find the best matching table name, column name or column value again based on certain thresholds

This best match functionality in the system is extremely important because it is not practical to expect users to:

1. Type FAQs exactly (and in the same order) as stored
2. Use exact table names, column names and column values (as in the BHF compendium) when asking data retrieval questions

Hence, the inclusion of the best match functionality ensures the system can handle a large variety of questions. This includes questions with typos, synonyms, different orderings in addition to just exact words/phrases that are already stored in the system.

Below is the flowchart describing the main program:



In addition to this, the system has been optimised to ensure performance is optimal. This means users can quickly receive answers for their questions.

Project Files & Structure

```
.
├── src/                              # Contains all the data and code required for the chatbot and web application
│   ├── FAQS                          # Contains all the data and code required for the FAQ chatbot
│   │   └── aux_functions.py          # Contains useful auxiliary functions
│   │   └── best_match.py             # A script to match questions to FAQs from the database
│   │   └── converse.py               # A helper script, part of the nltk library
│   │   └── FAQs.py                   # The main script for FAQ matching
│   │   └── FAQs_q.txt                # A text file containing FAQs
│   │   └── FAQs.txt                  # A text file containing question and answer pairs
│   ├── Retrieval                     # Contains all the data and code required for the data retrieval chatbot
│   │   └── data                      # Contains all the data in .csv and .json files
│   │   └── classifier_col.py         # A script to identify the appropriate column name
│   │   └── classifier_tab.py         # A script to identify the appropriate table name
│   │   └── create_col_name_json.py   # A script to create column names form .json format
│   │   └── process_questions.py      # A script for question pre-processing
│   │   └── retriever.py              # A script to retrieve filterd table from query
│   ├── static                        # Contains the css code required for the chatbots web application
│   │   └── style.css
│   └── templates                     # Contains the html code required for the chatbots web application
│       └── index.html
│   ├── unit_test.py                  # A python Script containing unit tests
│   ├── sys_test.py                   # A python Script containing system tests
│   ├── app.py                        # The main python file to run and start the web application for the chatbot
├── archive                           # A folder for unused code, that was left for potential future development
├── fix-dependencies.py               # A python file to automatically download all required dependencies
├── requirements.txt                  # A text file that contains all the libraries used in the chatbot code
├── sources.txt                       # A text file that contains all references for sections of code and images used
├── test_questionts.txt               # A text file that contains a list of example questions that can be used to test the chatbot
├── sources.md                        # A file with links to all the sources sued
├── README.md                         # Essential Information about the project
└── .gitignore                        # A script that tells Git which files to ignore when committing your project to the GitHub repository.
```

Deployment

Features

HeartBOT can:

- Answer general questions and greetings
- Answer FAQs
- Answer data retrieval questions related to the BHF compendium

Prerequisites

- Python 3.8
- Python libraries mentioned in requirements.txt

Build & Run

Note: the following instructions are also mentioned in README.md

Configuration & Installation

1. Download the code from GitHub
2. Download python 3.8 from *https://www.python.org/*
3. Open your terminal
4. Change your directory to *comp0016_team_20*
5. Run the following command in your terminal:

   ```
   python fix_dependencies.py
   ```

6. This will install all the dependencies required to run the chatbot

Operating & Running

1. Open your terminal
2. Change your directory to *comp0016_team_20/src*
3. Run the following command in your terminal:

   ```
   python app.py
   ```

4. Click on the link displayed in the terminal
5. This will open your browser with the chatbot web application
6. Interact with the chatbot using the web interface
7. Hover over help button in web interface if help is needed
8. To shut down the chatbot/web application, in your terminal press *CTRL+C* to quit
9. Please refer to the user manual for further guidance

Troubleshooting

- Sometimes you may need to use *python3* instead of *python*

Testing

The project has been tested using unit and system tests. The python framework *unittest* and *coverage* were used to carry out these tests.

To run the tests:

1. In a terminal, navigate to *comp0016_team_20/src*
2. Run unit tests by typing the following into your terminal:

   python3 unit_test.py

3. Run system tests by typing the following into your terminal:

   python3 sys_test.py

4. Run test coverage by typing the following into your terminal:

   coverage run -m unittest unit_test.py

   coverage html

Further Development

Understanding The Current Code & Current Modification Areas

The key files for FAQs sub-system include:

1. **FAQs.txt** – This text file contains the list of FAQs; where one line is a question and next line is the corresponding answer
2. **FAQs_q.txt** – This text file contains the list of FAQs (only questions); where each line is a FAQ
3. Note – HeartBOT can recognize more FAQs by adding more questions and answers into these files. To ensure this works correctly they must be added into the files *FAQs.txt* and *FAQs_q.txt* in the mentioned formats
4. **aux_functions.py** – This python file contains the pairs used for the general conversations and greetings. In the pairs list, it is possible to add further regex statements and answers to allow HeartBOT to answer a greater variety of general questions and greetings
5. **best_match.py** – This python file contains the code used to find the best matching FAQ. It currently works by comparing words in the request and stored FAQs. It then returns a percentage and the best matching FAQ. It is possible to extend this by adding additional checks like synonyms and n-grams checks. This will allow the chatbot to find the best matching FAQ more precisley

The key files for data retrieval sub-system include:

1. **data directory** – This directory contains two types of files csv and json files:
   a. CSV files with table names are files with the actual data from the compendium. These are used to extract data (e.g. admissions.csv)
   b. JSON files with table names are files that contain a dictionary of key-value pairs. Each key is a column name and values are column values. These are used when searching for the best matching column names and values from user request in classifier_col.py. Note it is possible to use the python file *create_col_name_json.py* to automatically create these JSON files
   c. table_names.json is a json file that contains a dictionary of key-value pairs. Where keys are table names, and values are possible variations of table names. Note it is

possible to add variations of table names like abbreviations in this file to allow HeartBOT to recognize a greater variety of table names

   d. col_names.json is a json file that contains a dictionary of key-value pairs. Where keys are the column names and values are possible variations of the column values. It is possible to add variations like abbreviations here to allow HeartBOT to recognize a greater variety of column values. While this functionality is not implemented, it is a possible extension to the code

2. **classifier_col.py** – This python file contains the code required to find the best matching column names and values (if any exist) from the user's request/question. It currently uses multiple checks like direct, spelling, synonym, and n-grams
3. **classifier_tab.py** - This python file contains the code required to find the best matching table names (if any exist) from the user's request/question. It currently uses multiple checks like direct, spelling and synonym. It is possible to extend the code by adding an n-grams check here, this will allow HeartBOT to recognize table names more precisely
4. **process_questions.py** – This python file contains the code that takes the users request/question and return a list of tokens. These tokens are tuples of the individual words that make up the request and their pos-tags. This section is important as unnecessary information like stop words and punctuation is removed. While we do not use the pos-tags, it is a possible area of extension where HeartBOT can be made more precise and hence is included in the code
5. **retriever.py** – This python file contains the code required to retrieve data from the compendium. It uses some of the files mentioned above to extract the table names, column names and column values from the user's request. If these values are found, a dataframe query is constructed and the relevant dataframe (constructed from the relevant csv files) is filtered to provide the correct response to the user

Other key files or concepts include:

1. **app.py** – This python file contains the code required to run HeartBOT and the web application
2. **test_questions.txt** – This text file contains some questions that you can ask HeartBOT. They are included as examples of possible questions.

<u>Extending the system*</u>

Some ideas for future development of the program (and possible solutions) include:

- Automatic addition of FAQs from an excel sheet
   a. Possible Solution - An excel sheet could store all FAQs and they could be automatically read (following the specified format) into the two files *FAQs.txt* and *FAQs_q.txt* mentioned in the above section
- Suggesting possible questions or alternatives for users
- Adding more synonyms/alternatives for table names and column names in table_names.json and col_names.json. The same thing can be implemented for column values.
- Adding statistics to the dataframes retrieved
- Adding a 'I'm feeling lucky' button

*Please see the project website (evaluation page) for further ideas