**Smart Chatbot**

**3.1 Project Proposal:**

The proposed idea is to build a versatile Chatbot that is used to answer frequently asked questions and has following features:

* User-friendly GUI
* Answer all the queries regarding our college academics and admission processes (any information which the user wants to know) such as Admission Process, Location Of rooms/faculties/labs etc, Procedure for Submission of forms such as Bonafide/Railway Concession and the list is endless.
* Speech Recognition capabilities
* Ability of conversing with users (Speaker Output), including some other features covered one by one later.

The basic idea is to build a machine learning model that processes the queries asked by the user via application and accordingly answer their questions. This is done by using **MLP (Multilayer Perceptron)** which is a class of **Feedforward Artificial Neural Network**. This application of Deep Neural Networks can be easily used to obtain the above mentioned features of the required Chatbot for our college to function as a smart ‘RAIT ASSISTANT’.

The whole software is implemented in **Python 3.6 64 bit** programming language.

**3.2 Proposed Work:**

In our system, there is a frontend application would be displayed as soon as the BOT reaches the user. This application acts as Natural User Interface for the user to ask their queries about our college. The query asked by the user is then processed by our deep learning model and the response is fetched from our own database and it replies to the user accordingly. This system is composed of two parts:

1) Backend: Deep Learning Model

2) Frontend: Using Kivy Python GUI Framework

**3.3 Backend: Deep Learning Model**

**3.3.1 Deep Learning Model**

We created a Multilayer Perceptron (MLP) which is a type of Artificial Feedforward Neural Network having two **hidden layers** of 8 perceptrons with **Linear Activation Function** each including input layer and output layer with number of perceptrons equal to the number of the categories included in our database. This model trains itself using a technique called **Multinomial Logistic Regression** which is **Supervised Learning Algorithm** and optimizes the weights of each perceptron using algorithm called **Backpropagation**. This algorithm works by computing the gradient of the loss function with respect to each weight by the [chain rule](https://en.m.wikipedia.org/wiki/Chain_rule), computing the gradient one layer at a time, [iterating](https://en.m.wikipedia.org/wiki/Iteration) backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule. Using this technique, our model has been trained with 98% of accuracy metric after about 3000 epochs of dataset scanning. To achieve **classification** of input data, the output layer of our Neural Network is activated by **Softmax activation** to classify inputs to multiple categories of data available in our dataset.  This softmax function transforms all values to the range [0, 1] and the sum of the elements is 1, giving the probability of the input value being in a specific class.This classificationof input is achieved at the accuracy determined by the **Maximum Likelihood** **Estimation**.

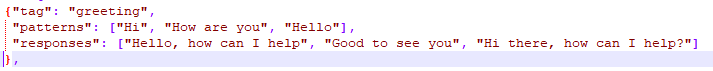
The above discussed model is implemented in python using **Tflearn** module for creating perceptrons and fully connected layers with appropriate activation functions and then predicting the correct classification of the user input into the appropriate category.

**3.3.2 Formation of Dataset**

Our dataset is created using **.JSON** file that contains a number categories/classes/tags/labels/ topics of data about which users are expected to ask. Thus, this forms the categorical data required to feed our model. Each category is identified with its name and consists:

* set of patterns/questions/queries that the user could frame his question pertaining to the category and
* Possible responses for such patterns.

Considering the example of ‘Greeting’ category, the dataset would look like following:



In this way, JSON file creates a bunch of messages that the user is likely to type in which are then mapped to a group of appropriate responses. The tag on each dictionary in the file indicates category that the user input would be classified in. With this data we will train a neural network to take a sentence of words and classify it as one of the tags in our file. Then we took a response from those groups and display that to the user. The JSON object is loaded in python using an inbuilt library in python called **json** module.

To ensure data security against eavesdropping and modification of the training data, the training data and model weights are encoded by converting those python objects into character stream so that it can easily be saved into the disk and making it difficult for the hacker to reverse engineer. This saves the files from getting eavesdropped or modified. We achieved this by encoding the training data and model weights in **Pickle Encoding Format,**

Using an inbuilt module in python called **pickle** Module which serializes/de-serializes data.

This format of dataset is chosen to:

* simplify the given task,
* ensure easy creation, maintenance, modification and extension of dataset by the admin
* less memory footprints

**3.3.3 Final Algorithm and Data Processing**

The dataset loaded from JSON file is processed to feed the data to the neural network. This is implemented using **nltk** (Natural Language Toolkit) module in python. The data processing includes following steps:

1. **Word-Tokenization**: Here, each format of query of all categories is split into individual words using space as a delimiter. These individual words form tokens. Word Tokenization is preferred over Sentence Tokenization to increase the tolerance of varied range of inputs provided by the user to fall in a particular category. In this way, user need not to submit his query which has same dependency parse between tokens as that registered in dataset.
2. **Filtering out Stop words**: All the stop words that frequently appear in sentences such as ‘a’, ‘and’, ‘the’, ‘.’, ‘?’; which are not meaningful are removed from the above tokens obtained from previous step to avoid the noise while performing statistical analysis.
3. **Stemming**: The set of unique words available from above step are then reduced to their parent words by reducing the inflections of the words. For example: words such as ‘playing’, ‘played’, ‘plays’ are inflections of the same word ‘Play. Stemming ensures less time and processing over Lemmatization. Moreover, in our case, exact semantics and grammar is not required. Stemming is implemented using **Lancaster** **Stemmer** available in nltk package which uses iterative algorithm with rules saved externally.
4. **One Hot Encoding**: The word list from above step forms the binary feature list. Now each pattern/ query in dataset is converted into a unique binary code depending on binary feature list. These codes are appended with each other and form the input of the labeled dataset. The corresponding category of each query is also one hot encoded and appended with each other with same index as that of input to form the output of labeled dataset.

Finally, this labeled dataset obtained is fed for training the model.

Similarly, user input is also processed in the same way before inputting it to the model for probabilistic classification. The category which is assigned the maximum probability by prediction and whose probability is greater than 70% is selected as the output category. Then a random response from the category is picked to answer the user’s query.

F L O W C H A R T

O F

C H A T B O T

S Y S T E M

Word Tokenization

Word Tokenization

Filtering out Stop Words

Filtering out Stop Words

Stemming

Stemming

One Hot Encoding

One Hot Encoding

Processed User Query

Processed Labeled Dataset

Multinomial Logistic Regression by trained model

Training model and saving the weights

Category with maximum probability > 70% is selected

Output probabilities for each category

Maximum Likelihood Estimation by the model

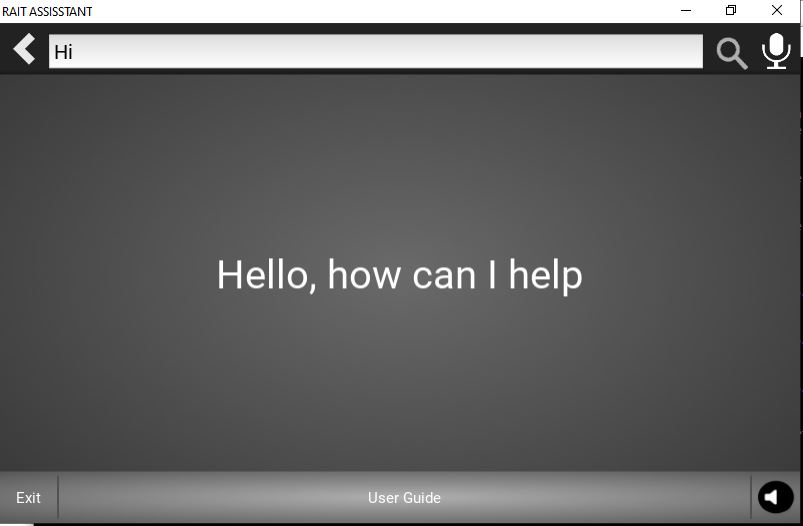
**3.4 Frontend: Using Kivy Python GUI Framework**

The frontend of our Chatbot system is developed using Kivy which is open source Python GUI framework. Kivy is chosen for our frontend application due to following reasons:

* It is open source Python library for rapid development of applications.
* Supports innovative user interfaces, such as multi-touch apps
* Cross Platform: Kivy runs on Linux, Windows, OS X, Android, iOS, and Raspberry Pi. Same code can be run on all supported platforms.
* The framework is stable
* The graphics engine is built over OpenGL ES 2, using a modern and fast graphics pipeline.
* The toolkit comes with more than 20 widgets, all highly extensible.
* Same codes can be burnt into Android device using Buildozer module. Hence, it is easy to build an Android App.
* Allows parallel processing and multi-threading processes enabling us to schedule events and contemporary processes.

**3.4.1 Application Design**

The application opens with the UI shown below:



1. **Home Button**: For coming back to the home page.
2. **Text Area/ Search Area**: User can enter their query here using a keyboard.
3. **Search Button**: Once the user is done writing his query in the text area, he can press this button to get answer or simply press ‘Enter’ on the keyboard.
4. **Microphone Input Button**: User can search for answers to his query simply by asking to this application instead of actually typing his query in search area. To do this, he can just speak up his question after clicking this button and immediately releasing it. The application may take some time to resolve the users’ query. The user needs to patiently wait for the application to respond.

\*\* The user is instructed not to long press this button.

1. **Exit Button**: Once the user is done with his search, he can press this button to exit the application and let other users use it. After pressing this button, the user can no more ask other questions from the same calling booth. As soon as this button is pressed, the BOT would return to its original Hotspot position.

Querying the application with phrases like ‘Bye’, ‘See you later’ also has the same outcome of pressing this button.

1. **User Manual Button**: This button is intended to guide the users about the usage of this application for their purposes. On pressing this button, application guides user to a ‘help’ page which explains the user about what all can the application do for him.
2. **Speaker Button**: This is a toggle button which is in ‘OFF’ state by default. By clicking this button ‘ON’, the application enables speaker. Hence, the user could hear answer to his queries in case he is unable to read or wants to hear the response to his query. It can be disabled by again clicking on it.
3. **Answer Area**: The responses to users query appear here.

**3.4.2 Features of Chatbot Application**

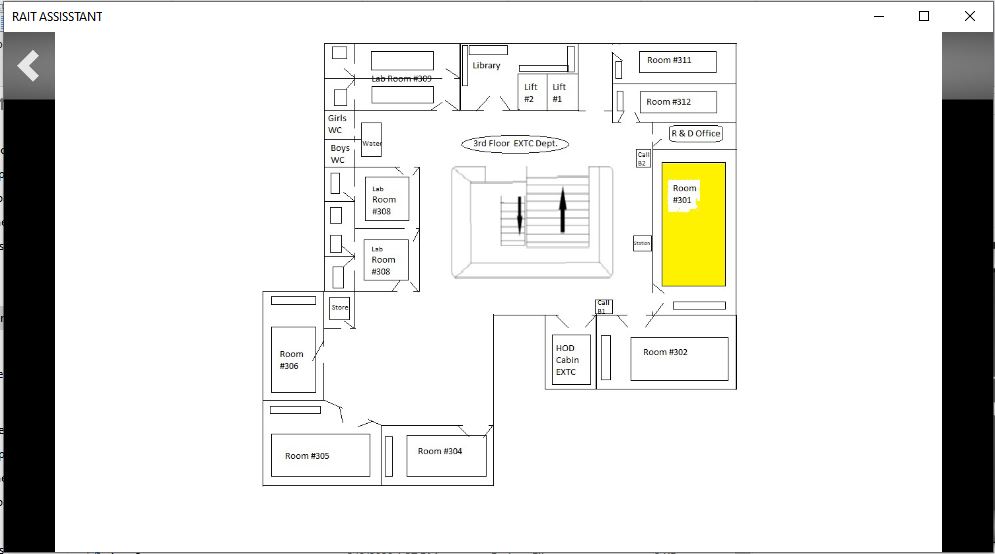
This Chatbot application solves the queries related:

* **Admission Process:**

The admission process for first new comers is a tedious task and most of them are confused about the exact procedure (‘where to go next?’, ‘what to do now?’).Our application answers such questions by pictorial representation of the procedure.

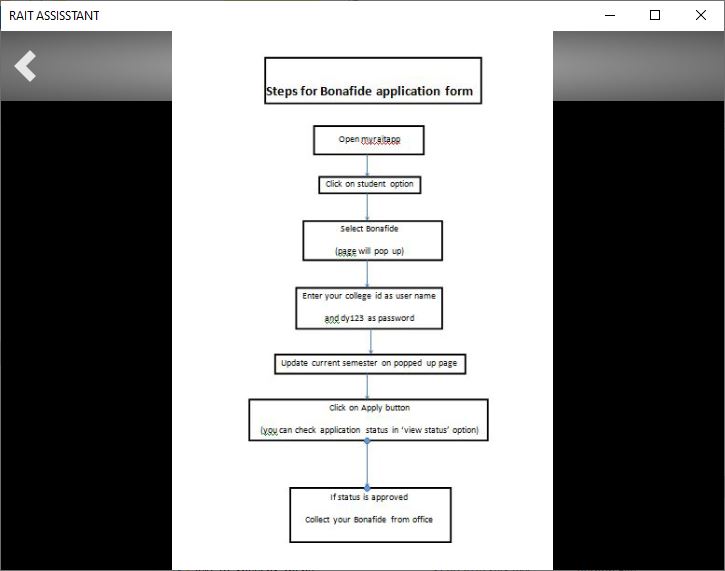
* **Location of class rooms/faculties/labs/ staff rooms and so on:**

Many students find trouble finding out where the faculty sits or where is a particular lab/ class room. Our application presents a viable solution for this. Our application pops the floor map for the user and highlights the destination required by the user. In this way, the user is guided to his destination. For example:

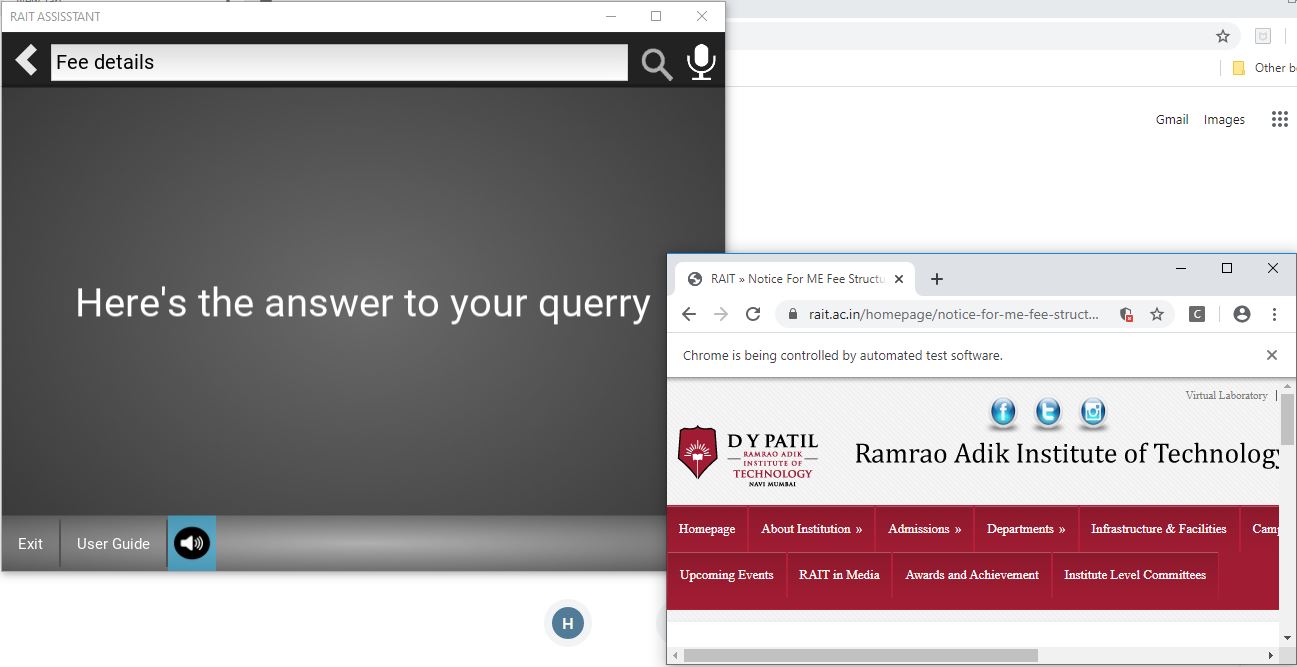


* **Procedure for Submitting forms such as Bonafide/Railway Concession:**

Many students are confused and generally don’t know the exact procedure for application of various day to day forms such as Bonafide, Scholarship, Railway Concession, KT forms and face problems finding the exact procedure. This application projects the exact steps in the form of a flowchart so that the procedure of such applications is clearly understood by the users.



* **Access any circulars such as time table, exam schedule, fee details updated by college in their website**: In this way, any news available in colleges’ website is made available directly to the user without scrapping over the website but just by querying in few simple words. When queried on, pops the browser window with the required link open for the user.



* **Quick ReadMe manual for users**: Our application provides a ‘Helper’ manual to guide the users about how to use the application. It also explains each components of the home screen and all the tasks that it can perform for them.

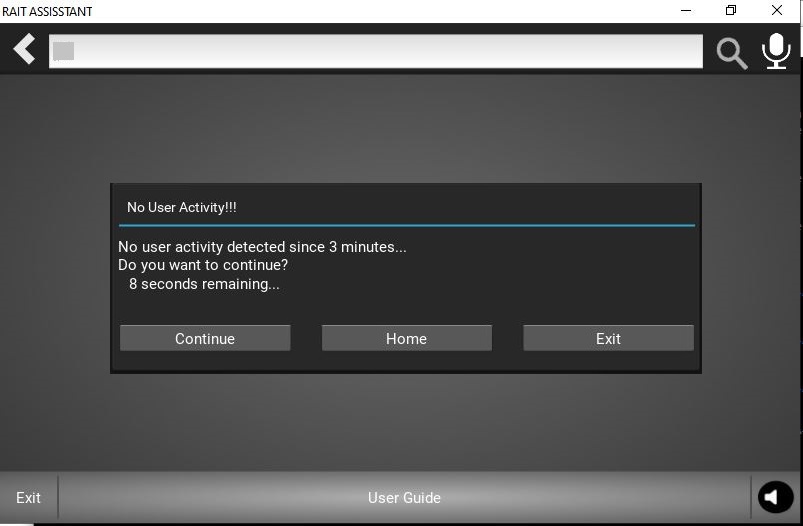
****

* **Detect idle activity of the user**: There might be cases where user hadn’t exited the application properly after he is finished with this query. This is a recipe for disaster since the BOT cannot know how long the user would take or whether the reader is still busy or he is done. The BOT might stand in the same hotspot assuming the user is busy.

To avoid such situation, the application pops up a dialog box after 3 minutes of user inactivity i.e. user is on same page or same query for 3 minutes. User can click **‘Continue’** button if he wants to continue reading or press **‘Exit’** button if he is done or press **‘Home’** to ask next question.

If none of the above options are chosen, the application would wait for 30 seconds for user to press any of the above buttons. If still nothing is pressed, the application would exit and the BOT would return to its original hotspot.

But, if the user presses ‘Continue’, again the dialog box would appear after every 3 minutes of user inactivity.



* **Security**: This is the most important and inevitable part of any system. Since there is a possibility for modification or eavesdropping of code files and other resources, our code files are compiled into C language using **Cython** module that makes our code files difficult to Reverse Engineer.

**3.5 Future Scope**

As rightly said, there should be no full stops for innovation, our Chatbot system can further be improvised by including following features:

* The application can be made multi-linguistic i.e. user can be given an option of asking queries in the language of his choice.
* In present system, we have included most of the data regarding our department (Electronics and Telecommunication Engineering) in our dataset. But data pertaining to other departments may also be included to cover whole about our college.
* In present sysytem, we have some latency issued while using Speech Recognition module for implementing Voice Input based answering system when Internet connection is slow (In our application only this part requires an Internet Connection, rest works well in offline conditions also). The algorithm for speech recognition may be optimized to reduce such latencies or it may be made offline so that the complete application flawlessly without internet connection.

|  |
| --- |
| References: |
|  | Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based |
|  | learning applied to document recognition." Proceedings of the IEEE, |
|  | 86(11):2278-2324, November 1998. |
|  |  |

<https://kivy.org/#home>