

QUERY CHATBOT

*A project report submitted in partial fulfillment of the requirements for the award of
the degree of*

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

submitted by

Kedarisetti Madhavi Prathyusha (18A31A0567)

Ramireddy Iswarya Krishna (19A35A0509)

Mamuduri Yashua Augustine (18A31A05B0)

Arigela Sai Kumar (18A31A0594)

Under the Guidance of

Mrs. P Srilakshmi

Asst. Professor, Dept. Of CSE



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**PRAGATI ENGINEERING COLLEGE
(AUTONOMOUS)**

(Approved by AICTE & Permanently Affiliated to JNTUK, Kakinada & Accredited by NAAC)

1-378, ADB Road, Surampalem, E.G.Dist., A.P, Pin-533437.

2020-2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PRAGATI ENGINEERING COLLEGE
(AUTONOMOUS)

(Approved by AICTE & Permanently Affiliated to JNTUK & Accredited by NAAC)

1-378, ADB Road, Surampalem, E.G.Dist., A.P, Pin-533437.



CERTIFICATE

*This is to certify that the report entitled “**QUERY CHATBOT**” that is being submitted by **K. Madhavi Prathyusha (18A31A0567), R. Iswarya Krishna (19A35A0509), M. Yashua Augustine (18A31A05B0), A. Sai Kumar (18A31A0594)** in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering, Pragati Engineering College is a record of bona fide work carried out by them.*

Supervisor

Mrs. P Srilakshmi

Assistant Professor, Dept. Of CSE

Head of the Department

Dr. M. Radhika Mani

Professor&HOD, Dept. Of CSE

ACKNOWLEDGMENT

In the accomplishment of this project successfully, many people have bestowed upon me their blessings and the heart pledged support, this time we are utilizing to thank all the people who have been concerned with this project.

Primarily we would like to thank our guide, **Mrs. Sri Lakshmi**, Assistant professor, department of Computer Science and Engineering because of her wholehearted and invaluable guidance throughout the report.

We would like to sincerely thank **Dr. M. Radhika Mani**, Professor & HOD, Computer Science and Engineering, for providing all the necessary facilities that lead to the successful completion of our report.

We would like to take this opportunity to thank our beloved principal and vice-principal, **Dr. S. Sambhu Prasad, Dr. K. Satyanarayana** for providing a great support to us in completing our project and for giving us the opportunity of doing the mini project report.

Finally, we would like to thank all the faculty members of the department of Computer Science and Engineering for helping us in completion of this project. And also, thanks to all of our friends and family members for their continuous help and encouragement.

K Madhavi Prathyusha (18A31A0567)

R Iswarya Krishna (19A35A0509)

M Yashua Augustine (18A31A05B0)

A Sai Kumar (18A31A0594)

ABSTRACT

Recently, chatbots received an increased attention from industry and diverse research communities as a dialogue-based interface providing advanced human-computer interactions. A chatbot is a computer program that simulates and processes human conversation. Chatbots are software programs that use natural language understanding and processing.

In this project we are going to develop a chatbot for our Pragati Engineering College. It is helpful for clarifying all the queries about college. At first, we developed a dataset in json format. Finally, we used Natural Language Toolkit (NLTK) in python for tokenization. We use various machine learning algorithms like Stochastic Gradient Descent (SGD) to train the data. We developed a web page by embedding the code using API. Through this web page one can interact with the chatbot for any quires related to the college activities without physically availability. The Chatbot analyses the query and responds to the user.

TABLE OF CONTENTS

ACKNOWLEDGMENT	iii	
ABSTRACT	iv	
LIST OF FIGURES	vii	
LIST OF TABLES	viii	
LIST OF ABBREIVATIONS	ix	
Chapter 1.	INTRODUCTION	
1.1	Chatbot	1
1.2	How Chatbot Works?	1
1.3	Types of chatbots	2
1.4	Objective	3
1.5	Requirements	3
Chapter 2.	LITERATURE SURVEY	
2.1	History	4
2.2	Early Approach	4
2.3	Modern Approach	5
2.4	Proposed System	5
Chapter 3.	METHODOLOGY	
3.1	Database	7
3.2	Required Libraries	8
3.3	Data Preprocessing	8
3.4	Creating Training Data	9
3.5	Creating Neural Network Model	10
Chapter 4.	CREATING INTERFACE	
4.1	Flask	12

4.2	Creating BoW for the input	13
4.3	Predicting Response	13
4.4	Text to Speech Conversion	14
4.5	Flow Chart	14
Chapter 5.	RESULTS AND DIMENSIONS	
5.1	Model Summary	16
5.2	Webpage	16
Chapter 6.	CONCLUSION AND FUTURE SCOPE	
6.1	Conclusion	18
6.2	Future Scope	18
BIBLIOGRAPHY		19
APPENDIX-A	Source Code	20
APPENDIX-B	Driver Code	23
APPENDIX-C	Template Files	26

LIST OF FIGURES

Figure No.	Name of the Figure	Page. No.
2.1	CleverBot	4
2.2	Block Diagram for Proposed System	6
3.1	JSON Database Format	7
3.2	Data Preprocessing	9
3.3	Training Dataset	10
3.4	Creating Model	10
3.5	Saving Model	11
4.1	Creating Flask App	12
4.2	Creating Bag of words for input	13
4.3	Predicting Response	14
4.4	Text to Speech Conversion	14
4.5	Flow Chart	15
5.1	Summary	16
5.2	Output	17
5.3	Webpage	17

LIST OF TABLES

Table No.	Name of the Table	Page. No.
3.1	Data frame	9

LIST OF ABBREVIATIONS

AIML	Artificial Intelligence Markup Language
NLU	Natural Language Understanding
NLP	Natural Language Processing
AI	Artificial Intelligence
IDE	Integrated Development Environment
JSON	Javascript Object Notion
NLTK	Natural Language Tool Kit
BoW	Bag of Words
API	Application Programming Interface
ReLU	Rectified Linear Units
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
WSGI	Web Server Gateway Interface

CHAPTER 1

INTRODUCTION

1.1 Chatbot

Chatbot is a piece of software that can communicate with people using natural language. A chatbot is an artificial intelligence (AI) program that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, mobile applications or by phone. For example, if you want to buy some shoes from your local retail store, you can access their website, find what you are looking for and buy it. But what if that store had a bot? It would only be necessary to write a message to the brand through Facebook and tell them what we want. And if you had doubts about size measurements you could get answers to your problem in a moment. One of the great advantages of chatbots is that, unlike applications, they are not downloaded, it is not necessary to update them, and they do not take up space in the phone's memory. Another one is that we can have several bots integrated in the same chat.

1.2 How Chatbot works?

Chatbots work by analyzing and identifying the intent of the user's request to extract relevant entities, which is the most important task of a chatbot. Once the analysis is done appropriate response is delivered to the user.

The chatbots work by adopting three classification methods. They are:

1.2.1 Pattern matching

Bots utilize pattern matches to group the text and it produces an appropriate response from the clients. Artificial Intelligence Markup Language (AIML) is a standard structured model of these patterns. A bot is able to get the right answer in the related pattern. The bots react to anything relating to the correlated patterns.

1.2.2 Natural language understanding (NLU)

Natural language understanding (NLU) is the ability of the chatbot to understand a human. It is the process of converting text into structured data for a machine to understand. NLU follows three specific concepts. They are entities, context, and expectations.

1.2.3 Natural language processing (NLP)

Natural Language Processing (NLP) bots are designed to convert the text or speech inputs of the user into structured data. The data is further used to choose a relevant answer. NLP includes important steps such as tokenization, sentiment analysis, entity recognition, and dependency parsing.

1.3 Types of Chatbots

Chatbots process data to deliver quick responses to all kinds of users' requests with pre-defined rules and AI based chatbots. There are two types of chatbots. They are:

1.3.1 Rule based chatbot:

Rule-based chatbots also referred to as decision-tree bots, use a series of defined rules. These rules are the basis for the types of problems the chatbot is familiar with and can deliver solutions for.

Key attributes or rule based chatbots:

- These bots follow predetermined rules. So, it becomes easy to use the bot for simpler scenarios.
- Interactions with rule based chatbots are highly structured and are most applicable to customer support functions.
- Rule based bots are ideally suitable for answering common queries such as an inquiry about business hours, delivery status, or tracking details.

1.3.2 Conversational AI chatbots:

Conversational chatbots combine the power of machine learning and NLP to understand the context and intent of a question before formulating a response. These chatbots generate their own answers to more complicated questions using natural-language responses. The more you use and train these bots, the more they learn and the better they operate with the user.

The conversational communication skills of the chatbot technology empower them to deliver what customers are looking for.

Key attributes of AI enabled chatbots:

- Conversational bots can understand the context and intent of complex conversations and try to provide more relevant answers.
- AI bots apply predictive intelligence and sentiment analysis to understand customer emotions closely.
- Machine learning bots learn from user behavior and provide more personalized conversations.

1.4 Objective

The goal of Query Chatbot project is to build a Conversation AI chatbot that adopts NLP classification method for building a chatbot that solves all the queries of Pragati Engineering College. In this chatbot we are likely to include details like teaching staff, extracurricular activities, placement opportunities, upcoming exams, result details etc.,

1.5 Requirements

- Laptop or PC with minimum 4gb ram
- Python 3.7 or more
- PyCharm IDE

CHAPTER 2

LITERATURE SURVEY

2.1 History of chatbots

Chatbot models usually take as input natural language sentences uttered by a user and output a response. There are two main approaches for generating responses. The traditional approach is to use hard-coded templates and rules to create chatbots. The more novel approach was made possible by the rise of deep learning. Neural network models are trained on large amounts of data to learn the process of generating relevant and grammatically correct responses to input utterances.

2.2 Early Approach

ELIZA is one of the first ever chatbot programs written. It uses clever handwritten templates to generate replies that resemble the user's input utterances. Since then, countless hand-coded, rule-based chatbots have been developed. An example can be seen in fig 2.1. Furthermore, a number of programming frameworks specifically designed to facilitate building dialog agents have been developed.

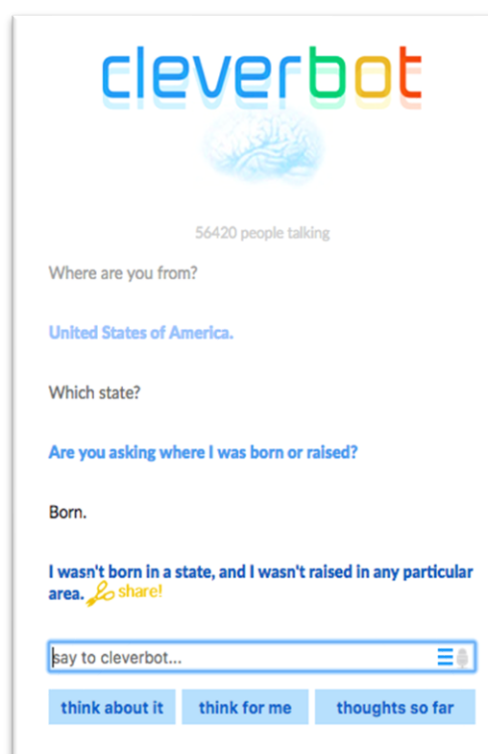


Fig.2.1 CleverBot

These chatbot programs are very similar in their core, namely that they all use hand-written rules to generate replies. Usually, simple pattern matching, or keyword retrieval techniques are employed to handle the user's input utterances. Then, rules are used to transform a matching pattern or a keyword into a predefined reply. A simple example is shown below in AIML.

<category>

<pattern>What is your name? </pattern>

<template>My name is Alice</template>

</category>

Here if the input sentence matches the sentence written between the brackets the reply written between the brackets is outputted.

2.3 Modern Approach – Encoder Decoder Model

The main concept that differentiates rule-based and neural network-based approaches is the presence of a learning algorithm in the latter case. An important distinction has to be made between traditional machine learning and deep learning which is a sub-field of the former. In this work, only deep learning methods applied to chatbots are discussed, since neural networks have been the backbone of conversational modeling and traditional machine learning methods are only rarely used as supplementary techniques.

2.4 Proposed System

The proposed system of query chatbot uses the modern approach. It uses sequential model and relu activation function as it is easier to train and achieve good performance. It takes input as string via webpage and result is displayed as text and voice. It is easy to use. It doesn't require any download or update. It doesn't require any login also. The block diagram for proposed system is as shown in fig 2.2.

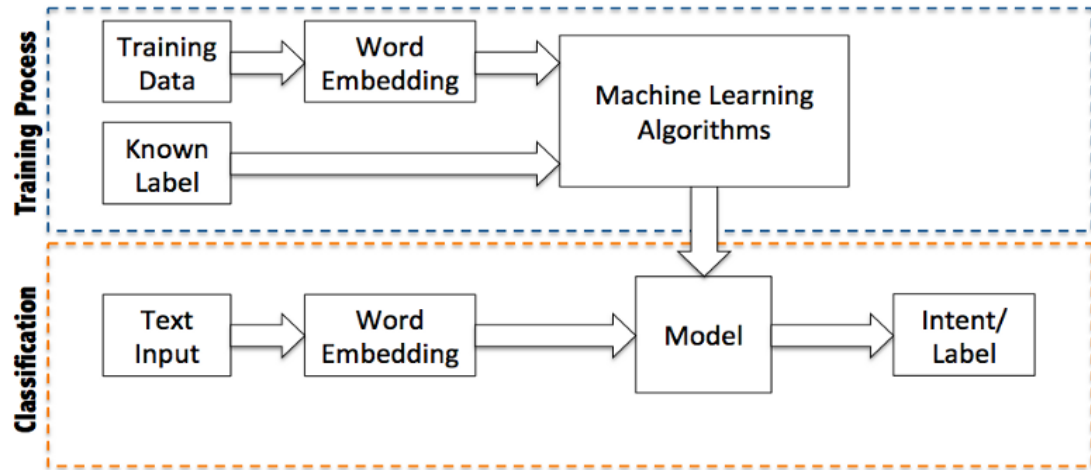


Fig.2.2 Block Diagram for Proposed System

CHAPTER 3

METHODOLOGY

3.1 Database

We need to set up an intents JSON file that defines certain intentions that could occur during the interactions with our chatbot. To perform this, we would have to first create a set of tags that user queries may fall into. For example:

- A user may wish to know the name of our chatbot, therefore we create an intention labelled with a tag called “name”.
- A user may wish to know the age of our chatbot, therefore we create an intention labelled with the tag “age”.

For each of the tags that we create, we would have to specify patterns. Essentially, this defines the different ways of how a user may pose a query to our chatbot. For instance, under the name tag, a user may ask someone's name in a variety of ways - “What’s your name?”, “Who are you?”, “What are you called?”.

The chatbot would then take these patterns and use them as training data to determine what someone asking for our chatbot's name would look like so that it could adapt to the different ways someone may ask to know our bot's name. Therefore, users wouldn’t have to use the exact queries that our chatbot has learned. It could pose the question as “What are you called?” and our chatbot would be able to infer that the user wants to know the name of our chatbot and then it would provide its name. The following fig 3.1 shows the example for designing the json database.

```
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": ["Hi there", "Is anyone there?", "Hey", "Hola", "Hello"],
      "responses": ["Hello!", "Good to see you again", "Hi there, how can I help?"]
    },
    {
      "tag": "wishes",
      "patterns": ["How are you", "Good Day", "How do you do?"],
      "responses": ["Greetings", "Fine Day", "Good Day"]
    },
    {
      "tag": "goodbye",
      "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],
      "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."]
    },
    {
      "tag": "thanks",
      "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
      "responses": ["Happy to help!", "Any time!", "My pleasure", "Your Welcome!"]
    }
  ]
}
```

Fig.3.1 JSON Database Format

3.2 Required Libraries

Before coding our bot, we require some Python built-ins, as well as popular libraries for NLP, deep learning, as well as the defacto library NumPy which is great for dealing with arrays. So, the following libraries should be downloaded.

nlTK	→	used for data preprocessing
json	→	used to work with json data
pickle	→	used in serializing and deserializing a Python object structure
numpy	→	provides a multidimensional array object
keras	→	used to create model
random	→	used for text-to-speech conversion
pyttsx3	→	used for text-to-speech conversion
flask	→	build up web-applications

3.3 Data Pre-processing

Data preprocessing is a process of preparing the raw data and making it suitable for a model. It is the first and crucial step while creating any machine learning model. So here we take the data in json dataset and perform data processing steps. The fig 3.2 describes the code for data preprocessing.

The steps in data preprocessing are:

- 1) **Importing essential libraries:** The library required for data pre-processing is nlTK (Natural Language Tool Kit). So, import nlTK library in our python file. Import json library since it is used to work with json database.
- 2) **Tokenization:** *Tokens* are individual words and “*tokenization*” is taking a text or set of text and breaking it up into its individual words or sentences. Tokenize all the sentences in the patterns key in json database and store them in a list along with the tag.
- 3) **Lemmatization:** Lemmatization is the process of grouping together the different inflected forms of words, so they can be analysed as a single item and is a variation of stemming. For example, “feet” and “foot” are both recognized as “foot”. Now the tokenized words in lists are lemmatized by using `lemmatize()` by ignoring the unnecessary symbols like `?,!` etc.,

- 4) **Pickle files:** The lemmatized words are sorted and stored in a list. These lists are dumped into pickle files with the help of pickle library.

```

words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('intents.json').read()
intents = json.loads(data_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'])
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)))
print (len(documents), "documents")
print (len(classes), "classes", classes)
print (len(words), "unique lemmatized words", words)
pickle.dump(words, open('words.pkl', 'wb'))
pickle.dump(classes, open('classes.pkl', 'wb'))

```

Fig.3.2 Data Preprocessing

3.4 Creating Training Data

Neural Networks expect numerical values, and not words, to be fed into them. Therefore, first we have to process our data so that a neural network could read what we are doing. In order to convert our data to numerical values, we are having many techniques like One-Hot encoding, Index-based encoding, Bag of words, Word Embeddings etc., In this project we are using Bag of words.

The Bag of Words encoding technique derives its name from the fact that any information or structure of the words in a document is discarded, therefore, it is like we put the set of words into a bag and given it a shake. It only cares if a word appears in a document and the number of times it appeared. It does not care where it appeared.

For example, if we have the list like:

Corpus = ["I can't wait to get out of lockdown", "India is soon going to be fine soon"]

	be	cant	fine	get	going	i	india	is	lockdown	of	out	soon	to	wait
0		1		1		1			1	1	1		1	1
1	1		1		1		1	1				2	1	

Table.3.1 Data frame

After implementing Bag of Words (BoW) on the corpus the data frame would be as shown in table 3.1.

So, after lemmatizing we create a bag of words array with 1, if word match is found in current pattern. Shuffle the data and convert it into an array as shown in figure 3.3.

```
training = []
output_empty = [0] * len(classes)
for doc in documents:
    bag = []
    pattern_words = doc[0]
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    for w in words:
        bag.append(1 if w in pattern_words else 0)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1
    training.append([bag, output_row])
random.shuffle(training)
training = np.array(training)
train_x = list(training[:, 0])
train_y = list(training[:, 1])
print("Training data created")
```

Fig.3.3 Training Dataset

3.5 Creating Neural Network Model

With our data converted into a numerical format, we can now build a Neural Network model that we are going to feed our training data into. The idea is that the model will look at the features and predict the tag associated with the features then will select an appropriate response from that tag.

Neural networks are mathematical models that use learning algorithms inspired by the brain to store information. Since neural networks are used in machines, they are collectively called an ‘artificial neural network.’ For creating neural network model, we use Sequential API model which is available in keras library. The Sequential model API is a way of creating deep learning models where an instance of the Sequential class is created, and model layers are created and added to it as shown in figure 3.4. Sequential API allows you to create models layer-by-layer.

```
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'))
print(model.summary())
```

Fig.3.4 Creating Model

We shall create a 3-layer output model. The first layer having 128 neurons, the second layer having 64 neurons, and the third layer contains the number of neurons equal to the number of intents to predict output intent with softmax. We shall be using “ReLU” activation function as it's easier to train and achieves good performance.

We compile the model with Stochastic gradient descent with Nesterov accelerated gradient that gives good results for this model as shown in fig 3.5. Now that we have our model trained, we need to fit it and save it in .h5 format for fast access from the Flask REST API.

```
sgd = SGD(lr=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=200, batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist)
print("model created")
```

Fig.3.5 Saving Model

CHAPTER 4

CREATING INTERFACE

After training our Deep Learning model, we have to create the actual features that would allow us to use our model. For it we have to create webpage. To create the webpage we use the HTML, CSS and Java Script. To create API, we used flask framework.

4.1 Flask

Flask is a micro web framework written in Python. It is an API of Python that allows us to build up web-applications. Flask Framework is the collection of modules and libraries that helps the developer to write applications without writing the low-level codes such as protocols, thread management, etc. Flask is based on WSGI (Web Server Gateway Interface) toolkit and Jinja2 template engine. To use the flask framework first we import Flask from the flask package. Along with it we load the required libraries and the saved neural network model and also the pickle files. Initialize the flask app to get the input from user through webpage and giving response to the user by using the routing methods as shown in figure 4.1.

```
import nltk
from nltk.stem import WordNetLemmatizer
import pickle
import numpy as np
from keras.models import load_model
import json
import random
lemmatizer = WordNetLemmatizer()
model = load_model('chatbot_model.h5')
intents = json.loads(open('intents.json').read())
words = pickle.load(open('words.pkl', 'rb'))
classes = pickle.load(open('classes.pkl', 'rb'))
from flask import Flask, render_template, request
app = Flask(__name__)
app.static_folder='static'
@app.route("/")
def home():
    return render_template("index.html")
@app.route("/get")
def get_bot_response():
    userText = request.args.get('msg')
    res=chatbot_response(userText)
    text_to_speech(res)
    return res
```

Fig.4.1 Creating Flask App

This `get_bot_response()` function in figure 4.1 will be called every time when a user sends a message to the chatbot and returns a corresponding response based on the user query. It will call the other functions which are used to clean up sentences and return a bag of words based on the user input.

4.2 Creating BoW for the input

After getting the input from the user through the webpage, the input is passed to the functions in chatbot using the methods in flask. We first clean up the sentence by tokenizing the sentences, lemmatizing the tokenized words and storing them in a list. This list is then used to create bag of words (BoW) by using the function `bow()` as shown in figure 4.2.

```
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words
def bow(sentence, words, show_details=True):
    sentence_words = clean_up_sentence(sentence)
    bag = [0] * len(words)
    for s in sentence_words:
        for i, w in enumerate(words):
            if w == s:
                bag[i] = 1
                if show_details:
                    print("found in bag: %s" % w)
    return np.array(bag)
```

Fig.4.2 Creating Bag of words for input.

4.3 Predicting response

The bag of words is used to predict the class by using the saved neural network model. It returns a list of classes and the probability which is sorted by the strength of probability. The strongest probability tag is traced through the database to get the matched tag. After getting the matched tag, a random response is returned as output as shown in figure 4.3.

```

def predict_class(sentence, model):
    p = bow(sentence, words, show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list

def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if (i['tag'] == tag):
            result = random.choice(i['responses'])
            break
    return result

def chatbot_response(msg):
    ints = predict_class(msg, model)
    res = getResponse(ints, intents)
    return res

```

Fig.4.3 Predicting Response

4.4 Text to Speech Conversion

For conversion from text to speech we use pyttsx3 library. Unlike alternative libraries, it works offline. An application invokes the pyttsx3.init() factory function to get a reference to a pyttsx3.Engine instance. it is a very easy to use tool which converts the entered text into speech. With engine instance we set the rate and volume of the text to be spoken as shown in figure 4.4.

```

import pyttsx3
def text_to_speech(text):
    engine = pyttsx3.init()
    engine.setProperty('rate', 175)
    engine.setProperty('volume', 1)
    engine.say(text)
    engine.runAndWait()

```

Fig.4.4 Text to Speech Conversion

4.5 Flowchart

The following figure 4.5 shows the process flow of our chatbot model.

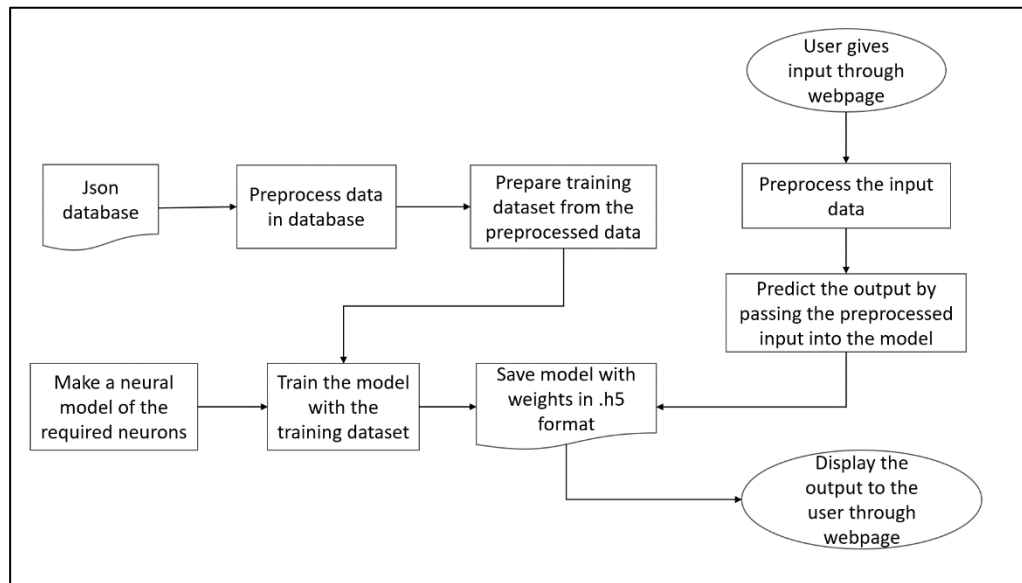


Fig.4.5 Flow Chart

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Model Summary

Keras provides a way to summarize our model. The summary is textual and includes information about:

- The layers and their order in the model.
- The output shape of each layer.
- The number of parameters (weights) in each layer.
- The total number of parameters (weights) in the model.

The summary can be created by calling the `summary()` function on the model that returns a string that in turn can be printed.

The following figure 5.1 shows the summary of our model created.

Model: "sequential"		
Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(None, 128)	12032

dropout (Dropout)	(None, 128)	0

dense_1 (Dense)	(None, 64)	8256

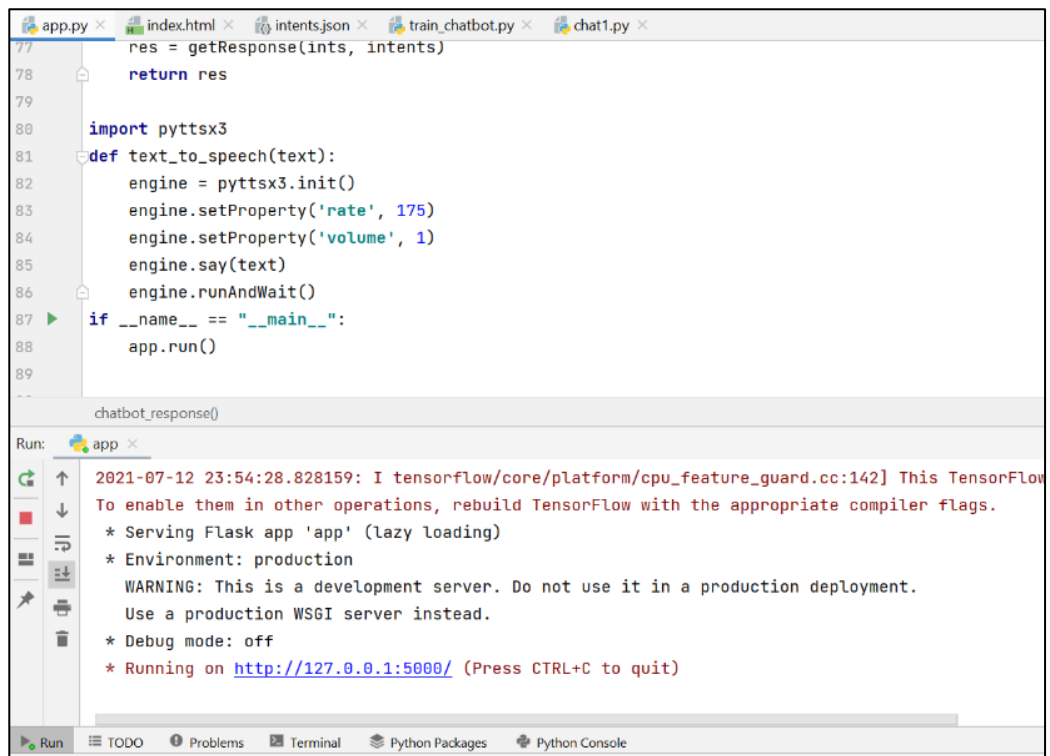
dropout_1 (Dropout)	(None, 64)	0

dense_2 (Dense)	(None, 17)	1105
=====		
Total params: 21,393		
Trainable params: 21,393		
Non-trainable params: 0		
=====		

Fig.5.1 Summary

5.2 Webpage

After training of model and creating the API to access the model we get the local host server website for using the chatbot which is as shown in figure 5.2.



```

app.py | index.html | intents.json | train_chatbot.py | chat1.py
77 res = getResponse(ints, intents)
78 return res
79
80 import pyttsx3
81 def text_to_speech(text):
82     engine = pyttsx3.init()
83     engine.setProperty('rate', 175)
84     engine.setProperty('volume', 1)
85     engine.say(text)
86     engine.runAndWait()
87 if __name__ == "__main__":
88     app.run()
89
chatbot_response()

Run: app
2021-07-12 23:54:28.828159: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

```

Fig.5.2 Output

On clicking the link, it navigates to local host website which is as shown in figure 5.3. We can chat with the chatbot by sending the messages and pressing either “Enter” button or by clicking send button on desktop. The output is displayed in the form of text and voice.

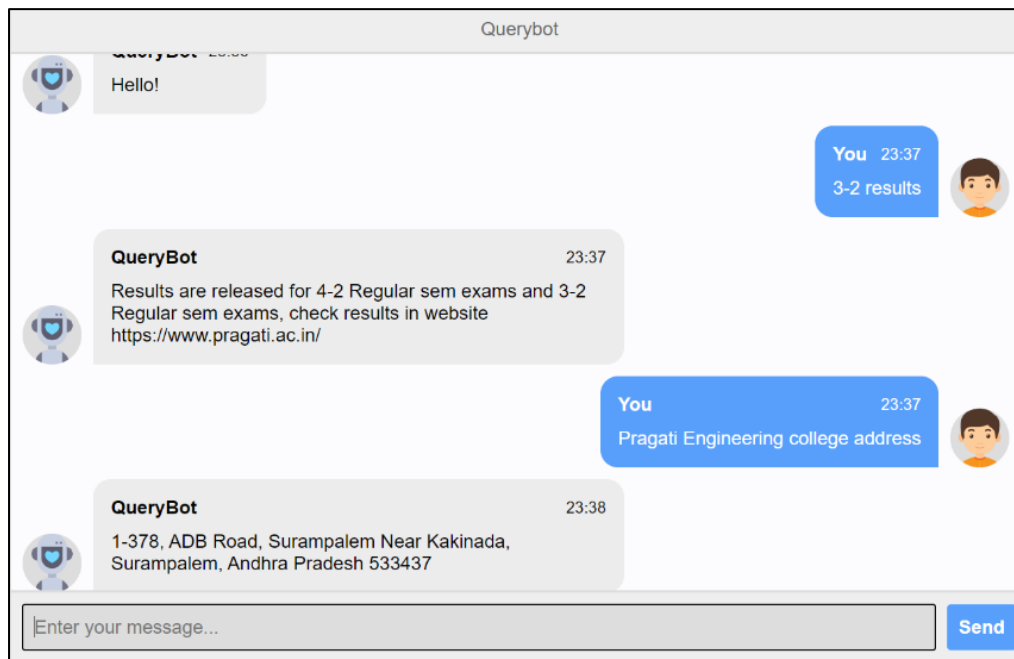


Fig.5.3 Webpage

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

- With the help of this chatbot we can get any new notifications regarding college.
- It is easy to handle.
- Anyone can use this application without any login.

6.2 Future Scope

We can also make chatbot smarter by adding voice recognition technique. This helps the user to communicate with chatbot with voice commands rather than the text.

BIBLIOGRAPHY

1. Natural Language Processing with Python by Steven Bird, Ewan Klein, Edward Loper, Released June 2009, Publisher(s): O'Reilly Media, Inc., ISBN: 9780596516499
2. https://dair.ai/notebooks/nlp/2020/03/19/nlp_basics_tokenization_segmentation.html
3. Weizenbaum, Joseph (*January 1966*), "*ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine*", *Communications of the ACM*, 9 (1): 36–45, doi:10.1145/365153.365168, S2CID 1896290
4. <https://machinelearningmastery.com/keras-functional-api-deep-learning/>
5. <https://towardsdatascience.com/a-simple-chatbot-in-python-with-deep-learning-3e8669997758>
6. <https://dev.to/dennismaina/how-to-create-an-ai-chatbot-in-python-and-flask-1c3m>
7. <https://towardsdatascience.com/a-guide-to-encoding-text-in-python-ef783e50f09e>
8. Ronacher, Armin. - <https://web.archive.org/web/20160604162342/http://mitsuhiko.pocoo.org/flask-pycon-2011.pdf> (PDF). Archived from <http://mitsuhiko.pocoo.org/flask-pycon-2011.pdf> (PDF) on 2016-12-17. Retrieved 2011-09-30.

APPENDIX-A

SOURCE CODE

#Code to train chatbot

```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random

words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('intents.json').read()
intents = json.loads(data_file)

for intent in intents['intents']:
    for pattern in intent['patterns']:

        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)

        #add documents in the corpus
        documents.append((w, intent['tag']))

# add to our classes list
```

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

# lemmatize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))

# sort classes
classes = sorted(list(set(classes)))

# documents = combination between patterns and intents
print (len(documents), "documents")

# classes = intents
print (len(classes), "classes", classes)

# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)


pickle.dump(words, open('words.pkl', 'wb'))
pickle.dump(classes, open('classes.pkl', 'wb'))


# create our training data
training = []

# create an empty array for our output
output_empty = [0] * len(classes)

# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []

    # list of tokenized words for the pattern
    pattern_words = doc[0]

    # lemmatize each word - create base word, in attempt to represent related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]

    # create our bag of words array with 1, if word match found in current pattern
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)
```

```
# output is a '0' for each tag and '1' for current tag (for each pattern)
output_row = list(output_empty)
output_row[classes.index(doc[1])] = 1

training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:, 0])
train_y = list(training[:, 1])
print("Training data created")

# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd
output layer contains number of neurons
# equal to number of intents to predict output intent with softmax
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'))
print(model.summary())

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient
gives good results for this model
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

# fitting and saving the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5,
verbose=1)
model.save('chatbot_model.h5', hist)
print("model created")
```

APPENDIX-B

DRIVER CODE

Driver code for running chatbot

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

lemmatizer = WordNetLemmatizer()
model = load_model('chatbot_model.h5')
intents = json.loads(open('intents.json').read())
words = pickle.load(open('words.pkl', 'rb'))
classes = pickle.load(open('classes.pkl', 'rb'))
from flask import Flask, render_template, request
app = Flask(__name__)
app.static_folder='static'
@app.route("/")
def home():
    return render_template("index.html")
@app.route("/get")
def get_bot_response():
    userText = request.args.get('msg')
    res=chatbot_response(userText)
    text_to_speech(res)
    return res

def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
```

```
# stem each word - create short form for word

sentence_words = [lemmatizer.lemmatize(word.lower()) for word in
sentence_words]

return sentence_words


# return bag of words array: 0 or 1 for each word in the bag that exists in the
sentence

def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0] * len(words)
    for s in sentence_words:
        for i, w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
            if show_details:
                print("found in bag: %s" % w)
    return np.array(bag)


def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words, show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list
```

```
def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if (i['tag'] == tag):
            result = random.choice(i['responses'])
            break
    return result
```

```
def chatbot_response(msg):
    ints = predict_class(msg, model)
    res = getResponse(ints, intents)
    return res
```

```
import pyttsx3
def text_to_speech(text):
    engine = pyttsx3.init()
    engine.setProperty('rate', 175)
    engine.setProperty('volume', 1)
    engine.say(text)
    engine.runAndWait()
if __name__ == "__main__":
    app.run()
```

APPENDIX-C

TEMPLATE FILES

HTML file:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Chatty</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="shortcut icon" href="#" />

  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <style>
    :root {
      --body-bg: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
      --msger-bg: #fff;
      --border: 2px solid #ddd;
      --left-msg-bg: #ececfc;
      --right-msg-bg: #579ffb;
    }

    html {
      box-sizing: border-box;
    }

    *,
    *:before,
    *:after {
```

```
margin: 0;
padding: 0;
box-sizing: inherit;
}
```

```
body {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background-image: var(--body-bg);
  font-family: Helvetica, sans-serif;
}
```

```
.msger {
  display: flex;
  flex-flow: column wrap;
  justify-content: space-between;
  width: 100%;
  max-width: 867px;
  margin: 25px 10px;
  height: calc(100% - 50px);
  border: var(--border);
  border-radius: 5px;
  background: var(--msger-bg);
  box-shadow: 0 15px 15px -5px rgba(0, 0, 0, 0.2);
}
```

```
.msger-header {
  /* display: flex; */
  font-size: medium;
  justify-content: space-between;
  padding: 10px;
  text-align: center;
```

```
border-bottom: var(--border);  
background: #eee;  
color: #666;  
}
```

```
.msger-chat {  
  flex: 1;  
  overflow-y: auto;  
  padding: 10px;  
}  
.msger-chat::-webkit-scrollbar {  
  width: 6px;  
}  
.msger-chat::-webkit-scrollbar-track {  
  background: #ddd;  
}  
.msger-chat::-webkit-scrollbar-thumb {  
  background: #bdbdbd;  
}  
.msg {  
  display: flex;  
  align-items: flex-end;  
  margin-bottom: 10px;  
}
```

```
.msg-img {  
  width: 50px;  
  height: 50px;  
  margin-right: 10px;  
  background: #ddd;  
  background-repeat: no-repeat;  
  background-position: center;  
  background-size: cover;  
  border-radius: 50%;
```

```
}  
.msg-bubble {  
  max-width: 450px;  
  padding: 15px;  
  border-radius: 15px;  
  background: var(--left-msg-bg);  
}  
.msg-info {  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  margin-bottom: 10px;  
}  
.msg-info-name {  
  margin-right: 10px;  
  font-weight: bold;  
}  
.msg-info-time {  
  font-size: 0.85em;  
}  
  
.left-msg .msg-bubble {  
  border-bottom-left-radius: 0;  
}  
  
.right-msg {  
  flex-direction: row-reverse;  
}  
.right-msg .msg-bubble {  
  background: var(--right-msg-bg);  
  color: #fff;  
  border-bottom-right-radius: 0;  
}  
.right-msg .msg-img {
```

```
margin: 0 0 0 10px;
}

.msger-inputarea {
  display: flex;
  padding: 10px;
  border-top: var(--border);
  background: #eee;
}
.msger-inputarea * {
  padding: 10px;
  border: none;
  border-radius: 3px;
  font-size: 1em;
}
.msger-input {
  flex: 1;
  background: #ddd;
}
.msger-send-btn {
  margin-left: 10px;
  background: #579ffb;
  color: #fff;
  font-weight: bold;
  cursor: pointer;
  transition: background 0.23s;
}
.msger-send-btn:hover {
  background: rgb(0, 180, 50);
}

.msger-chat {
  background-color: #fcfcfe;
}
```

```
</style>
</head>

<body>
  <!-- partial:index.partial.html -->
  <section class="msger">
    <header class="msger-header">
      <div class="msger-header-title">
        <i> Querybot </i>
      </div>
    </header>

    <main class="msger-chat">
      <div class="msg left-msg">
        <div class="msg-img" style="background-image:
url(https://image.flaticon.com/icons/svg/327/327779.svg)"></div>

        <div class="msg-bubble">
          <div class="msg-info">
            <div class="msg-info-name">Querybot</div>
            <div class="msg-info-time"></div>
          </div>

          <div class="msg-text">
            Hi, welcome to chat! We are here to solve any queries of Pragati Engineering
            College. 😊
          </div>
          <div class="msg-text">
            you can ask me anything related to college fee, bus fee, upcoming events etc..
          </div>
        </div>
      </div>
    </main>
  </section>
</body>
```



```
</main>

<form class="msger-inputarea">
  <input type="text" class="msger-input" id="textInput" placeholder="Enter your
message...">
  <button type="submit" class="msger-send-btn">Send</button>
</form>
</section>
<!-- partial -->
<script src='https://use.fontawesome.com/releases/v5.0.13/js/all.js'></script>
<script>

const msgerForm = get(".msger-inputarea");
const msgerInput = get(".msger-input");
const msgerChat = get(".msger-chat");

// Icons made by Freepik from www.flaticon.com
const BOT_IMG = "https://image.flaticon.com/icons/svg/327/327779.svg";
const PERSON_IMG = "https://image.flaticon.com/icons/svg/145/145867.svg";
const BOT_NAME = "QueryBot";
const PERSON_NAME = "You";

msgerForm.addEventListener("submit", event => {
  event.preventDefault();

  const msgText = msgerInput.value;
  if (!msgText) return;

  appendMessage(PERSON_NAME, PERSON_IMG, "right", msgText);
  msgerInput.value = "";
  botResponse(msgText);
});
```

```
function appendMessage(name, img, side, text) {  
  // Simple solution for small apps  
  const msgHTML = `  
<div class="msg ${side}-msg">  
  <div class="msg-img" style="background-image: url(${img})"></div>  
  <div class="msg-bubble">  
    <div class="msg-info">  
      <div class="msg-info-name">${name}</div>  
      <div class="msg-info-time">${formatDate(new Date())}</div>  
    </div>  
    <div class="msg-text">${text}</div>  
  </div>  
</div>  
`;  
};
```

```
msgerChat.insertAdjacentHTML("beforeend", msgHTML);  
msgerChat.scrollTop += 500;  
}
```

```
function botResponse(rawText) {  
  
  // Bot Response  
  $.get("/get", { msg: rawText }).done(function (data) {  
    console.log(rawText);  
    console.log(data);  
    const msgText = data;  
    appendMessage(BOT_NAME, BOT_IMG, "left", msgText);  
  
  });  
  
}
```

```
// Utils
```

```
function get(selector, root = document) {  
  return root.querySelector(selector);  
}  
  
function formatDate(date) {  
  const h = "0" + date.getHours();  
  const m = "0" + date.getMinutes();  
  
  return `${h.slice(-2)}:${m.slice(-2)}`;  
}  
</script>  
  
</body>  
  
</html>
```