

Table of Contents

Abstract3

Purpose of the problem statement 4

Proposed Method with Flow Diagram4

Modules Description5

Python Main Code5

Training Data8

JSON Files and error checking8

Implementation Requirements10

Output Screenshots11

Abstract:

The technology at the core of the rise of the chatbot is natural language processing (“NLP”). Recent advances in machine learning have greatly improved the accuracy and effectiveness of natural language processing, making chatbots a viable option for many organizations. This improvement in NLP is firing a great deal of additional research which should lead to continued improvement in the effectiveness of chatbots in the years to come.

A simple chatbot can be created by loading an FAQ (frequently asked questions) into chatbot software. The functionality of the chatbot can be improved by integrating it into the organization’s enterprise software, allowing more personal questions to be answered, like “What is my balance?”, or “What is the status of my order?”.

Purpose of the problem statement:

Problem:

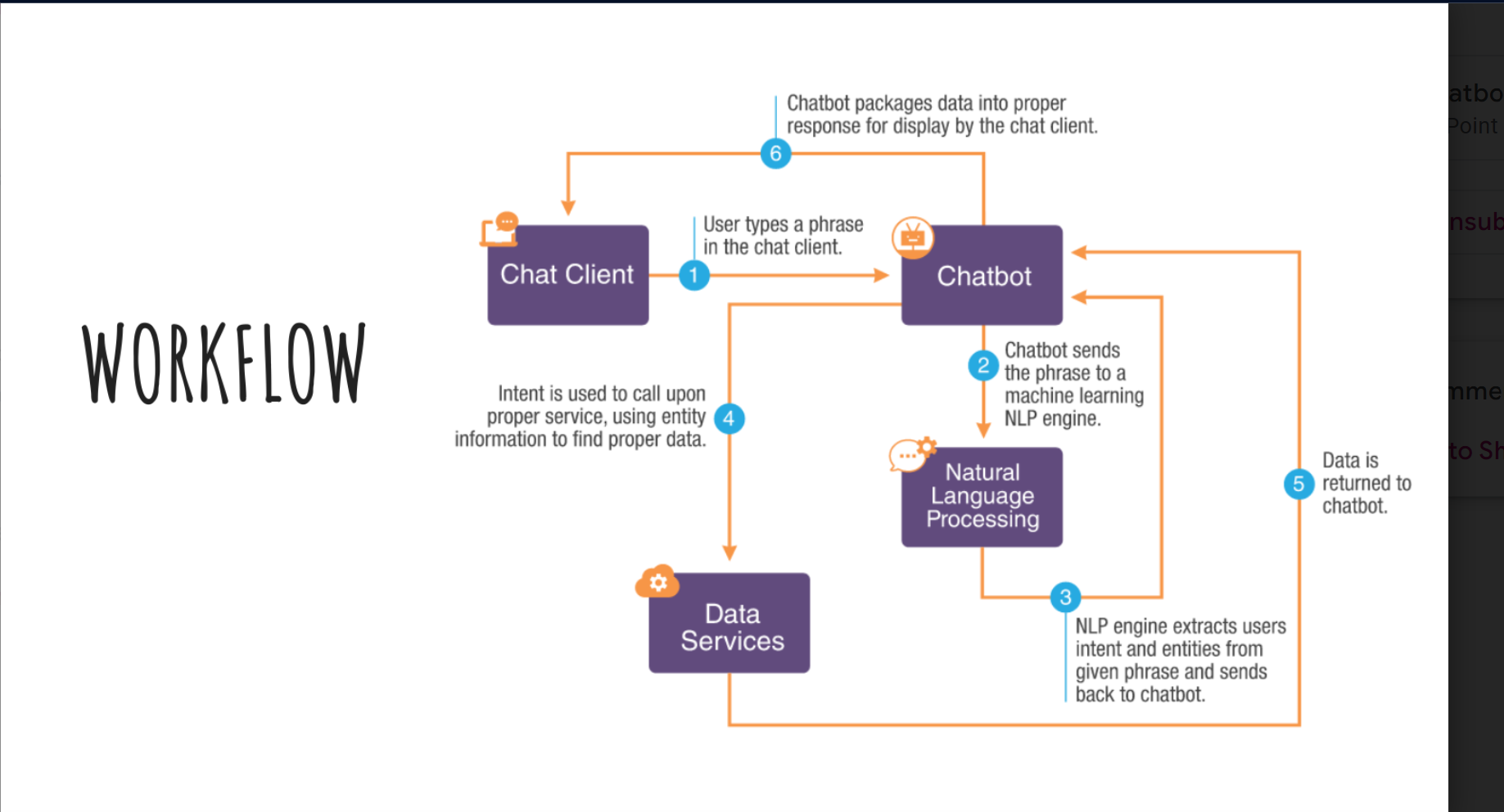
Hectic communication services for information transfer

Solution:

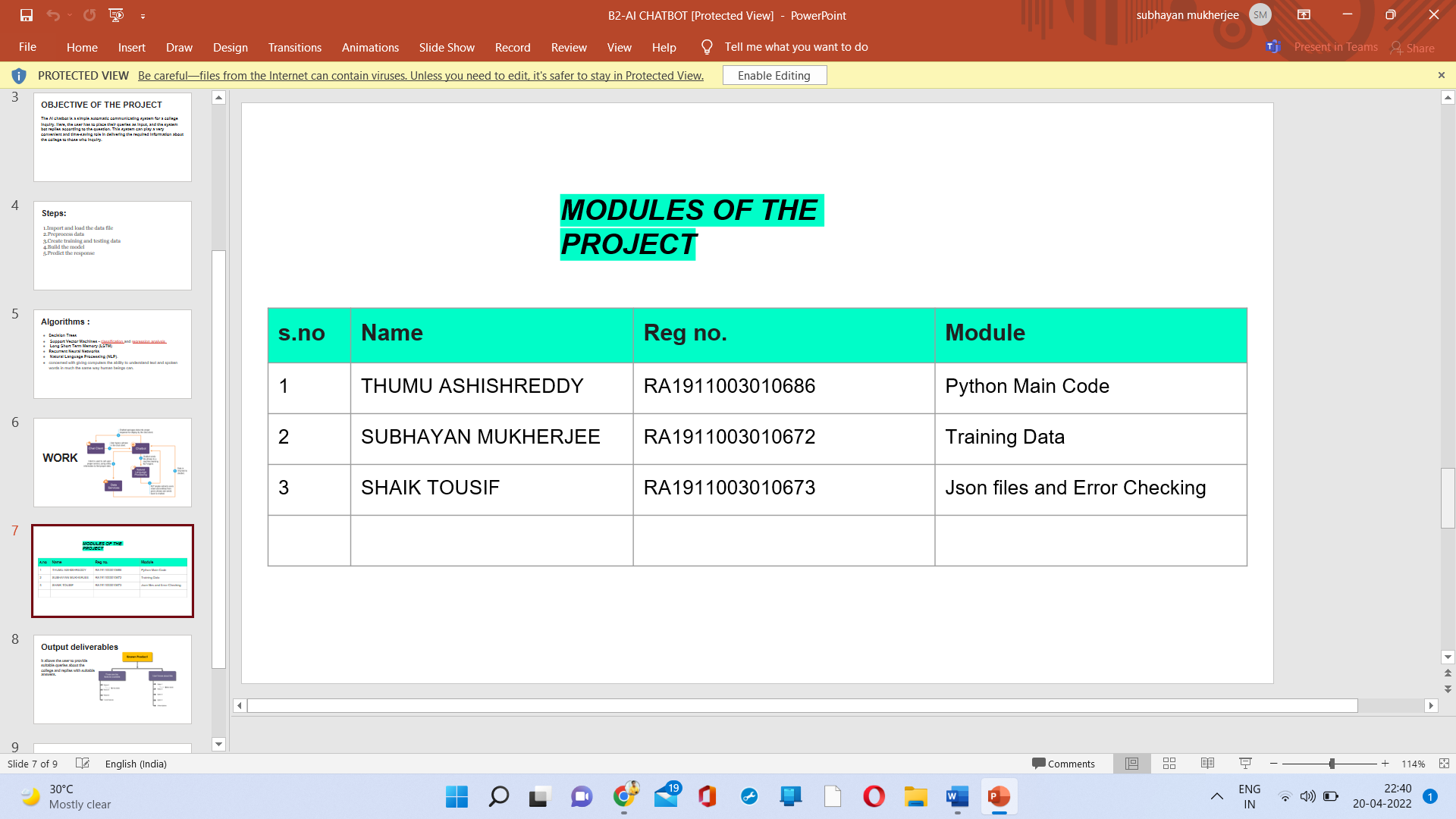
A chatbot eases data transfer without any hassles for standard information

Proposed Method with Flow Diagram:

1. User types their query in the chat.
2. Chatbot sends the query to a machine learning NLP engine.
3. NLP engine extracts users’ intent and entities from given query and previous queries, data.
4. Intent is used to call upon proper service, using entity information to find proper data.
5. Data is returned to chatbot.
6. Chatbot packages data into proper response for display by the chat client.



Modules Description:



PYTHON MAIN CODE:

1. Start by importing the required libraries:

Random

Json

Pickle

Numpy

Nltk

Tensor flow

1. Assign a variable lemmatizer as WordNetLemmatizer()
2. Open intents json file and read the content into intents.
3. Create a function clean\_up\_sentence to filter words from sentence.
4. To find keywords, create a function bag\_of\_words.
5. Create a function to predict the tag of the given statement, predict\_class.
6. Create a variable ERROR\_THRESHOLD, to predict a word if its occurrence is greater than our required threshold.

import random

import json

import pickle

import numpy as np

import nltk

from nltk.stem import WordNetLemmatizer

from tensorflow.keras.models import load\_model

lemmatizer = WordNetLemmatizer()

intents = json.loads(open('intents.json').read())

words = pickle.load(open('words.pkl', 'rb'))

classes = pickle.load(open('classes.pkl', 'rb'))

model = load\_model('chatbotmodel.h5')

def clean\_up\_sentence(sentence):

    sentence\_words = nltk.word\_tokenize(sentence)

    sentence\_words = [lemmatizer.lemmatize(word)  for word in sentence\_words]

    return sentence\_words

def bag\_of\_words(sentence):

    sentence\_words= clean\_up\_sentence(sentence)

    bag = [0] \* len(words)

    for w in sentence\_words:

        for i, word in enumerate(words):

            if word == w:

                bag[i] = 1

    return np.array(bag)

def predict\_class(sentence):

    bow = bag\_of\_words(sentence)

    res = model.predict(np.array([bow]))[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 get\_response(intents\_list,intents\_json):

    tag= intents\_list[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

print("|============= Welcome to College Equiry Chatbot System! =============|")

print("|============================== Feel Free ============================|")

print("|================================== To ===============================|")

print("|=============== Ask your any query about our college ================|")

while True:

    message = input("| You: ")

    if message == "bye" or message == "Goodbye":

        ints = predict\_class(message)

        res = get\_response(ints, intents)

        print("| Bot:", res)

        print("|===================== The Program End here! =====================|")

        exit()

    else:

        ints = predict\_class(message)

        res = get\_response(ints, intents)

        print("| Bot:", res)

Training Data:

import random

import json

import pickle

import numpy as np

import nltk

from nltk.stem import WordNetLemmatizer

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout

from tensorflow.keras.optimizers import SGD

lemmatizer = WordNetLemmatizer()

intents = json.loads(open('intents.json').read())

words = []

classes = []

documents = []

ignore\_letters = ['?', '!',',','.']

for intent in intents['intents']:

    for pattern in intent['patterns']:

        word\_list = nltk.word\_tokenize(pattern)

        words.extend(word\_list)

        documents.append((word\_list,intent['tag']))

        if intent['tag'] not in classes:

            classes.append(intent['tag'])

words = [lemmatizer.lemmatize(word) for word in words if word not in ignore\_letters]

words = sorted(set(words))

classes = sorted(set(classes))

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

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

training = []

output\_empty = [0] \* len(classes)

for document in documents:

    bag =[]

    word\_patterns = document[0]

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

    for word in words:

        bag.append(1) if word in word\_patterns else bag.append(0)

    output\_row = list(output\_empty)

    output\_row[classes.index(document[1])] = 1

    training.append([bag, output\_row])

random.shuffle(training)

training = np.array(training)

train\_x = list(training[:, 0])

train\_y = list(training[:, 1])

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(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('chatbotmodel.h5', hist)

print('Done')

JSON Files and Error checking:

Conversational AI often does not go the way we want it to. It leads to disappointing experiences and ultimately a rejection of the experience.

Luckily, there are ways to design conversational experiences in a way that makes chatbots and voice assistants more helpful, natural and persuasive. An important element of designing better experiences and unlocking the potential of conversational AI, is through error handling.

Conversations are messy. We misunderstand each other or our thoughts wander off and we did not listen to the question that we were asked. Humans have developed amazing techniques to deal with such situations. We ask for clarification, or we share more details to get the other person to engage with us.

If we translate this to conversation design, we also can say that we have techniques for when there is a no-input and when there is a no-match.

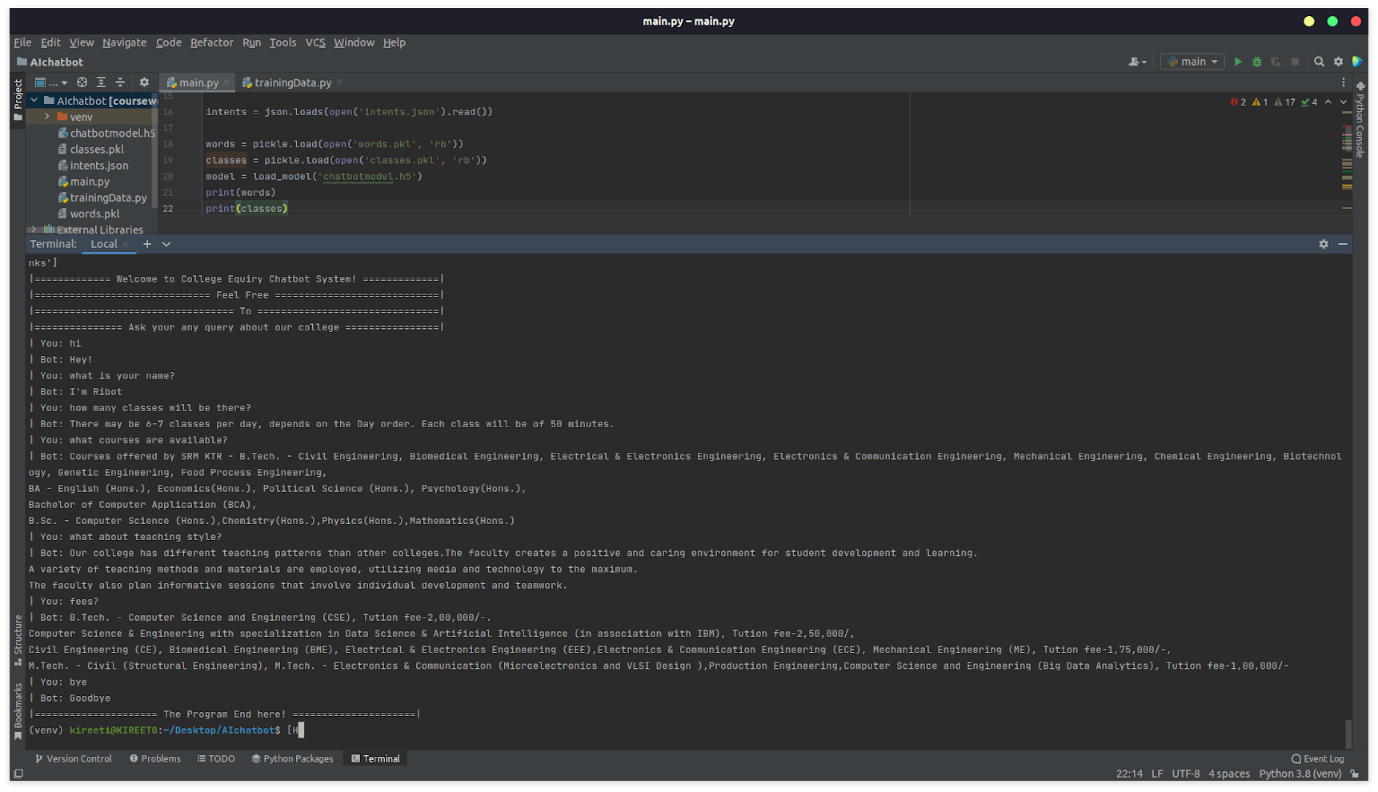
Implementation requirements:

* pip
* NumPy
* random
* nltk
* tensorflow
* windows 7

## System requirements﻿

| **Requirement** | **Minimum** | **Recommended** |
| --- | --- | --- |
| RAM | 4 GB of free RAM | 8 GB of total system RAM |
| CPU | Any modern CPU | Multi-core CPU. PyCharm supports multithreading for different operations and processes making it faster the more CPU cores it can use. |
| Disk space | 2.5 GB and another 1 GB for caches | SSD drive with at least 5 GB of free space |
| Monitor resolution | 1024×768 | 1920×1080 |
| Operating system | Officially released 64-bit versions of the following:   * Microsoft Windows 8 or later * macOS 10.14 or later * Any Linux distribution that supports Gnome, KDE , or Unity DE. PyCharm is not available for some Linux distributions, such as RHEL6 or CentOS6, that do not include [GLIBC](https://ftp.gnu.org/gnu/libc/) 2.14 or later.   Pre-release versions are not supported. | Latest 64-bit version of Windows, macOS, or Linux (for example, Debian, Ubuntu, or RHEL) |

Output Screenshots:



Conclusion:

This chat bot is used for queries related to SRMIST college by using NLP.