170102263_AI_CHATBOT_for_College_Website.pdf

by

Submission date: 11-May-2020 01:50PM (UTC+0530)
Submission ID: 1321529688
File name: 170102263_AI_CHATBOT_for_College_Website.pdf (2.69M)
Word count: 2331
Character count: 12920

ARTIFICIAL INTELLIGENCE
CHATBOT FOR COLLEGE INFORMATION

A Report submitted

By

Aishwarya Mishra (170102263)
Abhinav Pathak (170102264)
Yashika Mittal (170102252)

Under the Guidance of
Dr. Sarvesh Vishwakarma
Professor of
CSE Department

In partial fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING, DIT UNIVERSITY, DEHRADUN

(State Private University through State Legislature Act No. 10 of 2013 of Uttarakhand and approved by UGC)

Mussoorie Diversion Road, Dehradun, Uttarakhand - 248009, India.

DECLARATION

This is to certify that the Project entitled **"ARTIFICIAL INTELLIGENCE CHATBOT FOR COLLEGE INFORMATION"** in partial fulfillment of the requirement for the award of the **Degree of B.TECH** in Computer Science, submitted to **DIT University, Dehradun, Uttarakhand, India,** is an authentic record of bonafide work carried out by Aishwarya Mishra, Abhinav Pathak and Yashika Mittal under the guidance of Sarvesh Vishwakarma.

The matter embodied in this Project/Thesis/Dissertation has not been submitted for the award of any other degree or diploma to any University/Institution. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Students Name & Signature:

This is to certify that the above statement made by the candidate is correct to the best of my /our knowledge.

Date: Signature(s) of the Supervisor (s)

ACKNOWLEDGEMENT

First and foremost, we express our gratitude to the Almighty and the Management of DIT University for providing all the facilities needed for this project. We express our gratitude to the Head of Department, Prof. Vishal Bharti, for his immense support and encouragement. We express our gratitude to our project coordinator, Dr Radha Guha, Senior Professor, Department of Information Technology for her valuable suggestions and guidance. I would like to extend my gratefulness to my panel member co-ordinator Ms. Nishtha Rawat, Assistant Professor, Department of Computer Science and Engineering for her continuous support and overwhelming responses.

Last but not the least, I wish to thank all the staff of Computer Science and Engineering Department for their help and support.

ABSTRACT

Chatbots, or conversational interfaces as they are also known, present a new way for individuals to interact with computer systems. Traditionally, to get a question answered by a software program involved using a search engine, or filling out a form. A chatbot allows a user to simply ask questions in the same manner that they would address a human. The most well known chatbots currently are voice chatbots: Alexa and Siri. However, chatbots are currently being adopted at a high rate on computer chat platforms.

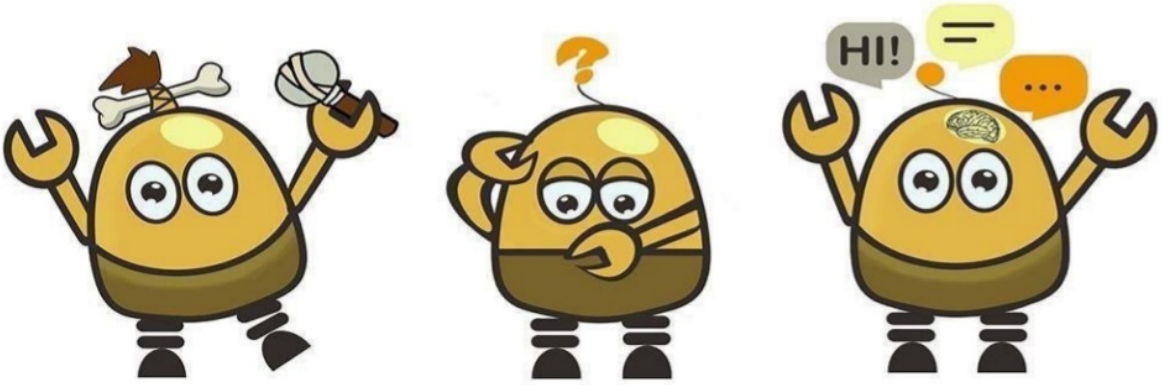
The technology at the core of the rise of the chatbot is natural language processing ("NLP"). Recent advances in machine learning have greatly improved the accuracy and effectiveness of natural language processing, making chatbots a viable option for many organizations. This improvement in NLP is firing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

TABLE OF CONTENTS

1. INTRODUCTION

1.1. PURPOSE

1.1.1. Purpose of a Chatbot?



Demo Image of Chat bot

The *purpose* of chat bots is to support and scale business teams in their relations with customers. It could live in any major chat applications like Facebook Messenger, Slack, Telegram, Text Messages, etc..

1.1.2.Modules of Project

The project will be concluding 2 main modules:

- Using framework – Dialog Flow.
- Using NLP.

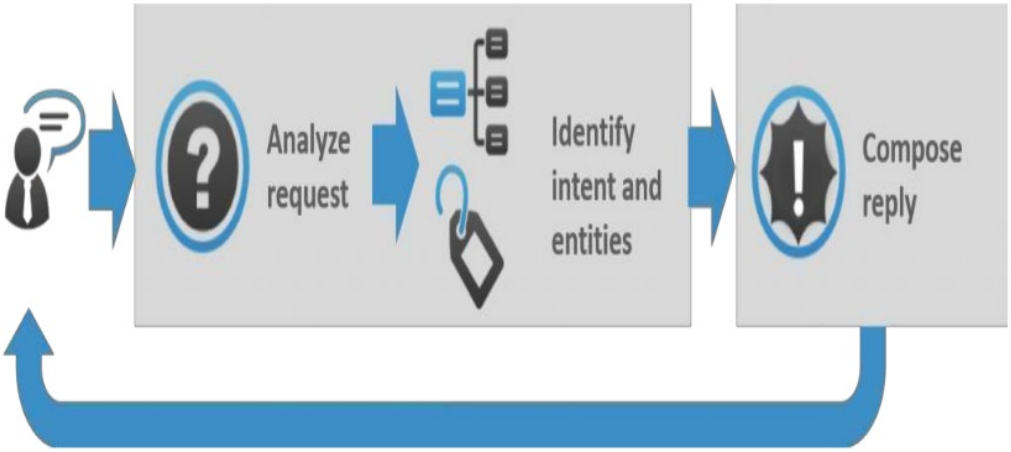
1.1.3.Chatbot will comprise of:

- Bot Chat: User can chat with the bot it implies as if enquiring to the college person about college related activities.
- Text to speech: Bot will receive the commands that are spoken.

1.2. Working of Chatbot

There are two different tasks at the core of a Chabot:

- User request analysis
- Returning the response



Work Flow of Chatbot

- **User request analysis**: This is the first task that a Chabot performs. It analyses the user's request to identify the user intent and to extract relevant entities.

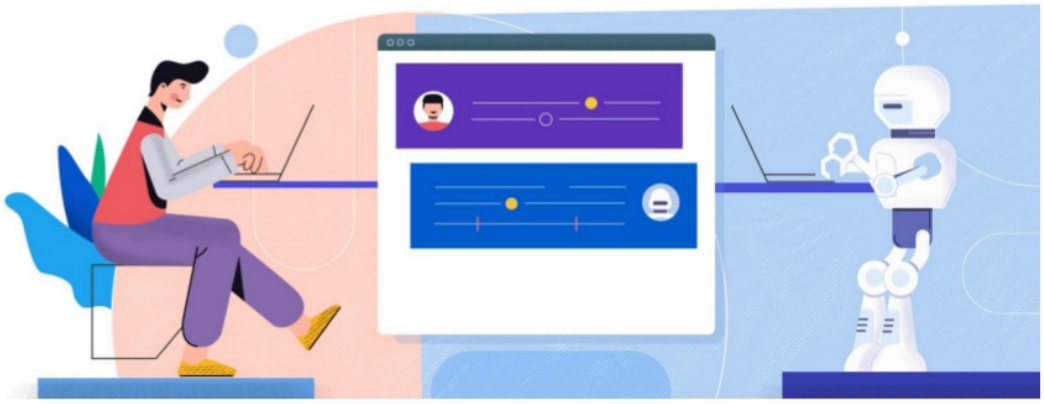
- **Returning the response**: Once the user's intent has been identified, the Chabot must provide the most appropriate response for the user's request. The answer may be:

1. A generic and predefined text
2. A text retrieved from a knowledge base that contains different answers 3. A contextualized piece of information based on data the user has provided
4. Data stored in enterprise systems.
5. A disambiguating question that helps the Chabot to correctly understand the user's request.

1.3. Definition and overview

A Chabot is artificial intelligence (AI) software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, and mobile apps or through the telephone. This could be text based, spoken, non-verbal conversation. This Chabot will give answer to query about college related issues e.g.: (admission, subjects, streams, fee structure etc.).

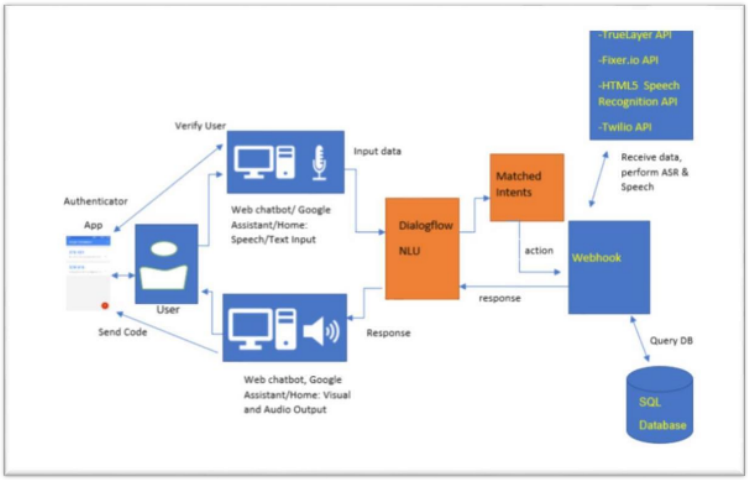
Importance: A Chabot is often described as one of the most advanced and promising expressions of interaction between humans and machines.



2. DESCRIPTIONS

2.1. Architectural Design

Figure provides a clear and concise abstract architectural design of the proposed chatbot.



Users will interact with the chatbot from the web client and Google assistant app for Android devices. Users will carry out their interaction with the chatbot in the form natural language voice or text-based phrases. With the Google assistant integration, users will receive rich responses, such as; images, and cards thus improving ease of use and interaction experience for users as less typing and effort will be required when interacting with the chatbot mainly through the use of text commands.

Project Modules:

Project is completed in two modules :

1) Implementation of chatbot through **Dialogflow**.
2) Implementation of chatbot through **NLP** (RNN – *Sequence to Sequence Model*)

2.2. Progress in work

2.2.1. Building of chatbot through dialogflow –

Dialog Flow: In this method we are going to follow the following steps which will lead in initiating the process. The steps are:

- Intents will be made as:

  1. Contexts

  2. Training phases

  3. Action and parameters

  4. Responses

- Entities

  It will contain all different types about what applicant want to enquire.

- Training

  It will make the bot learn about the statements which are asked frequently many times and answer default.

- History

  It will tell about the questions asked in past and if want to change the response or want to train can be done here.

- Fulfilment

  The inline editor will contain the code of handling request by user and action taken by agent.

- Integrations

  It will build the assistant to reach the users through the required home page.

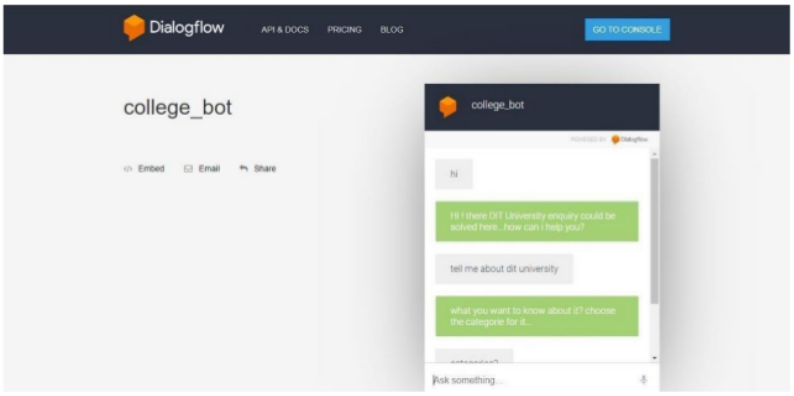
1

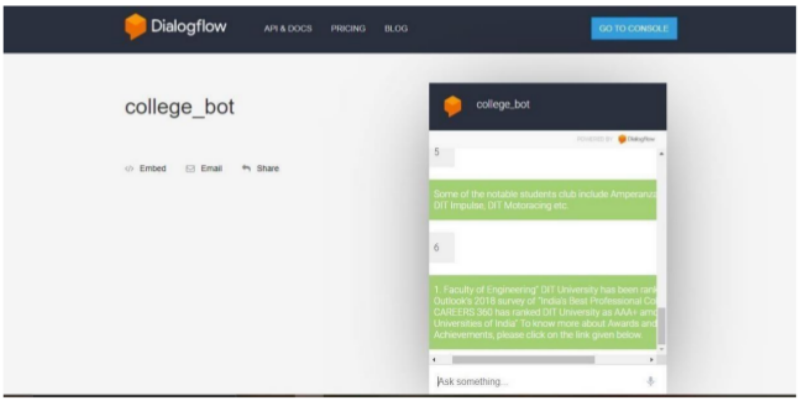
As main focus of interaction is between the user and the chatbot, which is carried out through natural language, the conversation design is a crucial aspect to consider when designing the chatbot. To achieve this; appropriate dialog will be designed through the Dialog flow console using follow-up and fall-back intents. A particular intent will be set, that

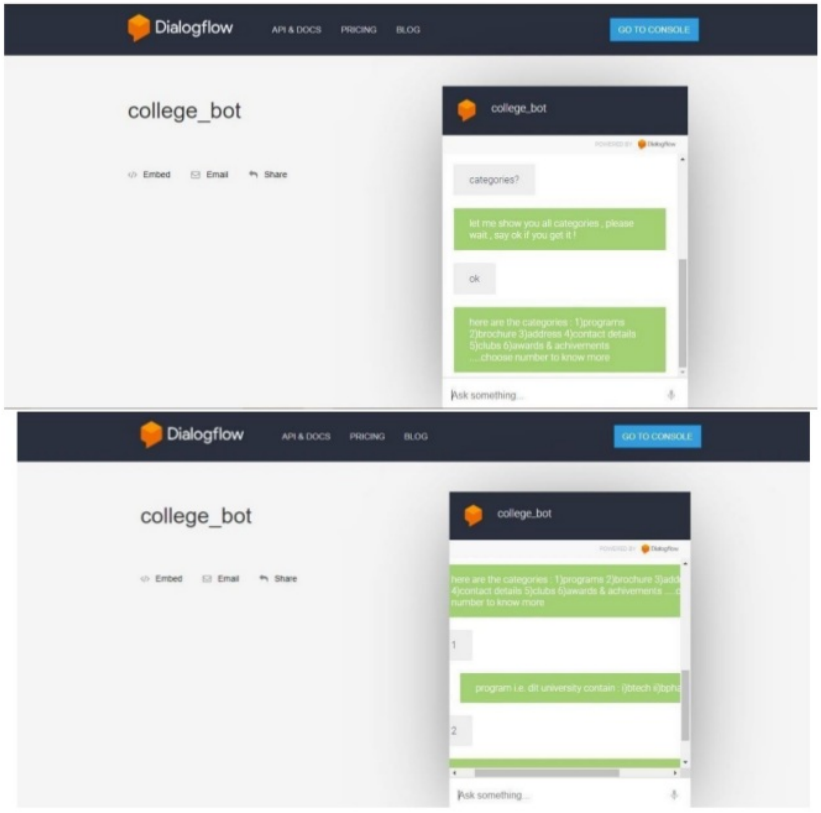
once invoked by the user, the chatbot will end the conversation effectively regardless of the conversational context. The dialog is the collection of words and phrases used to respond to user input. The dialog will be designed with human like phrases to respond to users with in order to sound more natural. Another design aspect will include fall-back intents. If a user utterance cannot be matched to an intent, the chatbot will respond with a message stating it did not match the input to an intent and prompt the user to repeat the query or provide more details. The dialog is also designed using follow-up intents, these provide a more natural conversational flow when interacting with the chatbot.

Dialog flow recommend designing a linear dialog for interactions where data present in the conversation is extracted to help the users achieve their goal, this is applied to the design specification is applied to the chatbot as it is an tasked-based interaction chatbot that consist of banking domain knowledge.

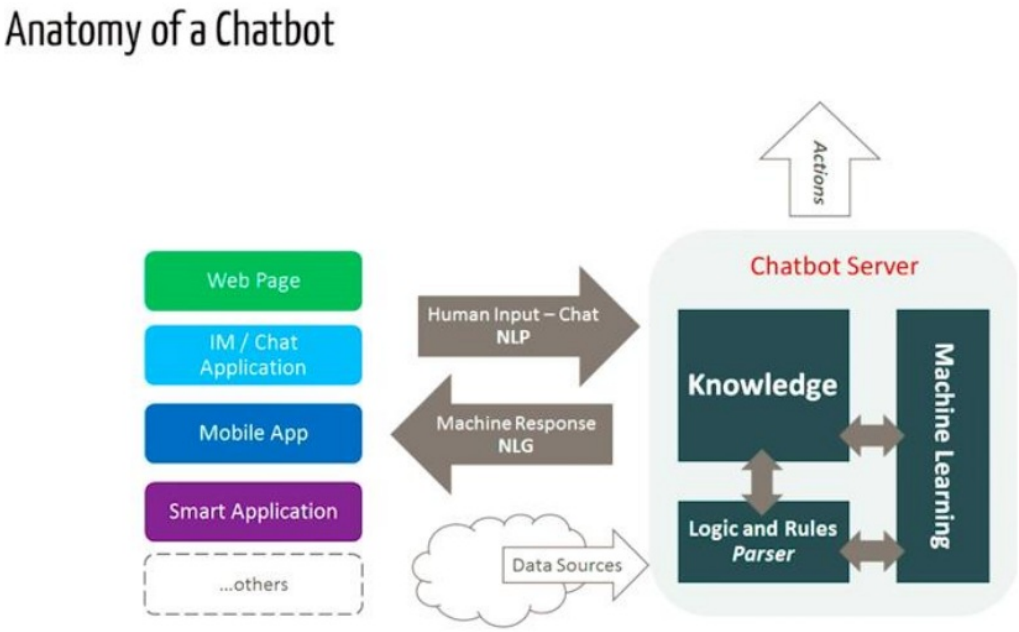
In this example the chatbot will extract the following information from the university:







CHECK THE WORKING HERE:
https://bot.dialogflow.com/0e2b5bf5-3628-47f8-bfb4-7847ce4a35f4



2.2.2. BUILDING OF CHATBOT WITH NLP –

Chat-bot have three phases to complete and get the output required. They are as follows:

1) Data Pre-processing
2) Sequence to Sequence Model
3) Data Training

DATA PRE-PROCESSING –

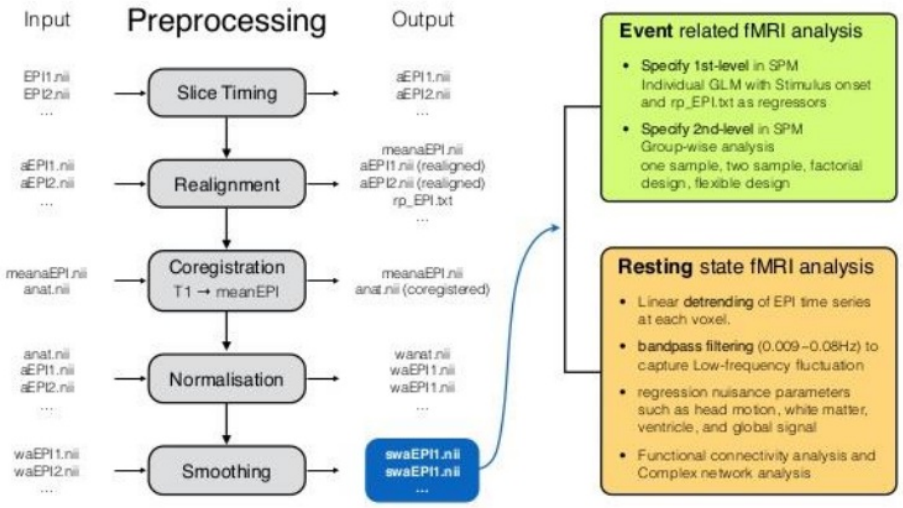
What is Data Pre-processing?

Data pre-processing is a data mining technique that involves transforming raw data into an understandable format. Real world data is often incomplete, inconsistent, or lacking in certain behaviours or trends, and is likely to contain many errors. Data pre-processing is proven method of resolving such issues.

Why we use Data Pre-processing?

In real world data are generated incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data. Noise: containing errors or outliers. Inconsistent: containing discrepancies in codes or names.

Summary of Preprocess

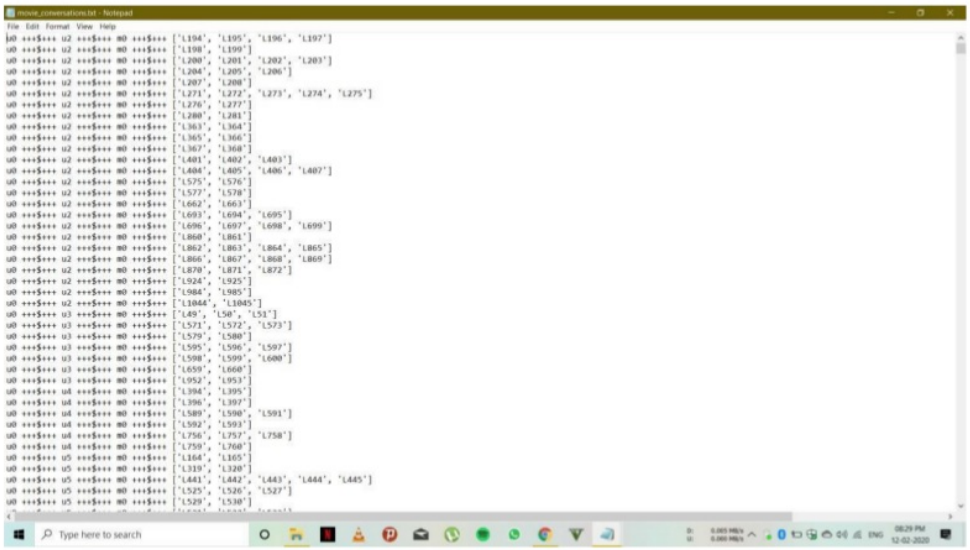


DATASET COLLECTED:

An effective chat bot requires a massive amount of training data in order to quickly solve user inquiries without human intervention. However, the primary bottleneck in chat bot development is obtaining realistic, task-oriented dialog data to train these machine learning-based systems.

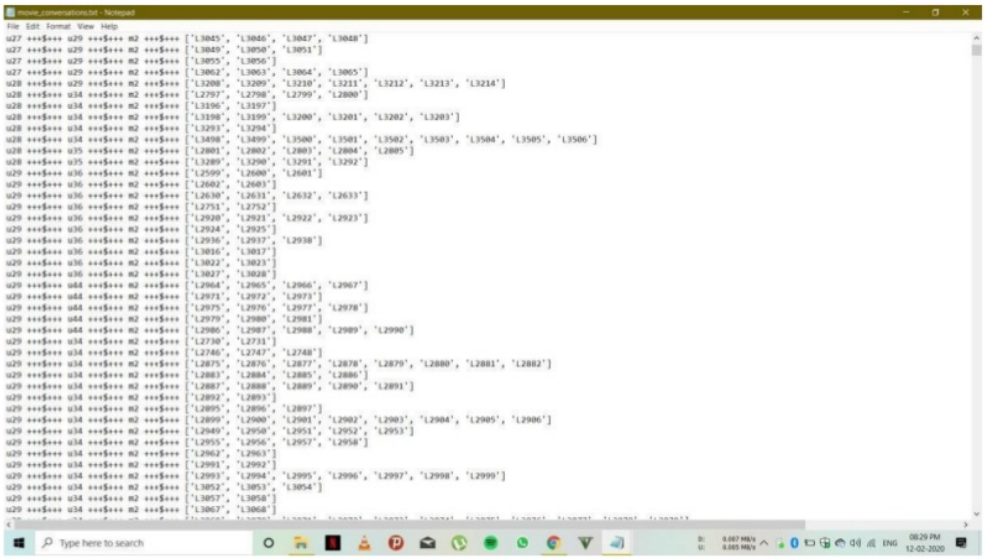
Steps in Data Preprocessing

1) Import the libraries 2) Import the data set
3) Check out the missing values
4) See the Categorical Values
5) Splitting the Dataset into training and test set
6) Feature scaling



COLLECTING DATASET:

- For effective and accurate Chatbot data is massive.
- Some screenshots are included for sample.



10

SEQUENCE TO SEQUENCE MODEL -

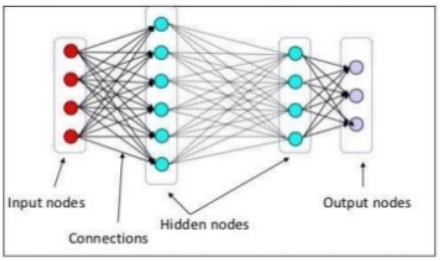
Sequence-to-sequence learning (Seq2Seq) is about training models to convert sequences from one domain (e.g. sentences in English) to sequences in another domain (e.g. the same sentences translated to French).

This can be used for machine translation or for free-from question answering (generating a natural language answer given a natural language question) -- in general, it is applicable any time you need to generate text.

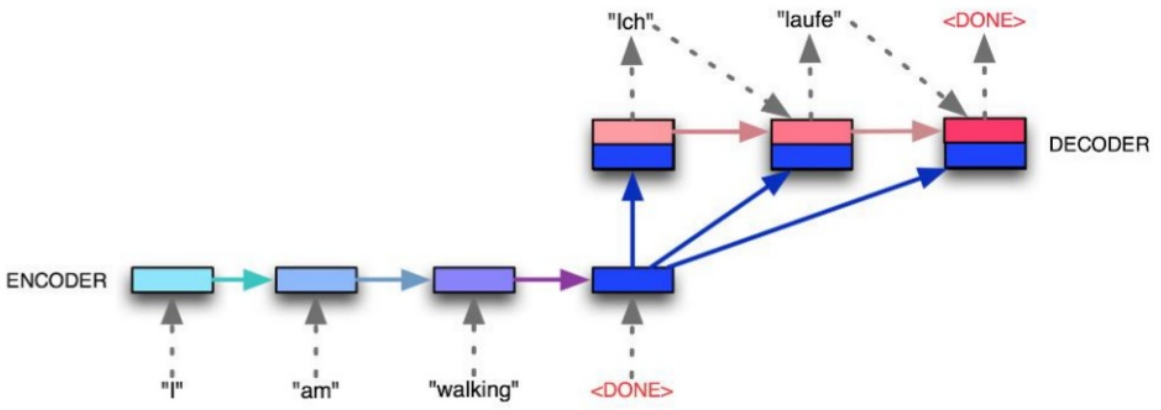
6

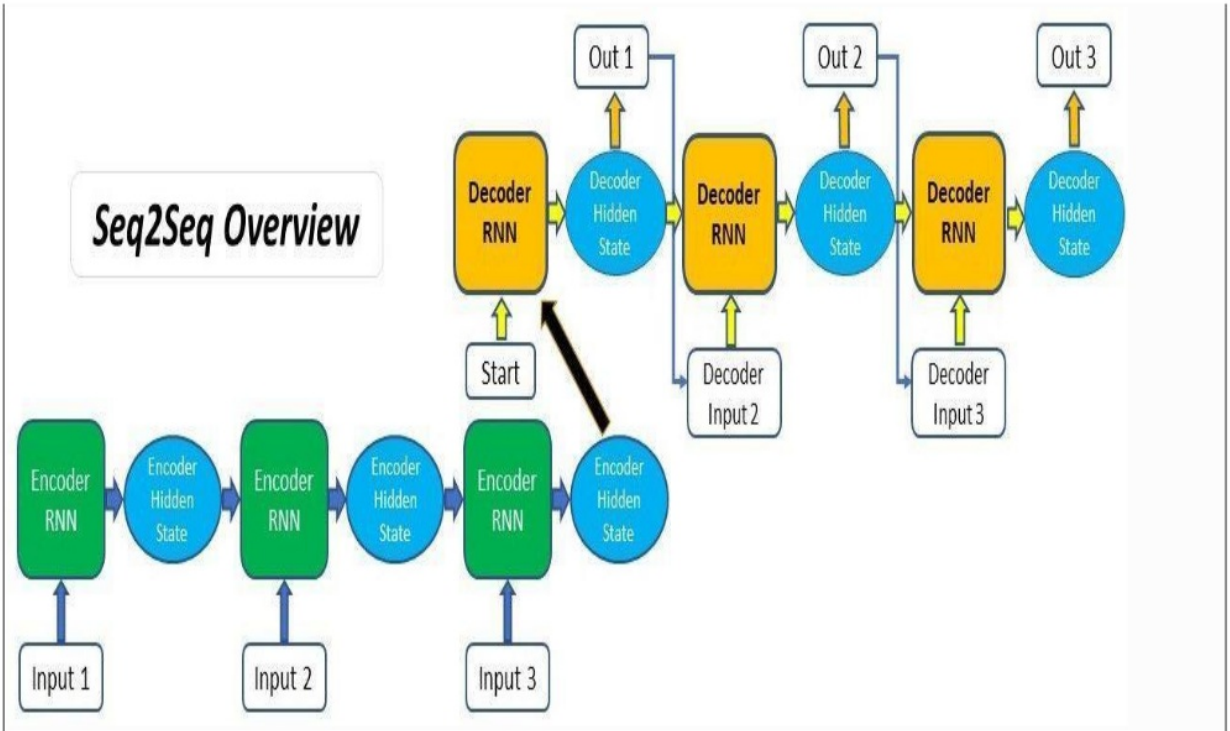
Let's illustrate these ideas with actual code.

For our example implementation, we will use a dataset of pairs of English sentences and their French translation. We will implement a *character-level* sequence-to-sequence model, processing the input character-by-character and generating the



output character-by-character. Another option would be a word-level model, which tends to be more common for machine translation. At the end of this post, you will find some notes about turning our model into a word-level model using embedding layers.




Seq2Seq Overview
Out 1
Out 2
Out 3
Decoder RNN
Decoder Hidden State
Decoder RNN
Decoder Hidden State
Decoder RNN
Decoder Hidden State
Start
Decoder Input 2
Decoder Input 3
Encoder RNN
Encoder Hidden State
Encoder RNN
Encoder Hidden State
Encoder RNN
Encoder Hidden State
Input 1
Input 2
Input 3

DATA TRAINING –

- The training data set in Machine Learning is the actual dataset used to train the model for performing various actions.
- This is the actual data the ongoing development process models learn with various API and algorithm to train the machine to work automatically.
- There are three types of data sets – Training, Dev and Test that are used at various stage of development. *Training dataset is the largest of three of them.*

2.3. Feasilibilty and Methodology

CODE FOR CHATBOT:

```
# Importing the libraries
import numpy as np
import tensorflow as tf
import re
import time



########## DATA PREPROCESSING ##########



# Importing the dataset
lines = open('movie_lines.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')
conversations = open('movie_conversations.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')

# Creating a dictionary that maps each line and its id
id2line = {}
for line in lines:
    _line = line.split(' +++$+++ ')
    if len(_line) == 5:
        id2line[_line[0]] = _line[4]

# Creating a list of all of the conversations
conversations_ids = []
for conversation in conversations[:-1]:
    _conversation = conversation.split(' +++$+++ ')[-1][1:-1].replace("'", "").replace(" ", "")
    conversations_ids.append(_conversation.split(','))
```

```
# Getting separately the questions and the answers
questions = []
answers = []
for conversation in conversations_ids:
    for i in range(len(conversation) - 1):
        questions.append(id2line[conversation[i]])
        answers.append(id2line[conversation[i+1]])

# Doing a first cleaning of the texts
def clean_text(text):
    text = text.lower()
    text = re.sub(r"i'm", "i am", text)
    text = re.sub(r"he's", "he is", text)
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"that's", "that is", text)
    text = re.sub(r"what's", "what is", text)
    text = re.sub(r"where's", "where is", text)
    text = re.sub(r"how's", "how is", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"n't", " not", text)
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "cannot", text)
    text = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", text)
    return text

# Cleaning the questions
clean_questions = []
for question in questions:
    clean_questions.append(clean_text(question))
```

```
# Cleaning the answers
clean_answers = []
for answer in answers:
    clean_answers.append(clean_text(answer))

# Filtering out the questions and answers that are too short or too long
short_questions = []
short_answers = []
i = 0
for question in clean_questions:
    if 2 <= len(question.split()) <= 25:
        short_questions.append(question)
        short_answers.append(clean_answers[i])
    i += 1
clean_questions = []
clean_answers = []
i = 0
for answer in short_answers:
    if 2 <= len(answer.split()) <= 25:
        clean_answers.append(answer)
        clean_questions.append(short_questions[i])
    i += 1

# Creating a dictionary that maps each word to its number of occurrences
word2count = {}
for question in clean_questions:
    for word in question.split():
        if word not in word2count:
            word2count[word] = 1
        else:
            word2count[word] += 1
for answer in clean_answers:
    for word in answer.split():
```

```
        if word not in word2count:
            word2count[word] = 1
        else:
            word2count[word] += 1

# Creating two dictionaries that map the questions words and the answers
threshold_questions = 15
questionswords2int = {}
word_number = 0
for word, count in word2count.items():
    if count >= threshold_questions:
        questionswords2int[word] = word_number
        word_number += 1
threshold_answers = 15
answerswords2int = {}
word_number = 0
for word, count in word2count.items():
    if count >= threshold_answers:
        answerswords2int[word] = word_number
        word_number += 1

# Adding the last tokens to these two dictionaries
tokens = ['<PAD>', '<EOS>', '<OUT>', '<SOS>']
for token in tokens:
    questionswords2int[token] = len(questionswords2int) + 1
for token in tokens:
    answerswords2int[token] = len(answerswords2int) + 1

# Creating the inverse dictionary of the answerswords2int dictionary
answersints2word = {w_i: w for w, w_i in answerswords2int.items()}

# Adding the End Of String token to the end of every answer
for i in range(len(clean_answers)):
    clean_answers[i] += ' <EOS>'
```

```
# Translating all the questions and the answers into integers
# and Replacing all the words that were filtered out by <OUT>
questions_into_int = []
for question in clean_questions:
    ints = []
    for word in question.split():
        if word not in questionswords2int:
            ints.append(questionswords2int['<OUT>'])
        else:
            ints.append(questionswords2int[word])
    questions_into_int.append(ints)
answers_into_int = []
for answer in clean_answers:
    ints = []
    for word in answer.split():
        if word not in answerswords2int:
            ints.append(answerswords2int['<OUT>'])
        else:
            ints.append(answerswords2int[word])
    answers_into_int.append(ints)

# Sorting questions and answers by the length of questions
sorted_clean_questions = []
sorted_clean_answers = []
for length in range(1, 25 + 1):
    for i in enumerate(questions_into_int):
        if len(i[1]) == length:
            sorted_clean_questions.append(questions_into_int[i[0]])
            sorted_clean_answers.append(answers_into_int[i[0]])
```

```
# Creating placeholders for the inputs and the targets
def model_inputs():
    inputs = tf.placeholder(tf.int32, [None, None], name = 'input')
    targets = tf.placeholder(tf.int32, [None, None], name = 'target')
    lr = tf.placeholder(tf.float32, name = 'learning_rate')
    keep_prob = tf.placeholder(tf.float32, name = 'keep_prob')
    return inputs, targets, lr, keep_prob

# Preprocessing the targets
def preprocess_targets(targets, word2int, batch_size):
    left_side = tf.fill([batch_size, 1], word2int['<SOS>'])
    right_side = tf.strided_slice(targets, [0,0], [batch_size, -1], [1,1])
    preprocessed_targets = tf.concat([left_side, right_side], 1)
    return preprocessed_targets

# Creating the Encoder RNN
def encoder_rnn(rnn_inputs, rnn_size, num_layers, keep_prob, sequence_length):
    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
    encoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
    encoder_output, encoder_state = tf.nn.bidirectional_dynamic_rnn(cell_fw = encoder_cell,
                                                                    cell_bw = encoder_cell,
                                                                    sequence_length = sequence_length,
                                                                    inputs = rnn_inputs,
                                                                    dtype = tf.float32)
    return encoder_state

# Decoding the training set
def decode_training_set(encoder_state, decoder_cell, decoder_embedded_input,
 sequence_length, decoding_scope, output_function, keep_prob, batch_size):
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function =
    tf.contrib.seq2seq.prepare_attention(attention_states, attention_option = "bahdanau",
```

```
# Decoding the training set
def decode_training_set(encoder_state, decoder_cell, decoder_embedded_input,
 sequence_length, decoding_scope, output_function, keep_prob, batch_size):
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function =
    tf.contrib.seq2seq.prepare_attention(attention_states, attention_option = "bahdanau",
      num_units = decoder_cell.output_size)
    training_decoder_function = tf.contrib.seq2seq.attention_decoder_fn_train(encoder_state[0],
                                                                              attention_keys,
                                                                              attention_values,
                                                                              attention_score_function,
                                                                              attention_construct_function,
                                                                              name = "attn_dec_train")
    decoder_output, decoder_final_state, decoder_final_context_state = tf.contrib.seq2seq.dynamic_rnn_decoder
    (decoder_cell,
      training_decoder_function,
      decoder_embedded_input,
      sequence_length,
      scope = decoding_scope)
    decoder_output_dropout = tf.nn.dropout(decoder_output, keep_prob)
    return output_function(decoder_output_dropout)

# Decoding the test/validation set
def decode_test_set(encoder_state, decoder_cell, decoder_embeddings_matrix, sos_id, eos_id, maximum_length,
                    num_words, decoding_scope, output_function, keep_prob, batch_size):
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function =
    tf.contrib.seq2seq.prepare_attention(attention_states, attention_option = "bahdanau",
      num_units = decoder_cell.output_size)
    test_decoder_function = tf.contrib.seq2seq.attention_decoder_fn_inference(output_function,encoder_state[0],
                                                                              attention_keys,attention_values,
                                                                              attention_score_function,
                                                                              attention_construct_function,
```

```
                                                                    attention_construct_function,
                                                                    decoder_embeddings_matrix,
                                                                    sos_id, eos_id,maximum_length,
                                                                    num_words,name = "attn_dec_inf")
    test_predictions, decoder_final_state, decoder_final_context_state =
    tf.contrib.seq2seq.dynamic_rnn_decoder(decoder_cell,test_decoder_function,scope = decoding_scope)
    return test_predictions

# Creating the Decoder RNN
def decoder_rnn(decoder_embedded_input, decoder_embeddings_matrix, encoder_state, num_words,
                sequence_length, rnn_size, num_layers, word2int, keep_prob, batch_size):
    with tf.variable_scope("decoding") as decoding_scope:
        lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
        lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
        decoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
        weights = tf.truncated_normal_initializer(stddev = 0.1)
        biases = tf.zeros_initializer()
        output_function = lambda x: tf.contrib.layers.fully_connected(x,
                                                                      num_words,
                                                                      None,
                                                                      scope = decoding_scope,
                                                                      weights_initializer = weights,
                                                                      biases_initializer = biases)
        training_predictions = decode_training_set(encoder_state,
                                                   decoder_cell,
                                                   decoder_embedded_input,
                                                   sequence_length,
                                                   decoding_scope,
                                                   output_function,
                                                   keep_prob,
                                                   batch_size)
        decoding_scope.reuse_variables()
        test_predictions = decode_test_set(encoder_state,
                                           decoder_cell,
```

```
# Building the seq2seq model
def seq2seq_model(inputs, targets, keep_prob, batch_size, sequence_length, answers_num_words,
                  questions_num_words, encoder_embedding_size, decoder_embedding_size, rnn_size, num_layers, questionswords2int):
    encoder_embedded_input = tf.contrib.layers.embed_sequence(inputs,
                                                              answers_num_words + 1,
                                                              encoder_embedding_size,
                                                              initializer = tf.random_uniform_initializer(0, 1))
    encoder_state = encoder_rnn(encoder_embedded_input, rnn_size, num_layers, keep_prob, sequence_length)
    preprocessed_targets = preprocess_targets(targets, questionswords2int, batch_size)
    decoder_embeddings_matrix = tf.Variable(tf.random_uniform([questions_num_words + 1, decoder_embedding_size], 0, 1))
    decoder_embedded_input = tf.nn.embedding_lookup(decoder_embeddings_matrix, preprocessed_targets)
    training_predictions, test_predictions = decoder_rnn(decoder_embedded_input,
                                                         decoder_embeddings_matrix,
                                                         encoder_state,
                                                         questions_num_words,
                                                         sequence_length,
                                                         rnn_size,
                                                         num_layers,
                                                         questionswords2int,
                                                         keep_prob,
                                                         batch_size)
    return training_predictions, test_predictions
```

```
# Training
batch_index_check_training_loss = 100
batch_index_check_validation_loss = ((len(training_questions)) // batch_size // 2) - 1
total_training_loss_error = 0
list_validation_loss_error = []
early_stopping_check = 0
early_stopping_stop = 100
checkpoint = "./chatbot_weights.ckpt"
session.run(tf.global_variables_initializer())
for epoch in range(1, epochs + 1):
    for batch_index, (padded_questions_in_batch, padded_answers_in_batch)
     in enumerate(split_into_batches(training_questions, training_answers, batch_size)):
        starting_time = time.time()
        _, batch_training_loss_error = session.run([optimizer_gradient_clipping, loss_error],
          {inputs: padded_questions_in_batch, targets: padded_answers_in_batch,lr: learning_rate,
          sequence_length: padded_answers_in_batch.shape[1], keep_prob: keep_probability})
        total_training_loss_error += batch_training_loss_error
        ending_time = time.time()
        batch_time = ending_time - starting_time
        if batch_index % batch_index_check_training_loss == 0:
            print('Epoch: {:>3}/{}, Batch: {:>4}/{}, Training Loss Error: {:>6.3f},Training Time on 100 Batches: {:d}
              format(epoch,epochs,batch_index,len(training_questions) // batch_size,
                total_training_loss_error / batch_index_check_training_loss,
                int(batch_time * batch_index_check_training_loss)))
            total_training_loss_error = 0
        if batch_index % batch_index_check_validation_loss == 0 and batch_index > 0:
            total_validation_loss_error = 0
            starting_time = time.time()
            for batch_index_validation, (padded_questions_in_batch, padded_answers_in_batch)
            in enumerate(split_into_batches(validation_questions, validation_answers, batch_size)):
                batch_validation_loss_error = session.run(loss_error, {inputs: padded_questions_in_batch,
                                                                       targets: padded_answers_in_batch,
                                                                       lr: learning_rate,
```

```
            for batch_index_validation, (padded_questions_in_batch, padded_answers_in_batch)
            in enumerate(split_into_batches(validation_questions, validation_answers, batch_size)):
                batch_validation_loss_error = session.run(loss_error, {inputs: padded_questions_in_batch,
                                                                       targets: padded_answers_in_batch,
                                                                       lr: learning_rate,
                                                                       sequence_length: padded_answers_in_batch.shape[1],
                                                                       keep_prob: 1})
                total_validation_loss_error += batch_validation_loss_error
            ending_time = time.time()
            batch_time = ending_time - starting_time
            average_validation_loss_error = total_validation_loss_error / (len(validation_questions) / batch_size)
            print('Validation Loss Error: {:>6.3f}, Batch Validation Time: {:d} seconds'.
              format(average_validation_loss_error, int(batch_time)))
            learning_rate *= learning_rate_decay
            if learning_rate < min_learning_rate:
                learning_rate = min_learning_rate
            list_validation_loss_error.append(average_validation_loss_error)
            if average_validation_loss_error <= min(list_validation_loss_error):
                print('I speak better now!!')
                early_stopping_check = 0
                saver = tf.train.Saver()
                saver.save(session, checkpoint)
            else:
                print("Sorry I do not speak better, I need to practice more.")
                early_stopping_check += 1
                if early_stopping_check == early_stopping_stop:
                    break
    if early_stopping_check == early_stopping_stop:
        print("My apologies, I cannot speak better anymore. This is the best I can do.")
        break
print("Game Over")
```

INNOVATION AND USEFULNESS:

- **Improve customer service**

It is the best option for those who don't want their customers to:

- Wait for operator's answer — "Stay on the line, your call is very important to us" is always annoying, isn't it?
- Search for an answer in the FAQ — as a rule users don't have time for scrolling dozens of pages with instructions.

- **Personalize communication**

A Chabot answers the specific questions of visitors instead of displaying a long list of information. The more attention a customer gets the greater his desire to buy something.

- **Improve a response rate**

About 90% of questions sent from Facebook business pages remain unanswered. Chatbot responds to 100% of messages and converts more visitors into buyers.

- **Automate repetitive tasks.**

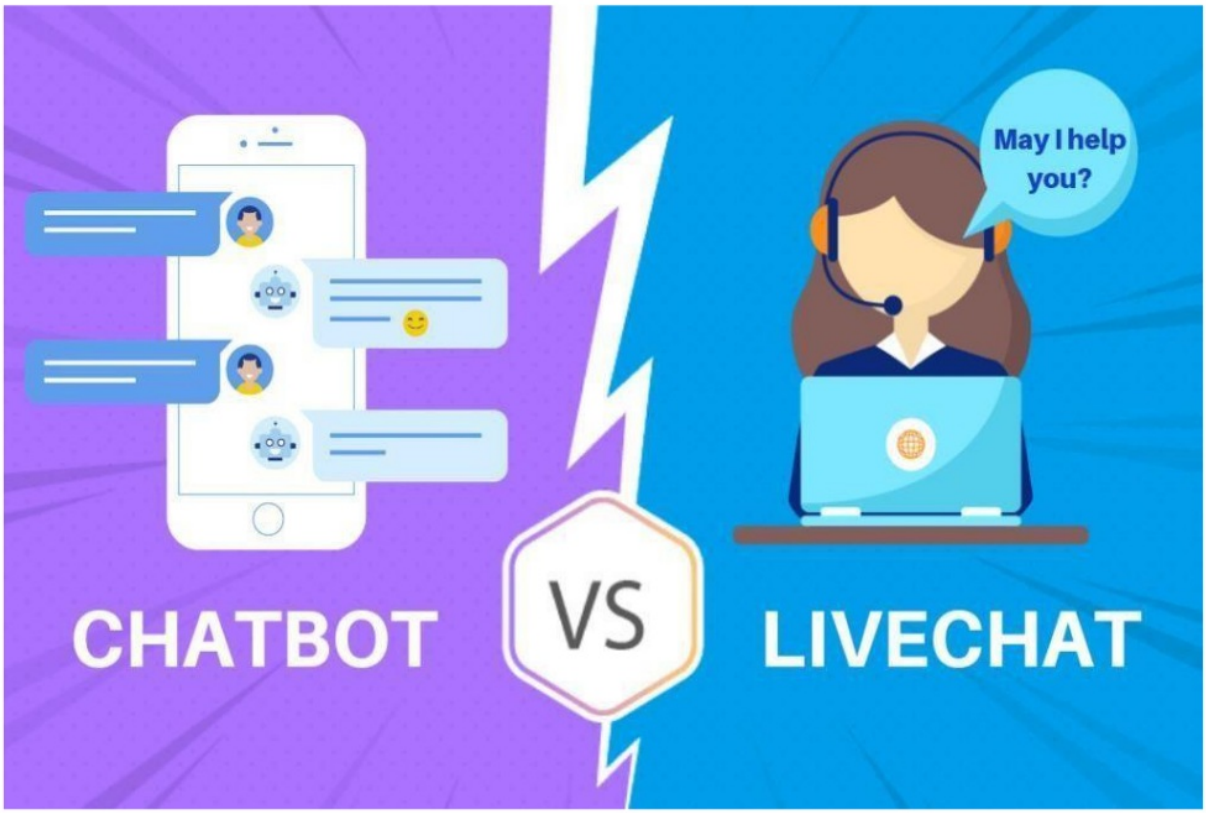
Most customers want to get answers on the same questions — when do you work? What is your location? Do you make deliveries? In order not to write the same answers every time, make a Chabot. It reduces your employees' workload.

3.1. FEASIBILITY STUDY

The Chabot industry is still in its early days, but growing very fast. Marketing motivations cannot be denied, but if Chabot meet the high expectations of the users, they will become indispensable tools for many use cases. The importance that tech giants like Google, Facebook, Microsoft, IBM, and Amazon are giving to Chabot is a strong indicator that this technology will play a key role in the future.

Clearly, Chabot is a rising trend and at Try labs we are witnessing a fast growing demand for them. If done right, this channel of communication with your users can increase engagement, give a better experience and also save costs. However, getting them right is not trivial.

Chatbot will improve customer service and will also help in improving the response rate. Catboats provide the assistance or access to information quickly and efficiently. Chatting with bots also helps to avoid loneliness, gives a chance to talk without being judged and improves conversational skills.



3.2. FACILITIES REQUIRED FOR PROPOSED WORK

Software Requirements:

Framework: Dialog Flow

NLP: Python IDLE / Google Colab / Visual Code

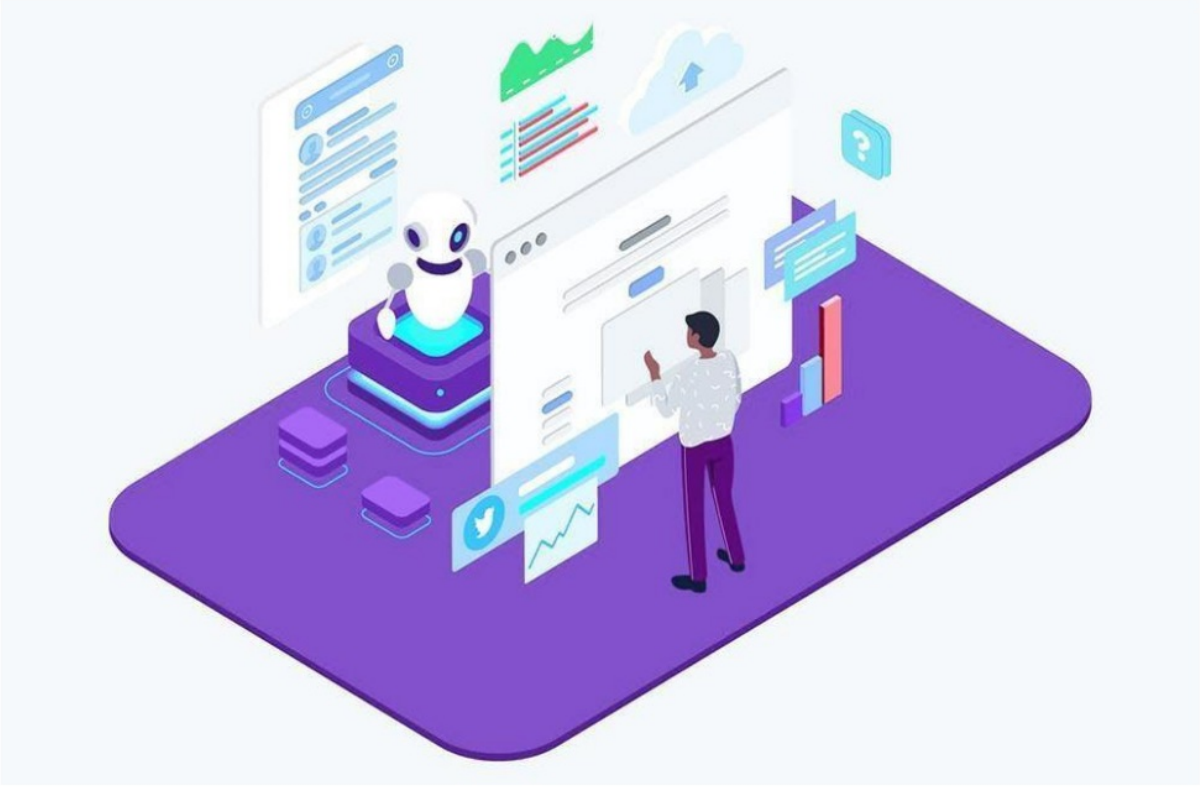
Hardware Requirements:

Processor: i3 Processor and above

RAM: 512 MB (minimum) and above

Hard disk: 64 GB (minimum) and above

Keyboard: 98 keys and above



3.3.REFERENCE AND BIBLIOGRAPHY

- https://console.dialogflow.com/api-client/#/agent/8929b00d-30b7-468e- 94f0c8ff8f76426e/fulfillment
- https://www.youtube.com
- https://www.youtube.com/results?search_query=nlp+for+chatbot

https://www.youtube.com/results?search_query=dialogflow+chatbot+tutorial

170102263_AI_CHATBOT_for_College_Website.pdf

ORIGINALITY REPORT

68%	66%	20%	51%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Ulster Student Paper	7%
2	www.coursehero.com Internet Source	7%
3	repozitorij.efzg.unizg.hr Internet Source	5%
4	tryolabs.com Internet Source	4%
5	Submitted to Symbiosis International University Student Paper	4%
6	blog.keras.io Internet Source	4%
7	Submitted to The Robert Gordon University Student Paper	4%
8	Dana Doherty, Kevin Curran. "Chatbots for online banking services", Web Intelligence, 2019 Publication	4%
9	hackernoon.com	

	Internet Source	3%
10	upcommons.upc.edu Internet Source	3%
11	www.scribd.com Internet Source	2%
12	roguevi.chaobear.com Internet Source	2%
13	Submitted to Universiti Malaysia Perlis Student Paper	2%
14	Submitted to ABV-Indian Institute of Information Technology and Management Gwalior Student Paper	2%
15	lionbridge.ai Internet Source	2%
16	www.irjet.net Internet Source	1%
17	www.portal.chat Internet Source	1%
18	expertsystem.com Internet Source	1%
19	www.mordorintelligence.com Internet Source	1%
	www.dituniversity.edu.in	

20	Internet Source	1%
21	oaji.net Internet Source	1%
22	ethesis.nitrkl.ac.in Internet Source	1%
23	machinelearningasaservice.weebly.com Internet Source	1%
24	Submitted to 9475 Student Paper	1%
25	chatbotslife.com Internet Source	1%
26	olovskog.wordpress.com Internet Source	<1%
27	clashfinder.com Internet Source	<1%
28	1000projects.org Internet Source	<1%
29	Jennifer L. French. "'Letras Terribles': Mourning and Reparation in Two Poems by Augusto Roa Bastos", Journal of Latin American Cultural Studies, 2019 Publication	<1%
30	Submitted to Institute of Technology, Nirma	

University

Student Paper

<1%

Exclude quotes	On	Exclude matches	< 5 words
Exclude bibliography	On		