Salesforce Account Handling with Voice Commands Using NLP

D. Kartik Yadav[1], Prof M. Sampath Kumar[2]
ANDHRA UNIVERSITY College of Engineering

***Abstract*—**Voice commands are the latest trending technology today which is very simple and easy to use by people of any age group. The paper depicts the voice-based application which is built upon the integration of three major components which are speech recognition [1], semantic similarity chatbot modal for NLP, and salesforce API. The first part is speech recognition handled by the google assistant, the second part is finding the best response of the speech which is converted into text by the google assistant using semantic similarity chatbot and the final part is the integration of the bundle with the salesforce API for the creation of new accounts and modification of the existing accounts.

Keywords*—Voice assistants, Natural Language Processing (NLP), Semantic Similarity Chatbot, Salesforce.*

I. INTRODUCTION

Nowadays the easily available luxury for people all over the world is an assistant who always listens for your call, anticipates your every need, and takes action when necessary. That luxury is now available to everyone thanks to the evolution of artificial intelligence assistants, aka voice assistants. Voice assistants come in somewhat small packages and can perform a variety of actions such as turning on lights, answering questions, playing music, placing online orders, etc.

Voice assistants are often mistaken with virtual assistants, which are people who work remotely and can therefore handle all kinds of tasks. Rather, voice assistants are technology-based. As voice assistants become more robust, their utility in both the personal and business realms will grow as well. A voice assistant is a digital assistant that uses voice recognition, speech synthesis, and natural language processing (NLP) to provide a service through a particular application on any mobile phone, personal computer or a tablet PC.

The training dataset which we look up for the response for the user is a large corpus of 3 million sentences. The sentence embedding of the corpus is generated by a model named BERT-base-nil-mean-tokens. The model uses the concept of sentence transformers to generate the sentence embeddings. BERT stands for Bidirectional Encoder Representations from Transformers is research papers published by the AI researchers at Google. The main reason behind using BERT base model is the size of the corpus we have in the dataset. If we use large and extra-large BERT models, we have a problem of overfitting. The usage of the BERT model depends upon the amount of data handled by an application.

The internal mechanism for finding the nearest sentence for a given voice command by the user is found out by the Semantic Similarity Chatbot[3]. The organization of this paper is as follows. Section-3 describes the proposed algorithm. Section-4 describes the detailed algorithm. Section-5 describes the conclusion of the research paper.

II. RELATED WORK

The basic idea of voice recognition has been originated by the conversion of speech to text. Many algorithms are there for the evaluation of polynomials at a large number of values.

[1] Ayushi Trivedi, Navya Pant, Pinal Shah, Simran Sonik, and Supriya Agrawal, ”Speech to text and text to speech recognition systems-Areview”[2] Google Cloud Speech to text conversion,https://cloud.google.com/speech-to-text/docs/streaming-recognize[3] Tatwadarshi P. Nagarhalli, Vinod Vaze, N. K. Rana, "A Review of Current Trends in the Development of Chatbot Systems", Advanced Computing and Communication Systems (ICACCS) 2020 6th International Conference on, pp.706-710, 2020.[4] Jasmina Đ. Novakovic, Alempije Veljovic, Sinisa S. Ilic, MilosPapic “Experimental Study Of Using The K-Nearest Neighbour Classifier With Filter Methods ”[5] Enhancing Customer Experience with Salesforce Chatbot Integration,https://medium.com/voice-tech-podcast/enhancing-customer-experience-with-salesforce-chatbot-integration-3f56361a2fe3

III. THE PROPOSED ALGORITHM

This section presents an outline of the proposed algorithm. Each step is explained as procuring more formally in the next section.

1. Voice Recognition by Google Assistant

Step 1: The Google Assistant is launched by the user and a command “Talk to my salesforce” is used to start the salesforce service which is used.

Step 2: All the available options on the screen are voiced out by the Google Assistant [2]and the response is awaited.

Step 3: A voice command is given by the user to select from the voiced-out options.

Step 4: The given voice command is converted into text by the Google Assistant.

2. Response of the Semantic Similarity Chatbot

Step 1: The Google Assistant gives the text form of the voice command input to the semantic similarity chatbot.

Step 2: Finding the nearest sentences from the corpus trained.

Step 3: Choosing the response of the sentence which is nearest to the input voice command.

Step 4: The nearest response is given for the input sentence to trigger the salesforce action.

3. Integration with Salesforce

Step 1: According to the given response from the semantic similarity chatbot the Salesforce action is triggered.

Step 2: Confirmation of the user is taken when necessary.

Step 3: After the completion of the triggered action the Google Assistant voices out the action performed.

Step 4: After the completion of the account creation or modification the account details are displayed on the screen and voiced out.

IV. DETAILED ALGORITHM

In this section, a detailed account of the Voice Recognition by Google Assistant, semantic similarity chatbot creation and Triggering Salesforce action is explained

1. Voice Recognition by Google Assistant:

The procedure below tells us about performing streaming speech recognition on an audio stream received from a microphone:

1. The voice command is received by the microphone of the mobile yielding the audio chunks.

2. The audio stream is run asynchronously to fill the buffer object.

3. Signal the generator to terminate so that the client streaming recognize method will not block the process termination.

4. The responses passed is a generator that will block until a response is provided by the server

5. Each response may contain multiple results, and each result may contain multiple alternatives.

6. In this case, responses are provided for interim results as well. If the response is an interim one, print

a line feed at the end of it, to allow the next result to overwrite it, until the response is a final one.

```
from __future__ import division
import re
import sys
from google.cloud import speech
from google.cloud.speech import enums
from google.cloud.speech import types
import pyaudio
from six.moves import queue
# Audio recording parameters
RATE = 16000
CHUNK = int(RATE / 10) # 100ms
class MicrophoneStream(object):
    """Opens a recording stream as a generator yielding the audio chunks."""
def __init__(self, rate, chunk):
    self._rate = rate
    self._chunk = chunk
    self._buff = queue.Queue()
    self.closed = True
def __enter__(self):
  self._audio_interface = pyaudio.PyAudio()
  self._audio_stream = self._audio_interface.open(
      format=pyaudio.paInt16,
      channels=1, rate=self._rate,
      input=True, frames_per_buffer=self._chunk,
      stream_callback=self._fill_buffer,
)
  self.closed = False
  return self
def __exit__(self, type, value, traceback):
  self._audio_stream.stop_stream()
  self._audio_stream.close()
  self.closed = True
  self._buff.put(None)
  self._audio_interface.terminate()
def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
    """Continuously collect data from the audio stream, into the buffer."""
  self._buff.put(in_data)
  return None, pyaudio.paContinue
def generator(self):
  while not self.closed:
      chunk = self._buff.get()
      if chunk is None:
    return
data = [chunk]
while True:
 try:
  chunk = self._buff.get(block=False)
```

```
    if chunk is None:
      return
    data.append(chunk)
  except queue.Empty:
    break
yield b''.join(data)
def listen_print_loop(responses):
  """Iterates through server responses and prints them."""
    num_chars_printed = 0
    for response in responses:
      if not response.results:
        continue
      result = response.results[0]

      if not result.alternatives:
        continue
      transcript = result.alternatives[0].transcript
      overwrite_chars = ' ' * (num_chars_printed - len(transcript))
      if not result.is_final:
        sys.stdout.write(transcript + overwrite_chars + '\r')
        sys.stdout.flush()
        num_chars_printed = len(transcript)
      else:
        print(transcript + overwrite_chars)
        if re.search(r'\b(exit|quit)\b', transcript, re.I):
          print('Exiting..')
          break
        num_chars_printed = 0
def main():
    language_code = 'en-US'
    client = speech.SpeechClient()
    config = types.RecognitionConfig(
    encoding=enums.RecognitionConfig.AudioEncoding.LINEAR16,
    sample_rate_hertz=RATE,
    language_code=language_code)
streaming_config = types.StreamingRecognitionConfig(
config=config,interim_results=True)
with MicrophoneStream(RATE, CHUNK) as stream:
      audio_generator = stream.generator()
      requests=(types.StreamingRecognizeRequest(audio_content=content)
          for content in audio_generator)
      responses=client.streaming_recognize(streaming_config,requests)
      listen_print_loop(responses)
if __name__ == '__main__':
  main()
```

2. Response of the Semantic Similarity Chatbot:

The procedure below tells us about performing streaming speech recognition on an audio stream received from a microphone:

1. The chatbot[3] has been developed in the Google Colab environment which is like a jupyter notebook, we can use any basic PC with a good internet connection. The resources such as RAM, GPU, HDD are taken care of by google with customizable configuration.

2. We use spaCy an open-source software library for advanced natural language processing, written in the programming languages Python.

3. For the training of the chatbot, we use dialog-corpus provided by Cornell University for general conversations.

4. The pre-processing of the corpus is done by splitting the lines and adding special characters in between different lines.

5. All the sentences in the dialog-corpus have been converted into their respective vector forms.

6. The user query is converted into a vector form and compared with the sentence vectors in the corpus using the technique K Near Neighbours[4].

7. The most similar sentence in the corpus has been picked up from the corpus using the concept mean of vectors.

8. A similar sentence of the query has a corresponding response which is given in the form of a reply to the user.

```
! pip install spacy
! python -m spacy download en_core_web_lg
import spacy
nlp = spacy.load('en_core_web_lg')
!curl-L-O http://www.cs.cornell.edu/~cristian/data/dialogs_corpus.zip
! unzip cornell_movie_dialogs_corpus.zip
dialog_lines = {}
for line in open("dialogs_corpus_lines.txt",
          encoding="latin1"):
```

```
    line = line.strip()
    parts = line.split(" +++$+++ ")
    if len(parts) == 5:
      dialog_lines[parts[0]] = parts[4]
    else:
      dialog_lines[parts[0]] = ""
import json
responses = {}
for line in open("dialogs_corpus_lines.txt",
      encoding="latin1"):
    line = line.strip()
    parts = line.split(" +++$+++ ")
    line_ids = json.loads(parts[3].replace("'", '"'))
    for first, second in zip(line_ids[:-1], line_ids[1:]):
      responses[first] = second
import numpy as np
def sentence_mean(nlp, s):
  if s == "":
  s = " "
  doc = nlp(s, disable=['tagger', 'parser'])
  return np.mean(np.array([w.vector for w in doc]),
axis=0)
sentence_mean(nlp, "This... is a test.").shape
!pip install simpleneighbors
from simpleneighbors import SimpleNeighbors
nns = SimpleNeighbors(300)
for i , line_id in
enumerate(random.sample(list(response.keys()),1000)
#show progress
    if i % 1000 == 0: print(i, line_id, dialog_lines[line_id])
    line_text = dialog_lines[line_id]
    summary_vector = sentence_mean(nlp, line_text)
    if np.any(summary_vector):
      nns.add_one(line_id, summary_vector)
nns.build()
sentence = "I like cooking."
picked = nns.nearest(sentence_mean(nlp, sentence), 5)[0]
response_line_id = responses[picked]

print("Your line:\n\t", sentence)
print("Most similar turn:\n\t", dialog_lines[picked])
print("Response to most similar turn:\n\t",
dialog_lines[response_line_id])
!pip install
https://github.com/aparrish/semanticsimilaritychatbot/archive/master.zip
from semanticsimilaritychatbot import
SemanticSimilarityChatbot
chatbot = SemanticSimilarityChatbot(nlp, 300)
sample_n = 10000
for first_id, second_id in
random.sample(list(responses.items()), sample_n):
      chatbot.add_pair(dialog_lines[first_id],dialog_lines[se
cond_id])
 chatbot.build()
 print(chatbot.response_for("Hello computer!"))
 my_turn = "The weather's nice today, don't you think?"
 for i in range(5, 51, 5):
    print("picking from", i, "possible responses:")
    print(chatbot.response_for(my_turn, i))
    print()
chatbot.save('dialoglines-10k-sample.annoy')
chatbot.save('dialoglines-10k-sample-data.pkl')
chatbot.save('dialoglines-10k-sample-chatbot.pkl')
chatbot = SemanticSimilarityChatbot.load("dialoglines-
10k-sample", nlp)
print(chatbot.response_for("I'm going to go get some
coffee."))
from google.colab import files
files.download('dialoglines-10k-sample.annoy')
files.download('dialoglines-10k-sample-data.pkl')
files.download('dialoglines-10k-sample-chatbot.pkl')
import IPython
from google.colab import output
display(IPython.display.HTML(chatbot_html + \
      "<script>let getResp = colabGetResp;</script>"))
def get_response(val):
    resp = chatbot.response_for(val)
    return IPython.display.JSON({'result': resp})
output.register_callback('notebook.get_response',
get_response)
```

3. Integration with Salesforce:

The procedure below tells us about performing the chatbot integration with salesforce[5]:

1. Salesforce chatbot integration enables businesses to quickly access customer information without hampering the task being performed.

2. The features of a bot toolkit are as follows:

I. Bot Handler: A bot command is functioned to create a mapping between expressions to understand the pattern of the user's questions. Also, an Apex class is featured to provide the logic to the generated response. For instance, The Bot toolkit and Apex handler classes have a series of standard bot commands. Also, users can add their bot commands and provide mappings between Apex handlers and regular expressions.

II. Apex Classes: The bot toolkit is facilitated with a series of Apex classes. The Bot controller is coordinating the conversation using the submit method to lodge a response. The submit method is the area where the bot controller tries to match the expression that the user has typed. In case a match is found, the bot controlled will use a handler method in a corresponding handler class. The handler class will then utilize a series of utility classes to select the appropriate format for a message response and send it to the user including bot record, bot item, bot message, bot field, and bot response.

III. Stateful Conversations: The Bot Toolkit will support straight question/answer interactions using a session object that will represent the state of the conversation. Also, the session object will be passed back and forth between the server and the client.

IV. Message Formatting: A Bot response consists of optional and bot-message objects. Bot messages can be in the form of simple text messages, records, post back buttons, lists of items, and images. The bot components format the message in various ways and respond with the most suitable content of each message.

3. When a user gives a voice commands like 'ABC Industries', the bot toolkit will pass the command to Apex handler 'new customer', where the name of the person will be identified as a parameter, '611 Avenue' is identified as a parameter 'address' etc.

4. The commands 'Connecticut' will be identified as parameter 'state' and '11058' is identified as parameter 'zipcode'. For parameters like address and zip code reconfirmation is asked to the user. After completion of the account creation or modification, details are provided for the user on-screen and also voiced out.

V. CONCLUSION

The Salesforce service is a voice-based application which can perform email account handling through voice. Any smartphone, personal computer, or tablet which satisfies the specified hardware and software requirements can be used There is no need for large memory or high-speed processor for this project. This application can change a basic smartphone, PC, or tablet into a voice-based assistant without any hardware or software additions. The main purpose of the application is to make voice-based assistants available to people of all economic classes. The only need is a good high-speed internet connection which changes the way of the usage of the website.

VI. REFERENCES

[1]. Ayushi Trivedi, Navya Pant, Pinal Shah, Simran Sonik, and Supriya Agrawal," Speech to text and text to speech recognition systems-Areview", IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661,p-ISSN:2278-8727, Volume 20, Issue 2, Ver. I (Mar.- Apr. 2018), PP 36-43

[2]. Google Cloud Speech to text conversion, "https://cloud.google.com/speech-to-text/docs/streaming-recognize "

[3]. Tatwadarshi P. Nagarhalli, Vinod Vaze, N. K. Rana, "A Review of Current Trends in the Development of Chatbot Systems", Advanced Computing and Communication Systems (ICACCS) 2020 6th International Conference on, pp. 706-710, 2020.

[4]. Jasmina Đ. Novakovic, Alempije Veljovic, Sinisa S. Ilic, MilosPapic "Experimental Study Of Using The K-Nearest Neighbour Classifier With Filter Methods",

[5]. Enhancing Customer Experience with Salesforce ChatbotIntegration,https://medium.com/voicetechpodcast/enhancing-customer-experience-with-salesforce-chatbot-integration-3f56361a2fe3

D. Kartik Yadav is an MTech student at Andhra University College of Engineering with specialization in Artificial Intelligence and Robotics. He is interested in the research of the field Natural Language Processing which is a rapidly growing technology.