COEN 296 Term Project

Nature Language Processing

Voice driven conversational agents using text processing

Team 5

Arushi Gupta

Subrahmanya Pramod Nanduri

Suraj Bagaria

Instructor

Prof Ming Hwa Wang

Santa Clara University

Preface:

With the evolution of latest personal assistants like Siri/Google assistant, chatbots became the biggest trend in every application. But all of them lack one important aspect called Context. Context based conversations are still an unsolved problem. We want to combine various techniques and provide our solution for conversation agents driven to speech to text conversions.

Acknowledgement

We would like to thank Dr. Ming-Hwa Wang for teaching us about the topics we used in this project.

Table of Contents

1. Introduction
    1.1. Objective
    1.2. What is the problem
    1.3. Why this is project related to this class
    1.4. Why other approach is no good
    1.5. Why you think your approach is better
    1.6. Area or scope of investigation

2. Theoretical bases and literature review
    2.1. Definition of the problem
    2.2. Theoretical Background Of The Problem
    2.3. Related research to solve the problem
    2.4. Advantage/Disadvantage of those research
    2.5. Your solution to solve this problem
    2.6. Where your solution different from others
    2.7. Why your solution is better

3. Hypothesis (or goals)
    3.1 Single/multiple hypothesis
    3.2 Positive or negative (only for proof correctness) hypothesis

4. Methodology
    4.1 How to generate/collect input data
    4.2 How to solve the problem

        4.2.1 Algorithm design
        4.2.2 Language used
        4.2.3 Tools used
    4.3 How to generate output
    4.4 How to test against hypotheses

5. Bibliography
6. Implementation
    6.1 Code (refer programming requirements)
    6.2 Design document and flowchart
7. Data Analysis and Discussion
    7.1 Output generation
    7.2 Output analysis
    7.3 Compare output against hypothesis
    7.4 Abnormal case explanation (the most important task if you have it)
    7.5 Static regression (if you have complex curve to fit)
    7.6 Discussion
8. Conclusions and recommendations
    8.1 Summary and conclusions
    8.2 Recommendations for future studies
9. Bibliography
10. Appendices
    10.1 program source code with documentation
    10.2 input/output listing

List of Figures:

1. Figure 1: Problems with chatbot
2. Figure 2: Google Language API
3. Figure 3: LSTM model
4. Figure 4: Good chatbot vs Bad chatbot
5. Figure 5: Process flow
6. Figure 6: User Interface
7. Figure 7: Chatbot Results
8. Figure 8: Chatbot server status
9. Figure 9: Chatbot Result using LSTM

Abstract

In this project, we are using natural language techniques to develop a conversational agent using sequence to sequence long short-term memory cell neural network (LSTM). Neural Networks can be used to do more than classification such as mapping of complex functions. The user will be able to converse with the system using audio and will receive audio responses with context to the conversation. The user can say anything to the system, the sentences are converted to text and pre-processed and the LSTM will be used to extract the semantic information between the sentences. The aim is to help the user make decisions by storing previous context.

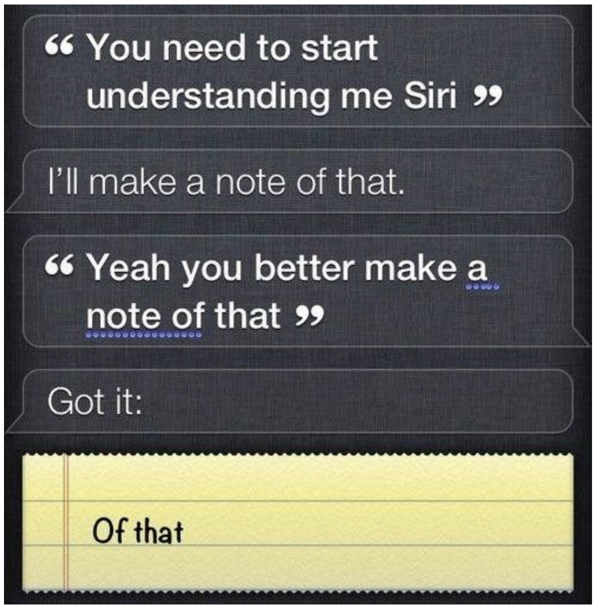
1. Introduction

1.1. Objective

The objective of this project is to create a voice based conversational agent which can mimic a human conversation by maintaining the state of the conversation and responding in the current context.

1.2. What is the problem

The problem with chatbots are that they are not trained to respond to all possible scenarios and often result in closed conversations. Also, the responses generated are based on the previous message it receives. To determine the best possible response for any given message it receives, the agent has to hold context. This will help the agent to respond to the message with the relevant information and ask follow up questions to continue the conversation. Figure 1 shows an example of how chatbots are not able to understand our intentions and are not able to get us the correct information.


“You need to start understanding me Siri”

I’ll make a note of that.

“Yeah you better make a note of that”

Got it:

Of that

Figure 1: Problems with chatbot

Mimicking a human conversation is complex since they include sarcasm, leverage contextual information for a response and reading between the lines. A bot cannot hold contextual information for more than a few chats and end up losing track of what the user was saying unless they are powered by natural language processing technology. Also developers forget to narrow the scope of focus and do not try to solve problems outside the scope.

Chatbots represent an effective way to scale messaging with users. But this sometimes leaves users frustrated if the chatbot is built without a clear understanding of the current pitfalls.

1.3. Why this is project related to this class

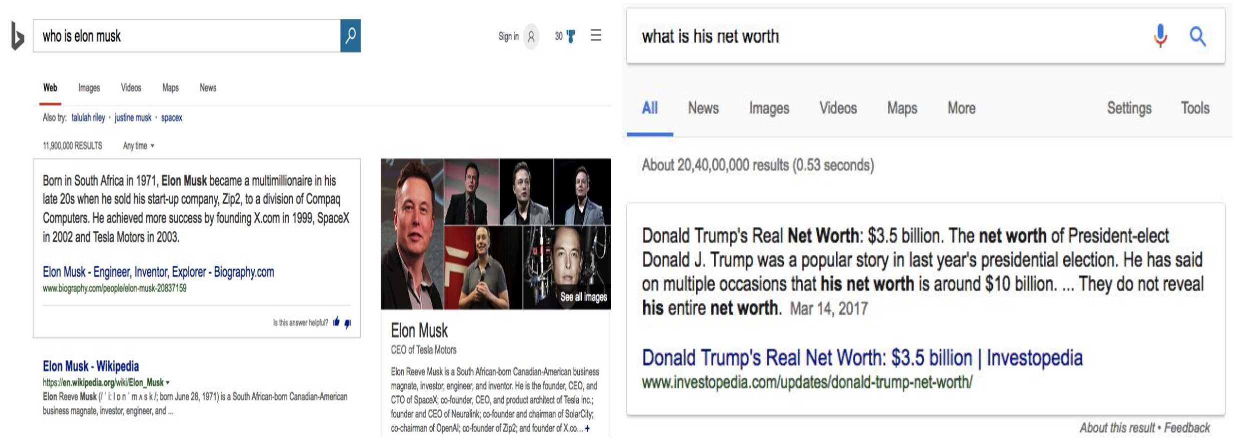
In this project, we are going to apply various models to extract useful information from the voice input message and will train the model to respond within the same context. In natural language processing, we care about understanding the key insight of the given information which is related to our project in which we are interested in discovering various techniques to model out chatbot that can be applied to many industries. The approach we will take to model a conversational agent to generate responses will utilize multiple techniques in natural language processing such as sequence to sequence modelling, LSTM networks. The voice input file will be preprocessed utilizing techniques in natural language processing that are crucial for accurate feature extraction and responding to users messages within the same context.

1.4. Why other approach is no good

Chatbots used to build using a rule based model which could not train the bot to respond to all possible scenarios. Machine learning methods have helped understand how responses can be generated instead of just using the responses used for training.

The challenge is to create a bot which can understand the context of a conversation, hold a long conversation, handle flow based conversations and multi turn dialogues.

The Google Language API had incorporated a contextual algorithm in its searches but the results were based on user search patterns and not the current flow of the conversation.



who is elon musk

Also try: talulah riley · justine musk · spacex

11,900,000 RESULTS Any time

Born in South Africa in 1971, **Elon Musk** became a multimillionaire in his late 20s when he sold his start-up company, Zip2, to a division of Compaq Computers. He achieved more success by founding X.com in 1999, SpaceX in 2002 and Tesla Motors in 2003.

Elon Musk - Engineer, Inventor, Explorer - Biography.com
www.biography.com/people/elon-musk-20837159

Elon Musk - Wikipedia
https://**en.wikipedia.org**/wiki/**Elon_Musk**
Elon Reeve **Musk** (/ ˈ iː l ɒ n ˈ m ʌ s k /; born June 28, 1971) is a South African-born Canadian-American business magnate, investor, engineer, and ...

Elon Musk
CEO of Tesla Motors

Elon Reeve Musk is a South African-born Canadian-American business magnate, investor, engineer, and inventor. He is the founder, CEO, and CTO of SpaceX; co-founder, CEO, and product architect of Tesla Inc.; founder and CEO of Neuralink; co-founder and chairman of SolarCity; co-chairman of OpenAI; co-founder of Zip2; and founder of X.co... +

what is his net worth

About 20,40,00,000 results (0.53 seconds)

Donald Trump's Real **Net Worth**: $3.5 billion. The **net worth** of President-elect Donald J. Trump was a popular story in last year's presidential election. He has said on multiple occasions that **his net worth** is around $10 billion. ... They do not reveal **his** entire **net worth**. Mar 14, 2017

Donald Trump's Real Net Worth: $3.5 billion | Investopedia
www.investopedia.com/updates/donald-trump-net-worth/

Figure 2: Google Language API

Most bots are built on a decision tree logic where the response depends on the keywords found in the input given by the user. There are very few bots with linguistic and natural language learning capabilities. Most companies try to be the first to deploy bots in a category, resulting in poor customer experience or irrelevant use cases.

1.5. Why you think your approach is better

In our project, we are trying to build a conversational agent that can respond to messages within the given context and also help make decisions for you. To help provide the user a more personalized experience, the user can provide an audio input and will get a response as an audio file. There are two types of chatbots: Retrieval based bots come with a set of written responses to minimize grammatical errors and solve simple recurring problems but does not come with responses for all possible scenarios. Generative chat bots come with responses which are generated by a program without any active human assistance. Their advantage over retrieval based is that they can easily be adapted or trained to be used in diverse domains.

To understand the context of a conversation, hold a long conversation, handle flow based conversations and multi turn dialogues, our proposed approach has the following steps:

1. Convert the audio message into a text file.
2. Text preprocessing: Tokenize the words, remove stop words, perform stemming and lemmatization.
3. Bag of words: Map each word to a n*n dimensional vector
4. TF-IDF: Rescale frequency of common words and use inverse document frequency to score rare words across documents.
5. Cosine similarity: Find similarity between 2 documents by taking their dot product.
6. Long short term memory (LSTM) neural network: A recurrent neural network can also be used to train the model but it suffers from two problems which makes it unstable. These problems are vanishing gradient and exploding gradient. LSTM is a form of RNN that learns long term dependencies of sequence inputs and remembers information for long periods of time. They predict the likelihood of a word to show up in a sentence given a sequence of previous input words.

   Figure 3 shows the components of a LSTM.

   a. Forget gate (forgetting) is used to filter the amount of past information the LSTM should keep such as when the subject of interest changes during a conversation.

b. Input gate (ignoring) decides which information should be input into the network.

c. Candidate state (selection) generates a vector of potential candidate

d. The ⊕ and ⊗ symbols are used to filter out possibilities.

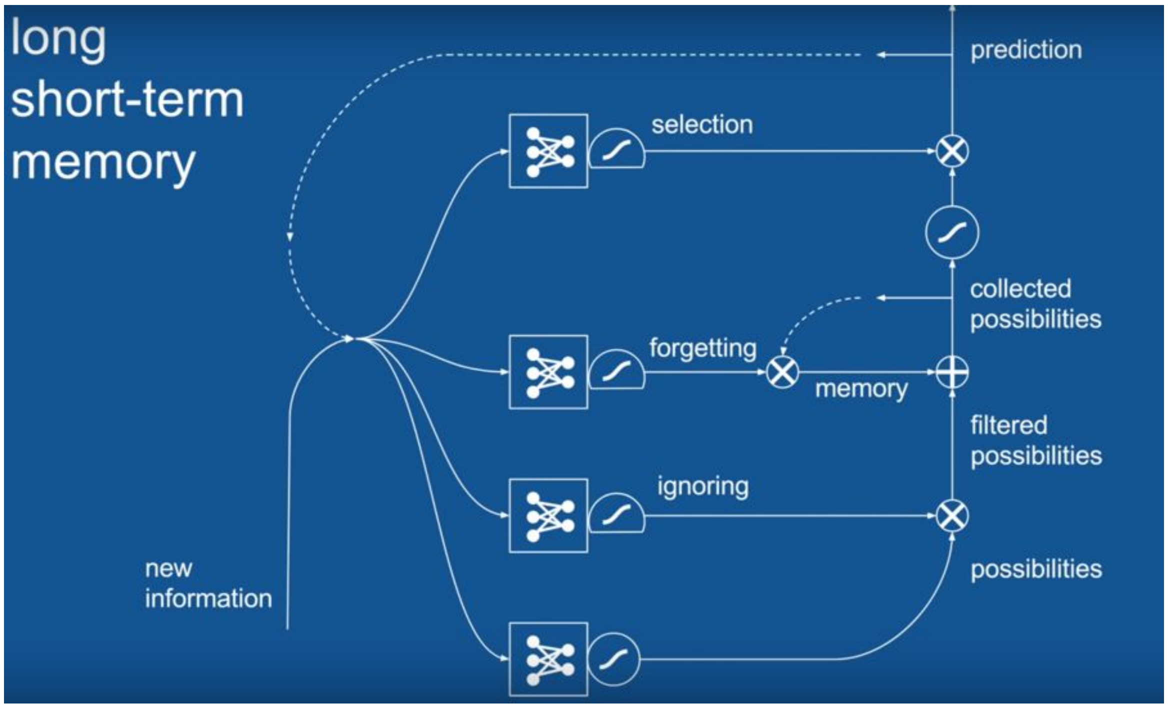
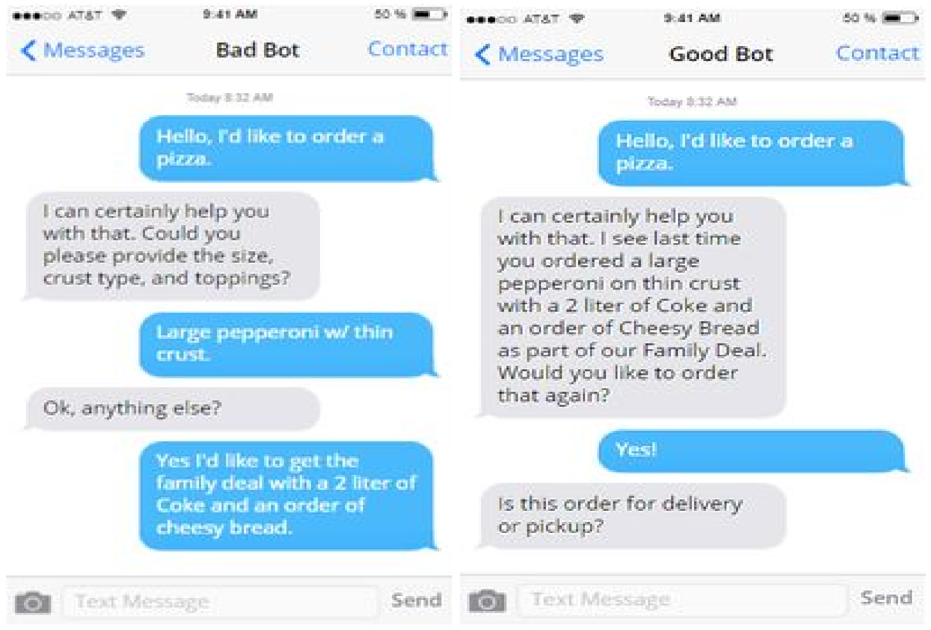


Figure 3: LSTM model

Figure 4 shows an example between a good chatbot and a bad chatbot. The good chatbot did a better job at automating the process and was faster than using a chatbot without context.



Hello, I'd like to order a pizza.

I can certainly help you with that. Could you please provide the size, crust type, and toppings?

Large pepperoni w/ thin crust.

Ok, anything else?

Yes I'd like to get the family deal with a 2 liter of Coke and an order of cheesy bread.

Hello, I'd like to order a pizza.

I can certainly help you with that. I see last time you ordered a large pepperoni on thin crust with a 2 liter of Coke and an order of Cheesy Bread as part of our Family Deal. Would you like to order that again?

Yes!

Is this order for delivery or pickup?

Figure 4: Good chatbot vs Bad chatbot

1.6. Area or scope of investigation

The proposed conversational agent should be able to hold both long and short conversations using a generative based model to generate responses. Training a model to mimic a human conversation is still a work in progress. Some of the challenges include: speech recognition and synthesis, syntactic and semantic analysis, real world knowledge. Linking all these technologies is the hardest part in order to create a conversational flow and avoid dead ends.

1. <u>Theoretical background of the problem:</u>

A chatbot (also known as a smartbots, talkbot, chatterbot, Bot, IM bot, interactive agent, Conversational interface or Artificial Conversational Entity) is a computer program or an artificial intelligence which conducts a conversation via auditory or textual methods. Such programs are often designed to convincingly simulate how a human would behave as a conversational partner, thereby passing the Turing test. Chatbots are typically used in dialog systems for various practical purposes including customer service or information acquisition. Some chatterbots use sophisticated natural language processing systems, but many simpler systems scan for keywords within the input, then pull a reply with the most matching keywords, or the most similar wording pattern, from a database.

The term "ChatterBot" was originally coined by Michael Mauldin (creator of the first Verbot, Julia) in 1994 to describe these conversational programs. Today, most chatbots are either accessed via virtual assistants such as Google Assistant and Amazon Alexa, via messaging apps such as Facebook Messenger or WeChat, or via individual organizations' apps and websites. Chatbots can be classified into usage categories such as conversational commerce (e-commerce via chat), analytics, communication, customer support, design, developer tools, education, entertainment, finance, food, games, health, HR, marketing, news, personal, productivity, shopping, social, sports, travel and utilities.

In 1950, Alan Turing's famous article "Computing Machinery and Intelligence" was published, which proposed what is now called the Turing test as a criterion of intelligence. This criterion depends on the ability of a computer program to impersonate a human in a real-time written conversation with a human judge, sufficiently well that the judge is unable to distinguish reliably—on the basis of the conversational content alone—between the program and a real human. The notoriety of Turing's proposed test stimulated great interest in Joseph Weizenbaum's program ELIZA, published in 1966, which seemed to be able to fool users into believing that they were conversing with a real human. However, Weizenbaum himself did not claim that ELIZA was genuinely intelligent, and the Introduction to his paper presented it more as a debunking exercise:

[In] artificial intelligence ... machines are made to behave in wondrous ways, often sufficient to dazzle even the most experienced observer. But once a particular program is unmasked, once its inner workings are explained ... its magic crumbles away; it stands revealed as a mere collection of procedures ... The observer says to himself "I could have written that". With that thought he moves the program in question from the shelf marked "intelligent", to that reserved for curios ... The object of this paper is to cause just such a re-evaluation of the program about to be "explained". Few programs ever needed it more.

ELIZA's key method of operation (copied by chatbot designers ever since) involves the recognition of cue words or phrases in the input, and the output of corresponding pre-prepared or pre-programmed responses that can move the conversation forward in an apparently meaningful way (e.g. by responding to any input that contains the word 'MOTHER' with 'TELL ME MORE ABOUT YOUR FAMILY'). Thus an illusion of understanding is generated, even though the processing involved has been merely superficial. ELIZA showed that such an illusion is surprisingly easy to generate, because human judges are so ready to give the benefit of the doubt when conversational responses are capable of being interpreted as "intelligent".

Interface designers have come to appreciate that humans' readiness to interpret computer output as genuinely conversational—even when it is actually based on rather simple pattern-matching—can be exploited for useful purposes. Most people prefer to engage with programs that are human-like, and this gives chatbot-style techniques a potentially useful role in interactive systems that need to elicit information from users, as long as that information is relatively straightforward and falls into predictable categories. Thus, for example, online help systems can usefully employ chatbot techniques to identify the area of help that users require, potentially providing a "friendlier" interface than a more formal search or menu system. This sort of usage holds the prospect of moving chatbot technology from Weizenbaum's "shelf ... reserved for curios" to that marked "genuinely useful computational methods".

The classic historic early chatbots are ELIZA (1966) and PARRY (1972). More recent notable programs include A.L.I.C.E., Jabberwacky and D.U.D.E (Agence Nationale de la Rechercheand CNRS 2006). While ELIZA and PARRY were used exclusively to simulate typed conversation, many chatbots now include functional features such as games and web searching abilities. In 1984, a book called The Policeman's Beard is Half Constructed was published, allegedly written by the chatbot Racter (though the program as released would not have been capable of doing so).

One pertinent field of AI research is natural language processing. Usually, weak AI fields employ specialized software or programming languages created specifically for the narrow function required. For example, A.L.I.C.E. uses a markup language called AIML, which is specific to its function as a conversational agent, and has since been adopted by various other developers of, so called, Alicebots. Nevertheless, A.L.I.C.E. is still purely based on pattern matching techniques without any reasoning capabilities, the same technique ELIZA was using back in 1966. This is not strong AI, which would require sapience and logical reasoning abilities.

Jabberwacky learns new responses and context based on real-time user interactions, rather than being driven from a static database. Some more recent chatbots also combine real-time learning with evolutionary algorithms that optimise their ability to communicate based on each conversation held. Still, there is currently no general purpose conversational artificial intelligence, and some software developers focus on the practical aspect, information retrieval.

Chatbot competitions focus on the Turing test or more specific goals. Two such annual contests are the Loebner Prize and The Chatterbox Challenge (offline since 2015, materials can still be found from web archives).

According to Forrester (2015), AI will replace 16 percent of American jobs by the end of the decade. Chatbots have been used in applications such as customer service, sales and product education. However, a study conducted by Narrative Science in 2015 found that 80 percent of their respondents believe AI improves worker performance and creates jobs.

2. <u>Related research to solve the problem:</u>

a. <u>Sample Efficient Deep Reinforcement Learning for Dialogue Systems With Large Action Spaces:</u>

In spoken dialogue systems, Aim was to deploy artificial intelligence to build automated dialogue agents that can converse with humans. A part of this effort is the policy optimization task, which attempts to find a policy describing how to respond to humans, in the form of a function taking the current state of the dialogue and returning the response of the system. In this paper, we investigate deep reinforcement learning approaches to solve this problem. Particular attention is given to actor-critic methods, off-policy reinforcement learning with experience replay, and various methods aimed at reducing the bias and variance of estimators. When combined, these methods result in the previously proposed ACER algorithm that gave competitive results in gaming environments. These environments, however, are fully observable and have a relatively small action set so, in this paper, we examine the application of ACER to dialogue policy optimization. It can be shown that this method beats the current state of the art in deep learning approaches for spoken dialogue systems. This not only leads to a more sample efficient algorithm that can train faster, but also allows us to apply the algorithm in more difficult environments than before. We thus experiment with learning in a very large action space, which has two orders of magnitude more actions than previously considered. We find that ACER trains significantly faster than the current state of the art.

b. *<u>A neural-network based chat bot:</u>*

In this paper, they explored the avenues of teaching computers to process natural language text by developing a chat bot. They took an experiential approach from a beginner level of understanding, in trying to appreciate the processes, techniques, the power and possibilities of natural language processing using recurrent neural networks (RNN). To achieve this, they kick started our experiment by implementing sequence to sequence long short-term memory cell neural network (LSTM) in conjunction with Google word2vec. Their results show the relationship between the number of training times and the quality of language model used for training our model bot affect the quality of its prediction output. Furthermore, they demonstrate reasoning and generative capabilities or RNN based chat bot.

Let's imagine for a minute world where instead of a human being at a customer support centre, a chat bot helps us fix our router and the internet starts working again. Such an invention would be of great convenience in this ever connected world where we cannot afford to wait several hours per year to fix our internet connection, it is the internet. But is it possible? Simple answer is yes. Today's applications and technologies like Apple's Siri, Microsoft Cortana, are at the forefront of highly personalised virtual assistants. They do not do what we have imagined but they are, by no reasonable doubt, a stone throw away from being able to do so. Now where do we begin?

The most obvious solution that leads us one step closer to living in our imaginary world is knowing that the chat bot must be able to understand messages we present it and how to respond appropriately. But computers are dumb. For starters, they use numbers raised to the powers of two, which is binary and that is all they know, whilst humans normally use decimal numbers, expressed as powers of ten, humans can read, write, and are intelligent. How do we make them understand our natural language when all they know is 0 and 1? Luckily for us there is a field of computer science called Natural Language Processing (NLP) and linguistics that comes to our rescue. As the name suggests, NLP is a form of artificial intelligence that helps machines “read” text by simulating the human skill to comprehend language. Given the benefit of time, driven by cognitive and semantic technologies, natural language processing will make great strides in human -like understanding of speech and text, thereby enabling computers to understand what human language input is meant to communicate.

A chat bot is a computer program capable of holding conversations in a single or multiple human languages as if it were human. Applications can range from customer assistance, translation, home automation to name just a few. This paper explores the technologies behind such innovation, implements a simple model and experiments with the model.

Recent research has demonstrated that deep neural networks can be trained to do more than classification but mapping of complex functions to complex functions. An example of this mapping is the sequence to sequence mapping

done in language translation. This same mapping can be applied to conversational agents, i.e. to map input to responses by using probabilistic computation of occurrence of tokens in sequences that belong to a finite set of formally defined model of utility such as a language model. We develop our own sequence to sequence model with the intention of experimenting with this technology in hope of understanding the underlying functionality, concepts and capabilities of deep neural networks.
We trained our model on a small conversational dataset to kick start a series of experiments with the model.

This research is inspired by the investigation initiated by a group of researchers who treated generation of conversational dialogue as a statistical machine translation (SMT) problem. Previously dialogue systems were based on hand-coded rules, typically either building statistical models on top of heuristic rules or templates or learning generation rules from a minimal set of authored rules or labels. The SMT is data driven and it learns to converse from human to human corpora. SMT researched on modelling conversation from micro blogging sites. In their research they viewed response generation as a translation problem where a post needed to be translated into a response. However, it was discovered that response generation is considerably more difficult than translating from one language to another due to lack of a plausible response and lack of phrase alignment between each post and response. Improved upon by re-scoring the output of a phrasal SMT-based conversation system with a SEQ2SEQ model that incorporates prior context. They did this using an extension of (Cho et al 2014) Hierarchical Recurrent

Encoder Decoder Architecture (HRED). Hidden state LSTMs that implement GRUs were the ingredient that improved on STM. Both the encoder and decoder make use of the hidden state LSTM/GRUs. However in their paper they were using triples not full length dialogue conversations which they mentioned in their future work. One of the biggest problems was that conversational NLP systems could not keep the context of a conversation but was only able to respond from input to input. Hochreitern et al. wrote a paper on long short memory (LSTM) retention in deep nets. At the time of publication of paper, they had not found practical solutions for the LSTM but now it is being used by conversational NLP systems to keep the context of a conversation as it uses the LSTM to keep previous responses and inputs of text into a deep net. Training proposes techniques of improving training in deep nets which overcome the vanishing gradient or exploding gradient in Neural nets by using a stack of restricted Boltzman machine. The vanishing gradient would make the layers closer to the output learn faster than ones closer the input. This would end up with a network that has the output layers trained whilst the input layers were not trained. Tsung-Hsien Wen at al. proposed a paper on NLG using LSTM for spoken dialog system (SDS) which help in the structuring of Chatbot. Finally, Mikolov et al. at google produced a paper that describes the efficient use of word vectors for large data sets which is currently being used by google for its Deep mind machine. Our Chat Bot will use Word2vec which is based on the publication of Mikolov et al. to convert words in its language into vectors representation.

Chat bot models come packaged in different shapes and sizes. To begin with, there are two types of Chat bots i.e. *Retrieval-based and Generative*.

The *Retrieval based* bots come with a set of written responses to minimize grammatical errors, improve coherence, and avoid a situation where the system can be hacked to respond in less appropriate ways. *Retrieval based* chat bots best suit closed domain systems.

Closed domain chat bot systems are built to specifically solve simple recurring problems for instance an elevator voice assistant. The draw backs with closed domain chat bots is that the set of data they come with does not come with responses for all possible scenarios.

Generative model conversational agents are more intelligent but more difficult to implement than retrieval based chat bots. With generative chat bots, responses are generated by a program without any active human assistance. Their advantage over retrieval based is that they can easily be adapted or trained to be used in diverse domains.

Open domain systems include popular virtual assistance services like *Siri* and *Cortana* which are not constrained to a particular type of a conversation domain. Apple users can ask *Siri* anything from time of day to business advice and still get reasonable human like responses.

c. *<u>A Neural Chatbot with Personality</u>*

Conversational modeling is an important task in natural language processing as well as machine learning. Like most important tasks, it's not easy. Previously, conversational models have been focused on specific domains, such as booking hotels or recommending restaurants. They were built using hand-crafted rules, like ChatScript, a popular rule-based conversational model.

In 2014, the sequence to sequence model being used for translation opened the possibility of phrasing dialogues as a translation problem: translating from an utterance to its response. The systems built using this principle, while conversing fairly fluently, aren't very convincing because of their lack of personality and inconsistent persona.

A basic sequence-to-sequence model, as introduced in Cho et al., 2014, consists of two recurrent neural networks (RNNs): an encoder that processes the input and a decoder that generates the output. The encoder maps a variable-length source sequence to a fixed-length vector, and the decoder maps the vector representation back to a variable length target sequence. Sequence-to-sequence is often used with attention-based that allows the decoder more direct access to the input. This model has been successfully used for many different natural language processing tasks, such as alignment, translation, and summarization.
Conversational modeling can be phrased as a mapping between utterances and responses, and therefore can benefit from the encoder-decoder setup. In our model, the encoder processes an utterance by human, and the decoder produces the response to that utterance. We train the word embeddings as we train the model. We also use attentional mechanism and experimenting with using GLoVe pre-trained word vectors to initialize our word embeddings.

To make the bot speak like a certain character, we train vector embeddings for different characters with the hope that these embeddings would be able to encode information and style of speech of these characters. These character embeddings are trained together with the word embeddings. This is inspired by Google's Zero-shot multilingual translation system.

3.5 Your solution to solve this problem

Conversational chatbot is one of the most researched topics that is going around. Our project involves Voice drives contextual chatbot. So we have various components involved for our process.

- Voice to text conversion
- Text interpretation
- Tracking and store previous context
- Respond to the text

Voice to text conversion:

Since our main goal is to build a conversational bot, we are going to use standard google text to voice conversion. We use standard microphone to record the voice . Feed the signal to voice to text conversion apis. The result of the voice to text conversion apis would be text that is interpreted from the text.

Text Interpretation:

Once we receive the text from the voice to text conversion api, we try to interpret the text as a question and answer agent. We interpret the text that came by feeding it to the model that we build from our system. Our model internally uses LSTM with various techniques to interpret the text and send appropriate response which we will discuss in the further sections.

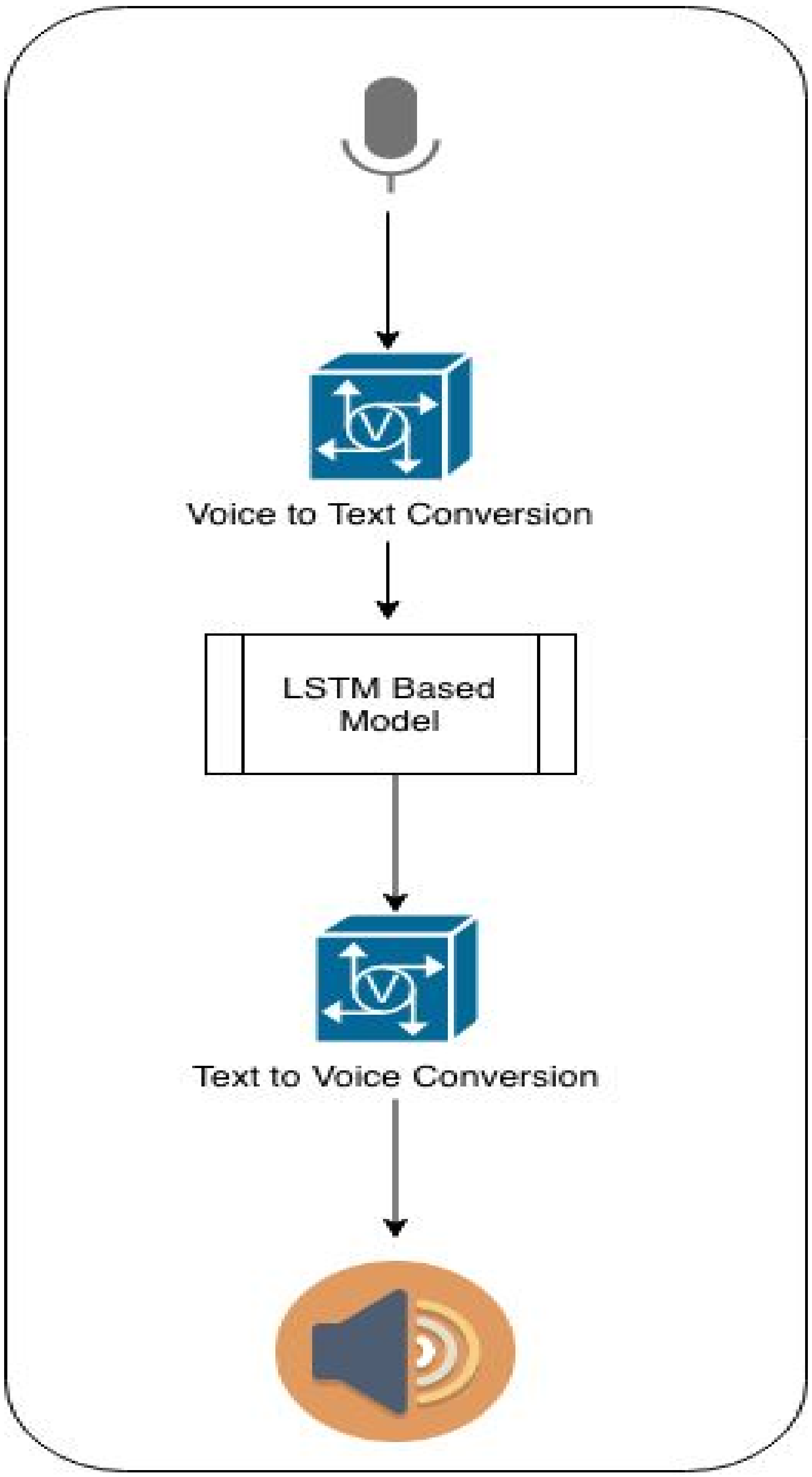
Tracking and store previous context:

As we take the text input and interpret the text, we store the text within the model . Our model uses the old text history that is part of the conversation and moves along with further answers. As we take more history of the conversation in place, the training process can take a lot of time further affecting the accuracy and other tuning parameters.

Respond to the text:

Once we have the response from the model for the conversation that is going on, we make the system read out to us. We use the text to speech conversion apis to convert the response output to the voice so that output seems to be a voice driven conversation.

The overall flow of the project working is as shown in the below picture:



Voice Driven Conversational Chatbot

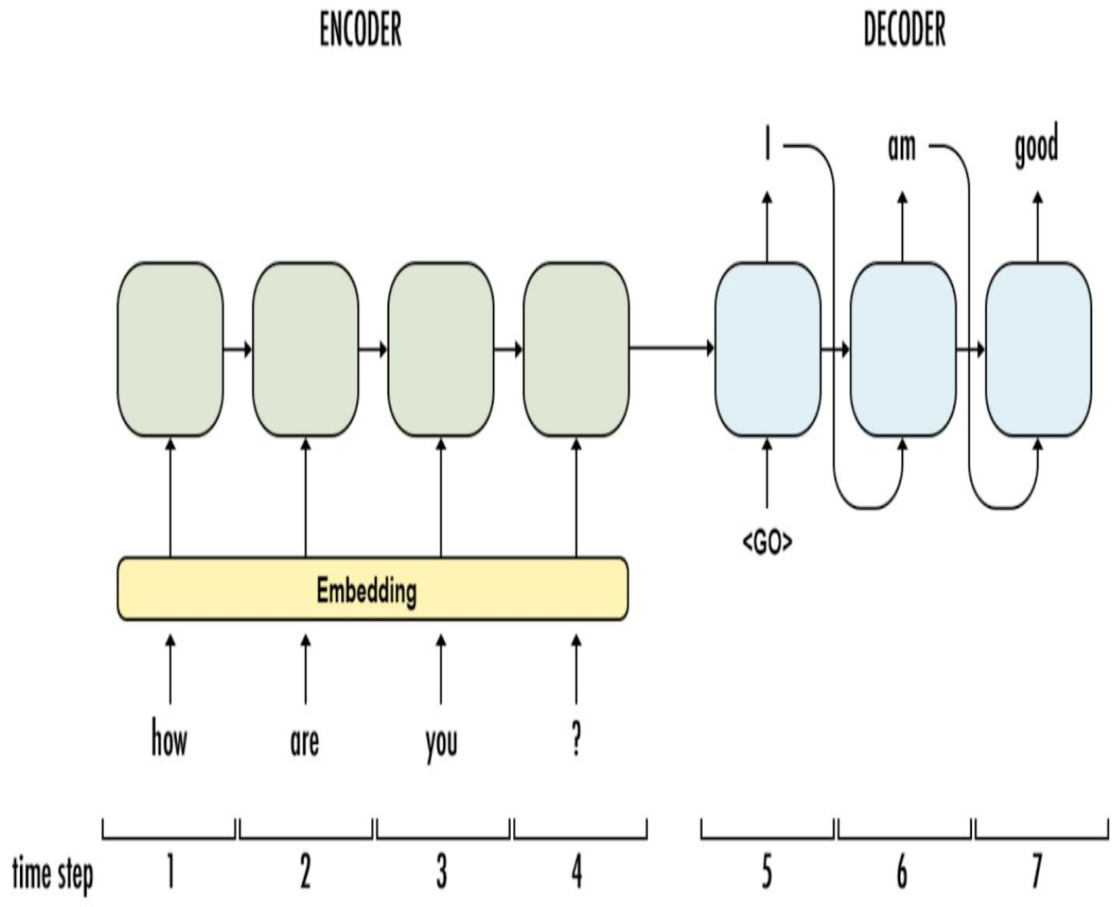
3.6 where your solution different from others

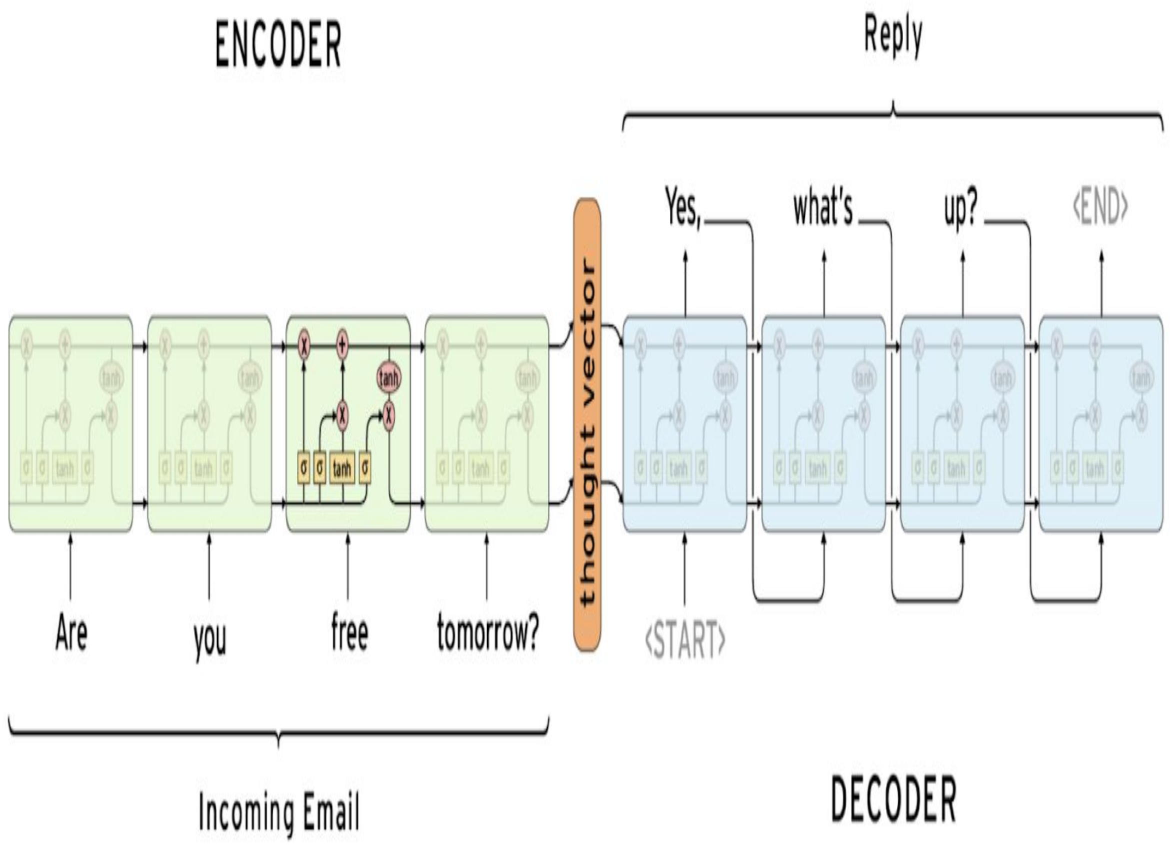
Standard chatbots includes text processing using LSTM algorithms. But we are planning to use a solution where we combine LSTM with Seq2Seq generation with the objective function of semantic coherence. So for semantic coherence, We need to measure the adequacy of responses to avoid situations in which the generated replies are highly rewarded but are un- grammatical or not coherent.We also need to measure the adequacy of responses to avoid situations in which the generated replies are highly rewarded but are un-grammatical or not coherent. Our model thus integrates the power of SEQ2SEQ systems to learn compositional semantic meanings of utterances with the strengths of reinforcement learning in optimizing for long-term goals across a conversation

3.7 why your solution is better :

Since our solution includes Seq2Seq model with an optimized Semantic coherence function, we are expecting the predicted accuracy would be higher than the normal. We have taken two samples of the algorithms as given below.

LSTM Sequence To Sequence Model




ENCODER
Reply
Yes,
what's
up?
<END>
thought vector
σ
tanh
Are
you
free
tomorrow?
<START>
Incoming Email
DECODER

3. Hypothesis

The goal of the project is to develop a user-friendly and efficient system for developing a conversational agent. The system receives an input in the form of a digital audio file and converts it to a text file. Given a text file, we want the system to be able to generate responses to the message by maintaining the state of the conversation and responding in the current context. There are many algorithms to solve this problem and we select the best ones so that we can find the best features to train the model.

3.1 Hypothesis 1

Our goal is to take real time audio files and convert them to text and create a chatbot to mimic a human conversation using an ensemble of data mining algorithms.

3.2 Hypothesis 2

The model should generate responses to the messages given by the user as input by maintaining the state of the conversation and responding in the current context. It should also be able to help the user make decisions using previous context and not just the response given in the previous dialogue.

5. Methodology

5.1 How to generate/collect input data?

We are going to use a dataset form the below sources:

Cornell Movie Dialog Corpus:

Ubuntu corpus

Microsoft's Social Media Conversation Corpus

5.2 How to solve the problem?

We are solving the problem using LSTM model using Seq2Seq generation.

As discussed in the previous sections, our system takes the voice , converts the voice to text . Converted text will be fed to the model which will result in the next possible conversation.

So the approach of the model is as follows:

LSTM is a special case of RNN. Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. The repeating module in a standard RNN contains a single layer. An RNN contains the number of hidden state vectors, which each represent information from the previous time steps. For example, the hidden state vector at the 3rd time step will be a function of the first 3 words. By this logic, the final hidden state vector of the encoder RNN can be thought of as a pretty accurate representation of the whole input text.

The decoder is another RNN, which takes in the final hidden state vector of the encoder and uses it to predict the words of the output reply. Let's look at the first cell. The cell's job is to take in the vector representation v, and decide which word in its vocabulary is the most appropriate for the output response. Mathematically speaking, this means that we compute probabilities for each of the words in the vocabulary, and choose the argmax of the values.

The 2nd cell will be a function of both the vector representation v, as well as the output of the previous cell. The goal of the LSTM is to estimate the following conditional probability.

$$p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

$Y_1, \ldots, y_{T'}$ = Output Sequence
$x_1, \ldots, x_T$ = Input Sequence
v = Vector Representation

When we deconstruct what that equation is: The left side refers to the probability of the output sequence, conditioned on the given input sequence. The right side contains the term p(yt|v, y1, …, yt-1), which is a vector of probabilities of all the words, conditioned on the vector representation and the outputs at the previous time steps. The Pi notation is simply the multiplication equivalent of Sigma (or summation). The right hand side can be reduced to p(y1|v) * p(y2|v, y1) * p(y3|v, y1, y2) ... and so on.

Let’s go over a quick example before moving on. Let’s take the input text we saw in the first image. Given the phrase “Are you free tomorrow?”, let’s think about how most people would answer the question. A majority will start with something along the lines of “Yes”, “Yeah”, “No”, etc. After we’re done training our network, the probability p(y1|v) will be a distribution that looks like the following.

$$\begin{array}{lcccccc} & \text{Apple} \ldots\ldots & \text{But} \ldots\ldots & \text{Dog} \ldots\ldots & \text{No} \ldots\ldots & \text{Yeah} \ldots\ldots & \text{Yes} \\ p(y_1|v) = [& 0 & 0.01 & 0 & 0.40 & 0.25 & 0.34\ ] \end{array}$$

The second probability we need to compute, p(y2|v, y1), will be a function of the word this distribution y1 as well as the vector representation v. The result of the Pi (product) operation will give us the most likely sequence of words, which we’ll use as our final response.The most important characteristics of sequence to sequence models is the versatility that it provides. When you think of traditional ML methods (linear regression, SVMs) and deep learning methods like CNNs, these models require a fixed size input, and produce fixed size outputs as well. The lengths of the inputs must be known before hand.
This is a significant limitation to tasks such as machine translation, speech recognition, and question answering. These are tasks where we don't know the size of the input phrase, and we'd also like to be able to generate variable length responses, not just be constrained to one particular output representation. Seq2Seq models allow for that flexibility.

5.3 Algorithm design

Our rough pseudo code of the algorithm is as follows:

The algorithm includes following steps :

Step 1: Voice to speech conversion.

```
String convertToSpeech(String inputWav);
```

Step 2: Text to next sentence prediction.

```
String getNextSentence(String inputText) {
//Feed the input text to the LSTM and Seq2Seq generation model with
//semantic coherence and get the the output text of the sequence.
}
```

Step 3: Convert output text from text 2 to voice:

```
Object textToSpeech(String text);
```

5.4 Language used

Python:

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace (wiki link)

5.5 Tools used:

NLTK Libraries

Tensor flow

Skykit py

6. Implementation

Used ChatterBot to train the model on Mongo database adapter and enhanced the model to give better results. Various datasets from different sources were used to train the model.

Chatterbot: ChatterBot is a Python library that makes it easy to generate automated responses to a user's input. ChatterBot uses a selection of machine learning algorithms to produce different types of responses. This makes it easy for developers to create chat bots and automate conversations with users.

An example of typical input would be something like this:

user: Good morning! How are you doing?
bot: I am doing very well, thank you for asking.
user: You're welcome.
bot: Do you like hats?

Language Independence: The language independent design of ChatterBot allows it to be trained to speak any language. Additionally, the machine-learning nature of ChatterBot allows an agent instance to improve it's own knowledge of possible responses as it interacts with humans and other sources of informative data.

How ChatterBot Works: An untrained instance of ChatterBot starts off with no knowledge of how to communicate. Each time a user enters a statement, the library saves the text that they entered and the text that the statement was in response to. As ChatterBot receives more input the number of responses that it can reply and the accuracy of each response in relation to the input statement increase.

The program selects the closest matching response by searching for the closest matching known statement that matches the input, it then chooses a response from the selection of known responses to that statement.

Process flow diagram:

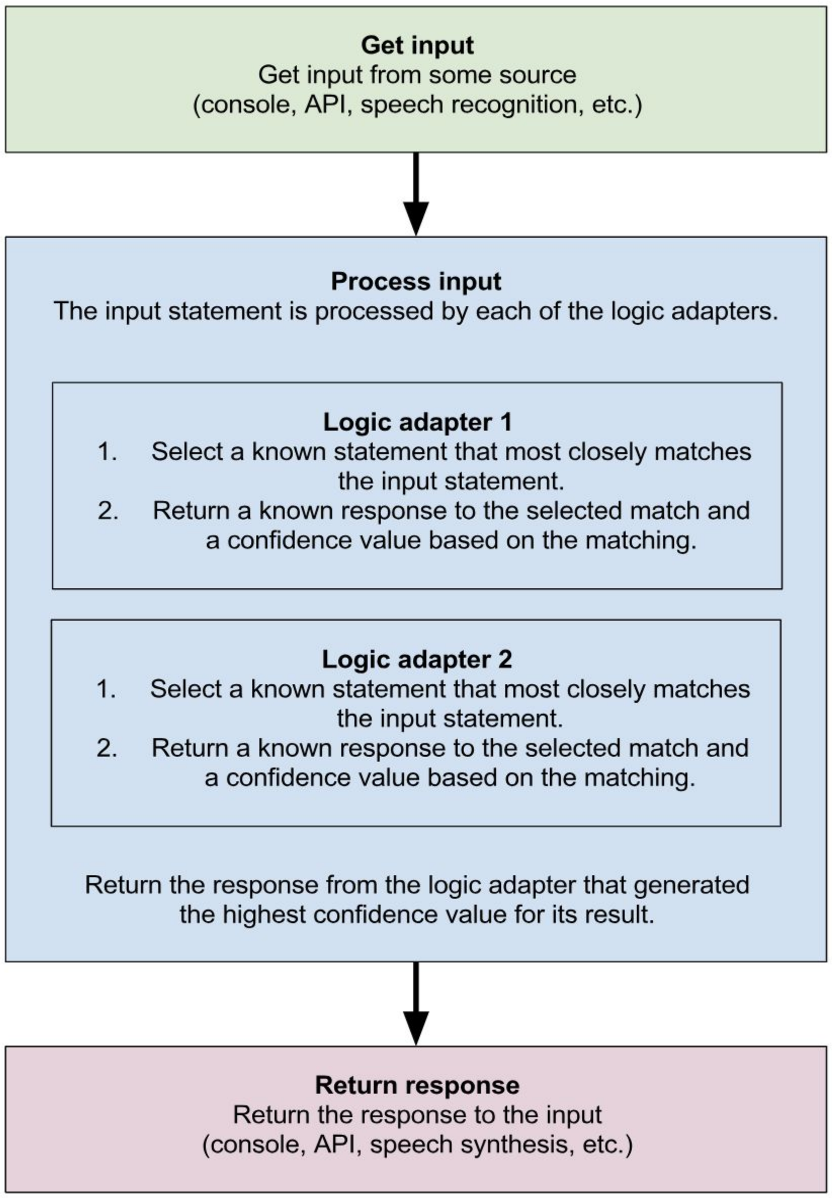

Get input
Get input from some source
(console, API, speech recognition, etc.)
Process input
The input statement is processed by each of the logic adapters.
Logic adapter 1
1. Select a known statement that most closely matches the input statement.
2. Return a known response to the selected match and a confidence value based on the matching.
Logic adapter 2
1. Select a known statement that most closely matches the input statement.
2. Return a known response to the selected match and a confidence value based on the matching.
Return the response from the logic adapter that generated the highest confidence value for its result.
Return response
Return the response to the input
(console, API, speech synthesis, etc.)

Figure 5: Process flow

6. Data analysis and discussion

6.1 Output generation

The input datasets used are:

ChatterBot Corpus: English words, Greetings , Conversations

Cornell Movie Dataset

These datasets were used to train the model. After training, the model is tested with sample user inputs to check the results.

6.2 Output analysis

User Interface:

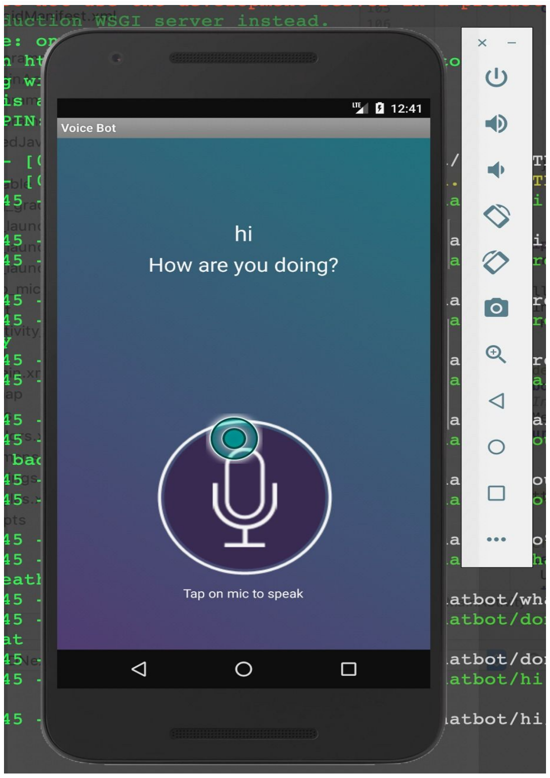
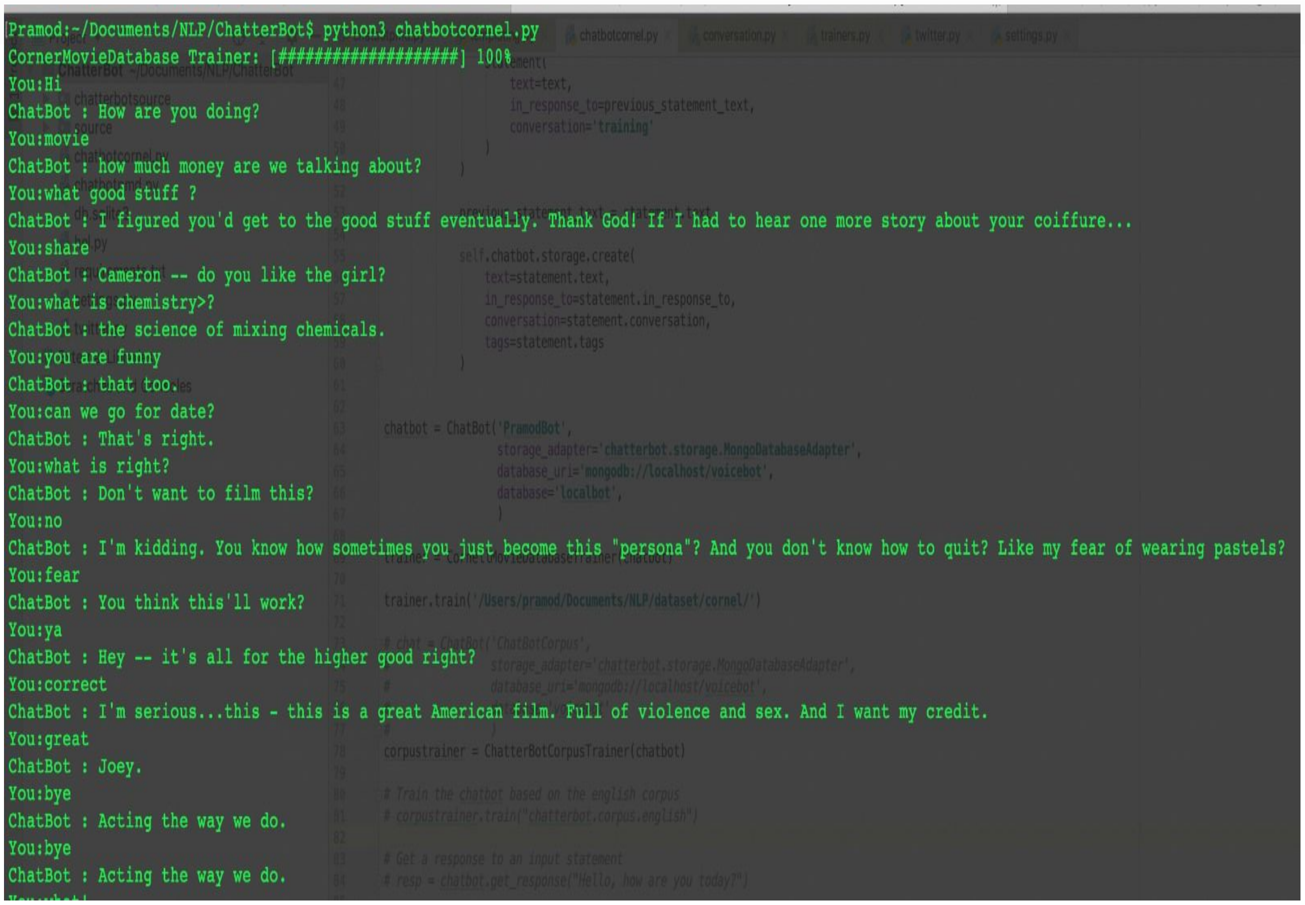


Figure 6: User Interface

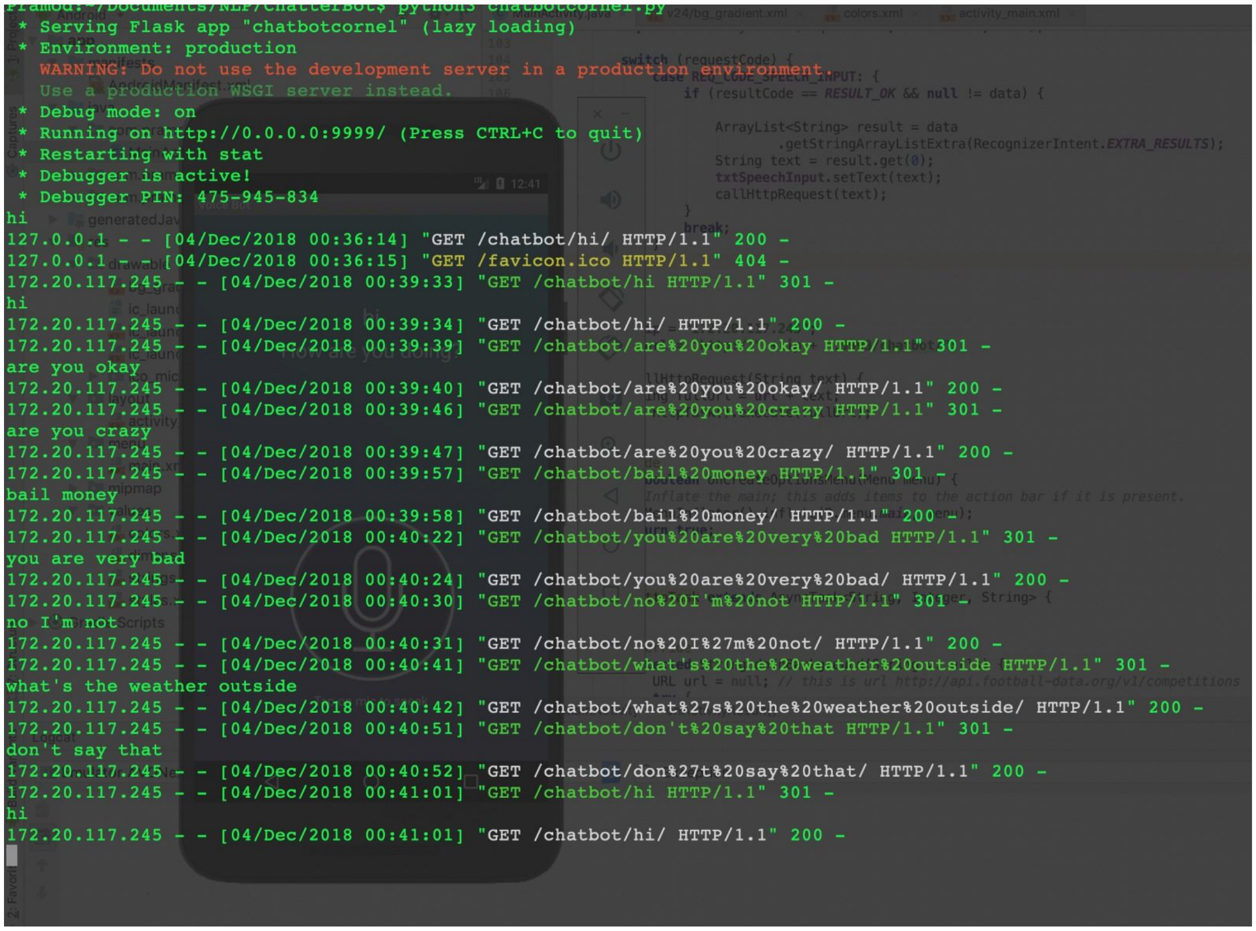
Chatbot results with sample user input:



```
Pramod:~/Documents/NLP/ChatterBot$ python3 chatbotcornel.py
CornerMovieDatabase Trainer: [##################] 100%
You:Hi
ChatBot : How are you doing?
You:movie
ChatBot : how much money are we talking about?
You:what good stuff ?
ChatBot : I figured you'd get to the good stuff eventually. Thank God! If I had to hear one more story about your coiffure...
You:share
ChatBot : Cameron -- do you like the girl?
You:what is chemistry>?
ChatBot : the science of mixing chemicals.
You:you are funny
ChatBot : that too.
You:can we go for date?
ChatBot : That's right.
You:what is right?
ChatBot : Don't want to film this?
You:no
ChatBot : I'm kidding. You know how sometimes you just become this "persona"? And you don't know how to quit? Like my fear of wearing pastels?
You:fear
ChatBot : You think this'll work?
You:ya
ChatBot : Hey -- it's all for the higher good right?
You:correct
ChatBot : I'm serious...this - this is a great American film. Full of violence and sex. And I want my credit.
You:great
ChatBot : Joey.
You:bye
ChatBot : Acting the way we do.
You:bye
ChatBot : Acting the way we do.
```

Figure 7: Chatbot Results

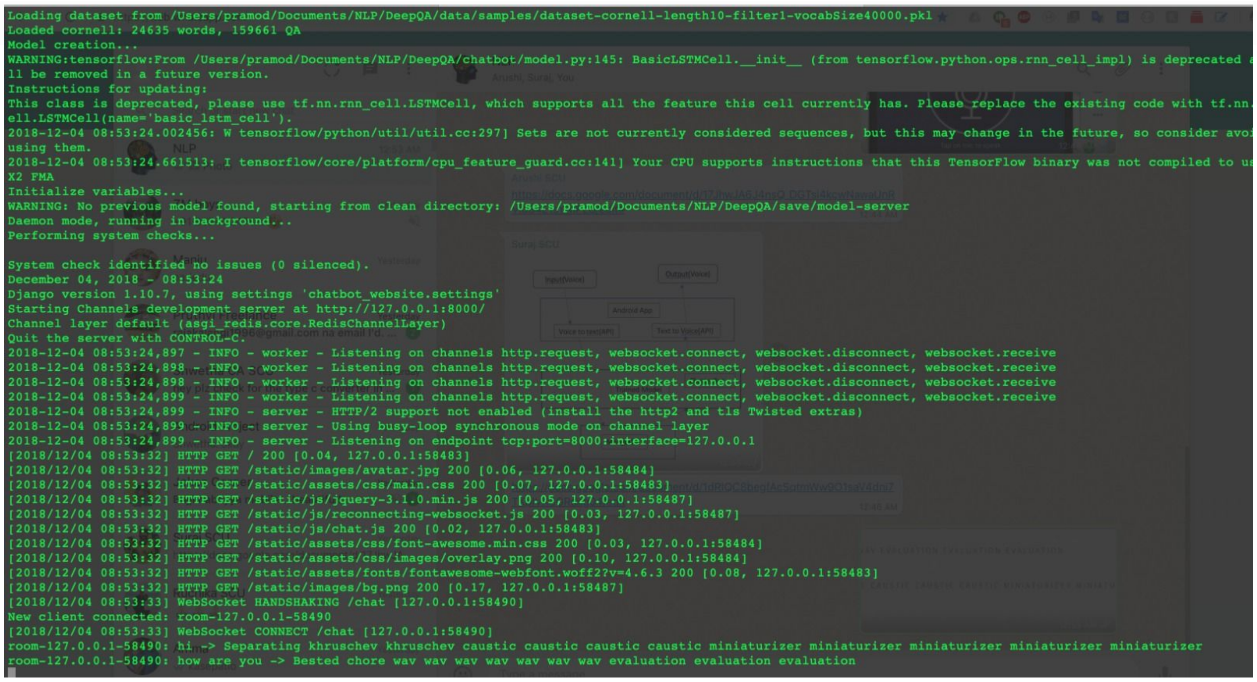
Chatbot status on Server side:



```
Pramod:~/Documents/NLP/ChatterBot$ python3 chatbotcornel.py
 * Serving Flask app "chatbotcornel" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:9999/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 475-945-834
hi
127.0.0.1 - - [04/Dec/2018 00:36:14] "GET /chatbot/hi/ HTTP/1.1" 200 -
127.0.0.1 - - [04/Dec/2018 00:36:15] "GET /favicon.ico HTTP/1.1" 404 -
172.20.117.245 - - [04/Dec/2018 00:39:33] "GET /chatbot/hi HTTP/1.1" 301 -
hi
172.20.117.245 - - [04/Dec/2018 00:39:34] "GET /chatbot/hi/ HTTP/1.1" 200 -
172.20.117.245 - - [04/Dec/2018 00:39:39] "GET /chatbot/are%20you%20okay HTTP/1.1" 301 -
are you okay
172.20.117.245 - - [04/Dec/2018 00:39:40] "GET /chatbot/are%20you%20okay/ HTTP/1.1" 200 -
172.20.117.245 - - [04/Dec/2018 00:39:46] "GET /chatbot/are%20you%20crazy HTTP/1.1" 301 -
are you crazy
172.20.117.245 - - [04/Dec/2018 00:39:47] "GET /chatbot/are%20you%20crazy/ HTTP/1.1" 200 -
172.20.117.245 - - [04/Dec/2018 00:39:57] "GET /chatbot/bail%20money HTTP/1.1" 301 -
bail money
172.20.117.245 - - [04/Dec/2018 00:39:58] "GET /chatbot/bail%20money/ HTTP/1.1" 200 -
172.20.117.245 - - [04/Dec/2018 00:40:22] "GET /chatbot/you%20are%20very%20bad HTTP/1.1" 301 -
you are very bad
172.20.117.245 - - [04/Dec/2018 00:40:24] "GET /chatbot/you%20are%20very%20bad/ HTTP/1.1" 200 -
172.20.117.245 - - [04/Dec/2018 00:40:30] "GET /chatbot/no%20I'm%20not HTTP/1.1" 301 -
no I'm not
172.20.117.245 - - [04/Dec/2018 00:40:31] "GET /chatbot/no%20I%27m%20not/ HTTP/1.1" 200 -
172.20.117.245 - - [04/Dec/2018 00:40:41] "GET /chatbot/what's%20the%20weather%20outside HTTP/1.1" 301 -
what's the weather outside
172.20.117.245 - - [04/Dec/2018 00:40:42] "GET /chatbot/what%27s%20the%20weather%20outside/ HTTP/1.1" 200 -
172.20.117.245 - - [04/Dec/2018 00:40:51] "GET /chatbot/don't%20say%20that HTTP/1.1" 301 -
don't say that
172.20.117.245 - - [04/Dec/2018 00:40:52] "GET /chatbot/don%27t%20say%20that/ HTTP/1.1" 200 -
172.20.117.245 - - [04/Dec/2018 00:41:01] "GET /chatbot/hi HTTP/1.1" 301 -
hi
172.20.117.245 - - [04/Dec/2018 00:41:01] "GET /chatbot/hi/ HTTP/1.1" 200 -
```

Figure 8: Chatbot server status

Chatbot result using LSTM:



```
Loading dataset from /Users/pramod/Documents/NLP/DeepQA/data/samples/dataset-cornell-length10-filter1-vocabSize40000.pkl
Loaded cornell: 24635 words, 159661 QA
Model creation...
WARNING:tensorflow:From /Users/pramod/Documents/NLP/DeepQA/chatbot/model.py:145: BasicLSTMCell.__init__ (from tensorflow.python.ops.rnn_cell_impl) is deprecated a
ll be removed in a future version.
Instructions for updating:
This class is deprecated, please use tf.nn.rnn_cell.LSTMCell, which supports all the feature this cell currently has. Please replace the existing code with tf.nn.
ell.LSTMCell(name='basic_lstm_cell').
2018-12-04 08:53:24.002456: W tensorflow/python/util/util.cc:297] Sets are not currently considered sequences, but this may change in the future, so consider avoi
using them.
2018-12-04 08:53:24.661513: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to us
X2 FMA
Initialize variables...
WARNING: No previous model found, starting from clean directory: /Users/pramod/Documents/NLP/DeepQA/save/model-server
Daemon mode, running in background...
Performing system checks...

System check identified no issues (0 silenced).
December 04, 2018 - 08:53:24
Django version 1.10.7, using settings 'chatbot_website.settings'
Starting Channels development server at http://127.0.0.1:8000/
Channel layer default (asgi_redis.core.RedisChannelLayer)
Quit the server with CONTROL-C.
2018-12-04 08:53:24,897 - INFO - worker - Listening on channels http.request, websocket.connect, websocket.disconnect, websocket.receive
2018-12-04 08:53:24,898 - INFO - worker - Listening on channels http.request, websocket.connect, websocket.disconnect, websocket.receive
2018-12-04 08:53:24,898 - INFO - worker - Listening on channels http.request, websocket.connect, websocket.disconnect, websocket.receive
2018-12-04 08:53:24,899 - INFO - worker - Listening on channels http.request, websocket.connect, websocket.disconnect, websocket.receive
2018-12-04 08:53:24,899 - INFO - server - HTTP/2 support not enabled (install the http2 and tls Twisted extras)
2018-12-04 08:53:24,899 - INFO - server - Using busy-loop synchronous mode on channel layer
2018-12-04 08:53:24,899 - INFO - server - Listening on endpoint tcp:port=8000:interface=127.0.0.1
[2018/12/04 08:53:32] HTTP GET / 200 [0.04, 127.0.0.1:58483]
[2018/12/04 08:53:32] HTTP GET /static/images/avatar.jpg 200 [0.06, 127.0.0.1:58484]
[2018/12/04 08:53:32] HTTP GET /static/assets/css/main.css 200 [0.07, 127.0.0.1:58483]
[2018/12/04 08:53:32] HTTP GET /static/js/jquery-3.1.0.min.js 200 [0.05, 127.0.0.1:58487]
[2018/12/04 08:53:32] HTTP GET /static/js/reconnecting-websocket.js 200 [0.03, 127.0.0.1:58487]
[2018/12/04 08:53:32] HTTP GET /static/js/chat.js 200 [0.02, 127.0.0.1:58483]
[2018/12/04 08:53:32] HTTP GET /static/assets/css/font-awesome.min.css 200 [0.03, 127.0.0.1:58484]
[2018/12/04 08:53:32] HTTP GET /static/assets/css/images/overlay.png 200 [0.10, 127.0.0.1:58484]
[2018/12/04 08:53:32] HTTP GET /static/assets/fonts/fontawesome-webfont.woff2?v=4.6.3 200 [0.08, 127.0.0.1:58483]
[2018/12/04 08:53:32] HTTP GET /static/images/bg.png 200 [0.17, 127.0.0.1:58487]
[2018/12/04 08:53:33] WebSocket HANDSHAKING /chat [127.0.0.1:58490]
New client connected: room-127.0.0.1-58490
[2018/12/04 08:53:33] WebSocket CONNECT /chat [127.0.0.1:58490]
room-127.0.0.1-58490: hi -> Separating khruschev khruschev caustic caustic caustic caustic miniaturizer miniaturizer miniaturizer miniaturizer miniaturizer
room-127.0.0.1-58490: how are you -> Bested chore wav wav wav wav wav wav wav evaluation evaluation evaluation
```

Figure 9: Chatbot Result using LSTM

6.3 Compare output against hypothesis

Hypothesis: Our goal is to take real time audio files and convert them to text and create a chatbot to mimic a human conversation using an ensemble of data mining algorithms.

We have taken a real time audio file, converted it to text and passed the input to the chatbot. The response generated by the chatbot is given to the user as an audio file for ease of conversation. The scope of responses and the length of the response to have a meaningful conversation is still limited.

6.4 Abnormal case explanation

With less training samples, the output generated was out of context and did not align with the users input. Training the chatbot on a larger dataset improved the accuracy but it was a very small improvement. The accuracy of the model was seen to increase as the type of data and source of data changed.

7. Conclusions and recommendations

7.1 Summary and conclusions

In this project, we have taken multiple datasets, preprocessed the data using different natural language preprocessing techniques like tokenization, stemming, bag of words. Given an input by the user, the chatbot selects one of the closest matching responses by searching for the closest matching statements using Jaccard similarity and uses Naive Bayes classification algorithm to check if the input meets a set of criteria to get a valid response.

7.2 Recommendations for future studies

This project can be further enhanced by training the model with larger data sources and data sources from different areas. This will help to have a more human like conversation.

9. Bibliography

[1] Weisz, Gellért, et al. "Sample efficient deep reinforcement learning for dialogue systems with large action spaces." *arXiv preprint arXiv:1802.03753* (2018).

[2] Varghese, Enza, and MT Rajappan Pillai. "A Standalone Generative Conversational Interface Using Deep Learning." *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. IEEE, 2018.

[3] Mutiwokuziva, Milla T., et al. "A neural-network based chat bot." *Communication and Electronics Systems (ICCES), 2017 2nd International Conference on*. IEEE, 2017.

[4] Su, Ming-Hsiang, et al. "A chatbot using LSTM-based multi-layer embedding for elderly care." *Orange Technologies (ICOT), 2017 International Conference on*. IEEE, 2017.

[5] Ochs, Magalie, Catherine Pelachaud, and Gary Mckeown. "A User Perception--Based Approach to Create Smiling Embodied Conversational Agents." *ACM Transactions on Interactive Intelligent Systems (TiiS)* 7.1 (2017): 4.

[6] Li, Jiwei, et al. "Deep reinforcement learning for dialogue generation." *arXiv preprint arXiv:1606.01541* (2016).

[7] Nguyen, Huyen, David Morales, and Tessera Chin. *A Neural Chatbot with Personality*. Stanford University working paper, 2017.

[8] Setiaji, Bayu, and Ferry Wahyu Wibowo. "Chatbot using a knowledge in database." *7th International Conference on Intelligent Systems, Modelling and Simulation*. 2016.

10. Appendices

10.1 Program source code with documentation

1)Android code for user interface:

```
package com.pramod.voicebot;


import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Locale;


import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.speech.RecognizerIntent;
import android.speech.tts.TextToSpeech;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;

import com.pramod.voicebot.R;

public class MainActivity extends Activity {

   private TextView txtSpeechInput;
   private TextView outputText;
   private ImageButton btnSpeak;
   private final int REQ_CODE_SPEECH_INPUT = 100;
   private TextToSpeech textToSpeech;



   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      txtSpeechInput = (TextView) findViewById(R.id.txtSpeechInput);
      btnSpeak = (ImageButton) findViewById(R.id.btnSpeak);
```

```
        outputText = findViewById(R.id.outputText);
        // hide the action bar
        //getActionBar().hide();

        btnSpeak.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                promptSpeechInput();
            }
        });

            textToSpeech = new TextToSpeech(getApplicationContext(), new
TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int status) {
                if (status == TextToSpeech.SUCCESS) {
                    int ttsLang = textToSpeech.setLanguage(Locale.US);

                    if (ttsLang == TextToSpeech.LANG_MISSING_DATA
                            || ttsLang == TextToSpeech.LANG_NOT_SUPPORTED) {
                        Log.e("TTS", "The Language is not supported!");
                    } else {
                        Log.i("TTS", "Language Supported.");
                    }
                    Log.i("TTS", "Initialization success.");
                } else {
                            Toast.makeText(getApplicationContext(), "TTS Initialization
failed!", Toast.LENGTH_SHORT).show();
                }
            }
        });

      }

      /**
       * Showing google speech input dialog
       */
      private void promptSpeechInput() {
                                            Intent      intent      =      new
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
                                  intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
Locale.getDefault());
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
                getString(R.string.speech_prompt));
        try {
            startActivityForResult(intent, REQ_CODE_SPEECH_INPUT);
        } catch (ActivityNotFoundException a) {
            Toast.makeText(getApplicationContext(),
                    getString(R.string.speech_not_supported),
                    Toast.LENGTH_SHORT).show();
        }
    }

    /**
     * Receiving speech input
     */
    @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        super.onActivityResult(requestCode, resultCode, data);

        switch (requestCode) {
            case REQ_CODE_SPEECH_INPUT: {
                if (resultCode == RESULT_OK && null != data) {

                    ArrayList<String> result = data
                            .getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                    String text = result.get(0);
                    txtSpeechInput.setText(text);
                    callHttpRequest(text);
                }
                break;
            }

        }
    }

    String ip = "172.20.117.245";
    String url = "http://" + ip + ":9999/chatbot/";

    void callHttpRequest(String text) {
        String fullUrl = url + text;
        new HttpTask().execute(fullUrl);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the main; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }



    class HttpTask extends AsyncTask<String, Integer, String> {



        @Override
        protected String doInBackground(String... urls) {
                                            URL url = null; // this is url
    http://api.football-data.org/v1/competitions
            try {
                url = new URL(urls[0]);
                            HttpURLConnection urlConnection = (HttpURLConnection)
    url.openConnection();
                try {
                            BufferedReader bufferedReader = new BufferedReader(new
    InputStreamReader(urlConnection.getInputStream()));
                    StringBuilder stringBuilder = new StringBuilder();
                    String line;
                    while ((line = bufferedReader.readLine()) != null) {
                        stringBuilder.append(line).append("\n");
                    }
                    bufferedReader.close();
                    return stringBuilder.toString();
                } finally {
                    urlConnection.disconnect();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

            return null;
        }

        @Override
        protected void onPostExecute(String output) {
            outputText.setText(output);

            int speechStatus = 0;
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                                                    speechStatus   =   textToSpeech.speak(output,
TextToSpeech.QUEUE_FLUSH, null, null);
            } else {
                                                    speechStatus   =   textToSpeech.speak(output,
TextToSpeech.QUEUE_FLUSH, null);
            }
            if (speechStatus == TextToSpeech.ERROR) {
                Log.e("TTS", "Error in converting Text to Speech!");
            }
        }
    }

    public void onDestroy() {
        super.onDestroy();
        if (textToSpeech != null) {
            textToSpeech.stop();
            textToSpeech.shutdown();
        }
    }

}
```

2) Training and Pre-processing code for Cornell Movie Dataset:

```
from chatterbot import ChatBot
from chatterbot.trainers import ChatterBotCorpusTrainer
from flask import Flask, render_template
from chatterbot.conversation import Statement
from chatterbot.trainers import Trainer
from chatterbot import utils
from flask import Flask, render_template

app = Flask(__name__)

chatbot = ChatBot('PramodBot',
                storage_adapter='chatterbot.storage.MongoDatabaseAdapter',
                database_uri='mongodb://localhost/voicebot',
                database='localbot',
                )


@app.route("/chatbot/<string:query>/")
```

```
def index(query):
    print(query)
    reply = chatbot.get_response(query)
    return str(reply)



class CornellMovieDatabaseTrainer(Trainer):

    def preprocess(self, path):
        with open(path + 'xaa', 'r', encoding="ISO-8859-1") as file:
            core_move_data = file.readlines()
        cornel_data = []
        for line in core_move_data:
            line_list = line.split("\n")[0].split("+++$+++")
            cornel_data.append(line_list[3:])

        result = []
        last_point = ''
        for i in range(0, len(cornel_data)):
            if last_point == cornel_data[i][0]:
                result[-1] += " " + cornel_data[i][1]
            else:
                result.append(cornel_data[i][1])
                last_point = cornel_data[i][0]
        return result

    def train(self, datasetpath):
        """
        Train the chat bot based on the provided list of
        statements that represents a single conversation.
        """
        dataset = self.preprocess(datasetpath)
        previous_statement_text = None

        for conversation_count, text in enumerate(dataset):
            if self.show_training_progress:
                utils.print_progress_bar(
                    'CornelMovieDatabase Trainer',
                    conversation_count + 1, len(dataset)
                )

            statement = self.get_preprocessed_statement(
                Statement(
                    text=text,
                    in_response_to=previous_statement_text,
                    conversation='training'
                )
            )
```

```
            previous_statement_text = statement.text

            self.chatbot.storage.create(
                text=statement.text,
                in_response_to=statement.in_response_to,
                conversation=statement.conversation,
                tags=statement.tags
            )


# trainer = CornellMovieDatabaseTrainer(chatbot)

# trainer.train('/Users/pramod/Documents/NLP/dataset/cornel/')

# chat = ChatBot('ChatBotCorpus',
#               storage_adapter='chatterbot.storage.MongoDatabaseAdapter',
#               database_uri='mongodb://localhost/voicebot',
#               database='voicebot',
#               )
corpustrainer = ChatterBotCorpusTrainer(chatbot)

if __name__ == '__main__':
    app.debug = True
    app.run(host='0.0.0.0', port=9999)
```

```
# Train the chatbot based on the english corpus
# corpustrainer.train("chatterbot.corpus.english")

# Get a response to an input statement
# resp = chatbot.get_response("Hello, how are you today?")

# while True:
#     message = input('You:')
#     if message.strip() != 'Bye':
#         reply = chatbot.get_response(message)
#         print('ChatBot :', reply)
#     if message.strip == 'Bye':
#         print('ChatBot : Bye')
#         break
```

3) Training Twitter Dataset on Chatbot model using LSTM:

```
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer


TWITTER = {
   "CONSUMER_KEY": "sohYJD3hOgAVRMemXiY0dS6Z4",
                                                    "CONSUMER_SECRET":
"8tIdlylU2x0S9LpXcQKEJEndesN7lu0e8SqeO0Gfj5t16WsCon",
                                                       "ACCESS_TOKEN":
"219348556-dO5Tu28dmltK6fOMTIERGBr2b5KTGGr6KpCJ6bDE",
                                                     "ACCESS_TOKEN_SECRET":
"SoPpEzqRwX86Y7ntWomEEfnMXXcCHafJSHTOt3bRBVcoB"
}

chatbot = ChatBot(
   "TwitterBot",
   logic_adapters=[
     "chatterbot.logic.BestMatch"
   ],
   input_adapter="chatterbot.input.TerminalAdapter",
   output_adapter="chatterbot.output.TerminalAdapter",
   database="./twitter-database.db",
   twitter_consumer_key=TWITTER["CONSUMER_KEY"],
   twitter_consumer_secret=TWITTER["CONSUMER_SECRET"],
   twitter_access_token_key=TWITTER["ACCESS_TOKEN"],
   twitter_access_token_secret=TWITTER["ACCESS_TOKEN_SECRET"],
)

cbot = ChatBot('ListBot',
        storage_adapter='chatterbot.storage.MongoDatabaseAdapter',
        database_uri='mongodb://localhost/pmd_chatterbot',
        database='pmd_chatterbot',
        )

trainer = ListTrainer(cbot)
```

```
trainer.train([
    "Hi there!",
    "Hello",
])

trainer.train([
    "Greetings!",
    "Hello",
])

#trainer.train()

chatbot.logger.info('Trained database generated successfully!')

while True:
    message = input('You:')
    if message.strip() != 'Bye':
        reply = cbot.get_response(message)
        print('ChatBot :', reply)
    if message.strip == 'Bye':
        print('ChatBot : Bye')
        break
```

10.2 Input/output listing

The input will be the user input for interacting with the chatbot using audio. The output will be the response generated by the chatbot for that particular user input based on the datasets it has been trained with.

README:

Install chatterbot:

pip intall chatterbot

Install chatterbot_corpus dataset.

Clone the dataset:

From cornel database:

https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

To Train the model:

Put the files in a directory:

For training cornel database:

Run: python3 chatbotcornel.py

For training Chatterbot corpus dataset

Run: python3 chatbotcornel.py with uncommented lines for train methods

This will start a server on a port 9999

Input the machine server ip in the Android app and start the app in the android studio

Start speaking with the app and see the output printing and narrating on the screen.