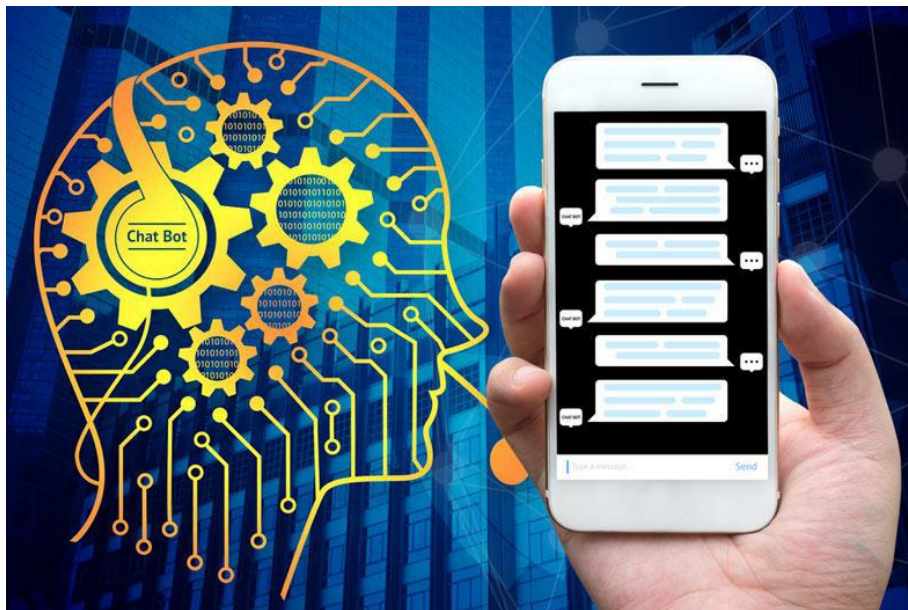


MTECH PROJECT REPORT

ISS CHAT BOT



COGNITIVE SYSTEMS project report submitted in part
fulfilment of the degree of **M-Tech Intelligent Systems**

TEAM MEMBERS

YONG QUAN ZHI, TOMMY A0195353Y
SANTHIYAPILLAI RAJEEVAN PAUL A0195399E
RITESH MUNJAL A0195304H
GARY NG JIAN ZHE A0195367N
ANI SIVARAM PORKALANGAD AYYAPPAN A0195404E
SOOREJ MOHANADAS GANGA A0195397J

May 31, 2019

Table of Contents

Abstract	3
Chapter 1: Introduction.....	4
Chapter 2: Project Lifecycle	5
Chapter 3: System Architecture	7
Chapter 4: Dialogflow	8
Chapter 5: Python NLTK	11
Chapter 6: NGROK	23
Chapter 7: AWS Lightsail	24
Chapter 8: Conclusion.....	25

Abstract

The power of chatbots is not lost on the business world. As brands focus on promoting personalized experiences, more and more intelligent chatbots are being built to engage users and improve brand image. That said, it is a rarity to find a live intelligent chatbot, also called as AI chatbot. As the thought of a chatbot springs up, we know it is not a real person for sure. What we know is that these chatbots bring a human touch. For that to become a reality, they need to be really intelligent. The crux is not the chatbot, rather it is the intelligence quotient that bring the human touch.

It is the intelligence that gives power to the AI chatbot to learn from conversations and handle any and every situation that comes its way. As chatbots move into complex territories, raising the intelligence quotient becomes increasingly difficult.

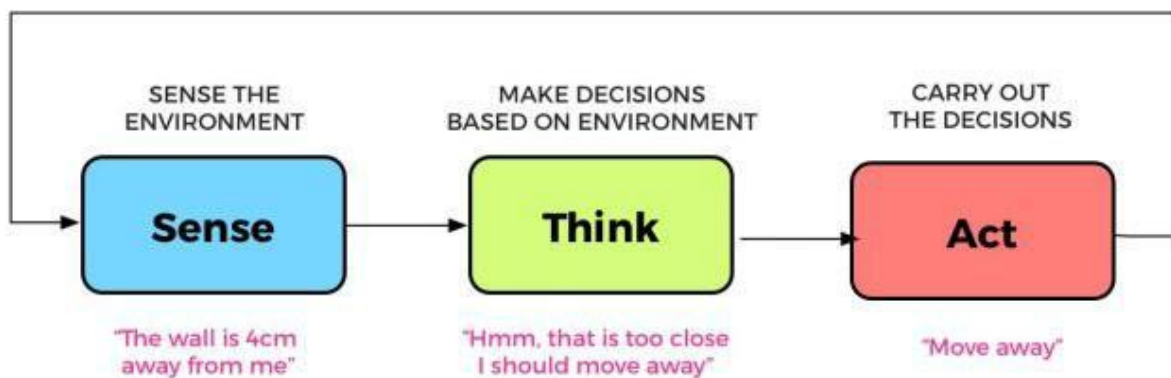
Today, several of successful chatbots including Google assistant have been built on intelligent platforms. In this scenario, the platform becomes the intelligent agent, and the interface becomes a sensor for this intelligent agent. The intelligent platform works to find out the goal, collect information, process, store and convert information to realize the goal. Then the challenge is not about infusing intelligence into a chatbot but creating an intelligent platform. With this in mind, our focus was on ways to define the goal and factor sense-think-act capability into the platform.

We leveraged on Dialogflow and Python NLTK to build an intelligent chatbot for NUS ISS, which has been deployed into production at <http://www.isschatbot.ga>

Our sincere thanks to lecturers at NUS-ISS, who has guided us on the working of Dialogflow and how it could interface with Python to pull data intelligently from scrapped website contents.

Chapter 1: Introduction

A Chatbot is a program that generates response based on given input to emulate human conversations in text or voice mode. These applications are designed to simulate human-human interactions. Chatbots are predominantly used in business and corporate organizations including government, non-profit and private. Their functioning can range from customer service, product suggestion, product inquiry to personal assistance. Many of these chat agents are built using rule based techniques, retrieval techniques or simple machine learning algorithms. In retrieval based techniques, chat agents scan for keywords within the input phrase and retrieves relevant answers based on the query string. They rely on keyword similarity and retrieved text is pulled from internal or external data sources including world wide web or organizational database. Some other advanced chatbots are developed with natural language processing (NLP) techniques and machine learning algorithms. Also, there are many commercial chat engines available, which help build chatbots based on client data input.



Recently, there has been major increase of interest in use and deployment of dialogue generation systems. Many major tech companies are using virtual assistant or chat agent to fill the needs of customers. Some of them include Google Assistant, Microsoft Cortana and Amazon Alexa. Though they are primarily question answering systems, their adoption by major corporations has peaked interest in customers and seems promising for more advanced conversational agent system research and development.

For our Cognitive Systems Group project, our team has been tasked to build an intelligent chatbot for NUS-ISS website. This chat bot has to receive all queries from visitors, and provide intelligent responses which simulate a human feel.

Chapter 2: Project Lifecycle

The first phase was **PROJECT INITIATION** where all pre-project work was done. This involved our team defining the opportunity, discussions on the merits of project options, and getting the consensus to move forward.



Figure 1: Project Life Cycle

After careful consideration several factors such as:

- Automatic Speech Recognition
- Speech Synthesis
- Natural Language Comprehension & Processing
- Rule based NER systems
- Natural Language Generation
- Availability of resources for developing front-end, back-end and integration
- Complexity & Feasibility of implementation within the stipulated timeline

The team had decided to proceed with the Dialogflow + Python NLTK combination for project implementation.

In the **PROJECT PLANNING** phase, our team sat down for discussions on requirement analysis.

For the requirement analysis, the key tasks include:

- Break down all the work that needs to be done into tasks
- Estimate the time, resources and budget required to complete each task, and
- Define acceptance criteria

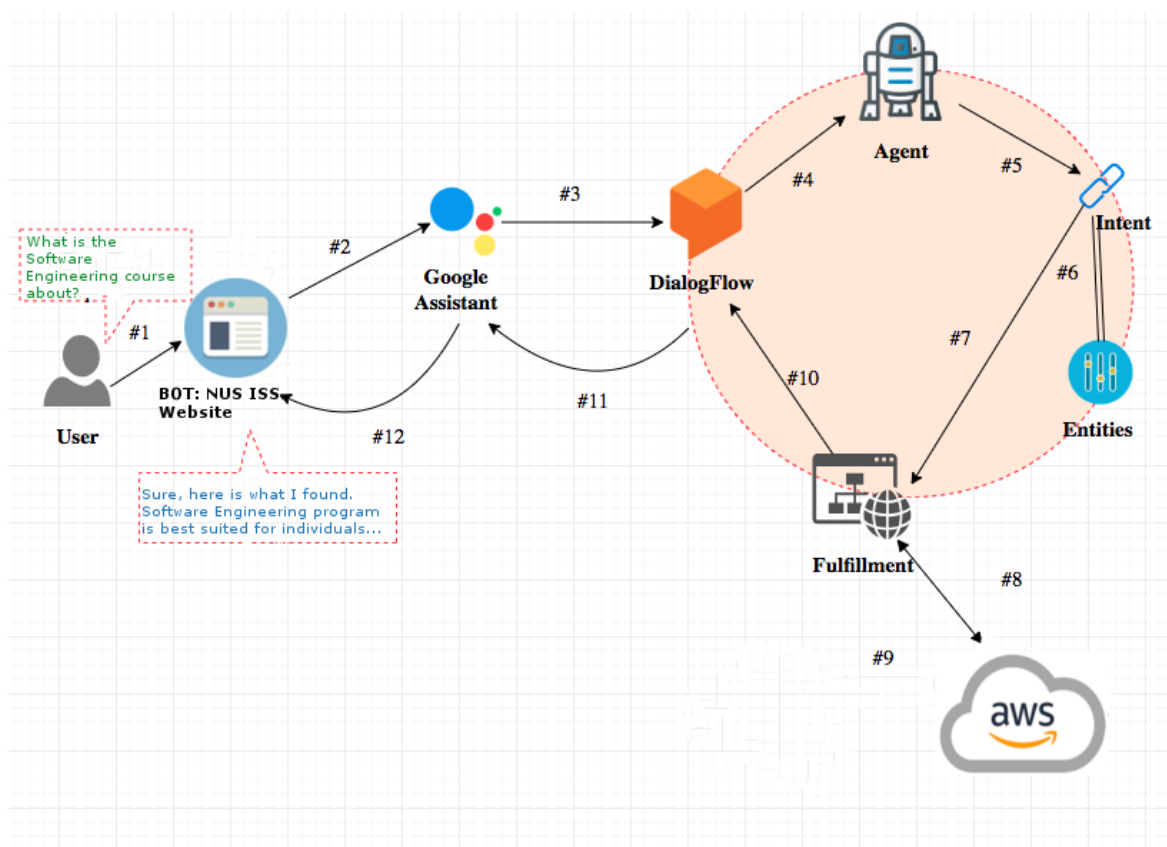
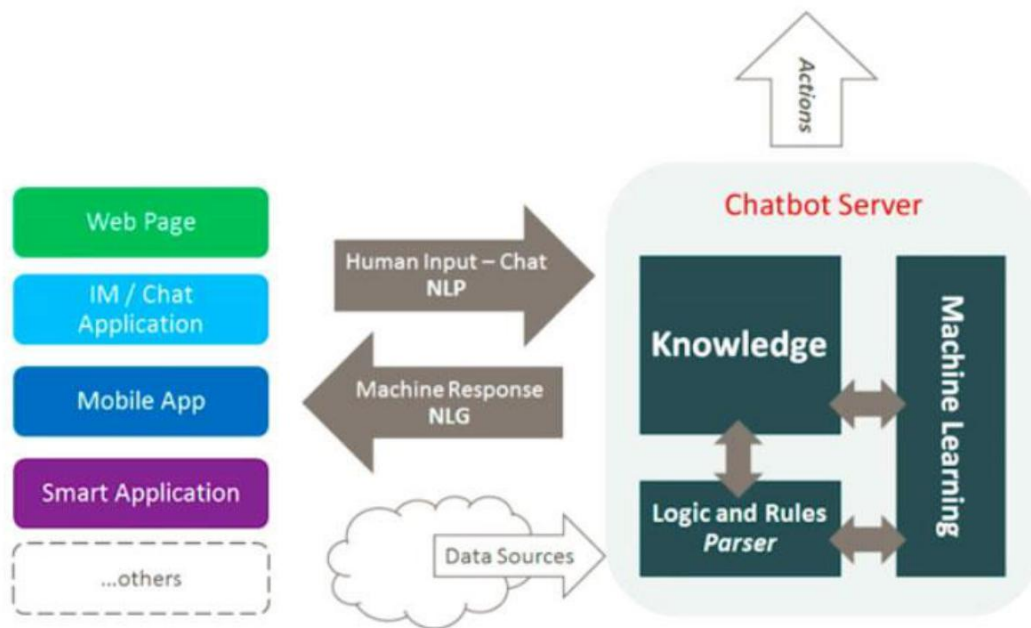
This was followed by the Project implementation and testing phases. The following table shows a summary of our efforts for developing chatbot for NUS ISS.

Deliverables	Owner
System Design & System Architecture	Gary, Tommy, Ritesh, Paul
Frontend Code development	Ani, Paul
Dialogflow development works	Ritesh, Ani
Dialogflow training questions	Ritesh, Soorej
Python Code development	Gary
Bug fixing	Tommy, Paul, Gary, Ritesh, Ani
Integration of front-end & back-end	Ani, Gary, Ritesh
AWS hosting, DNS & Webhook	Ani, Paul
Web scrapping for data	Tommy
Testing	Gary, Ritesh, Ani
Project report	Ani, Soorej
Project Video	Ritesh
Deployment (Github & Luminus)	Ani, Gary

Table 1: Requirement Analysis

Chapter 3: System Architecture

A high-level architecture is shown in the following diagram:

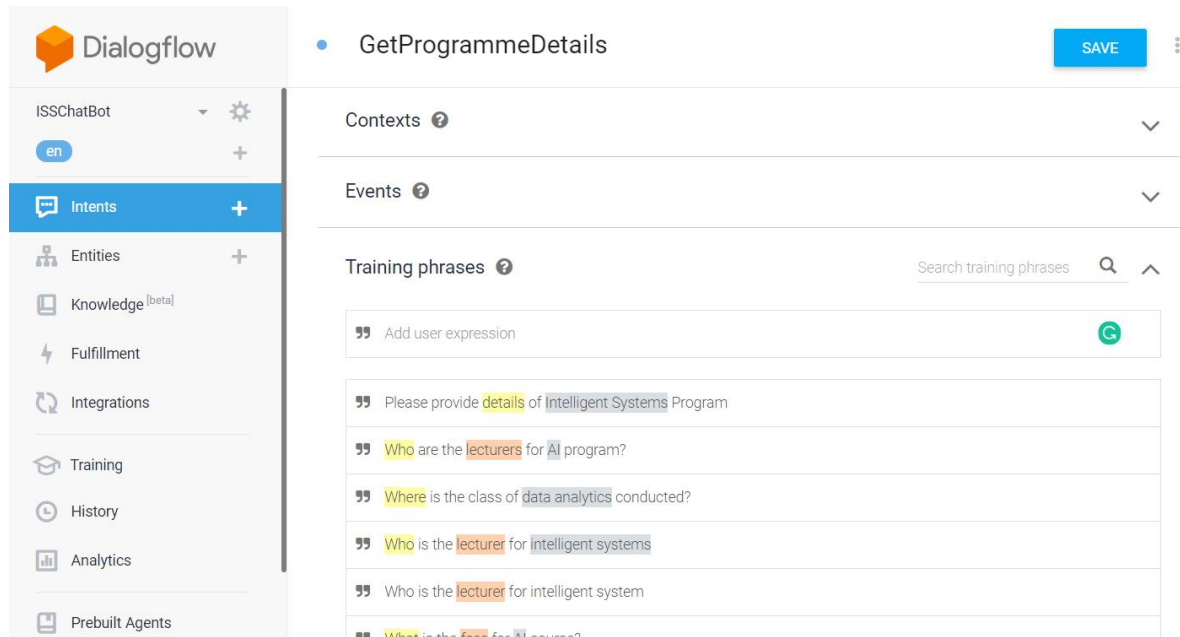


In this project, we have leveraged on Google's Dialogflow as its a great platform for developing chatbots for multiple platforms.

Among all the services taken into consideration, Dialogflow is certainly one of the most impressive. The power of Google's machine learning makes the difference: the natural language processing (NLP) engine is among the best on the market. As indeed its slogan recites: "Dialogflow is user-friendly, intuitive and makes sense". It is also very easy to integrate with Google Cloud Speech-to-Text and third-party services such as Google Assistant, Amazon Alexa, and Facebook Messenger.

Chapter 4: Dialogflow

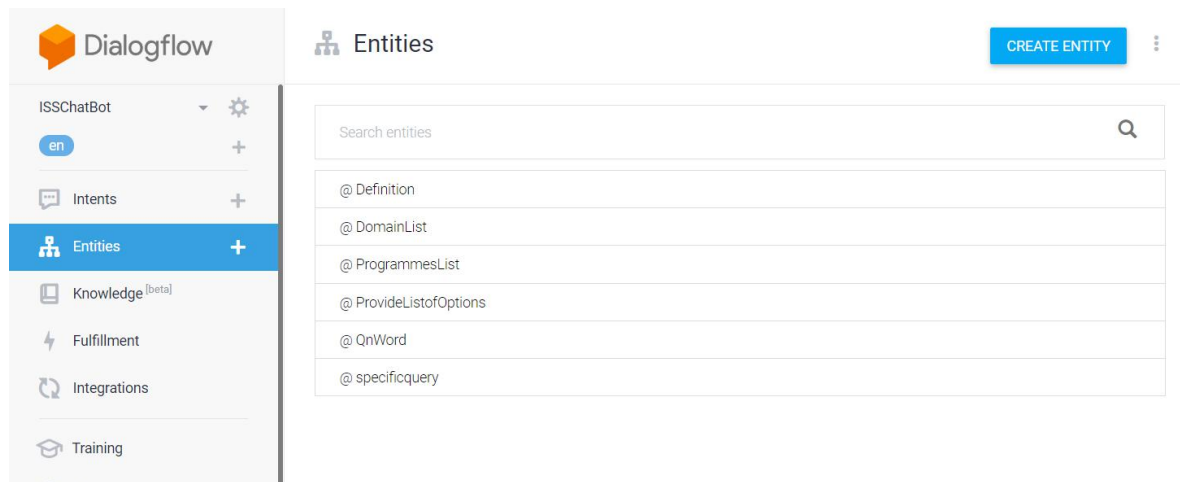
Dialogflow is based on two main concepts: intent and context. The intent is to accurately identify the purpose of the sentence that the user has sent to the bot. On the other side, the context is used to give coherence and fluency to the discussion, preserving the key concepts that have already been used in the conversation.



The screenshot shows the Dialogflow console interface. On the left is a sidebar with navigation options: ISSChatBot, Intents, Entities, Knowledge [beta], Fulfillment, Integrations, Training, History, Analytics, and Prebuilt Agents. The 'Intents' section is selected. The main area displays the configuration for the 'GetProgrammeDetails' intent. It includes sections for Contexts, Events, and Training phrases. The Training phrases section contains a list of example utterances with their corresponding entities highlighted in color (e.g., 'Please provide details of Intelligent Systems Program', 'Who are the lecturers for AI program?', 'Where is the class of data analytics conducted?', 'Who is the lecturer for intelligent systems', 'Who is the lecturer for intelligent system', 'Where is the class for AI course?'). A 'SAVE' button is visible in the top right corner.

Intent creation is the central part of the Dialogflow chatbot logics.

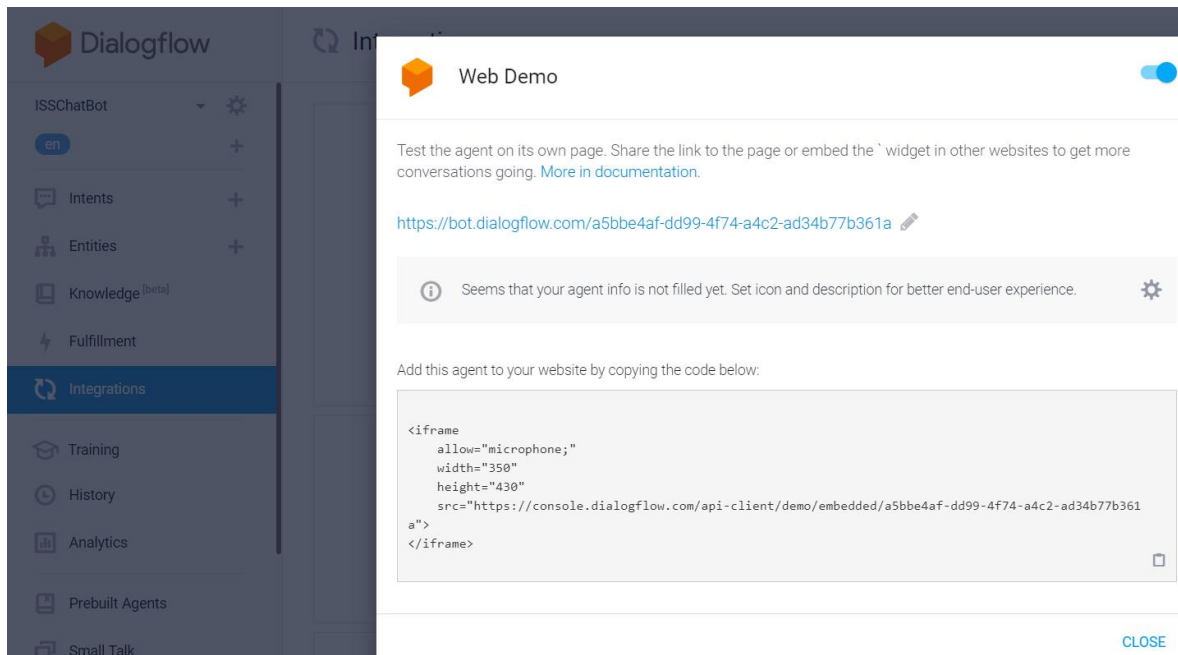
A further key concept is that of the entities, which allow us to identify common or parametrically recurring concepts in the discussion. There are a number of predefined types of entities that the system provides ready to use, such as dates or colors. Using these values we could make the chatbot very versatile.



The screenshot shows the Dialogflow console interface for the 'Entities' section. The sidebar on the left is the same as in the previous screenshot, but 'Entities' is now selected. The main area shows the 'Entities' configuration page with a 'CREATE ENTITY' button in the top right. Below the button is a search bar labeled 'Search entities'. A list of predefined entity types is displayed, including: @ Definition, @ DomainList, @ ProgrammesList, @ ProvideListOfOptions, @ QnWord, and @ specificquery.

Now that the Dialogflow has been configured, next step is to integrate it into the website.

The easiest way to integrate Dialogflow into an HTML page is to use the iframe. Select “Integrations” from the menu on the left and make sure that “Web Demo” is enabled. Simply copy and paste the HTML code to view the agent directly on your site.



Now that everything about the agent is set up , it’s time to train it. There are two modes to train the model:

1. Hybrid mode
2. ML only mode

Hybrid mode is good when you have less training samples. If you have a large set, you should go for the ML mode

ML Classification Threshold is the minimum confidence score that the model should have. Only then can it take an action on an intent, otherwise it will fall back to the “Default Intent”.

A Webhook to process requests:

Dialogflow is configured to send information to Python code through Webhooks configured as follows :

⚡ Fulfillment

Webhook

ENABLED ☒

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*	<input type="text" value="https://c2950628.ngrok.io"/>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/>
HEADERS	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	+ Add header	
SMALL TALK	<input type="checkbox"/> Disable webhook for Smalltalk ▼	

Chapter 5: Python NLTK

The field of study that focuses on the interactions between human language and computers is called Natural Language Processing or NLP for short. It sits at the intersection of computer science, artificial intelligence, and computational linguistics. NLP is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation.

NLTK:

NLTK (Natural Language Toolkit) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

NLTK has been called “a wonderful tool for teaching and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

To install NLTK: run `pip install nltk`

Test installation: run `python` then type `import nltk`

The main issue with text data is that it is all in text format (strings). However, the Machine learning algorithms need some numerical feature vector in order to perform the task. So before we start with any NLP project we need to pre-process it to make it ideal for working. Basic **text pre-processing** includes:

Converting the entire text into **uppercase or lowercase**, so that the algorithm does not treat the same words in different cases as different **Tokenization**: Tokenization is just the term used to describe the process of converting the normal text strings into a list of tokens i.e words that we actually want. Sentence tokenizer can be used to find the list of sentences and Word tokenizer can be used to find the list of words in strings. Removing **Noise** i.e everything that isn't in a standard number or letter. Removing **Stop words**. Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called *stop words* **Stemming**: Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form. Example if we were to stem the following words: “Stems”, “Stemming”, “Stemmed”, “and Stemtization”, the result would be a single word “stem”.

Lemmatization: A slight variant of stemming is lemmatization. The major difference between these is that, stemming can often create non-existent words, whereas lemmas are actual words. So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma. Examples of Lemmatization are that “run” is a base form for words like “running” or “ran” or that the word “better” and “good” are in the same lemma so they are considered the same.

Bag of Words:

After the initial preprocessing phase, we need to transform text into a meaningful vector (or array) of numbers. The bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

Why is it called a “*bag*” of words? That is because any information about the order or structure of words in the document is discarded and the model is only concerned with **whether the known words occur in the document, not where they occur in the document.**

The intuition behind the Bag of Words is that documents are similar if they have similar content. Also, we can learn something about the meaning of the document from its content alone.

For example, if our dictionary contains the words {Learning, is, the, not, great}, and we want to vectorize the text “Learning is great”, we would have the following vector: (1, 1, 0, 0, 1).

TF-IDF Approach

A problem with the Bag of Words approach is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content”. Also, it will give more weight to longer documents than shorter documents.

One approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized. This approach to scoring is called **Term Frequency-Inverse Document Frequency**, or **TF-IDF** for short, where:

Term Frequency: is a scoring of the frequency of the word in the current document.

$$TF = (\text{Number of times term } t \text{ appears in a document}) / (\text{Number of terms in the document})$$

Inverse Document Frequency: is a scoring of how rare the word is across documents.

$IDF = 1 + \log(N/n)$, where, N is the number of documents and n is the number of documents a term t has appeared in.

Tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus

Example:

Consider a document containing 100 words wherein the word ‘phone’ appears 5 times.

The term frequency (i.e., tf) for phone is then $(5 / 100) = 0.05$. Now, assume we have 10 million documents and the word phone appears in one thousand of these. Then, the inverse document frequency (i.e., IDF) is calculated as $\log(10,000,000 / 1,000) = 4$. Thus, the Tf-IDF weight is the product of these quantities: $0.05 * 4 = 0.20$.

Tf-IDF can be implemented in scikit learn as:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Cosine Similarity

TF-IDF is a transformation applied to texts to get two real-valued vectors in vector space. We can then obtain the **Cosine** similarity of any pair of vectors by taking their dot product and dividing that by the product of their norms. That yields the cosine of the angle between the vectors. **Cosine similarity** is a measure of similarity between two non-zero vectors. Using this formula we can find out the similarity between any two documents d1 and d2.

$$\text{Cosine Similarity}(d1, d2) = \text{Dot product}(d1, d2) / \|d1\| * \|d2\|$$

where d1,d2 are two non zero vectors

Importing the necessary libraries

```
import nltk
import numpy as np
import random
import string # to process standard python strings
```

Corpus

For our project, we have used the following python script to scrape the entire NUS-ISS website and have categorized them into multiple text files.

```
import urllib.request

import os

from bs4 import BeautifulSoup

#Define functions
```

```
def processText(divs, type):

    final_text = ""

    divText = ""

    if type is "graduate":

        for div in divs:

            divText = div.text

            if divText is not "":

                # removes whitespace

                lines = (line.strip() for line in divText.splitlines())

                # merge into 1 line

                chunks = (phrase.strip() for line in lines for phrase
in line.split(" "))

                # remove blank lines

                divText = ' '.join(chunk for chunk in chunks if chunk)

                final_text += divText+'\n\n'

                divText=""

    elif type is "general":

        for idx, div in enumerate(divs):

            divText = div.text

            if idx > 1 and idx < (len(divs)-2):

                # removes whitespace

                lines = (line.strip() for line in divText.splitlines())

                # merge into 1 line

                chunks = (phrase.strip() for line in lines for phrase
in line.split(" "))

                # remove blank lines
```

```

        divText = ' '.join(chunk for chunk in chunks if chunk)

        divText.replace('\n', ' ')

        if divText is not "" and divText != "Print":

            final_text += divText+'\n\n'

            divText=""

    elif type is "executive":

        for idx, div in enumerate(divs):

            divText = div.text

            if idx > 9 and idx < (len(divs)-3):

                # removes whitespace

                lines = (line.strip() for line in divText.splitlines())

                # merge into 1 line

                chunks = (phrase.strip() for line in lines for phrase
in line.split(" "))

                # remove blank lines

                divText = ' '.join(chunk for chunk in chunks if chunk)

                divText.replace('\n', ' ')

                if divText is not "" and divText != "Print":

                    final_text += divText+'\n\n'

                    divText=""

    return final_text

def stripSoup(containsContact):

    #Strips away all scripts, styles and

    hyperlinks if containsContact:

        for script in soup(["script", "style"]):

```



```
        script.extract()

    else:

        for script in soup(["script", "style"]):

            script.extract()

def checkDirectories():

    #Check directories

    directory1 = "generalInfo"

    directory2 = "graduateProgrammes"

    directory3 = "executiveEducation"

    directory4 = "stackableProgrammes"

    if not os.path.exists(directory1):

        os.makedirs(directory1)

    if not os.path.exists(directory2):

        os.makedirs(directory2)

    if not os.path.exists(directory3):

        os.makedirs(directory3)

    if not os.path.exists(directory4):

        os.makedirs(directory4)

#Define variables

categories = ["General Information", "Graduate Programmes",
"Executive Education", "Stackable Programmes"]

directoryNames = ["generalInfo", "graduateProgrammes",
"executiveEducation", "stackableProgrammes"]

generalInfoURLs = ["https://www.iss.nus.edu.sg/about-us/director-
ceo's-welcome",
```

```
        "https://www.iss.nus.edu.sg/about-us/our-story",
        "https://www.iss.nus.edu.sg/about-us/our-
achievements",
        "https://www.iss.nus.edu.sg/about-us/our-management-
board",
        "https://www.iss.nus.edu.sg/about-us/why-NUS-ISS",
        "https://www.iss.nus.edu.sg/about-us/contact-us",
        "https://www.iss.nus.edu.sg/about-us/getting-to-nus-
iss"]

graduateProgrammeURLs = ["https://www.iss.nus.edu.sg/graduate-
programmes/programme/detail/graduate-diploma-in-systems-analysis",

        "https://www.iss.nus.edu.sg/graduate-
programmes/programme/detail/master-of-technology-in-enterprise-
business-analytics",

        "https://www.iss.nus.edu.sg/graduate-
programmes/programme/detail/master-of-technology-in-digital-leadership",

        "https://www.iss.nus.edu.sg/graduate-
programmes/programme/detail/master-of-technology-in-
software-engineering",

        "https://www.iss.nus.edu.sg/graduate-
programmes/programme/detail/master-of-technology-in-intelligent-systems"]

executiveEducationURLs = ["https://www.iss.nus.edu.sg/executive-
education/discipline/detail/artificial-intelligence",

        "https://www.iss.nus.edu.sg/executive-
education/discipline/detail/cybersecurity",

        "https://www.iss.nus.edu.sg/executive-
education/discipline/detail/data-science",

        "https://www.iss.nus.edu.sg/executive-
education/discipline/detail/digital-agility",

        "https://www.iss.nus.edu.sg/executive-
education/discipline/detail/digital-innovation-design",

        "https://www.iss.nus.edu.sg/executive-
education/discipline/detail/digital-strategy-leadership",
```

```
        "https://www.iss.nus.edu.sg/executive-
education/discipline/detail/digital-products-platforms",

    "https://www.iss.nus.edu.sg/collaboration/professional-
conversion-programmes",

        "https://www.iss.nus.edu.sg/executive-
education/skillsfuture-series",

        "https://www.iss.nus.edu.sg/professional-
diploma-in-smart-health-leadership",

        "https://www.iss.nus.edu.sg/executive-
education/discipline/detail/software-systems",

        "https://www.iss.nus.edu.sg/executive-
education/discipline/detail/stackup---startup-tech-talent-development"]

stackableProgrammesURLs = ["https://www.iss.nus.edu.sg/stackable-
certificate-programmes/business-analytics",

        "https://www.iss.nus.edu.sg/executive-
education/course/detail/nus-iss-certificate-in-digital-solutions-
development",

        "https://www.iss.nus.edu.sg/stackable-
certificate-programmes/Intelligent-systems",

        "https://www.iss.nus.edu.sg/stackable-
certificate-programmes/Software-engineering"]

container = ""

checkDirectories()

for idx, category in enumerate(categories):

    print("Processing: "+category)

    if category is "General Information":

        for url in generalInfoURLs:

            strSplit = url.split("/")
```

```
        fileName = strSplit[-1].lower()

        f = open(directoryNames[idx]+"/"+fileName+".txt",
"w", encoding='utf-8')

        html = urllib.request.urlopen(url).read()

        soup = BeautifulSoup(html, "lxml")

        containsContact = "contact" in url or "getting" in url

        stripSoup(containsContact)

        container = processText(soup.findAll("div",
{"class":"sfContentBlock"}), "general")

        f.write(container)

        f.close()

    elif category is "Graduate Programmes":

        for url in graduateProgrammeURLs:

            strSplit = url.split("/")

            fileName = strSplit[-1].lower()

            f = open(directoryNames[idx]+"/"+fileName+".txt",
"w", encoding='utf-8')

            html = urllib.request.urlopen(url).read()

            soup = BeautifulSoup(html, "lxml")

            stripSoup(False)

            container = processText(soup.findAll("div",
{"class":"main-content-entry with-break"}), "graduate")

            f.write(container)

            f.close()

    elif category is "Executive Education":
```

```
for url in executiveEducationURLs:

    strSplit = url.split("/")

    fileName = strSplit[-1].lower()

    f = open(directoryNames[idx]+"/"+fileName+".txt",
"w", encoding='utf-8')

    html = urllib.request.urlopen(url).read()

    soup = BeautifulSoup(html, "lxml")

    stripSoup(True)

    container = processText(soup.findAll("div", {"class":"row"}),
"executive")

    f.write(container)

    f.close()

elif category is "Stackable Programmes":

    for url in stackableProgrammesURLs:

        strSplit = url.split("/")

        fileName = strSplit[-1].lower()

        f = open(directoryNames[idx]+"/"+fileName+".txt",
"w", encoding='utf-8')

        html = urllib.request.urlopen(url).read()

        soup = BeautifulSoup(html, "lxml")

        stripSoup(True)

        container = processText(soup.findAll("div",
{"class":"content-wrap js-contentFontResize"}), "stackable")

        f.write(container)

        f.close()
```

Reading in the data

We will read in the txt files and convert the entire corpus into a list of sentences and a list of words for further pre-processing.

```
f=open('chatbot.txt','r',errors = 'ignore')
raw=f.read()
raw=raw.lower()# converts to lowercase
nltk.download('punkt') # first-time use only
nltk.download('wordnet') # first-time use only
sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences
word_tokens = nltk.word_tokenize(raw)# converts to list of words
```

Pre-processing the raw text

We shall now define a function called LemTokens which will take as input the tokens and return normalized tokens.

```
lemmer = nltk.stem.WordNetLemmatizer()
#WordNet is a semantically-oriented dictionary of English included
in NLTK.
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in
tokens] remove_punct_dict = dict((ord(punct), None) for
punct in string.punctuation)
def LemNormalize(text):
    return
LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict))
```

Keyword matching

Next, we shall define a function for a greeting by the bot i.e if a user's input is a greeting, the bot shall return a greeting response. ELIZA uses a simple keyword matching for greetings. We will utilize the same concept here.

```
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", "I am
glad! You are talking to me"]
def greeting(sentence):
```

```
    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)
```

Generating Response

To generate a response from our bot for input questions, the concept of document similarity will be used. So we begin by importing necessary modules.

From scikit learn library, import the Tfidf vectorizer to convert a collection of raw documents to a matrix of TF-IDF features.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Also, import cosine similarity module from scikit learn library

```
from sklearn.metrics.pairwise import cosine_similarity
```

This will be used to find the similarity between words entered by the user and the words in the corpus. This is the simplest possible implementation of a chatbot.

We define a function **response** which searches the user's utterance for one or more known keywords and returns one of several possible responses. If it doesn't find the input matching any of the keywords, it returns a response: "I am sorry! I don't understand you"

```
def response(user_response):
    robo_response=''
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize,
stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx=vals.argsort()[0][-2] flat =
    vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if(req_tfidf==0):
        robo_response=robo_response+"I am sorry! I don't understand you"
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response
```

Finally, we feed the lines that we want our bot to say while starting and ending a conversation depending upon user's input.

```
flag=True
print("ROBO: My name is Robo. I will answer your queries about Chatbots.
If you want to exit, type Bye!")
while(flag==True):
    user_response = input()
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            print("ROBO: You are welcome..")
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response))
            else:
                print("ROBO: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("ROBO: Bye! take care..")
```

Chapter 6: NGROK

Ngrok allows you to expose a web server running on your local machine to the internet. Just tell ngrok what port your web server is listening on.

When you start ngrok, it will display a UI in your terminal with the public URL of your tunnel and other status and metrics information about connections made over your tunnel.

```
ngrok by @inconshreveable

Tunnel Status      online
Version            2.0/2.0
Web Interface      http://127.0.0.1:4040
Forwarding         http://92832de0.ngrok.io -> localhost:80
Forwarding         https://92832de0.ngrok.io -> localhost:80

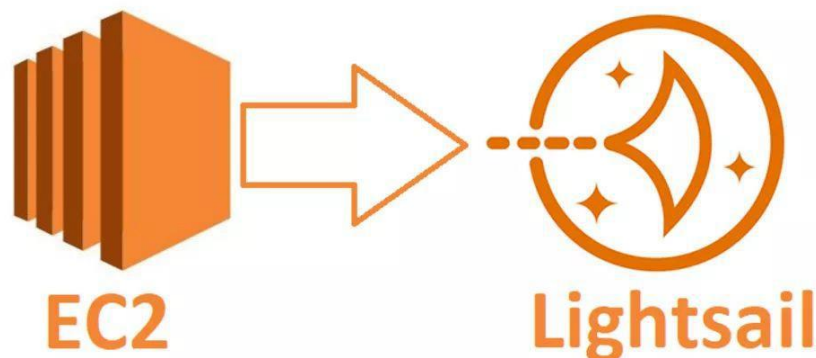
Connections      ttl    opn    rt1    rt5    p50    p90
                  0      0      0.00   0.00   0.00   0.00
```

ngrok provides a real-time web UI where you can introspect all of the HTTP traffic running over your tunnels. After you started ngrok, just open <http://localhost:4040> in a web browser to inspect request details.

Try making a request to your public URL. Then look back at the inspection UI. You will see all of the details of the request and response including the time, duration, headers, query parameters and request payload as well as the raw bytes on the wire.

Chapter 7: AWS Lightsail

Amazon Lightsail is the easiest way to get started with AWS for developers who just need virtual private servers. Lightsail includes everything you need to launch your project quickly – a virtual machine, SSD-based storage, data transfer, DNS management, and a static IP – for a low, predictable price.



Lightsail is for developers. You can choose an image for your Lightsail instance that jumpstarts your dev project so you don't have to spend as much time installing software or frameworks.

If you're an individual developer or hobbyist working on a personal project, Lightsail can help you deploy and manage basic cloud resources. You might also be interested in learning or experimenting with cloud services, such as virtual machines or networking. Lightsail provides a quick way to get started.

Lightsail has images with base operating systems, development stacks like LAMP, LEMP (Nginx), and SQL Server Express, and applications like WordPress, Drupal, and Magento.

As your project grows, you can add block storage disks and attach them to your Lightsail instance. You can take snapshots of these instances and disks and easily create new instances from those snapshots. You can also peer your VPC so that your Lightsail instances can use other AWS resources outside of Lightsail.

You can also create a Lightsail load balancer and attach target instances to create a highly available application. You can also configure your load balancer to handle encrypted (HTTPS) traffic, session persistence, health checking, and more.

Chapter 8: Conclusion

Chatbots are artificial intelligence systems that we interact with via text or voice interface. Those interactions can be straightforward, like asking a bot about the weather report, or more complex, like having a bot troubleshoot a problem with your internet service. Through this project, we have created a chatbot for NUS-ISS website.

The great power of this intelligent chat bot is that we were able to design our own business logic through the use of an intuitive console and easily integrate external modules. Moreover, Dialogflow can scale to thousands of users, being built on Google Cloud Platform, the scalable cloud infrastructure provided by Google.

In conclusion, chatbots are the future. Everyone should be ready to integrate one with their business. With so many chatbot platforms at our disposal, it has become extremely easy to build a chatbot. It's safe to say—a chatbot revolution coming.