JAC: Just Another Chatbot

A simple chatbot implementation

19MAT117 -MIS2



BL.EN.U4AIE19044 Suneel.N

BL.EN.U4AIE19045 Sai Prakash. N. N. V

BL.EN.U4AIE19067 Sudharsan Reddy V

**Abstract:**

JAC: Just Another Chatbot is a run of the mill chat bot designed to carry a normal conversation and try to mimic a real human interaction between two people as well as it possibly can. We have been assigned the text domain and have decided to construct a simple, yet useful chatbot; mainly owing to its tremendous functionality. The chatbot includes features like text prediction, auto-correction and so on. Concepts like linear algebra, breaking down the written text into numbers, store them in matrices which helps in processing the data; and probability that fuels text predictions form the backbone of this project. Another concept crucial to JAC, is text similarity; and the Maths behind it involves cosines and dot products to calculate similarity between sentences and suggest better predictions. JAC will ask the user about basic information such as your name, hobbies, likes and dislikes, and such and try to follow up with appropriate queries. The project will be written in Python and we plan to use datasets to provide the chatbot with its Knowledge Base.

**Introduction:**

A chatbot is generally meant for saving time for companies or organizations to help with their customer service that can be easily accessed by others to make use of the company’s services.

These chatbots helps in communicating with users and strengthening the relationship between user and the organization.

**Advantages:**

Chatbots are time savers. Involving personnel from the organizations may result in larger wait time and customers may not always get undivided attention. This may cause customers dissatisfaction, seeing as their valuable time is wasted.

Chatbots are able to communicate with several people at a time. This will be an added advantage when compared to the humans doing the same work, which leads to more human power.

Chatbots will help us to have a history of the persons and their requirements, which in turn helps the machine to learn and respond accordingly and more appropriately. This will also take care of emotional bias, always guaranteeing a polite and courteous reply to the customer in a gentle way, regardless of their behaviour.

**Methodology :**

Every chatbot will have some basic features irrespective of its domain in which it is used. Those include Text analysis, Text Autocomplete, Text Autocorrection.

This generally means that the user input will be analysed and based on the text from the customer, the chatbot will respond. User may enter some text wrongly. This also should be understood by the chatbot. So, chatbot needs to correct the input from the customer in the sense of spell-checks.

Chatbots also can predict the word, which the user is going to enter, prior to the entry in the chat space.

This is achieved in this project by analyzing each word for auto-correction, and considering a data set for auto-completion, which consumes time in typing.

In this project we have used Bayes’ theorem from probability.

https://miro.medium.com/max/276/0*BUkBDRtOlgBXwgpj.

This helps us to find the word that tries to enter by the customer.

In this process we tries to find the word with all possibilities of a word can go wrong while typing. We check the word with one letter error and two letter error. This has again four types of error check for any word.

1. There can be an alphabet typed extra
2. There can be an alphabet typed less
3. There can be misplace of two adjacent alphabets
4. There can be an alphabet replaced with other alphabet

These cases will be evaluated and, based on the baye’s theorem, the final correction will be done from the obtained list of words from those specified methods.

As we have a data set, we can check for the words which may satisfy the entering input, and can be returned to the user as the auto-completion. That can also be trained by the further chatting with customers.

**Code:**

Autocorrect.py:

import re  
from collections import Counter  
  
  
def words(text):  
 return re.findall(r'\w+', text.lower())  
  
  
CorpusFile = open('Corpus.txt').read()  
WORDS = Counter(words(CorpusFile))  
  
  
def Prob(word, N=sum(WORDS.values())):  
 # Probability of word.  
 return WORDS[word] / N  
  
  
def correction(word):  
 # Most probable spelling correction for word.  
  
 return max(correctwords(word), key=Prob)  
  
  
def correctwords(word):  
 # All possible spelling corrections for word.  
 return knownword([word]) or knownword(editDistance1(word)) or knownword(editDistance2(word)) or [word]  
  
  
def knownword(words):  
 # The subset of words that appear in the dictionary of WORDS.  
 return set(w for w in words if w in WORDS)  
  
  
def editDistance1(word):  
 alphabets = 'abcdefghijklmnopqrstuvwxyz'  
 # Remove any one character from the word.  
 Removeonechar = [(word[:i] + word[i + 1:]) for i in range(len(word))]  
 # Adding one character any where in the word.  
 InsertOneChar = [(word[:i] + c + word[i:]) for i in range(len(word) + 1) for c in alphabets]  
 # Transposing the order of any two adjacent characters.  
 Transpose = [word[:i] + word[i + 1:i + 2] + word[i:i + 1] + word[i + 2:] for i in range(len(word) - 1)]  
 # Substituting any character in the word with another character.  
 Replacing = [(word[:i] + c + word[i + 1:]) for i in range(len(word)) for c in alphabets]  
 return set(Removeonechar + InsertOneChar + Transpose + Replacing)  
  
  
def editDistance2(word):  
 return (editDis2 for editDis1 in editDistance1(word) for editDis2 in editDistance1(editDis1))

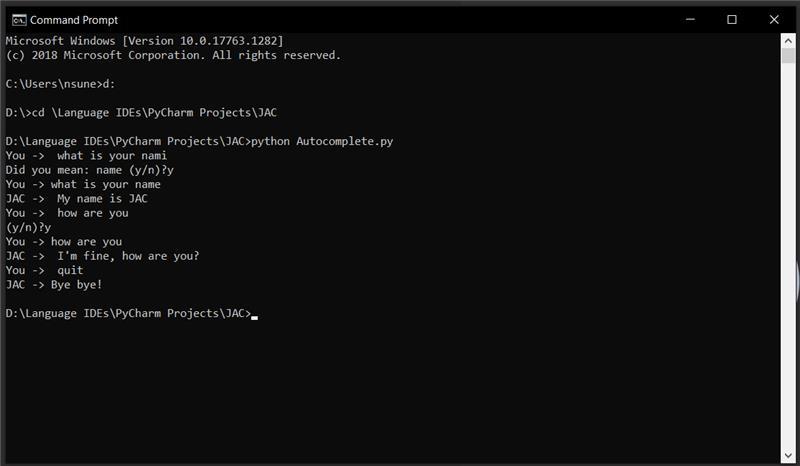
Autocomplete.py:

import readline  
from AutoCorrect import correction  
from ChatBot import Converse  
  
  
class MyCompleter(object): # Custom completer  
  
 def \_\_init\_\_(self, options):  
 self.options = sorted(options)  
  
 def complete(self, text, state):  
 if state == 0: # on first trigger, build possible matches  
 if text: # cache matches (entries that start with entered text)  
 self.matches = [s for s in self.options  
 if s and s.startswith(text)]  
 else: # no text entered, all matches possible  
 self.matches = self.options[:]  
  
 try:  
 return self.matches[state]  
 except IndexError:  
 return None  
  
  
Corpus = open('Corpus.txt', 'r')  
words = []  
for line in Corpus:  
 words.append(str(line.split(" ")[0].lower()))  
completer = MyCompleter(list(words))  
readline.set\_completer(completer.complete)  
readline.parse\_and\_bind('tab: complete')  
  
while True:  
 print("You -> ", end=" ")  
 userInput = input()  
 if userInput == "quit":  
 print("JAC -> Bye bye!")  
 break  
 else:  
 words = userInput.split(" ")  
 for i in words:  
 auto = correction(i.lower())  
 if auto != i.lower():  
 print("Did you mean: " + auto, end=" ")  
 userInput = userInput.replace(i.lower(), auto)  
 choice = input("(y/n)?")  
 print("You ->", userInput)  
 if choice == "y":  
 Converse(userInput)

Chatbot.py:

from random import randrange  
  
  
knowledge\_base = [  
 ["hey",  
 [  
 "Hey there!"  
 ]  
 ],  
 ["what is your name",  
 [  
 "My name is JAC",  
 "You can call me JAC"  
 ]  
 ],  
  
 ["what is your age",  
 [  
 "I'm just a few days old",  
 "I'm just a computer program; I don't have any age"  
 ]  
 ],  
  
 ["how are you",  
 [  
 "I'm fine, thank you for asking!",  
 "I'm fine, how are you?",  
 "Everything is going well"  
 ]  
 ],  
  
 ["I’m fine",  
 [  
 "Nice to hear that",  
 "Great!!!",  
 "Awesome!!!"  
 ]  
 ],  
  
 ["how is the weather",  
 [  
 "There are high chances of rain",  
 "It's very cold, currently, due to the frequent rains"  
 ]  
 ]  
]  
  
  
def Converse(User):  
 if User in dict(knowledge\_base):  
 print("JAC -> ", dict(knowledge\_base)[User][randrange(0, len(dict(knowledge\_base)[User]))])  
 else:  
 print("JAC -> I don't understand, sorry")

**Results:**

****

**Conclusion:**

We were able to develop a simple chatbot program implementing autocorrect and autocomplete features through Bayes’ Theorem, various models and Damerau-Levenshtein Distance. This sets the base for our future work to implement other advanced features and enhance our chatbot.

**Future work:**

Going forward, we hope to make JAC smarter by incorporating machine learning models and re-building our program with the help of neural networks and better datasets. A GUI interface to make user-friendly interactions is another venue we hope to explore. Finally, we want to centre our chatbot around a certain theme - A FAQ Chatbot for the university is a rather enticing possibility that we look forward to working on. One thing for sure, the end goal for is this long-term project is a useful, robust and highly intelligent chatbot.