The brain behind CharLando

Omkar Singh, Pratyush Tiwary, Anurag Pandey, Akash Chatterjee
Department of DS, Thakur College of Science and Commerce Mumbai, India

Abstract: **The use of chatbots evolved rapidly in numerous fields in recent years, including Marketing, Supporting Systems, Education, Health Care, Cultural Heritage, and Entertainment. In this paper, we present the algorithms used to create CharLando and how we tried several algorithms before using Levenshtein distance. Next, we discussed how the whole system is designed, where we tried to segment the system into 2 parts and explain them separately. Moreover, we highlight how the program is not dependent on the distances algorithm. Furthermore, we also talked about the elements used to teach CharLando.**

Keywords – charlando, rule-based chatbot, chatbot package, chatbot.

I. INTRODUCTION

Artificial Intelligence (AI) increasingly integrates our daily lives with the creation and analysis of intelligent software and hardware, called intelligent agents. Intelligent agents can do various tasks ranging from labor work to sophisticated operations. A chatbot is a typical example of an AI system and one of the most elementary and widespread examples of intelligent Human-Computer Interaction (HCI). It is a computer program, which responds like a smart entity when conversed with through text or voice and understands one or more human languages by Natural Language Processing (NLP). In the lexicon, a chatbot is defined as "A computer program designed to simulate conversation with human users, especially over the Internet". Chatbots are also known as smart bots, interactive agents, digital assistants, or artificial conversation entities.

Chatbots can mimic human conversation and entertain users but they are not built only for this. They are useful in education, information retrieval, business, and e-commerce. They became so popular because chatbots have many advantages for users and developers. Most implementations are platform-independent and instantly available to users without needed installations. Contact to the chatbot is spread through a user's social graph without leaving the messaging app the chatbot lives in, which provides and guarantees the user's identity. Moreover, payment services are integrated into the messaging system and can be used safely and reliably and a notification system re-engages inactive users. Chatbots are integrated with group conversations or shared just like any other contact, while multiple conversations can be carried forward in parallel. Knowledge in the use of one chatbot is easily transferred to the usage of other chatbots, and there are limited data requirements. Communication reliability, fast and uncomplicated development iterations, lack of version fragmentation, and limited design efforts for the interface are some of the advantages for developers too.
CharLando is an advanced python package that streamlines the creation of complex chatbots, and its user interface makes it accessible to non-technical users as well. This paper will provide an in-depth examination of the workings of CharLando, as well as explore potential areas for improvement. Furthermore, the paper will scrutinize the package's internal architecture and how it determines the correct rule to activate based on the user's input. The paper will also outline the trials and errors we encountered in developing the package and how we arrived at its current state.

II. ALGORITHMS/TECHNIQUES.

A. *Cosine Similarity*

So in the beta version of CharLando we decided to use cosine similarity. Cosine similarity is a measure of similarity between two sequences, in our cases, these sequences will be strings. The algorithm loops through each rule's cases and required words to map the similarity between the query and the cases or required words. This technique was not used later on

as cosine similarity is not a good measure of similarity between 2 sentences. It is generally used to find similarities between 2 vectors.

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Fig. 1 Cosine Similarity Formula

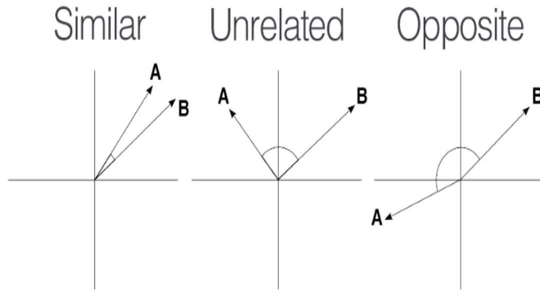

Fig. 2 Cosine Distance

B. SequenceMatcher

In the second version, we used python's inbuilt SequenceMatcher class. This class can be used for comparing pairs of input sequences. The basic idea is to find the longest contiguous matching subsequence (LCS) that contains no "junk" elements. This does not yield minimal edit sequences but does tend to yield matches that "look right" to people. This was removed in the later version because of its poor performance.

Let's say we want to compare the following strings.

1. "Charlando is a chatbot python package",
2. "Chatbot"

To anyone the natural match is chatbot, as follows:
Charlando is a chatbot python package
……………....chatbot………………..

However, the LCS will match it as follows
Charlando is a chatbot python package
Cha……………...tbot………………..
Since longer common subsequences may appear less natural to a human expert than a shorter one. Therefore we argue the fact that SequenceMatcher tries to find out the output which is more human-friendly.

We also no longer use it because the result may depend on the order of the arguments passed.

Fig. 3 SequenceMatcher FlowChart

C. Levenshtein distance

The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word into the other. The current version of CharLando uses this to measure the similarity between queries and cases.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}\big(\operatorname{tail}(a), b\big) \\ \operatorname{lev}\big(a, \operatorname{tail}(b)\big) \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) \end{cases} & \text{otherwise,} \end{cases}$$

Fig. 4 Levenshtein Distance Formula

III. DESIGN

CharLando uses 2 different approaches to detect a single rule for activation. The first approach is where the previously introduced algorithms come into play. For a given query, CharLando iterates through each rule and its cases, for each case it finds similarity between case and the query. This similarity is stored in a list, which is later on aggregated such that only the max similarity remains in the list. The image below nicely explains how this works in a graphical manner.

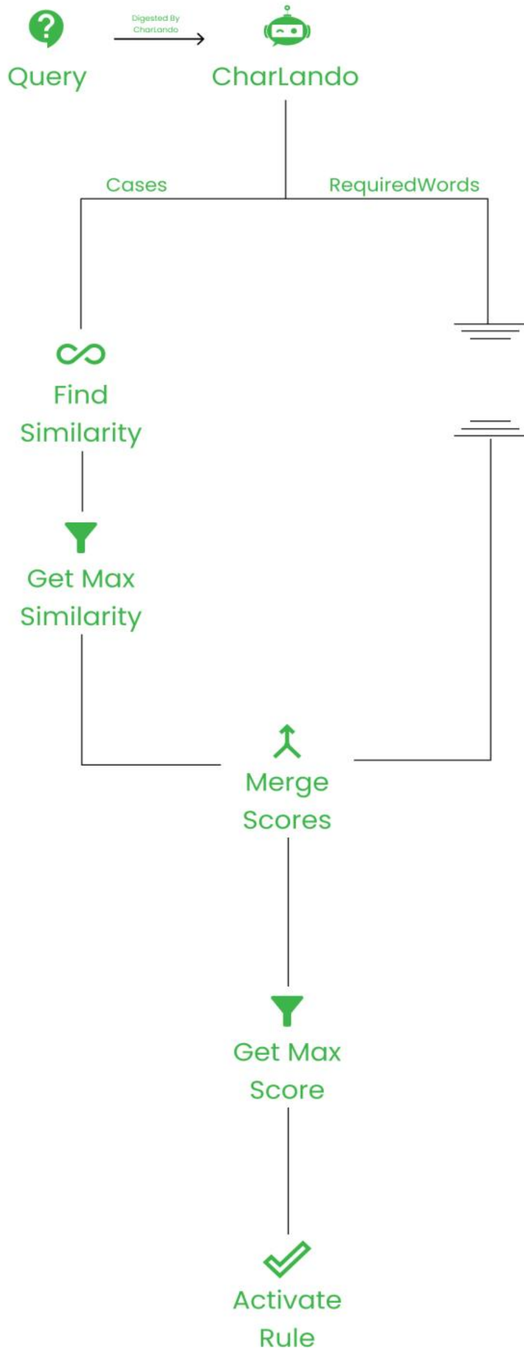

Fig. 5 CharLando's Case Logic

Here in the later part we are referring to similarity as score because required_words doesn't use the concept of similarity, instead they give score based on the occurrences of the words. In this diagram, we are also missing the pre-processing stage that happens at the time of digestion. Now that we know how cases contribute to a rule's activation, let's take a look at how required_words contribute to a rule's activation.

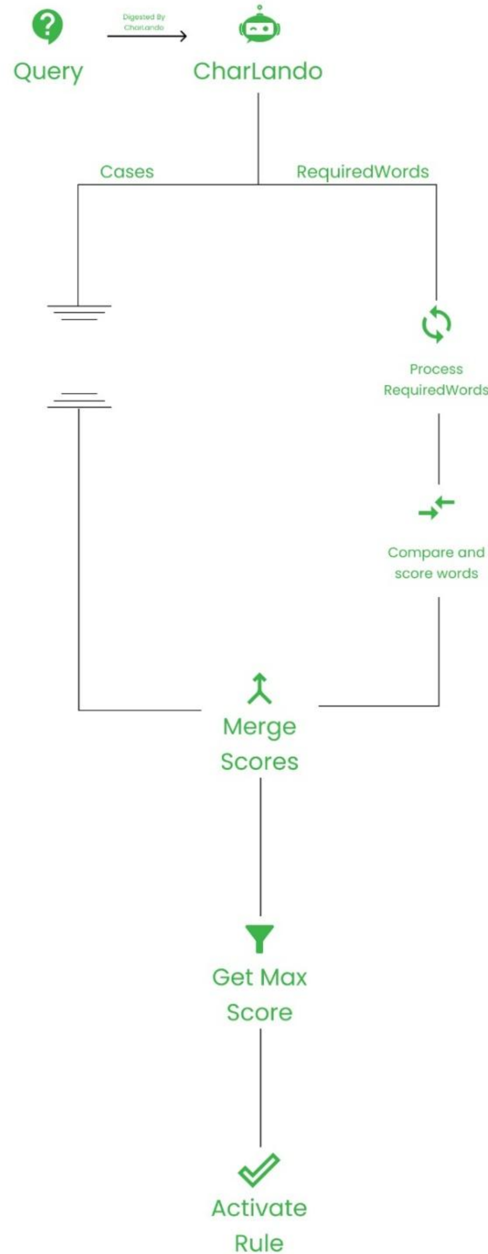

Fig. 6 CharLando's RequiredWords Logic

A lot of magic happens in the process requiredWords stage, there variables are removed from the list, optional words are broken down and all words are converted into lowercase. The compare and score words try to score query based on requiredWords.

Finally, after all the cases and requiredWords processes a rule along with variables and confidence

is returned, predict_raw function returns a dictionary of scores and variables which is later on processed by the predict function to activate a rule.

IV. RESULT/OUTCOME

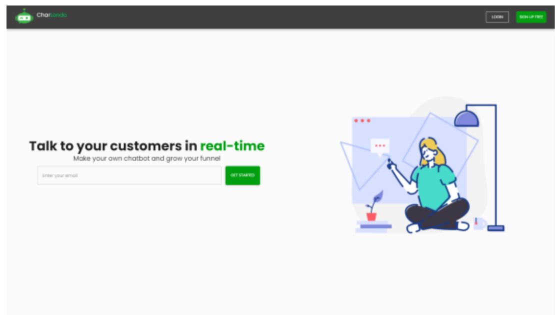

Fig. 7 CharLando's Homepage

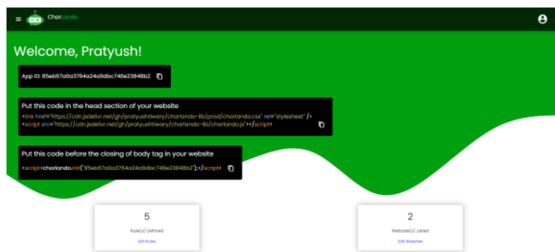

Fig. 8 CharLando's App Dashboard Page

Fig. 9 CharLando's Rule Editor Page

V. CONCLUSION

Minimal human interference in the use of devices is the goal of our world of technology. Chatbots can reach out to a broad audience on websites and be more effective than humans. At the same time, they may develop into a capable information-gathering tools. They provide significant savings in the operation of customer service departments.

Creating a chatbot that is flexible enough to change according to your needs is still pretty tough. You can use if-else statements to create a chatbot but it will not be very flexible and scalable, and the developer experience will be very bad. CharLando tries to fix these issues by making it easy for anyone to create a chatbot in minutes without losing the flexibility and control over how their bot acts. Even though there are several libraries like chatterbot but they are no longer supported and are not in the active development phase, which makes CharLando a suitable choice for making chatbots with future support. You can still use Keras to train a custom chatbot that can act very humanely but that will introduce a lot of problems, like in order to change a specific behavior of your bot you will have to train the whole model which makes it very inefficient. We try to solve this problem by removing the training phase totally and making it so that the bot doesn't have to be updated anytime there is a change.

Future work of this research would be exploring in detail how to implement a translation service that will make chatbots language independent. It would also be interesting to try different similarity algorithms.

REFERENCE

[1] Adamopoulou, Eleni, and Lefteris Moussiades. "An overview of chatbot technology." IFIP International Conference on Artificial Intelligence Applications and Innovations. Springer, Cham, 2020.

[2] Cosine similarity. (2023, January 13). In Wikipedia. https://en.wikipedia.org/wiki/Cosine_similarity

[3] Levenshtein distance. (2022, December 22). In Wikipedia. https://en.wikipedia.org/wiki/Levenshtein_distance

[4] Jaiswal, Nikhil. "SequenceMatcher in Python." Medium, April 2019, towardsdatascience.com/sequencematcher-in-python-6b1e6f3915fc.