International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 6, Issue 8, August 2017 **ISSN 2319 - 4847**

Remote Intelligent Assistant (REIA)

Maanav Shah[1]

[1]Pune Institute of Computer Technology, Pune

ABSTRACT

One of the fundamental means of interaction with any operating system is the command-line user interface (CLI). The use of CLI helps users to leverage the full potential of operating system. A widespread problem among CLI users is to learn the shell commands of operating system. This hinders the ability of users to effectively use any operating system. We aim to provide a natural language interface in the form of an Intelligent Assistant (IA) as a solution. This IA would understand natural language and then translate them into a logically and semantically correct sequence of Linux shell commands. The highlight of this assistant is that, it allows this using remote access. This IA accesses the full capabilities of Linux CLI. We propose to understand and extract information from the user's natural language input and then generate corresponding shell commands in an application. Our attempt to provide this solution has yielded significant results in natural language translation to shell commands. This paper demonstrates the application of natural language processing that can be extended as an IA in a Linux system.

Keywords: Command-Line Interface (CLI), Intelligent Assistant (IA), Natural Language Processing (NLP)

1. INTRODUCTION

Remote Intelligent Assistant (REIA) is a Linux based desktop application. It is an intelligent assistant that allows the user to communicate with Linux machine in natural language. This application bridges the gap between users and Linux CLI. REIA provides a solution to overcome the limitation of accessing any remote machine using natural language [1]. REIA understands English sentences and converts them into a logically and semantically correct sequence of Linux Bash commands. This way we can provide an abstract natural language interface to any underlying operating system. Remote access to the machine running REIA can further simplify the problem of accessing the machine from mobile devices and browsers from any location in the world [2]-[3]. This application also includes a natural language interface for CLI and can handle multiple users simultaneously. If it does not understands the user input, it also provides a recommendation system that suggests the user a list of commands, similar to the user input. It also takes account of the performance by logging and calculating the accuracy of the system. We provide a uniform interface of the application and also ease the task of using the command line effectively.

2. LITERATURE SURVEY

2.1 Exiting Methodologies

There are few existing methodologies that overlap with our idea, however, there are differences that separate our application.

2.1.1 Betty

Betty is an open source application that translates plain English sentences to Linux terminal commands. The development of natural language input systems, however, is the main design goal. The tool employs the use of pattern matching. On the other hand, REIA has proposed the pipelined combined classifier for natural language processing. It also does not have any facility for remote access. The command set is still comparatively small and the functionality limited to a few tasks. The project is still inviting commits from everywhere.

2.1.2 Sirius

Sirius is an open source end to end stand-alone intelligent personal assistant. Sirius receives queries in the form of speech or images and returns result in the form of natural language. It is one of the major advances in intelligent assistants. Sirius implements functionalities such as speech recognition, image processing, and natural language processing. It is basically a question-answer bot which provides the answer in natural language.

2.2 Proposed Methodology

The proposed methodology is Remote Intelligent Assistant that is a Linux based desktop application. It is a tool which converts English sentences into Linux shell commands. REIA is a natural language interface to Linux and it extends its service by providing remote access [1]. This will allow naïve users easy access to the system and without any prior knowledge of the operating system commands. This application provides a remote access feature to the user, enabling them to communicate with their machine from anywhere. The application consists of a natural language processing and

a machine learning engine for processing the text and keywords, structured with the help of a combination of pipelined combined classifiers – Vectorizer, Term Frequency – Inverse Document Frequency (TF-IDF) and OneVsRest classifier. The system also trains to adapt to the user's language style, in order to deliver better accuracy even with ambiguity in input. REIA is flexible and scalable as we have integrated Slack for communication. It allows multiple members of the team to access the system remotely. The queuing system ensures that the real-time messages from multiple users are handled. The application can be extended to handle voice input as well, further simplifying the user effort and at almost no cost to the system.

3. DESIGN

3.1 Higher Level Architecture

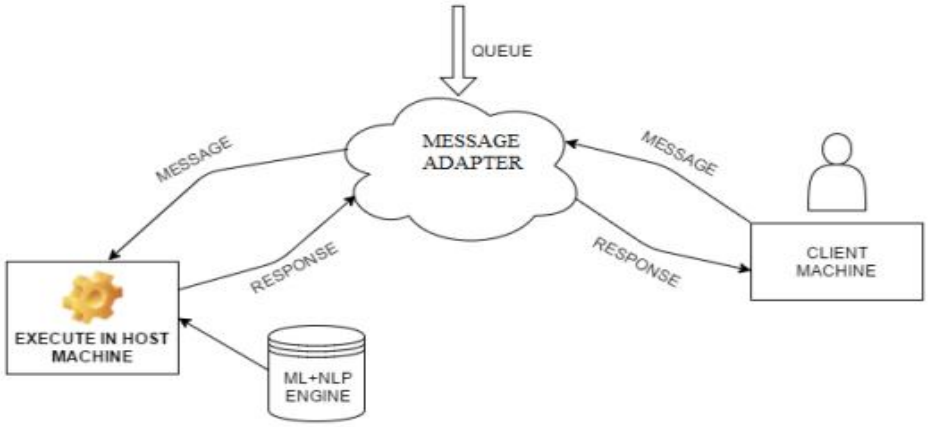

Figure 1 Higher Level Architecture

3.1.1 Client Machine

The user initiates the request by sending a message in English language to the required host machine using the message adapter channel which is slack.

3.1.2 Message Adapter

It assigns the messages to relevant user tokens and places them in queue which are grabbed by the host machine on a real-time basis.

3.1.3 Natural Language Processing and Machine Learning Engine

It tokenizes and classifies the keywords based on which the statements are converted into terminal commands.

3.1.4 Host Machine

It is the machine where the execution of the generated commands occurs. The execution engine runs the sequence of commands thereby generating the required result and forwarding them again back to the user.

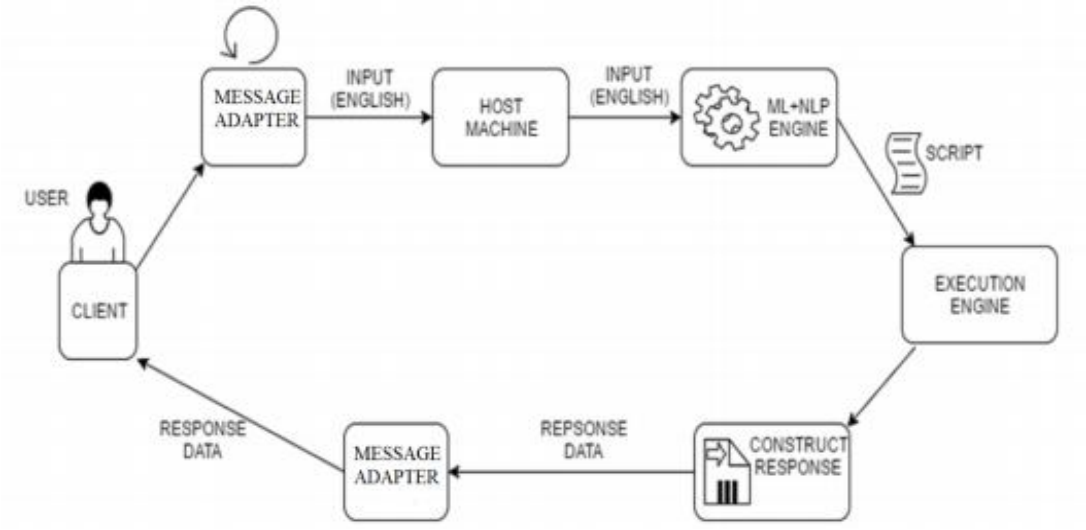

Figure 2 Modular Architecture

4. IMPLEMENTATION

4.1 Implementation Flow

User connects to REIA through a slack channel, under the registered domain, in this case, illuminatipict.slack.com. The user communicates with REIA by posting messages on the channel, starting each message by tagging REIA using @reia, that allows us to directly access the bot integrated into slack. The message is read from the slack channel and stored in a queue. A script continuously listens for messages and distinguishes between messages via their timestamp, id, and tags. Another script reads messages from the queue and further processing is performed. The two scripts work in parallel. In processing, first, the user metrics are extracted from the message. This includes the user posting it and his slack specific credentials. The user is then authenticated. Once the user is verified, the message is then tokenized. It is then run through a pipelined combined classifier – Vectorizer, Term Frequency –Inverse Document Frequency, OneVsRest classifier, that classifies it to a specific class of command namely file, folder, network, system, general etc. This helps to map that the input is either a file related command or a folder related command, etc. Then, using a distance similarity algorithm – Jaccardian distance and Jaro Wrinkler distance, we try and map the command to one in the dataset, and by further extension to its Linux command like cp, mv, mkdir, etc. We use Part-Of-Speech Tagger **(*POS Tagger*)** here to identify the nouns, proper noun etc so that, according to the class of command, we extract the relevant parameters like source and destination, flags. Using the above data, the command is constructed and executed on the machine. The response is returned to slack, along with the user it is addressed to.

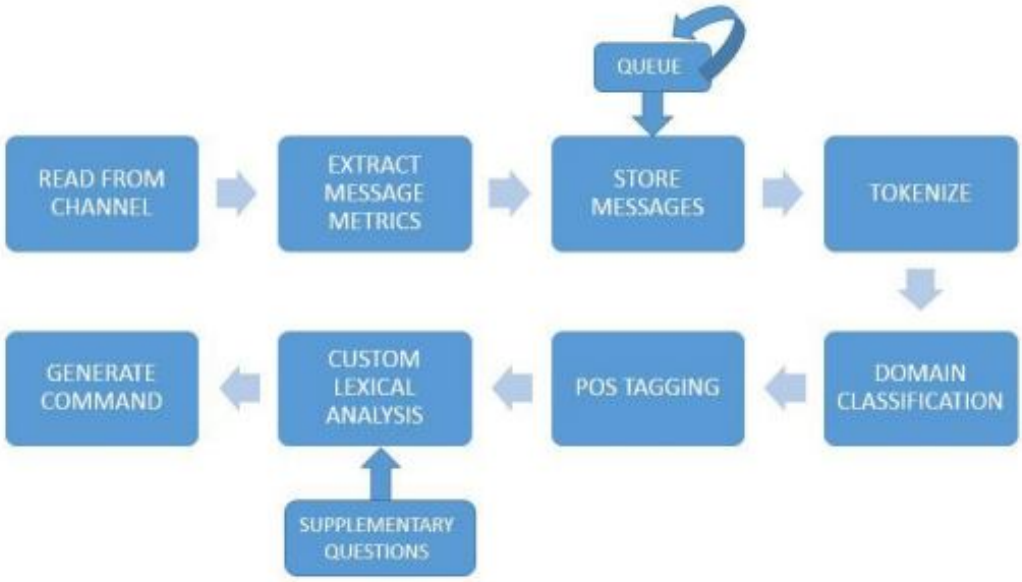

Figure 3 Modular Flow

4.2 Algorithms

4.2.1 Vectorizer classifier

It converts a set of documents to count matrix. The counts are than modelled into a sparse representation. The feature count is equivalent to the vocabulary size calculated unless feature extraction is done.

4.2.2 TF-IDF

The term frequency-inverse document frequency is used to describe importance of a word to a document in a particular collection. Generally used as a weighting factor in information retrieval and text mining. It works by calculating the frequency of words in a document and helps in removing the bias towards longer documents by calculating augmented frequency instead of just Boolean frequency which reduces misclassification to a big extent. Augmented term frequency can be given as –

$$tf(t, d) = 0.5 + 0.5 \cdot ft,d \max\{ft,d : t' \epsilon d\} \quad (1)$$

The inverse document frequency gives a measure of the importance of a word across all documents and is given as –

$$idf(t,D) = \log N \, |\{d \in D : t \in d\}| \quad (2)$$

with N as total number of documents in the corpus $N = |D|$ □ $|\{ d \in D : t \in d\}|$: Number of documents where the term t appears (i.e. $.tf(t, d) \neq 0$). If the term is not in the corpus, this will lead to a division-by-zero error. It is therefore, common to adjust the denominator to $1 + |\{d \in D : t \in \}|$

Term frequency–Inverse document frequency is calculated as -

$$tf\text{-}idf(t, d,D) = tf(t, d) \cdot idf(t,D) \quad (3)$$

with the term appearing in more documents the tf-idf value approaches zero.

4.2.3 OneVsRest Classifier

It is also called as the one-versus-all classifier. This basically consists of per class fitting one classifier. For each classifier, the class is fitted against all other classes. Interpretability is the most important advantage of using this classifier. It does not allow the mapping into a particular class to be biased. It provides a fair default choice and is a commonly used classifier for multiclass classification.

4.2.4 Jaro Wrinkler Distance

The Jaro–Winkler distance is a measure of distance between two documents. The Jaro-wrinkler distance calculated represents the minimum number of single-character transpositions required to change one word into the other. The distance is inversely proportional to similarity, similarity increases with a decrease in Jaro-wrinkler distance. Jaro distance can be given as,

$$dj = \{ \begin{matrix} 0 & if\ m = 0 \\ 1/3(\ m/\ |s1\ | + m/\ |s2\ | + m - t\ /m\ ) & otherwise \end{matrix} \} \quad (4)$$

where,$|si\ |$ is the length of the string si, m is the number of matching characters, t is half the number of transpositions.

4.2.5 Jaccard distance

The Jaccard coefficient is a measure of similarity between sample sets, and is defined as the intersection size divided by the union size of the sample sets –

$$J(A,\ B) = |A \cap B|\ |A \cup B| = |A \cup B| - |A \cap B|\ |A \cup B| \quad (5)$$

If A and B are both empty, we define $J(A,\ B) = 1$ The Jaccard distance, that measures dissimilarity between sets, is calculated by subtracting the Jaccard coefficient from 1 –

$$dj\ (A,\ B) = 1 - J(A,\ B) = |A \cup B| - |A \cap B|\ |A \cup B| \quad (6)$$

4.2.6 PartsOfSpeech Tagger

A Part-Of-Speech Tagger (POS Tagger) is a software that reads a text in a particular language and assigns parts of speech to each word, such as noun, verb, adjective, etc. In our project, we have used Stanford POS Tagger.

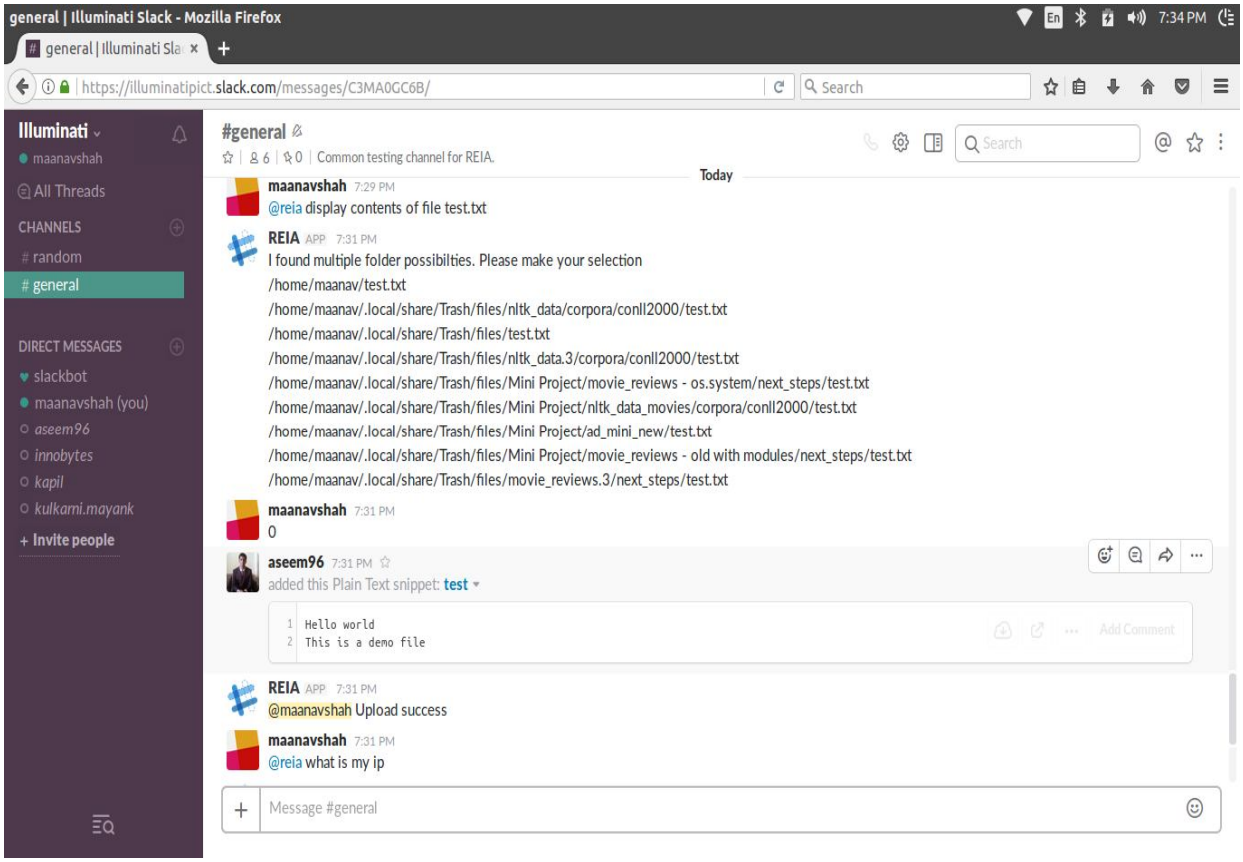

Figure 4 Slack Client Interface – 1

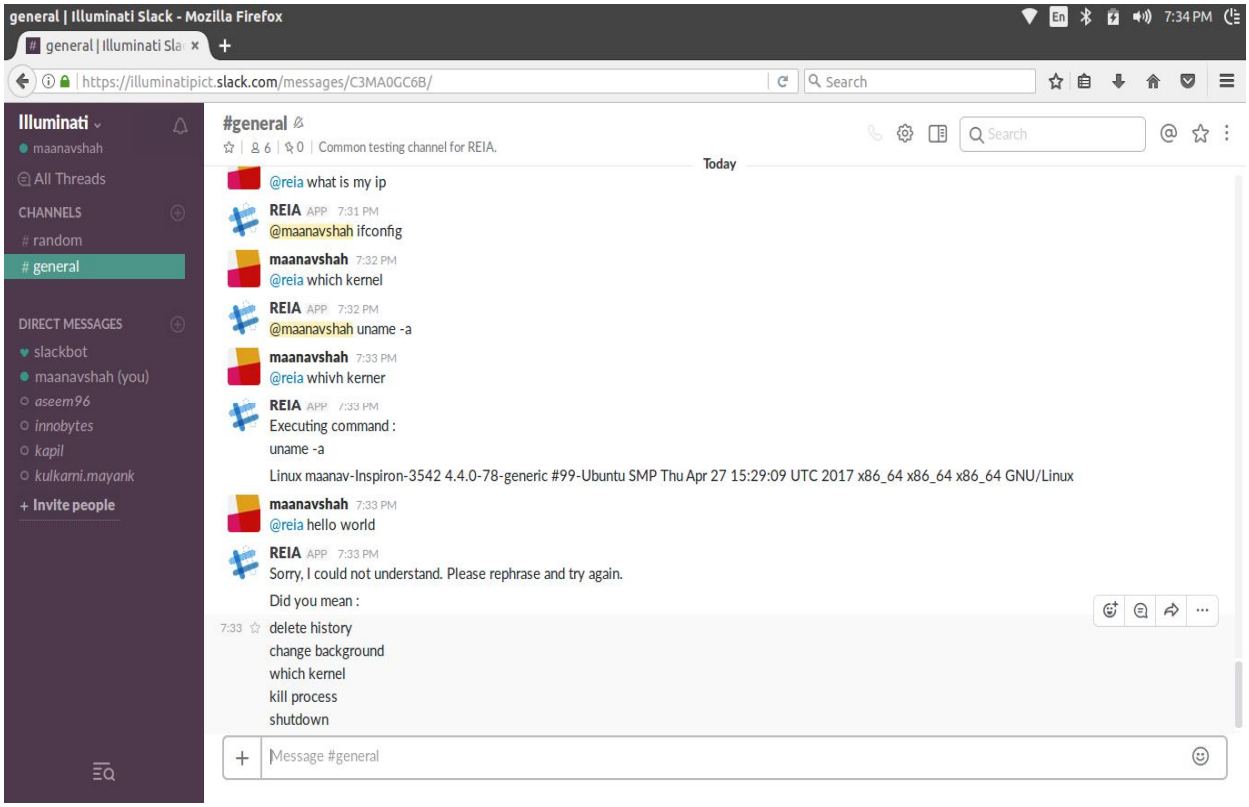

Figure 5 Slack Client Interface – 2

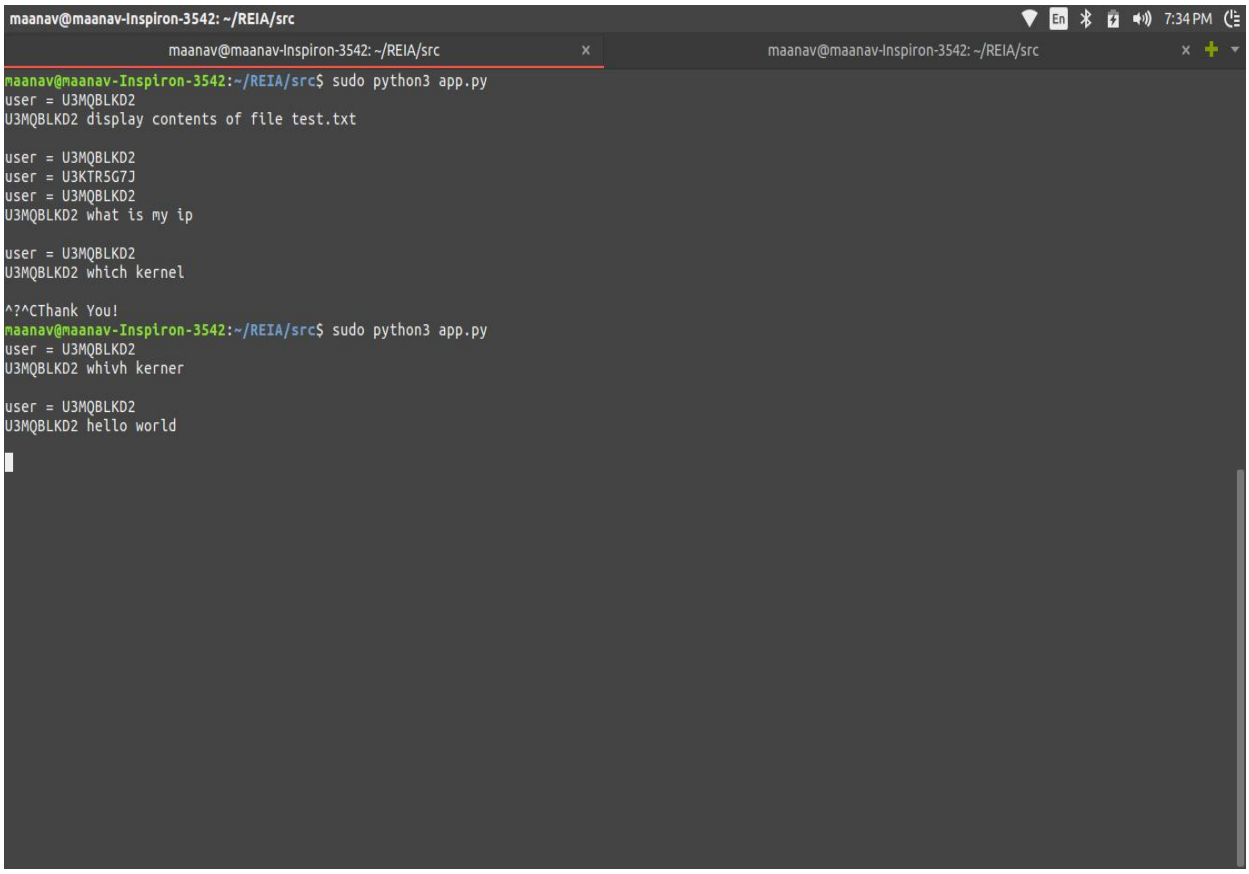

Figure 6 Listening client messages (src/app.py)

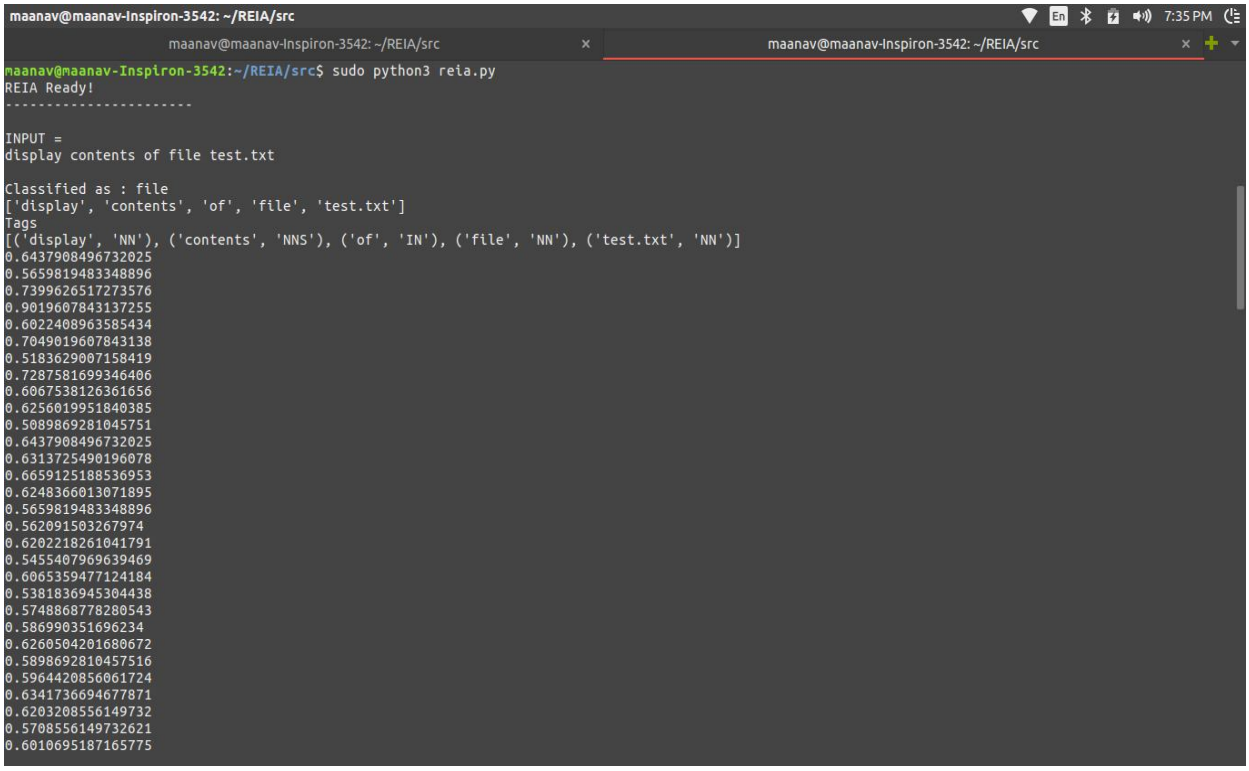

Figure 7 Message Processing and Execution Script - 1

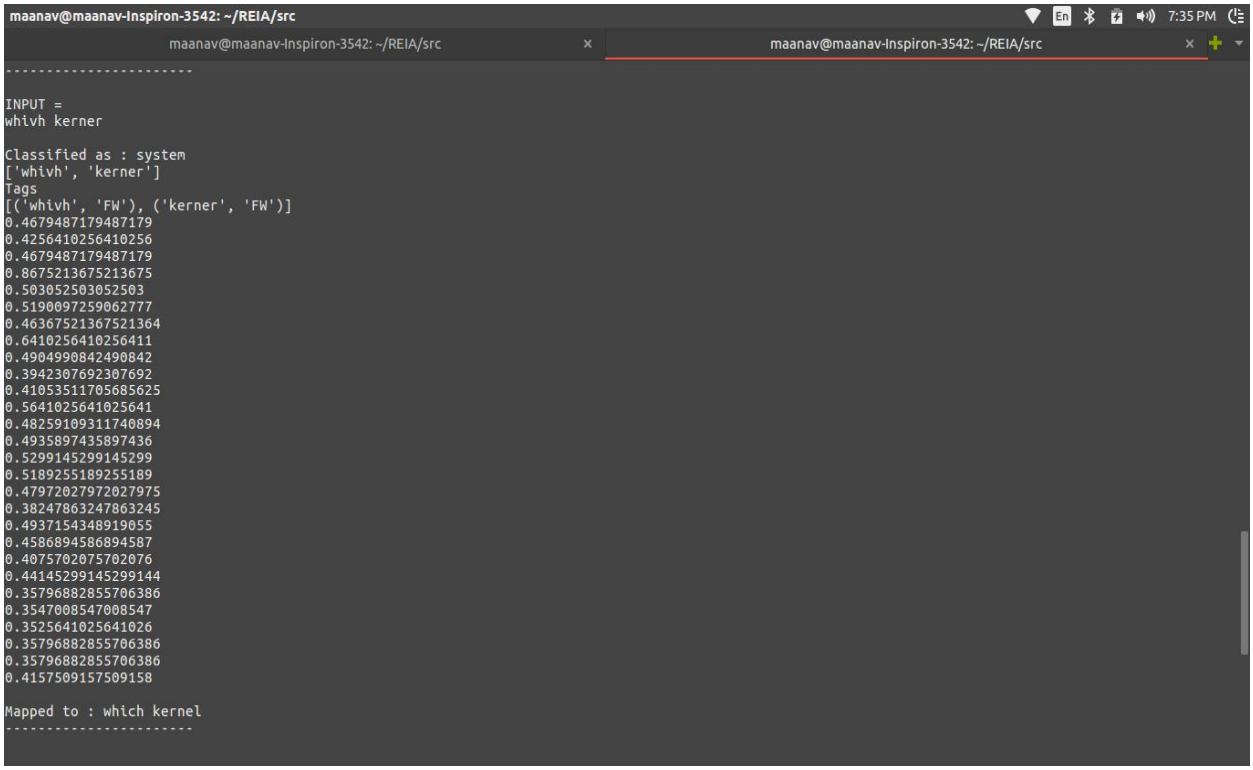

Figure 8 Message Processing and Execution Script – 2

5. RESULTS AND EVALUATION

5.1 Results

This application converts English statements into commands and executes with a response time of less than one second depending on internet connection. Multiple variations of the same command are handled. Multiple users can interact with the system. Each user is tagged and responded to individually and thus, no message is missed if system quits

before a message is processed, it picks up the message next time from history maintained in the queue. It also provides suggestion if it does not understand a message and also handles language ambiguity such as spelling mistake, etc.

5.2 Accuracy

The input received was logged in a log report, and the accuracy of the system classifying the input correctly or incorrectly was calculated. The accuracy of the system was calculated by –

$$\text{Accuracy} = \frac{\text{Statements correctly classified}}{\text{Total input statements}} \times 100$$

Figure 9 Log Report

6. CONCLUSION

Remote Intelligent Assistant (REIA) is an abstract natural language interface to Linux systems. REIA provides an easy reliable solution that allows the user to access the remote machine using natural language, without compromising on user's privacy. REIA also allows the user to communicate with the remote machine and control it virtually from anywhere in the world. It can handle multiple users simultaneously. REIA is accessible through desktop as well as mobile. It provides an interface for text input and can be extended to speech input as well. The extensibility of this application has led to huge applicability. Since our application allows communication through natural language, it will also enable even novice Linux users to interact with the system without any prior knowledge of terminal commands. Additionally, the application will easily automate the task of Linux server administrators.

References

[1] Artificial Intelligence – Making an Intelligent personal assistant, Indian Journal of Computer Science and Engineering (IJCSE) , Vol. 6 No.6 Dec 2015-Jan 2016, ISSN : 0976-5166

[2] Applying Chatbots to the Internet of Things: Opportunities and Architectural Elements, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 7, No. 11, 2016

[3] Donna Interactive Chat-bot acting as a Personal Assistant, International Journal of Computer Applications (0975 – 8887) Volume 140 – No.10, April 2016

[4] Personal Assistant and Intelligent Home Assistant via Artificial Intelligence Algorithms, International Journal of Research in Engineering & Technology (IMPACT: IJRET) ISSN(P): 2347-4599; ISSN(E): 2321-8843 Vol. 4, Issue 6, Jun 2016, 9-14

AUTHOR

Maanav Shah has pursued B.E in Information Technology from Pune Institute of Computer Technology, Pune. He received his Diploma in Computer Engineering from Shri Bhagubhai Mafatlal Polytechnic, Mumbai. His area of interest include programming and algorithms.