Unix Shell Assistant

HARSH KUNDNANI (HK17B)
TEJAS HASARALI (TDH17)
XIANGZHEN SUN (XS14)

DECEMBER 7TH, 2018

Introduction

There have been complaints about hard times of memorizing Unix commands from the man page, let alone using these commands with the right syntax in various situations. Through our research and study, we haven't found even one solution that can help users with Unix command line. The most popular solution so far is to search the usage of a specific command via the Google Search or its equivalents. Indeed, Google Search does provide a smart and convenient way to find answers to a wide range of problems with high accuracy. There are still drawbacks of using it. One of the drawbacks is that Google Search requires the assistance of a browser. One must resort to browser's search bar to feed queries into the search engine. Modern computer systems all come with a browser, yet in certain cases where a computer is used as a server or network management station, people might prefer to reduce the browser in exchange of functionality and security. In that case, a search engine would be a luxury at any cost. Closely connected to this drawback is another deadly drawback: the reliance on accessing the external network. As far as we know, search engines do not have local copies of the solutions to every question. Even if it does, it merely saves a small number of queries to the cache table. There is no guarantee that the server can return a satisfying result without being connected to the network. Besides, the search results from Google Search are greatly influenced by the way people edit them. In other words, the results are flexible all the time. However, in the Unix system, the syntax of commands does not change over time. Nor do people have the necessity to edit these commands. Therefore, we proposed this project, Unix Shell Assistant, in order to focus on the addressed issues in Unix command line tools.

In this project, we implement a command line tool on the local computer system. The programming language is mainly Python because many of the modern distributions of Unix system have their applications written and installed with Python script. There won't be any compatibility issues at all. Since the program was implemented to the local system, users do not rely on the Internet or other special configurations on their computers. This also adds a layer of safety to a user's computer. The mechanism of our command line tool is very succinct. It employs Natural Language Processing tools to parse and tokenize a user's input. Once a user's input, i.e., human natural language, is transformed into a list of tokens, we analyze these tokens to extract the action verbs and the entity nouns from them. Since the action and entity of a sentence can basically define the intention of a speaker, we can therefore safely apply our algorithms to rank the occurrence of these action and entity words among all Unix man pages. The highest ranked man page contains the result we want with the largest probability. In extreme cases, the item returned from our algorithm might still not be what we want. That is why we add a user feedback module to our program. Basically, if the result returned from our command line tool is not desirable, the user can consistently send negative feedback to it and the command line tool will recursively return the second highest ranked result until the user indicates "Yes".

Our project report consists of four major parts. Part 1 is an overview of our project. In part 2, we give details on Natural Language Processing (NLP) and the tools used in our project to implement NLP. We also talk about the algorithm used by our application to rank man pages. Part 3 is the

design and implementation of the program. In part 4 and 5, we demonstrate some promising results as well as our conclusions.

Literature Review

Natural Language Processing (NLP): It is the field of science that is concerned with the interaction between computers and human languages. It focuses on enabling computers to understand and process human languages.

NLP has several steps to make computer really understand the meaning of the spoken words:

1. *Sentence Segmentation:* In this step, it breaks down a paragraph or a text file into individual sentences. We assume that each sentence is a separate thought or idea and it would be a lot easier to make the process a single sentence or a line than the whole paragraph or file.
2. *Word Tokenization:* In this step, it splits a sentence into individual words or tokens. One should pay special attention to the tokenize punctuations since even punctuations have special meaning in our sentences.
3. *Part of Speech Tagging:* In this step, it tries to classify the tokens into a different part of speech. Each word is fed to a pre-trained POS classification model to obtain the desired results. The POS model has been originally trained by feeding it millions of English sentences with each word's part of speech already tagged and having it learn to replicate the same behavior. The model doesn't really understand the meaning of each word but rather statically tried to predict the part of speech of a word, for example, a word following "the" in English is most likely a noun.
4. *Stemming:* Stemming process works by cutting off the end or beginning of the word, considering a list of common prefixes and suffixes that can be found in a word without the knowledge of the context.
5. *Text Lemmatization:* Lemmatization is the process of figuring out the most basic form or lemma of each word in a sentence. It properly uses vocabulary and morphological analysis of the words to remove the endings and return the base or dictionary form of a word.
6. *Removal of Stop Words:* Stop words are the filler words like 'a', 'an' and 'the' which appear very frequently in an English sentence. For many use cases, it is best to remove the stop words as they introduce a lot of noise in a text document. There is usually a default list of stop words against which each word is compared to filter out the stop words.

Python NLP toolkits:

- *Natural Language Toolkit (NLTK):* NLTK can be considered as a standard tool for NLP in Python. It contains a large variety of tools, algorithms, and corpora for all the steps mentioned above of NLP. Several algorithms for a single problem can be extremely helpful for a researcher to customize their model according to their needs but may be problematic for a developer as it will increase the size of the project and may affect the performance.
- *SpaCy:* This toolkit contains the best algorithm for each problem and the developers keep the algorithms updated as the state-of-the-art progress. It is not huge in size when compared to the

NLP and can be effective for the development and production related work. spaCy uses Hidden Markov model for POS tagging. The model calculates the POS tags based on two probabilities: Transition probability which calculates the probability of the current word 's POS tag based on the previous word's POS tag. Emission Probabilities calculates the probability of a word having certain value based on the POS tag.

- *TextBlob:* This toolkit is built on top of the NLTK and provides an intuitive interface for NLTK. It provides simple API to perform all the tasks of NLP such as POS tagging, sentiment analysis, phrase extraction, and translation.

Page Content Rank (PCR) Algorithm:

In this algorithm, the importance of a page is determined based on the importance of the terms contained in the page and the importance of a term is specified with respect to the query.

PCR works in the following steps:

1. Term Extraction: A parser will extract the terms from the page and an inverted list is built based on these terms.
2. Term Frequency Calculation: A document that mentions the query term more often has more to do with the query and should receive a higher rank. We assign a weight for that term which is the frequency or number of occurrences of the term.

We only focus on the number of occurrences of each term hence the term "remove directory" will be similar to the term "directory removed" after performing the lemmatization on the document.

Design and Implementation

User Query Processing:

The computers are really good at handling standardized and structured data, but when it comes to unstructured data which do not have a format cannot be used to perform any task on the computer. Whereas we humans understand both structured and unstructured data easily. The language is the basic form of communication used by every human being, but the language doesn't have a standard format which can be readily understood by a computer. Even though language has rules, a word can mean different things in different context, this is not understood by the computer. Hence the language has to be converted into a machine-understandable standard format. This is where the natural language processing plays a key role, in our shell assistant a user can talk to the assistant how he talks to any other human being and the assistant is capable of understanding it. This is achieved by preprocessing the data to create a standard machine-understandable format. Natural language processing is the field which is responsible for converting the unstructured human language into a structured machine-understandable format.

The first step in converting the human language into the machine-understandable format is the cleaning of data, a sentence typed in by a human can contain many punctuations, symbols, and extra spaces. These have to be removed because they add extra noise to the data and interfere in converting them to a standard format. This was achieved by writing a simple regular expression

which would filter everything other than characters. Numbers were also filtered to reduce the noise further because a sentence "how to copy 2 files" and "how to copy files" should give the same result. Removing the number '2' from the first sentence will give us the second sentence, this was the case in most of the scenarios. Hence everything other than characters were filtered out. The next step in preprocessing of the data is performing lemmatization or stemming on the words. Both lemmatization and stemming perform the same task, removing the word suffixes like ing, ies, s. In stemming the words are converted into their root stem based on specific rules of porter's algorithm. It is based on the idea that suffixes of English language are made up of smaller and smaller stems, hence it will start by removing the smallest stem first. Stemming mainly has two problems, under stemming and over stemming, in under stemming the word may not even be converted into its root stem and in over stemming the root stem created may not even have any meaning. These problems are solved in lemmatizations, here the suffixes are removed only if the root word generated is present in the dictionary, this avoids over stemming. But the problem of under stemming still remains if the under stemmed word also is a meaningful word. But the trade-off in lemmatization is speed. The extra step of cross verifying every word generated against the dictionary makes the overall process really slow compared to stemming. Even though lemmatization is a slower process, we choose to use lemmatization as the results obtained through stemming were inconsistent and resulted in erroneous outputs. We choose three different natural language processing toolkits for lemmatization – NLTK, spaCy, and Text Blob. In our experiments NLTK performed the worst, spaCy performed the best and Text Blob was in between spaCy and NLTK. Even though spaCy performed the best among the three, we used Text Blob as the speed of spaCy lemmatization was really slow. In most of the cases Text Blob's output was consistent, hence we selected Text Blob for speed with a trade-off in accuracy. The lemmatization was carried out on both the query given by the user and the all the relevant man pages obtained to make the ranking more consistent.

After obtaining the root words through lemmatization, the next step was to extract the entity and action from the lemmatized sentence to produce different combinations for effective page ranking. To obtain the entities and action from the sentence, first we have to obtain the parts of speech of every word, this task is not as simple as identifying the parts of speech of every word as the same word can belong to different parts of speech. In different instances. Take the example of a list, in the sentence 'list all the countries' list, is a verb but in 'this is a list of countries', it is a noun. There are two ways in which POS tags can be generated rule-based tagging and stochastic parts of speech tagging. In rule-based tagging the tags are generated based on the predefined rules, here all the ambiguous words like list are classified based on linguistic properties of the language but this is very rigid, and the error rate is really high. But stochastic parts of speech (POS) tagging are more probabilistic approach where words are classified based on previous knowledge by generating a probabilistic model. We used Hidden Markov Model (HMM) of stochastic POS tagging, the HMM works by finding the probability of a word having a particular tag in the training dataset and the probability occurrence of previous n tags with the given word tag. We experimented on the three best NLP toolkits NLTK, spaCy, and Text Blob, here the Text Blob performed the worst, spaCy performed the best and NLTK in between spaCy and Text Blob. The execution speed of spaCy

was slow compared to Text Blob and NLTK, but due to the smaller length of query strings the difference was negligible. Hence, we used Hidden Markov Model provided by the spaCy toolkit to extract POS tags. Using spaCy POS tagging the verbs and nouns (Singular, Proper and Plural) in the sentence are extracted. Usually, the verb will be the action and noun will be the entity. In our project, we have limited the actions to one per query, but the entities can be of any number. In some instances, the HMM algorithm may classify the action as an entity in such cases, there won't be any actions.

Algorithm 1 Unix Shell Assistant

```
procedure BOT(query)
    Input: User query
    Output: Unix Command
    resultImprovement ← read from the file
    query ← filterTheCharacters(query)
    query ← TextBlob_lemmatization(query)
    POSDictionary ← SpaCy_POSTag(query)
    for tag in POSDictionary do
        if tag startsWith NN then
            entity ← tag
        end if
        if tag startsWith VB then
            action ← tag
        end if
    end for
    combinations ← Different Combinations(action, entity)
    commandList ← man -k (combinations)
    for page in commandList do
        page ← lemmatization(stopWordRemoval(filterTheCharcters(query)))
        highestRankedPage ← termFrequency(commandList)
    end for
    if highestRankedPage (score is negative) in resultImprovement then
        highestRankedPage ← next highest ranked page
    end if
    parsedDictionary ← parse(highestRankedPage)
    display ← command
    resultImprovement ← user feedback
end procedure
```

Unix Shell Assistant Algorithm

From the entities and action obtained from the POS tagging we will produce a different combination of action and entity i.e., the action is merged with different length combination of entities, along with this action is also added as one of the entries. In the produced combinations, the lengthiest combination is used as the keyword for search in the first step, then we will move on to the next highest length. The search is continued until we find a clear winner through term frequency algorithm. In some of the corner cases the action will be misunderstood by the model as an entity in such situations, the action is completely ignored, and all the entities as a whole are taken. In NLP, a query's stop words are removed before extracting the POS tagging. Stop words refers to common words of a language which are present in most of the sentences, which don't have any importance in a term frequency or searching algorithm. Removal of stop words are done before extracting the POS tags to ensure the focus is only on important parts of the sentences, but in our case removal of stop, words resulted in the classification of actions as entities. This is due to loss of sentence context after removal of stop words. Hence, we choose to skip the stop word removal and directly perform the POS tag extraction from the query.
The man pages obtained, are also processed in the same way as the user query by filtering and lemmatizing. But the stop words are removed from the man pages in contrast to user queries, this is done to reduce the time taken by the term frequency algorithm by reducing the search space. This also helps in searching of specific entities and actions together without the unnecessary noise.

Finding the relevant man pages:
We use a built-in feature provided by the man application which lists all the man pages containing the searched string. For example, man -k "remove directory" will give the following list of man pages: rm (1) - removes files or directories.
We take each file as an input and perform following steps to find the most relevant man page for our queried term.

1. We perform the sentence segmentation based on the period(.). Each sentence is converted to tokens and passed through the model to identify the stop words. We remove the stop words to reduce the unwanted tokens in the man pages and then pass each word through the lemmatization model to convert them to their basic root form.
2. After the lemmatization, the final text is used for searching the tokens extracted from the first step. We count the occurrence of each token and store it as the weight of that token for that man page. For example, if we find that our query term 'remove' occurs 10 times in a man page then we store the weight of remove as 10 for that man page. We repeat this process for each man page obtained in step 1.
3. We get a list of weighted tokens for each man page in step 2. We sort this list in descending order to obtain the man page with the highest frequency of the available tokens. In case of a tie we use the first available man page for that token. This step gives us the man page which most likely contains the command for the user query.

We again parse the man page obtained in step 3 but this time we perform segmentation line by line and we store the parsed results based on the titles like SYNOPSIS, DESCRIPTION, OPTION etc.

The parsed results are again lemmatized which is an important step since we need to obtain the command and the possible options form the man page which will be our result. Since the user query was lemmatized the man page document also need to be lemmatized to find the exact queried terms.
Once we get the command and the relevant option based on the queried terms, we display it to the user on the terminal.

Results Improvement based on User feedback:
Human and computer interaction has come a long way since the emergence of Natural Language Processing and Machine Learning, but we still have a long way ahead to make the computer understand human language like a human being. This causes the applications built on natural language processing to produce wrong results sometimes. Hence, we have built a user feedback mechanism which is used to improve the accuracy of the results. The users can give optional feedback at the end of the result, this feedback is used to improve the future results of the algorithm. Every feedback given by the user is saved along with the query and the result. When the algorithm has to choose a command, it will check the user feedback, if a greater number of users have given negative feedback, then the algorithm will choose the next highest ranking. If all the results for a combination have negative feedbacks, then it will go to the next combination to select the commands and the algorithm proceeds. When the results from all the combinations are exhausted, it will choose the one which has the least number of negative feedbacks. This help in selecting the best available command to improve the results of the model. More the application used, the better will be the results.

Results

In this part, we will demonstrate some real-time results returned by our Unix Shell Assistant. The main purpose of this demonstration is not only a validation of our command line tool but also a guide for our users on how to ask questions properly so that you can get the right feedback most efficiently. Without losing generality, we will provide examples of both correct and incorrect queries. By correct query, we mean the question was asked in a way that Natural Language Processing has no ambiguity of extracting its intention.
The first example, as it is shown in Figure 1, was to ask the bot “How can I remove the directory”. In Unix system, if we want to delete a directory along with all files under that directory, we need to do it recursively. Therefore, the command should be: rm -r /PATH/TO/DIR. Since the query phrase contains only one action verb “remove”, and one entity noun “directory”, the Unix Shell Assistant had no trouble interpreting the intention of this question. As we can see, the usage of the command “rm”, especially the recursively option “-r”, were both returned correctly onto the terminal. Similarly, if we ask the bot “How can I change file permission”, the Unix Shell Assistant

can instantly grab the words “change” and “permission” from the phrase. From the result shown in Figure 2, we know the bot is telling us to use: chmod MODE /PATH/TO/FILE.

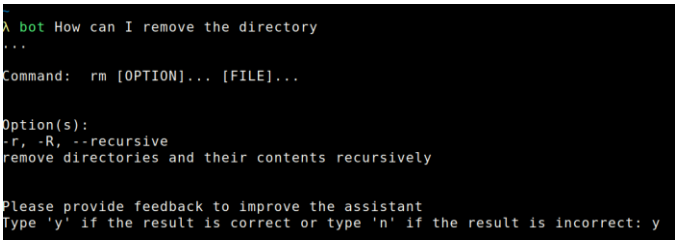

Fig.1 “How can I remove the directory”

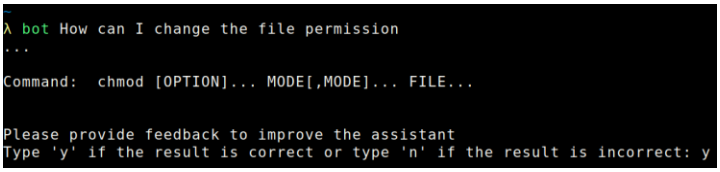

Fig.2 “How can I change file permission”

In some cases where a user has trouble pinpointing or describing the question, the bot may return undesirable solutions. In that case, a user’s feedback would largely help enhance the experience. For example, as it is shown in Figure 3, we ask the bot “How can I list the directories”. Instead of giving us: ls -d /PATH/TO/DIR/, the bot returned: dir OPTION FILE. This is because the man page that has the most occurrences of the words “list” and “directory” happened to be the page with “dir” command. Therefore, we can correct the returned result by indicating “NO” and feeding the very same question back to the bot again. This time, the bot rectified its searching results and returned us the correct syntax from man pages. Particularly, it also extracted the most related options along with the command. In some rare cases, a user can also learn from the bot’s feedback get the solution. As it is shown in Figure 4, we notice that a user asked, “How to delete the file” twice without getting what he wanted, so he modified his question by replacing “delete” with “remove”. This time, the bot immediately returned the optimal answer to this user.

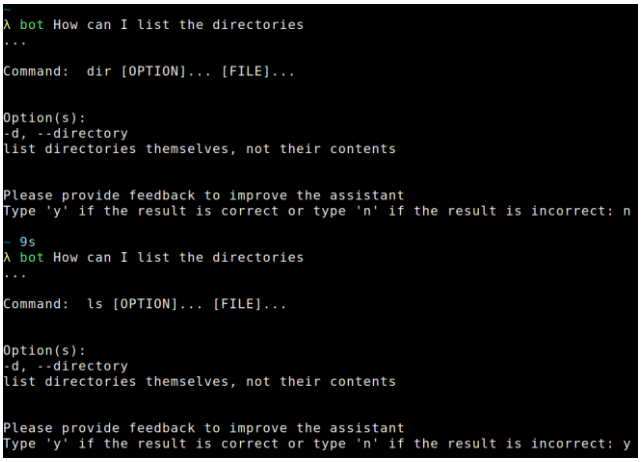

Fig.3 “How can I list the directories”

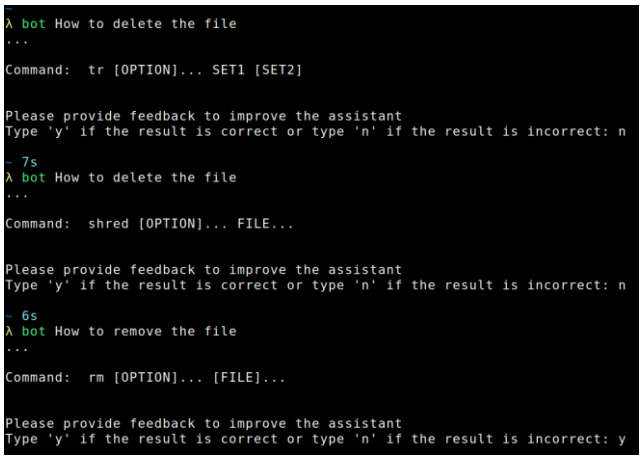

Fig.4 “How to delete the file”

The results also imply a fact that through queries, on the one hand, the bot "learns" from human language through iterations of queries. On the other hand, a user can learn from bot's feedback and make much more efficient queries at the same time. We call this a Mutual Learning Process.

Conclusion and Future Work

To summarize our work, we developed a command line assistant for Unix shell. The command line assistant takes questions related to using Unix commands and returns the usage of a command along with options. Natural language processing techniques were used in processing user inputs. Specifically, we used tokenization to extract individual words from a sentence and we performed lemmatization using TextBlob to get the root words and POS tagging using spaCy to get the entity and action of the user query. Ranking algorithms were extensively used in searching for man page with the highest frequency of keywords. To optimize the result, we add an extra module that asks a user for feedback. It helps improve the accuracy of the result through iterations of feedback. Theoretically, the bot will always return the desired result to the user, though it might take several iterations. This mechanism works in a similar way as the machine learning algorithm does. It leans from human and improves itself over time, which is exactly want we initially designed to do.

In this project, we have overcome several challenges. As we know, the details of the man page may vary based on the system. For example, different distributions of Linux machine may have different man pages; GNU and BSD have some different sets of commands implemented separately. This increases the complexity of parsing the man page. Fortunately, we use NLP tools to process input and intention words as keys to do the searching algorithms. No matter how different a man page looks like, the intention words always follow the same rule. Another challenge was dependent on the context of the user queries. In some cases, a word can be tagged as either a noun or a verb. In other cases, a query may contain more than one action words as well as multiple entity nouns. We conquered this challenge by using combinations. If a pair is defined as a verb plus a noun, alternatively an action plus an entity, we find and store all such combinations in our queries and feed them all into the searching algorithm. Therefore, we won't misrepresent a single possible intention in our result.

In the future, we plan to add more features to our Unix Shell Assistant and extend its usage to a wider range of applications. We will first refine our text mining process. The target is to have fewer iterations of feedback and a shorter waiting period for generating the results. Apart from this, we also plan to give support to multiple Linux based systems as it is crucial to satisfy users of different preferences. Of course, in the long run, we hope to add support for voice and other forms of inputs. This will help extend our product to a larger user market.

References:

[1]. Duhan, N., Sharma, A.K. and Bhatia, K.K., 2009, March. Page ranking algorithms: a survey. In Advance Computing Conference, 2009. IACC 2009. IEEE International (pp. 1530-1537). IEEE.

[2]. Natural Language Processing with Python Written by Steven Bird, Ewan Klein and Edward Loper.

[3]. https://www.nltk.org/book/

[4]. https://textblob.readthedocs.io/en/dev/

[5]. https://spacy.io/api/

[6]. https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72

[7]. https://medium.freecodecamp.org/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24