ISSN:0975-9646

Classifying Linux Shell Commands using Naive Bayes Sequence Model

Prof. Darshika Lothe[1],Pradumna Gite[2],Amit Mishra[3],Anubhav Yadav[4],Snehal Kadlag[5]

[1]M.Tech in Software Engineering,
RGPV Bhopal,India
[2,3,4,5] B.E. in Computer Engineering, Imperial College Of Engineering and Research,
Pune,Maharshatra,India

***Abstract*—Using Linux shell commands is a challenging task for most of the people new to Linux. This paper presents the idea of conversion of natural language to equivalent Linux shell command. To achieve the conversion we make use of a Naive Bayes text classifier. However there could be a case of a series of flags and combination of commands. This is handled by a sequence of Naive Bayes text classifier. Owing to the small amount of data set for every command the performance of naive Bayes is equivalent to that of the other discriminative classifiers like maximum entropy models. We improve the classification accuracy by combining the naive Bayes model with linear interpolation for predication of combination of multiple commands and flags.**

Keywords—Natural Language Processing; Linux; ; Naïve Bayes;

I. Introduction

Using Linux command line is one of the difficult parts of using Linux for new users. It is difficult to remember different commands and all the flags associated with the commands. This paper proposes the use of natural language to shell syntax conversion. Any English sentence will be converted to an equivalent Linux shell command. This involves the prediction of a combination of multiple commands and flags for a particular English statement. This conversion is achieved using a naive Bayes text classifier.

Here Naive Bayes performs faster than other discriminative classifiers like maximum entropy. Also because of a small training data set for all commands it is a reasonable choice to make use of a naive Bayes model instead of discriminative classifier. Example of Linux command

Create a new tar archive

tar cvf archivename.tar

Although a naive bayes classifier assumes conditional independence among the words in a sentence it performs relatively well for a small corpus.We make use of add one smoothing to avoid zero probabilities for newly occuring words. It works on the principle of maximum likelihood estimates.

When one of the commands is predicted by the classifier the next task is to find the combination of that command with other commands and also the flags associated with that command. This is done by evaluating the probability of several other possible combinations that can come along with the initially predicted command. For example if "find" is one command predicted by the classifier , for the sentence "search all the strings in the folder 'etc' " , the other possible combination for this command could be "grep" or "rm" or "cat" etc .To find this we calculate the probability of "find" given "grep" , "find" given "rm" and "find" given "cat" and then select the highest amongst it.

The section 2 of the paper introduces the multinomial naive bayes model. Then section 3 of the paper contains the use of the text classifier and linear interpolation of features to generate sequence of commands and flags . The section 4 concludes the paper with future scope and possible optimizations.

II. Model of Multinomial Naive Bayes

For a sentence s and class c we have by bayes theorem

$$P\left(c|s\right) = \frac{P\left(s|c\right)P\left(c\right)}{P\left(s\right)}$$

Maximum a posterior (MAP) is the most likely class

$$C_{MAP} = argmax_{c\epsilon C} P\left(c|s\right)$$

$$= argmax_{c\epsilon C} \frac{P\left(s|c\right)P\left(c\right)}{P\left(s\right)}$$

Where

$$d \rightarrow \left(x_1, x_2, x_3, ...\right)$$

P(s) can be eliminated. P(s) tells us how likely the sentence is which will be identical for all classes.

$$= argmax_{c\epsilon C} P\left(s|c\right)P\left(c\right)$$

$$C_{MAP} = argmax_{c\epsilon C} P\left(x_1, x_2, x_3, ..., x_n|c\right)P\left(c\right)$$

Here we make two assumptions. First assumption is that the position of the words does not affect the calculated value. Next we assume conditional independence amongst the words in a sentence.That means the word probabilites P(x|c) are independent given the class c.

$$P\left(x_1, x_2, x_3, ..., x_n|c\right) = P\left(X_1|c\right) * P\left(X_2|c\right) * P\left(X_3|c\right) * ... * P\left(X_n|c\right)$$

$$C_{NB} = argmax_{c_j\epsilon C} \prod_{i\epsilon position} P\left(x_i|c_j\right)$$

Here we perform Laplace smoothing to counter the problem of zero probabilities for newly occurring words.

$$P\left(x_i|c\right) = \frac{count(x_i, c)}{\sum_{x\epsilon V} count(x, c)}$$

$$P\left(x_u|c\right) = \frac{count(x_i, c) + 1}{\left(\sum_{x\epsilon V} count(x, c)\right) + V + 1}$$

The numerator in the equation is the number of times word wi came in the data set for class c. The first term in the denominator is the total number of words in the class c and V represents the total number of words in the vocabulary.

We create data sets for each command and flag containing the possible sentences that could be used for that class. When an input is provided by the user in the form of English sentence, its broken into individual words. These words act as an input to the classifier. These words are directly matched with the data sets of each class to calculate the probabilities. We then get the first possible command for the sentence.

III. Predicting sequence of commands

A Linux command could be a combination of more than one command and flags. To predict the other possible combination with the first command we create a list of all the possible combinations that can occur with the any given command. For example with the command "find" we can have combinations with "grep" , "rm", "cat" and other possible commands and flags that operate with a file found by the "find" command.

The list for every command contains its possible combinations with the command for which the list is made. Every entity in the list contains feature words related to that particular given command. For example a list for the command "find" will contain "grep" as an entity. "grep" will have feature words like "search", "find" ,"locate" and bigrams like "search word" ,"containing word" , "containing string" and also certain trigram combinations for grep.

To find the next possible combination with the initial command we provide the same English sentence as input to the classifier. However this time we find the probabilities only for those commands present in the list of the initially predicted command. The probability of every item in the list is multiplied by a confidence value for that entity of the list. The feature word of an entity in a list could be a feature word of other entities in the list as well. A confidence value tells us how likely the feature belongs to an entity as compared to the other entities.

Confidence value of an entity e is given by

$$C\left(e\right) = \sum_{i=1}^{N} f_i \lambda_i$$

Here f subscript i is a binary feature indicating the presence or absence of a feature. Lambda i gives us the weight of that feature. When the feature is a part of other entities as well we set lambda i equal to the linear interpolation of feature over trigram, bigram and unigram level.

The value of lambda i for a trigram feature is calculated as follows

$$\begin{aligned}\lambda_i f_i &= q\left(w_i|w_{i-2}, w_{i-1}\right) = q\left(f_i\right) \\ &= m_1 * q_{ml}\left(w_i|w_{i-2}, w_{i-1}\right) \\ &+ m_2 * q_{ml}\left(w_i|w_{i-1}\right) + m_3 * q_{ml}\left(w_i\right)\end{aligned}$$

Here q_{ml} is the maximum likelihood estimate of that feature.

Here

$$q\left(w_i|w_{i-2}, w_{i-1}\right) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})}$$

Here we choose m_1, m_2 ,m_3 to maximize

$$X\left(m_1, m_2, m_3\right) = \sum_{w_1, w_2, w_3} n\left(w_1, w_2, w_3\right) * log\left(q\left(w_3|w_1, w_2\right)\right)$$

Here c^1 is the number of times we found the given trigram in the validation data taken out from the data set of that class(entity).Thereby calculate the values of m_1, m_2 ,m_3 for all the trigram features in the class.

$$q\left(w_i|w_{i-1}\right) = m_1 * q_{ML}\left(w_i|w_{i-1}\right) + m_2 * q_{ML}\left(w_i\right)$$

We use the same technique for bigram features and use only the maximum likelihood estimate for the unigram features of a class (entity).This helps us to compensate for the assumption of conditional independence in naive Bayes.

Once the second command is predicted we look for the list of newly predicted command and repeat the procedure. This time if the confidence value of all the entities in the list multiplied by the naive Bayes classifier probability is below a threshold, we stop looking for further combinations.

Once we have the possible commands and flags for a sentence we generate permutations of those commands and flags. These permutations are matched with predefined sequences of possible combinations for those commands and flags. If an incorrect output is generated the user may enter the right combination which is then added to the predefined sequences.

IV. Predicting objects of interest in the sentence

Objects are words file name, string name or other words that serve as the parameters of the command. The words that follow the words like folder, file , string or disk and other similar words are the objects of interest in the sentence. However the sentences that do not contain such words preceding the object it is difficult to predict such objects.

A solution to this problem is to predict an object on the basis of the words preceding and following the object in the sentence. That is to find out the bigrams (preceding word, object) and (object, following word) in the database of the corresponding predicted command.This approach can be further enhanced by keeping information about the relation between the words.
For Example consider the sentence: Enlist the contents of the directory /applet.
Here the verb is 'enlist' and the relationship is 'contents of'
Example 2: Print the files in /etc.
Here the verb is 'Print" and the relationship is "files in"
So we keep a database of verbs and their possible relationships to predict an object.
Thereby we build a form of finite automata between verbs and relationships.

V. FUTURE SCOPE

The system can be extended to include more than two or three commands and possibly predict a larger set of flags and commands. The accuracy of the system can be increased by using word similarity tables increasing the classification accuracy.

Plug-ins can be added to the system to update status on social networking sites like twitter.

The system can be made more extensible to include a larger database for new commands. This prototype can be made into functional end user system software.

Voice control can be added to increase the flexibility of use for the end user.

VI. CONCLUSION

The model of predicting Linux commands from English sentence is a prototype which can be extended to operate most of the system through natural language. The model works effectively for predicting up to two commands. Larger sequences are also predicted correctly but with comparatively less accuracy.

ACKNOWLEDGMENT

This research would not have been feasible without encouragement and guidance of *Prof. Darshika Lothe*. We are heavily indebted to her. She patiently discussed ideas with us and gave in JSAPI grammar format and many more suggestion.

We are grateful to *Prof. Vinod S. Wadne*, Head Of Computer Department, Imperial College Of Engineering and Research for always being ready to help with most diverse problem that we have encountered along the way.

We express our sincere thanks to all staff and colleagues who have helped us directly or indirectly in completing this paper.

We pay our respects and love to our parents and all other family members and friends for their love and encouragement throughout our career.

REFERENCES

[1] http://www.baselinemag.com/c/a/Business-Intelligence/40-Fast-Facts-on-Linux-727574/

[2] http://www-nlp.stanford.edu/fsnlp/

[3] http://www.cs.colorado.edu/~martin/slp.html

[4] Part-Of-Speech Tagging using Neural network Ankur Parikh 2009

[5] Part-of-Speech tagging based on artificial neural networks Salvador Tortajada Velert

[6] A Fully Bayesian Approach to Unsupervised Part-of-Speech Tagging Sharon Goldwater

[7] Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? Christopher D. Manning

[8] Kristina Toutanova and Christopher D. Manning. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. *In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70.*

[9] Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. *In Proceedings of HLT-NAACL 2003, pp. 252-259..*