Autocompletion for Network Configurations

Ahsan Mahmood, Aaron Gember-Jacobson

Colgate University

Abstract

In this proposal, we state the need for an auto-completion engine for writing network configurations. The tools and technology available today are simply inadequate to help network operators in this process. We propose a simple, yet powerful model inspired by code completion techniques and NLP research. We show how the current state of the model gives encouraging results. We also outline additional work that is required to tune our model specifically for network configurations before we can truly realize our goal. Once completed, we believe our engine will be a strong first step in creating a holistic tool similar to IDEs that can assist network operators.

1 Introduction

Research has shown that configuring control planes can be extremely complex in modern networks [1]. Consequently, this causes configurations to be prone to errors, most of which are only uncovered during operation after a failure has already dealt significant damage [4]. Network operators thus try to minimize errors by reusing existing configurations that have been known to work in the past. Most routers offer some form of simple built-in Command Line Interface (CLI), where operators can use vendor specific languages to update router configurations. Often, these CLIs will offer rudimentary tab completion, where they will alphabetically suggest all the options available for a token from the invocation point. These suggestions are often unhelpful as the user then has to search for the desired completion.

We propose a different approach that can serve to complement existing techniques for writing routing configurations by consider the problem of writing network configurations to be analogous to writing software code. Most configurations are written using vendor specific languages, that make use of rules and keywords similar to traditional programming languages. We envision an interactive system inspired by code completion engines that could be invoked by network operators as they are writing router configurations to offer them suggestions for what to put in next, or list the options available from the invocation point.

Recent research on software systems has shown that codebases tend to contain regularities, much like natural languages [2]. This has motivated further research on using traditional Natural Language Processing techniques for code completion and token suggestion, resulting in fairly accurate models [2, 3]. We hypothesize a similar regularity for network configurations, especially since they tend to be homogeneous by design, reusing the same set of keywords/tokens. Our analysis of router configurations from a large research university showed that configurations shared between

85% and 99% of tokens across different routers. This prompted us to explore simple NLP techniques that could leverage these token similarities to produce useful suggestions or completions. Our preliminary results show that using an off-the-shelf NLP algorithm with minor modifications, can give us up to 93% accuracy for some configurations.

2 Preliminary Work

Prior research in computer networking hints that we should expect network configurations to share a common set of tokens. As Hindle,*et al.* 2012 [2] pointed out, regularities in bodies of texts can be easily exploited by NLP techniques. We took configurations from a large university network and split up each configuration file into a list of tokens. Tokens included all keywords and subnets with punctuation and newline characters stripped off. For every file we then plotted the percentage of tokens that exist in other router configuration files.

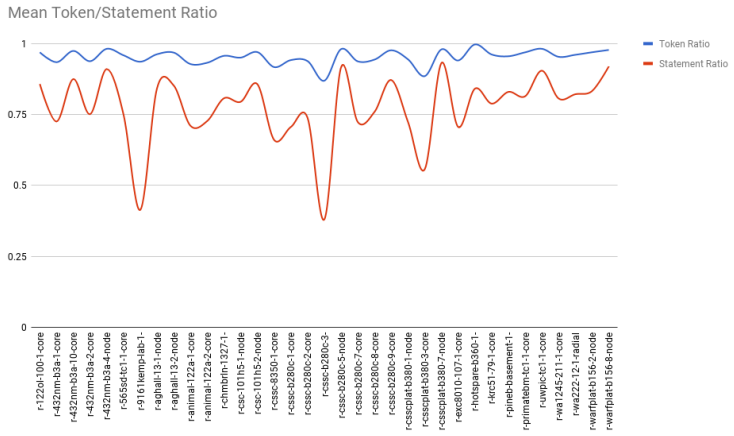

Figure 1: The plot shows how many tokens and statements a router configuration holds in common with the rest of the network. The data was taken from a large research university.

Our results show that most of each file could be rebuilt from existing statements in routing configurations due to the amount of tokens they share. Given our results, and the observations made by [1] about how networks are configured, we can confidently hypothesize that most token suggestions can be generated from analyzing other existing configurations. This effectively makes all router configuration histories a part of the search space for our model. We use an N-gram model to generate predictions using likelihood estimators to score our n-grams. In particular, we made use of bigrams and trigrams, the latter of which performed consistently better.

We preprocessed the data and added placeholders for certain tokens like IP addresses, subnet masks etc. This helped to reduce some noise and improved our accuracies.

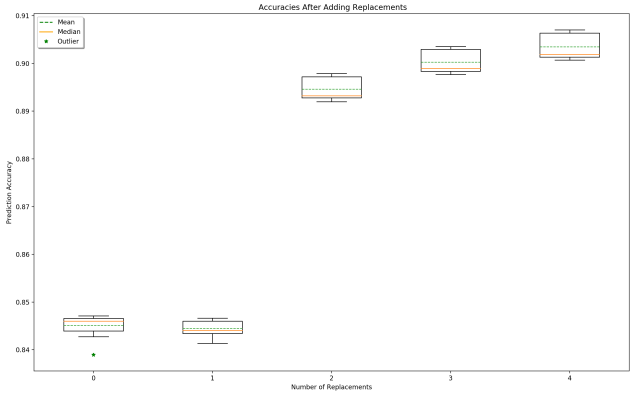

Figure 2: Accuracy increase as we utilize placeholders for noisy tokens

3 Results

To test the accuracy of our model, we perform Leave One Out (LOO) Cross Validation. This form of cross validation involves usingone observation as the validation set and the remaining observations as the training set. A suggestion is marked as successful when the correct prediction lies within the top three results generated by the model. We analyzed various large university networks, varying sample sizes and number of devices

Our analyses help direct our attention towards areas of improvements for the model. The variance seen in our device analysis suggests that having different modles for router pf different "roles" could help improve prediction accuracies.

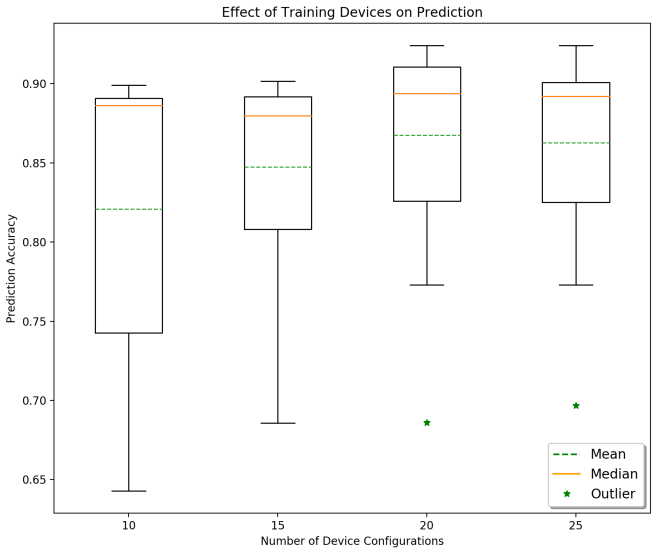

Figure 3: The bar charts show the accuracy of the model for each set of network configurations used as the validation set.

4 Conclusion

Compared to software developers, network operators are often left neglected when it comes to development tools. Our work tries to bridge that gap by providing a simple completion engine that could be incorporated into a more extensive tool. Our initial findings show that we can get fairly respectable accuracies with off-the-shelf NLP techniques. However, we propose a myriad of refinements to the model that could potentially improve our accuracies to desirable numbers. We plan to implement these features as part of a complete senior thesis.

5 Acknowledgments

I would like to thank Professor Aaron Gember-Jacobson for his continuing guidance and support throughout the semester. I would also like to thank the Colgate Computer Science department for allowing this independent study to be carried out (and hopefully continued).

References

[1] T. Benson, A. Akella, and D. Maltz. Unraveling the complexity of network management. pages 335–348, 2009.

[2] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. T. Devanbu. On the naturalness of software. *Commun. ACM*, 59(5):122–131, 2016.

[3] V. Raychev, M. Vechev, and E. Yahav. Code completion with statistical language models. *SIGPLAN Not.*, 49(6):419–428, June 2014.

[4] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy. An empirical study on configuration errors in commercial and open source systems. pages 159–172, 2011.