Smarter Code Autocomplete

Adrien Fallou James Gippetti Susan Tu

CS221

afallou | jgippetti | sctu @stanford.edu

November 12, 2014

1 Progress Report

Our project is motivated by the popularity of autocompletion plugins such as Jedi (for Python autocompletion in Vim)[**?**] andVisual Studio's Intellisense. Our goal is to build a smarter, language-agnostic autocomplete whose suggestions are (mostly, if not always) consistent with scope rules of the relevant language. We hope that applying AI techniques will allow us to better predict (beyond obeying rules such as the grammar and scoping rules of the language) the intention of the programmer.

We implemented a baseline model based on the n-gram scoring that was given to us in assignment 3. If we are trying to predict the last token of a 3-gram, for example, we simply compute the score for all possible 3-grams (i.e., we try all possible tokens as the 3rd token) and select the one that gives us the lowest score. (We obtain our tokens by splitting on whitespace and removing underscores.)

We ran this baseline on the Django web framework, which contains approximately 130,000 lines of code, almost entirely in Python. We trained on 80% of this text. In the remaining text, we chose 10000 $n-1$-grams and for each attempted to predict the next word. We considered the complete source code to be the oracle, and we counted our prediction as correct only if it exactly matched the next token. For $n = 2$, we were correct 0.8% of the time. Given that there are about 17000 unique tokens, if we select the next token randomly, we would expect to be correct only $6 * 10^{-3}\%$ of the time. So our very simple baseline model does better than random, but it is very far from the oracle and very far from useful. We also tried this for $n = 3$ and were correct 0.27% of the time. The following table shows a few reasonable autocompletions that our simple model found:

Selected token(s)	Predicted next token
while	not
def	init
for x	in

We should likely try this with higher n and smoothing (such as Laplace, additive discounting, or Kneser Ney [**?**]) to account for the fact that for higher n, it may be that the ngram has never been seen before. We will also try comparing against a less optimistic oracle: It seems unlikely that any autocomplete tool would be able to correctly predict the next token 100% of the time (since this would be equivalent to being able to automatically write code based on nothing more than than existing code), so another option for the oracle might be a program that attempts to predict a token based on all the text, i.e., both the code that comes before and the code that comes after the missing token. Furthermore, while we weren't able to obtain this prior to the proposal due date, we plan to look at some existing work done by one of Professor Liang's former students on smarter autocomplete as we try to build a better model.

Our repository is at `https://github.com/afallou`

References

[1] `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5431761`

[2] `https://github.com/davidhalter/jedi-vim`

[3] `http://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf`