API Code Recommendation using Statistical Learning from Fine-Grained changes

Summary

This paper purposes a new way to do API code recommendation by using statistical learning to look at fine grained changes on historical data. The main idea behind the approach is that when developers are doing changes to existing code, it's highly likely to co-changed elements are highly related. The result of the paper reflects the idea. Compare to traditional approaches, this paper's approach showed much higher accuracy when recommending results. However, due to the infrastructure of the approach there exists out of vocabulary problems. The approach only look at historical data, therefore could not recommend any result that does not appear in history commits. The paper also showed that if the learning data is on the developer's historical commits, it's able to predict the developer's future usage much higher.

Things I would like to see discussed

- Within project recommendation. It seems to me like the learning and testing focused on API called of JDK libraries. I wonder when dealing with calls within the project could this approach be used.

- The accuracy seems to be plateaued at a certain percentage (~80%). I wonder given a human who is familiar with the development but not the actual author of the code what percentage could be the correctness be.

- When a developer is writing code. Traditional auto-complete functionality purely based on phrase are already powerful enough. What kind of improvements does API recommendation bring over these.