

Learning from Large Codebases

Raychev V. Learning from Large Codebases, 2016.

Natalia Korepanova

January 10, 2018

Learning from Large Datasets

- natural language processing (Google Translate)
 - computer vision (Facebook photo service)
 - recommendation systems
- etc.

“Big Code”:

GitHub (<https://github.com/>),

BitBucket (<https://bitbucket.org/>),

and others.

However, learning from large datasets has not had great impact on programming tools!

Main Thesis Question

How to leverage large datasets of code to build practical programming tools?

Main Goals

- Usability of the resulting tools (e.g. scalability and precision)
- Generality of the underlying techniques

Core Research Challenges

- **Creating a suitable probabilistic model for programs**
Solution: design of program analysis that capture semantic information



Figure 1.1: Impact of the amount of training data and probabilistic model on the accuracy of the SLANG code completion system.

- **Learning from a large corpus of training data**
Solution: efficient learning procedures

Tools Showcase

JSNICE (<http://jsnice.org/>)

Takes as input JavaScript code, renames its local variables and function parameters, and outputs semantically equivalent JavaScript code, but more human readable.



Figure 1.3: Histogram of query sizes to <http://jsnice.org/> sent by its users in the period May 10, 2015 – May 10, 2016.

Tools Showcase

SLANG

API code completion system, capable of predicting one or multiple method invocations at once.

```
Activity currentActivity;  
  
void test(boolean no_bars) {  
    WebView view = new WebView(ctx);  
    if (no_bars) {  
        view.setVerticalScrollBarEnabled(false);  
        view.setHorizontalScrollBarEnabled(false);  
    }  
    view.  
}
```



The screenshot shows a code editor with a snippet of Java code. The cursor is positioned at the end of the line 'view.', which is highlighted with a blue background. A dropdown menu is open, displaying two suggestions: 'currentActivity.setContentView(View view): void - Activity' and 'view.loadUrl(String url): void - WebView'. The second suggestion is selected, indicated by a green dot to its left.

Figure 1.4: A code completion plugin capable to predict multiple statements at once.

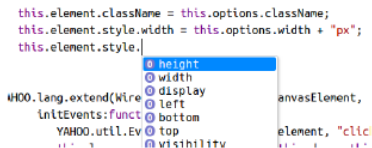
Tools Showcase

DEEPSYN

General technique that automatically synthesizes code completion systems described by a domain specific language.

```
    this.element.className = this.options.className;
    this.element.style.width = this.options.width + "px";
    this.element.style.|

```

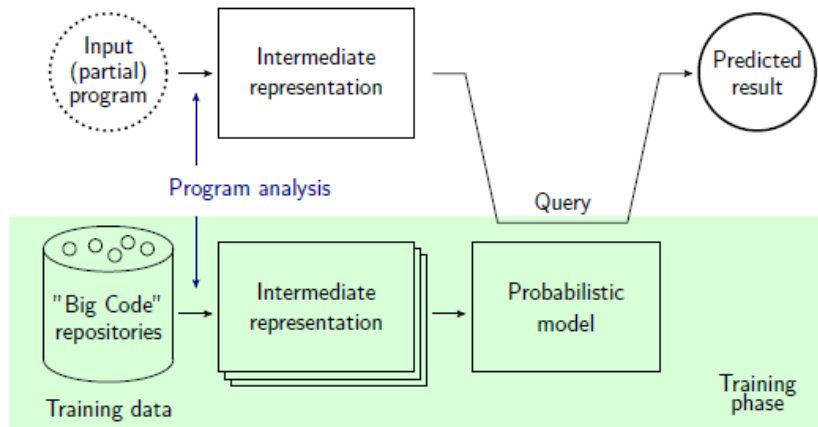


```
YAHOO.lang.extend(Wire, CanvasElement,
    initEvents: function() {
        YAHOO.util.Event.addListener(
            this.element, "click",
            this, this.handleClick);
    }
);
```

Figure 1.5: The DEEPSYN tool for completing JavaScript code. In this snippet of code, standard type analysis cannot resolve the types of the used variables. Our statistical model predicts that if width was set, height may need to be set as well.

*This technique synthesized a code completion system for JavaScript that predicts around 50% of API calls and almost 40% of field access correctly. (Good result due to lack of static type information in JavaScript.)

General Architecture of "Big Code" tools



Problem Dimensions

Dimensions	Instantiations in this thesis
Application	Deobfuscation (§2) Type Prediction (§2) Code Synthesis (§3, §5)
Program analysis	Scope Analysis, Type Analysis (§2) TypeState and Alias Analysis (§3) Domain Specific Languages (§5)
Intermediate representation	Factor Graphs (§2) Sequences (§3, §5)
Probabilistic model	CRF, Structured SVM (§2) Statistical Language Models (§3, §5)
Query	MAP Inference (§2, §3)

Challenges

- **Vast number of labels**

Trying every possible label at query time is practically infeasible.

- **Dependent predictions**

To predict the names of n variables, we need to consider up to n^k possible assignments if a variable name ranges over a set of size k .

- **Estimating probabilities and learning**

Valid probabilities need to be positive and sum to one over all possible outcomes. With the large number of predictions and specifically the presence of constraints it becomes intractable to even count the number of possible predictions.

- **Feature engineering**

In JSNICE and SLANG tools feature functions are defined manually. The synthesis technique behind DEEPSYN system replaces the feature engineering process.

Main Contributions (1)

- A new approach for probabilistic prediction of program properties and the JSNICE system based on this approach. This is the first connection of conditional random fields to the problem of learning from programs.
- A connection of static analysis with statistical language models, implemented in the SLANG tool and illustration of application of recent advances in deep learning to the problem of learning from programs.

Main Contributions (2)

- A new framework for program synthesis with noise, connecting traditional program synthesis with statistical learning, that:
 - ① enables existing programming-by-example engines to deal with noise,
 - ② provides a fast procedure for empirical risk minimization in machine learning
 - ③ serves as a basis for developing learning procedures from “Big Code” with high precision.
- A new learning approach based on the developed framework for program synthesis, implemented in the DEEPSYN system, which precision significantly improves over existing approaches.