Discriminative Models for Predicting Program Properties (Part 1)

Raychev V. Learning from Large Codebases, 2016. Chapter 2.

Natalia Korepanova

January 10, 2018

General Scheme

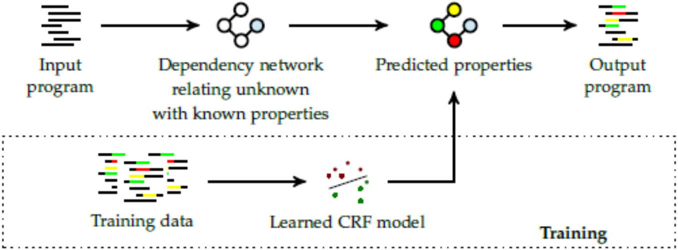


Figure 2.1: Statistical Prediction of Program Properties.

Properties:

- *syntactic* (e.g. variable names);
- *semantic* (e.g. optional type annotations).

CRF = conditional random fields

Input Program

(a) JavaScript program with minified identifier names

```
function chunkData(e, t) {
  var n = [];
  var r = e.length;
  var i = 0;
  for (; i < r; i += t) {
    if (i + t < r) {
      n.push(e.substring(i, i + t));
    } else {
      n.push(e.substring(i, r));
    }
  }
  return n;
}
```

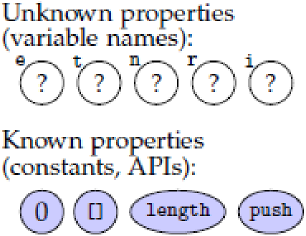
Output Program

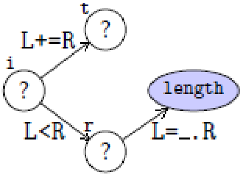
(e) JavaScript program with new identifier names (and type annotations)

```
/* str: string, step: number, return: Array */
function chunkData(str, step) {
  var colNames = [];
/* colNames: Array */
  var len = str.length;
  var i = 0;  /* i: number */
  for (;i < len;i += step) {
    if (i + step < len) {
      colNames.push(str.substring(i, i + step));
    } else {
      colNames.push(str.substring(i, len));
    }
  }
  return colNames;
}
```

Example: Name Inference Procedure (Steps 1, 2)

```
function chunkData(e, t) {
  var n = [];
  var r = e.length;
  var i = 0;
  for (; i < r; i += t) {
    if (i + t < r) {
      n.push(e.substring(i, i + t));
    } else {
      n.push(e.substring(i, r));
    }
  }
  return n;
}
```

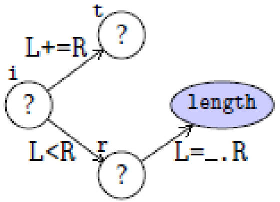




`i += t` $\rightarrow \langle i, t, L+=R \rangle$
`var r = e.length` $\rightarrow \langle r, length, L = \_.R \rangle$
`i < r` $\rightarrow \langle i, r, L < R \rangle$

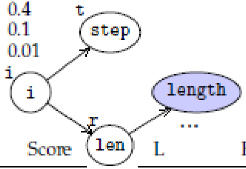
* Here and further arrows in a dependency network mean the direction of relation, not the direction of dependence. If two nodes in a relation they are dependent from each other.

Example: Name Inference Procedure (Step 3)



$\Longrightarrow$

L	R	Score
i	step	0.5
j	j	0.4
i	j	0.1
u	q	0.01



L	R	Score
i	len	0.8
i	length	0.6

L	R	Score
length	length	0.5
len	length	0.4

Output of the training phase

Maximum a Posteriori (MAP) inference $\longleftrightarrow$ Maximize the sum of scores

Notation: Programs

Let X be a set of all possible programs in some language, and $x \in X$ be a program.

$n, m : X \to \mathbb{N}$, where

- $n(x)$ is the total number of elements (e.g. variables) of a program x for which we are interested in inferring properties,
- $m(x)$ is the total number of elements with known properties of a program x.

Notation: Labels, Properties, Predictions

Let

$Labels_U$ denotes all possible values that a predicted property can take,
$Labels_K$ denotes a set of values that a known property can take,
$Labels = Labels_U \cup Labels_K$ denotes the set of all property values.

For a program x, the vector $z^x = \{z_1^x, ..., z_{m(x)}^x\}$ denotes the set of properties that are already known, where $z_i^x \in Labels_K$ for $i = 1, ..., m(x)$.

The notation $y = (y_1, ..., y_{n(x)})$ is used to denote a vector of predicted program properties, where $y \in Y$ and $Y = (Labels_U)^*$ in general.

Problem Definition

Let $D = \{\langle x^{(j)}, y^{(j)} \rangle\}_{j=1}^{t}$ denote the training data: t programs annotated with corresponding program properties.

Goal

Learning a model that captures the conditional probability $Pr(y|x)$.

Prediction (MAP or Maximum a Posteriori query)

Given a new program x, find $y = \arg\max_{y' \in \Omega_x} Pr(y'|x)$,
where $\Omega_x \subseteq Y$ describes the set of possible assignments of properties y' for the program x.

Ω_x allows restricting the set of possible property assignments and is useful for problem-specific constraints.

Conditional Random Fields (CRFs)

A model for the conditional probability of labels y given observations x is called **(log-linear) conditional random field**, if it is represented as:

$$Pr(y|x) = \frac{1}{Z(x)} \exp(score(y, x)),$$

with

- the *partition function*

  $$Z(x) = \sum_{y \in \Omega_x} exp(score(y, x)),$$

  which returns a real number depending only on the program x;
- $$score(y, x) = \sum_{i=1}^{k} w_i f_i(y, x) = w^T f(y, x),$$

  where f is a vector of *feature functions* $f_i : Y \times X \to \mathbb{R}$ and w is a vector of *weights* w_i.

Making Predictions for Programs with CRFs

Step 1: Build dependency network

Step 2: Define feature functions

Step 3: Score a prediction y

Dependency Network

Let *Rels* be the set of all element relations.

A multi-graph $G^x = \langle V^x, E^x \rangle$ is called a **dependency network** of the program x if

- $V^x = V_U^x \cup V_K^x$ denotes the set of program elements and consists of elements for which we would like to predict properties V_U^x and elements with known properties V_K^x;
- the set of edges $E^x \subseteq V^x \times V^x \times Rels$ denotes the fact that there is a relationship between two program elements and describes this relationship.

Feature Functions

Let $\{\psi_i\}_{i=1}^{k}$ be a set of **pairwise feature functions** s.t. $\psi_i : Labels \times Labels \times Rels \to \mathbb{R}$ scores a pair of program properties when they are related with the given relation.

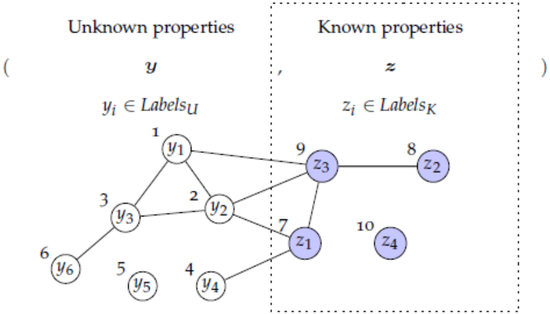
$$\psi_{example}(l_1, l_2, e) = \begin{cases} 1 & \text{if } l_1 = \mathtt{i} \text{ and } l_2 = \mathtt{step} \text{ and } e = \mathtt{L+=R} \\ 0 & \text{otherwise} \end{cases}$$

Let the assignment vector $A = (y, z^x)$ be a concatenation of the unknown properties y and the known properties z^x in x, and the property of the j'th element of vector A is accessed via A_j. Then the **feature function**[1] f_i is defined as:

$$f_i(y, x) = \sum_{\langle a,b,rel\rangle \in E^x} \psi_i\big((y, z^x)_a, (y, z^x)_b, rel\big).$$

[1] feature functions are defined independently of the program being queried

Edges and Dependencies of Program Properties



- Predictions for node 5 can be made independently.
- All undirected paths from node 2 to node 4 go through nodes with known properties, therefore the properties for them can be assigned independently (conditional inference property of CRFs).

MAP Inference in CRFs

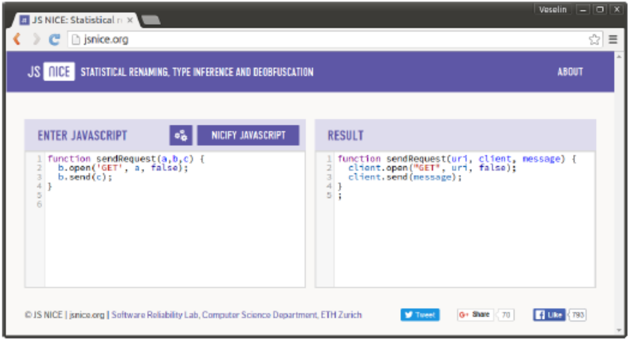
$$y = \arg\max_{y' \in \Omega_x} Pr(y'|x)$$

$$\Updownarrow$$

$$y = \arg\max_{y' \in \Omega_x} score(y', x)$$

A naive but inefficient way to solve this query is to score all possible $y' \in \Omega_x$.
Other exact and inexact inference algorithms exist, but they are too slow to be usable for programs.

JSNice (http://jsnice.org/)



Previous Approaches to Deobfuscation

- Application of certain predefined fixes to the names.
- Probabilistic models for prediction of one identifier name in the context of other good identifiers.

JSNice (http://jsnice.org/)

Existing Java Script Extensions Adding Optional Type Annotations

- TypeScript
- Google Closure Compiler

These extensions help discover type errors and improve code documentation, but require manual effort.

Application 1: Probabilistic Name Prediction

Goal

Predicting the names of local variables in a given program x.

V_K^x = all constants, object properties, methods and global variables of the program x.
$Labels_K = JSConst \cup JSNames$,
where *JSNames* is a set of all valid identifier names, and *JSConst* is a set of possible constants.

V_U^x = all local variables of the program x.
$Labels_U = JSNames$.

Application 2: Probabilistic Type Annotation Prediction

Goal

Predicting the type annotations of functions parameters. Important for languages lacking type annotations (e.g. JavaScript).

Simplified Language

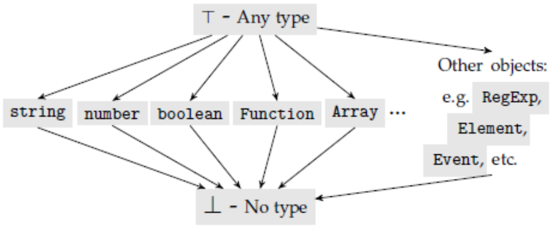
Expression: $expr ::= val \mid var \mid expr_1(expr_2) \mid expr_1 \odot expr_2$

Value: $val ::= \lambda val : \tau.expr \mid n$

$n \in JSConsts$,
var ranges over the program variables,
$\odot$ ranges over binary operators,
$\tau = JSTypes$.

Probabilistic Type Annotation Prediction

$JSTypes = \{?\} \cup L$, where ? stands for unknown type and L is a complete lattice of JavaScript types.



$JSTypes$ is built during training, therefore is finite.

Probabilistic Type Annotation Prediction

$[]_x : expr \rightarrow JSTypes$ - obtaining the type of a given expression in a given program. A shortcut for $[]_x(e) = [e]$ when program x is clear from the context.

$$V_U^x = \{e \mid e \text{ is } var, [e] = ?\}$$

$$Labels_U = JSTypes$$

$$V_K^x = \{e \mid e \text{ is } expr, [e] \neq ?\} \cup \{n \mid n \in JSConsts\}$$

$$Labels_K = JSTypes \cup JSConsts$$

$$\Omega_x = (JSTypes)^{n(x)}$$

Relating Expressions (Syntactic Relationship)

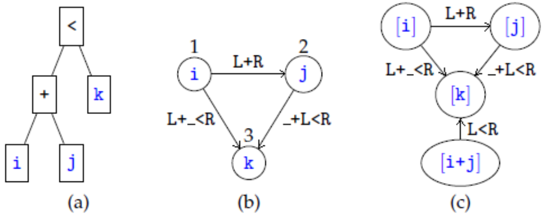


Figure 2.5: (a) the AST of expression i+j<k, and two dependency networks built from the AST relations: (b) for name predictions, and (c) for type predictions.

$$rel_{ast} ::= rel_L(rel_R) \mid rel_L \odot rel_R$$

$$rel_L ::= L \mid rel_L(\_) \mid \_(rel_L) \mid rel_L \odot \_ \mid \_ \odot rel_L$$

$$rel_R ::= R \mid rel_R(\_) \mid \_(rel_R) \mid rel_R \odot \_ \mid \_ \odot rel_R$$

*AST stands for Abstract Syntax Tree

Aliasing Relations (Semantic Relationship)

Let $alias(e)$ denotes the set of expressions that may alias with the expression e.

`ARG_TO_PM` relationship: relates arguments of a function invocation with parameters in the function declaration.

Transitive aliasing relationship $(r, \texttt{ALIAS})$: let a and b related via the relationship r which ranges over the grammar defined earlier. Then for all $c \in alias(b)$ where c is a variable, we include the edge $(a, c, (r, \texttt{ALIAS}))$.

Function Name Relationship

$(f, g, \texttt{MAY\_CALL})$: relates a function name f with names of other function g that f may call.

$(f, \mathit{fld}, \texttt{MAY\_ACCESS})$: relates a function name f with object fields fld to which this function has an access.

Obtaining Pairwise Feature Functions

$$all\_features(D) = \bigcup_{j=1}^{t} \{ ((y^{(j)}, z^{x^{(j)}})_a, (y^{(j)}, z^{x^{(j)}})_b, rel) \mid (a, b, rel) \in E^{x^{(j)}} \}$$

Let the $all\_features(D) = \{\langle l_i^1, l_i^2, rel_i\rangle\}_{i=1}^{k}$. Then the pairwise feature functions are defined as:

$$\psi_i(l^1, l^2, rel) = \begin{cases} 1 & \text{if } l^1 = l_i^1 \text{ and } l^2 = l_i^2 \text{ and } rel = rel_i \\ 0 & \text{otherwise} \end{cases}$$

Next Step: Learning feature weights $\{w_i\}_{i=1}^{k}$.

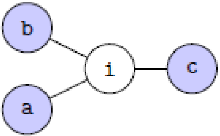
Tractable CRF Model

The CRF model is too general to ensure practical applicability, and solving MAP inference may be undecidable.

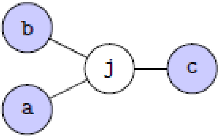
Additional Restrictions Used to Make the Problem Tractable

1. The predictions y is a vector of a given size $n(x)$ known before a prediction made.
2. Feature functions f are introduced through pairwise feature functions that relate only pairs of properties.
3. Pairwise feature functions are indicator functions in oder to enable fast inference.

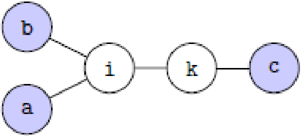
Illustration of Restriction 1



(a) Initial configuration



(b) A possible candidate configuration in the search space



(c) Configuration outside of the search space