

Machine Learning for Context-Sensitive Search in Agda



Andrej Bauer¹³, Matej Petković¹², Ljupčo
Todorovski¹²

University of Ljubljana, Faculty of Mathematics and Physics

Jožef Stefan Institute, Ljubljana, Slovenia

Institute for Mathematics, Physics and Mechanics, Ljubljana, Slovenia



Agenda

- Motivation
- Data Transformation
- Two learning approaches
- Efficiency Issues and Solutions



... is a dependently typed functional programming language that is mostly used as a proof assistant

```
data _≤_ : ℕ → ℕ → Set where
  z≤n : {n : ℕ} → zero ≤ n
  s≤s  : {n m : ℕ} → n ≤ n → suc n ≤ suc m

one : ℕ
one = suc zero

two : ℕ
two = suc one
```

```
proof : one ≤ two
proof = s≤s (z≤n)
```



We formulated a theorem ...

... i.e., the type of an expression, but – when writing the proof (the body of the expression)

we do **not remember/ know** which lemma from the library to use.

A **recommender system** should suggest us appropriate candidates given the **current context**.

```
proof : one ≤ two
proof = ?
```

```
proof : one ≤ two
proof = s ≤ s (?)
```



Machine learning (ML) to the rescue

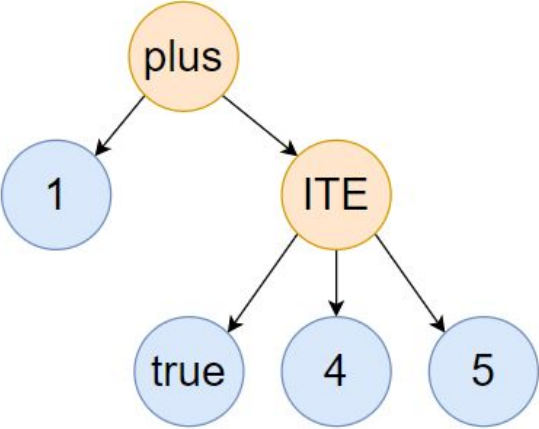
The task of learning a model that gives us suggestions can be formalized in ML as

- creating a recommender system,
- solving a multi-label classification.

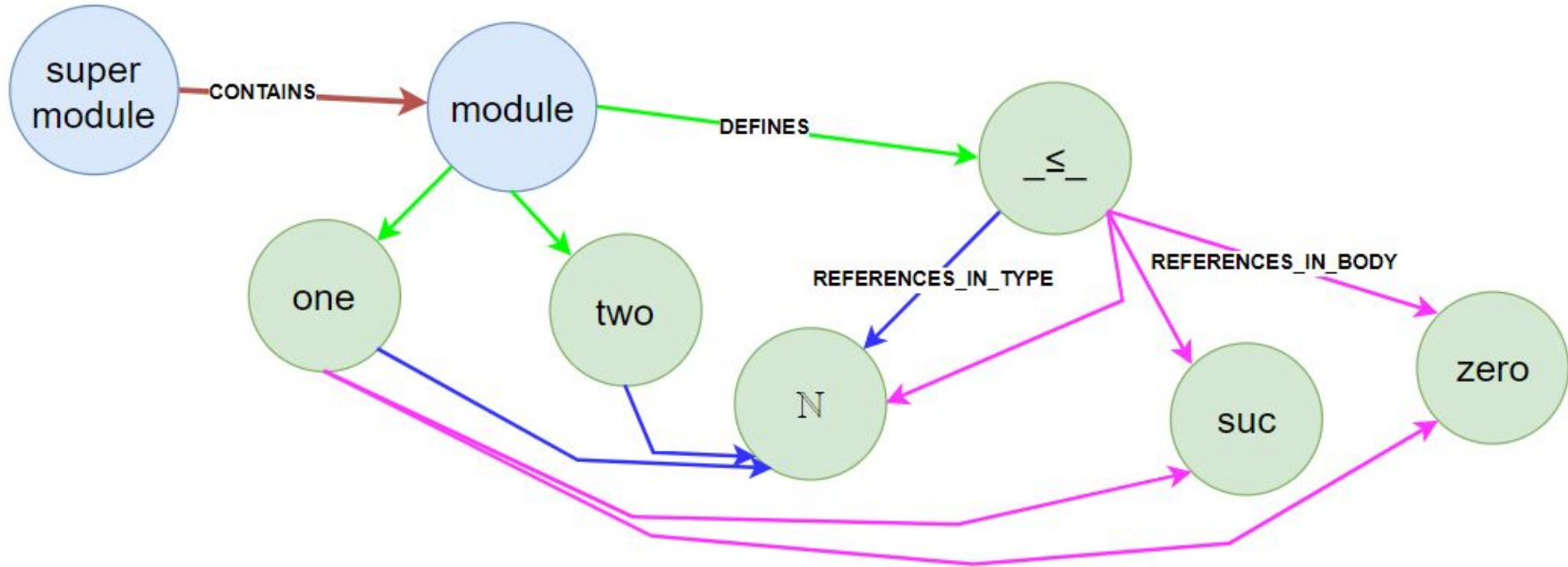
but first ... we need to represent Agda code in **machine-learnable** format.



Conversion of a definition

Agda code	<code>1 plus (if true then 4 else 5)</code>
Abstract Syntax Tree	 <pre>graph TD; plus((plus)) --> 1((1)); plus --> ITE((ITE)); ITE --> true((true)); ITE --> 4((4)); ITE --> 5((5));</pre>
s-expression	<code>(plus (1) (if_then_else_ (true) (4) (5)))</code>

Conversion of a library



where each green node represents an AST.



Agda libraries: stdlib, unimath

function	20 000
constructor	1000
record	600
data	150
axiom	100
primitive	100
sort	10



Some statistics: graphs

STDLIB:

- 16,193 nodes
- 223055 edges
- 7 weak components:
 - max: 16,187
- 14,459 strong components:
 - max: 30

UNIMATH:

- 11,846 nodes
- 176,330 edges
- 7 weak components:
 - max: 11,838
- 11,355 strong components:
 - max: 16



Recommender system

Given some **database** of candidates, use the context that user provided (**a query**) to find an appropriate response.

In our case:

- database = an Agda library of definitions
- query = a partially written definition



Recommender system

Given a definition that is still missing some connections to the other definitions in a library, **identify the missing edges**.

To make a prediction/suggestion, use

- the definition itself (e.g., words used),
- the position of the definition in the graph.



Recommender system

The recommendations are currently made in two stages:

- compute the fitness of every other definition in the library
- sort the candidates with respect to their fitness and return the top K ones



Efficiency issues #1

1. $O(n^2)$ fitness computations
2. Potentially very expensive fitness definition:
 - direct comparison of ASTs

Currently, we do not need the solution for 1, but at some point approximations would be needed.

The second issue was addressed with embeddings.



Embeddings

Most of the machine learning models can handle only vectors of numbers.

Instead of developing new methods (that would handle words, images or graphs), researchers tend to develop **embeddings** - mappings that map complex data to \mathbb{R}^n .



Relevant embeddings

- word2vec (Mikolov et al.)
- node2vec (Grover et al.)
- code2vec (Alon et al.)
- ...



Efficiency issues #2

Those embeddings are based on neural networks, which need **a lot** of examples to achieve great performance.

We use the pretrained models (instead of, for example, learning English from scratch) – if possible.



Multi-label classification

- Classification: given a finite set of **mutually exclusive** classes, assign a given object the correct class
 - *assign a person their blood type*
- Multi-label classification: given a finite set of **labels**, assign a given object **all the relevant labels**
 - *assign an image all the objects it contains*



Multi-label classification in our case

- Every label in the label corresponds to a definition in a library
- A label d is relevant for a definition D iff D references d
- The context is provided by a fixed set of features that describe the definition. Model uses them to decide which labels are relevant
- Those features can be anything: embeddings, graph-based ...



Preliminary experiments

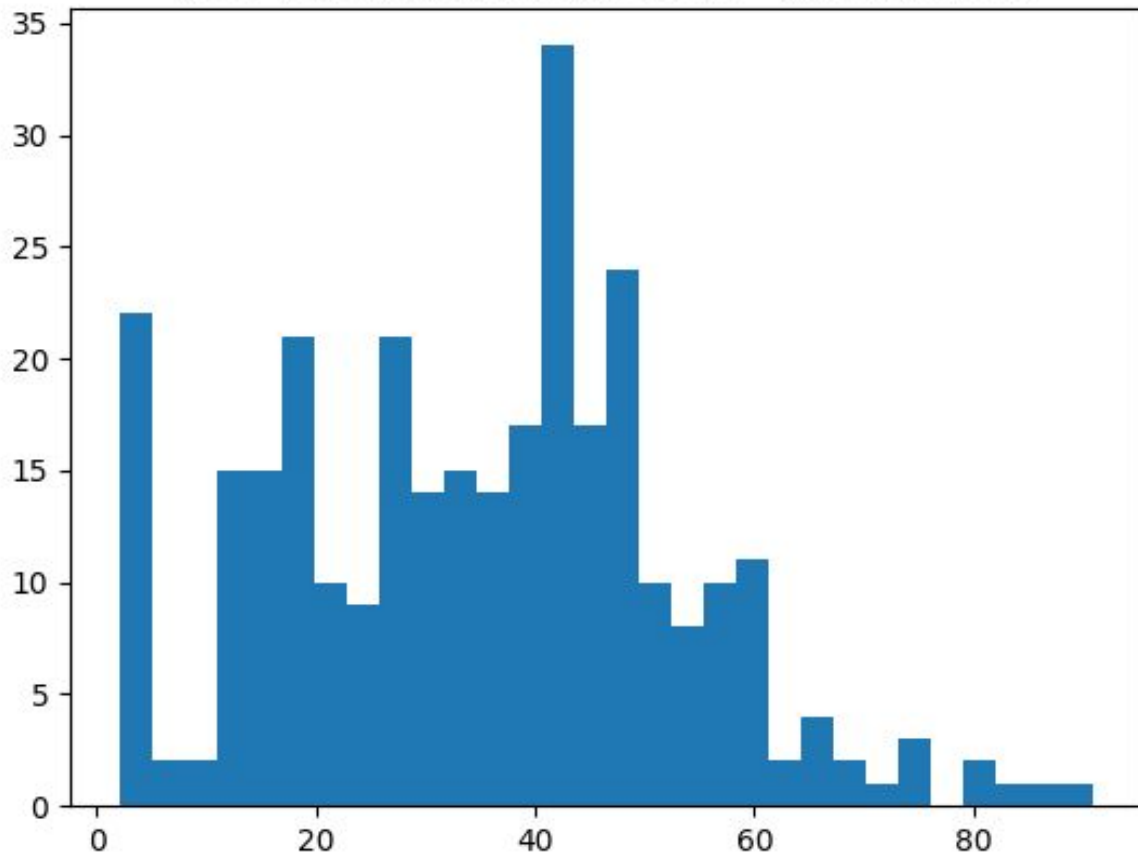
Nearest neighbours: bag of words	numba
Nearest neighbours: TFIDF	Sparse representation (and distance computation)
Nearest neighbours: word2vec embeddings	Pretrained models
Analogies (<code>left-thm</code> uses <code>left-lemma</code> , then <code>right-thm</code> uses ??)	$O(E V ^2)$: extensive vectorisation and usage of C-compiled libraries
Neighbours of neighbours	C-compiled libraries
...	...



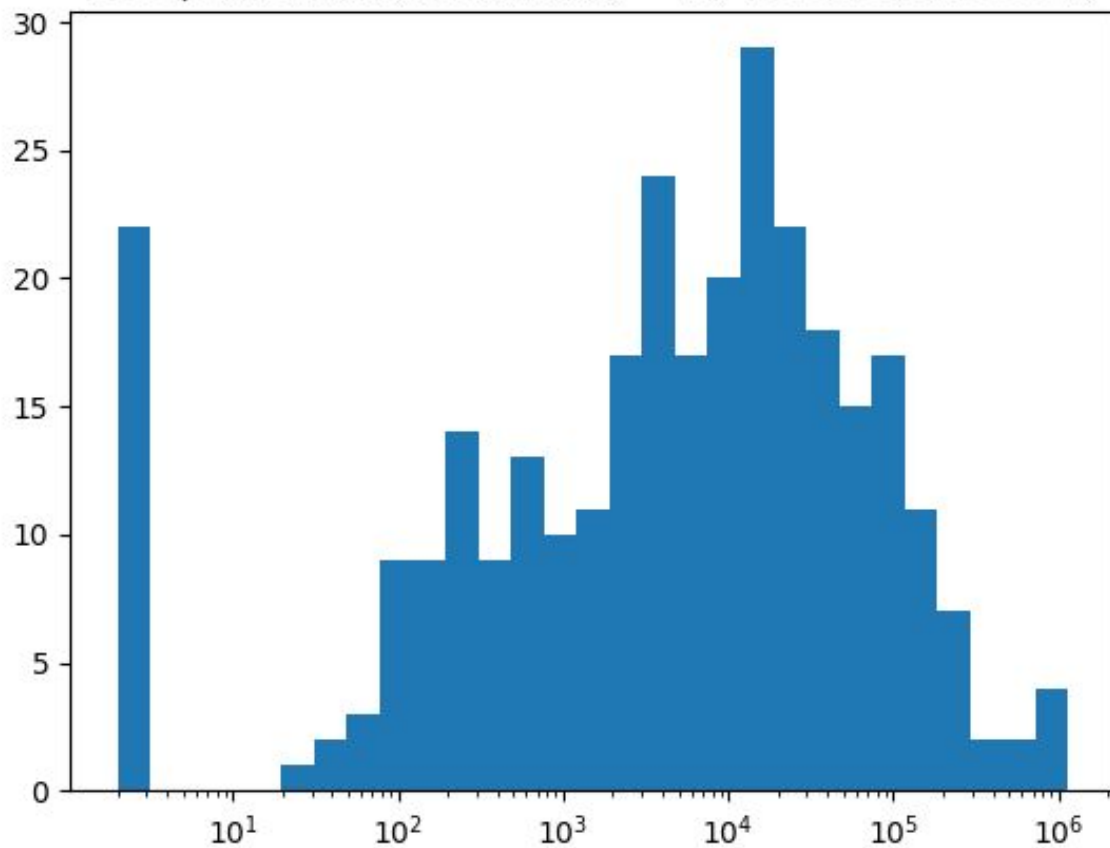
Thank you.

Questions?

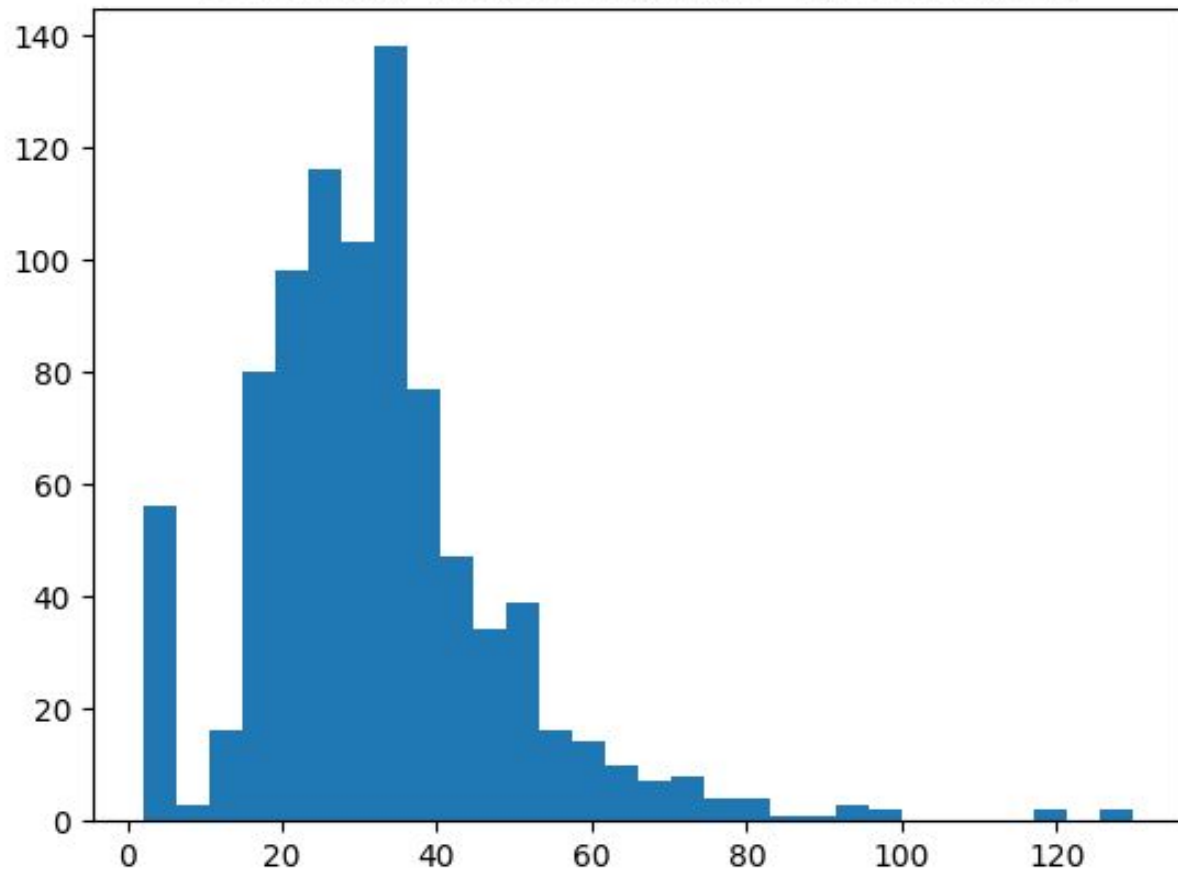
stdlib: Distribution of depth over the modules
(308 points, [min, mean, max] = [2, 34.70, 91])



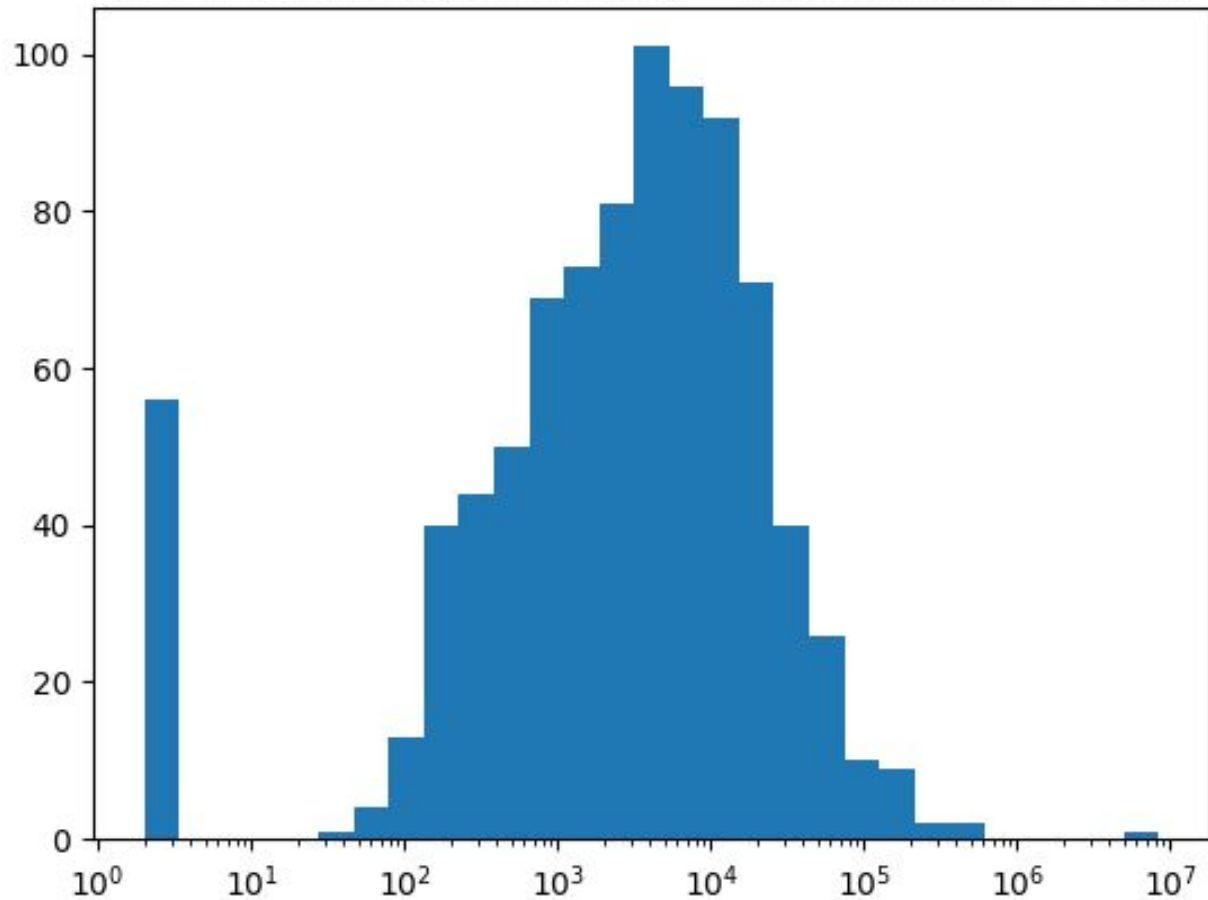
stdlib: Distribution of number of nodes over the modules
(308 points, [min, mean, max] = [2, 42536.51, 1150860])



unimath: Distribution of depth over the modules
(881 points, [min, mean, max] = [2, 31.95, 130])



unimath: Distribution of number of nodes over the modules
(881 points, [min, mean, max] = [2, 21822.73, 8488481])





Jaccard distance

- Defined as

$$J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

where A and B are two (multi)sets

- In our case, A and B are the **multisets** of “words” that describe a given definition.



Jaccard distance: example

Recall the definition of \mathbb{N} :

```
4 data N : Set where
5   zero : N
6   suc  : N → N
```

Its multiset is $A = \{\text{data}: 1, \mathbb{N}: 4, \text{where}: 1, \text{zero}: 1, \text{suc}: 1\}$.

Some preprocessing is done, e.g., `left-inverse` is split to words `left` and `inverse`.



TFIDF-based distance

- term frequency–inverse document frequency
- In our case, document = s-expression

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$



TFIDF: comments

- TFIDF actually maps a document into n-dimensional Euclidean space where N is the number of different terms:

$$\text{vec}(d) = [\text{tfidf}(d, w_1), \dots, \text{tfidf}(d, w_N)]$$

- After TFIDF is computed for every s-expression, standard Euclidean distance for finding nearest neighbours can be used



Examples

```
right-unit-law-mul-Group :  
  (x : type-Group) → Id (mul-Group x unit-Group) x  
right-unit-law-mul-Group = pr2 (pr2 is-unital-Group)
```

```
left-unit-law-mul-Group :  
  (x : type-Group) → Id (mul-Group unit-Group x) x  
left-unit-law-mul-Group = pr1 (pr2 is-unital-Group)
```

d = 0.0049

```
inner-inverse-law-mul-Inverse-Semigroup :  
  (x : type-Inverse-Semigroup) →  
  Id ( mul-Inverse-Semigroup  
      ( mul-Inverse-Semigroup x (inv-Inverse-Semigroup x)  
        ( x))  
      ( x))  
inner-inverse-law-mul-Inverse-Semigroup x =  
  pr1 (pr2 (center (is-inverse-semigroup-Inverse-Semigroup x)))
```

```
associative-mul-Inverse-Semigroup :  
  (x y z : type-Inverse-Semigroup) →  
  Id ( mul-Inverse-Semigroup (mul-Inverse-Semigroup x y) z)  
      ( mul-Inverse-Semigroup x (mul-Inverse-Semigroup y z))  
associative-mul-Inverse-Semigroup =  
  associative-mul-Semigroup semigroup-Inverse-Semigroup
```

d = 0.0871