Analyzing Software using Deep Learning

Token Vocabulary and Code Embeddings (Part 1)

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Summer 2020
Overview

- Token vocabulary problem
- Pre-trained token embeddings
- Joint embedding space for NL & PL

Recommended papers:

- ”Distributed representations of words and phrases and their compositionality”, NIPS, 2013
- ”Big Code != Big Vocabulary - Open-Vocabulary Models for Source Code”, ICSE, 2020
- ”Deep Code Search”, ICSE, 2018
Tokens: Building Blocks of Code

- Source code = Sequence of tokens
- Reasoning about large code snippets: Need to reason about tokens first

// From Angular.js
browserSingleton.startPoller(100,
  function(delay, fn) {
    setTimeout(delay, fn);
  });
Kinds of Tokens

Two categories of tokens

■ Fixed by programming language
  □ Operators, parentheses, keywords, etc.

■ Chosen by developers
  □ Identifiers, literals
Vocabulary Problem

- Large code corpus:
  - Huge number of tokens
- Difficult to represent and reason about
- Relevant for
  - Models that take code as an input
  - Models that produce code as an output
Vocabulary Problem (2)

Size of vocabulary for 14k projects

"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)
Vocabulary Problem (2)

Size of vocabulary for 14k projects

Almost 12 million tokens!

"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)
Vocabulary Problem (2)

Size of vocabulary for 14k projects

"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)
Vocabulary Problem (2)

Size of vocabulary for 14k projects

"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)
Vocabulary Problem (2)

Size of vocabulary for 14k projects

For all ways of modeling the vocabulary:
Linear growth when new projects are added

"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)
## Handling the Vocabulary Problem

<table>
<thead>
<tr>
<th>Abstract tokens</th>
<th>Consider N most frequent tokens only</th>
<th>Embed tokens into a vector space</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Much smaller vocabulary</td>
<td>- Covers large fraction of all tokens</td>
<td>- Constant vector size when code corpus grows</td>
</tr>
<tr>
<td>- Looses valuable information</td>
<td>- Out-of-vocabulary problem</td>
<td>- Non-trivial to obtain an effective embedding</td>
</tr>
</tbody>
</table>
Abstraction Token

Result:

\[ t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_2 \]

\[ a_1 \quad a_2 \]

\[ a_n \quad a_n \quad a_2 \quad a_2 \quad a_n \quad \ldots \]
Abstraction based on kind of token

```java
if (file != null) {
    line = file.read();
}
```

```
keyword operator identifier operator literal ...
```

```
OR
```

```
if (identifier != null) {
    identifier = identifier.identifier();
}
```
Consistent Renaming

if (file != null) {
    line = file. read();
}

if (id1 != null) {
    id2 = id1. id3();
}
Observation: Vocabulary has a “long-tail” distribution
- Few tokens occur frequently
- Many other tokens occur infrequently

Keep only N most frequent tokens

Represent others as special “unknown” token
Keeping Top-N Tokens (2)

Top-N approach on $\approx 100k$ JavaScript files

| $|V_{out}|$ | Percentage of unique names covered | Percentage of names covered |
|-------|----------------------|-------------------------------|
| 1,000 | 0.40                 | 63.19                         |
| 5,000 | 1.99                 | 75.07                         |
| 10,000 | 3.97                | 79.48                         |
| 20,000 | 7.95                | 83.82                         |
| 30,000 | 11.92               | 86.38                         |
| 40,000 | 15.89               | 88.16                         |
| 50,000 | 19.87               | 89.56                         |
| **60,000** | **23.84** | **90.74**                     |
| 70,000 | 27.81               | 91.62                         |
| 80,000 | 31.79               | 92.41                         |
| 90,000 | 35.76               | 93.19                         |
| 100,000 | 39.74              | 93.98                         |

"Context2Name: A Deep Learning-Based Approach to Infer Natural Variable Names from Usage Contexts" (Bavishi et al., 2018)