Natural Language Models for Predicting Programming Comments

Dana Movshovitz-Attias, William Cohen

Aug 5, 2013

ACL 2013
Modeling Software Code

• Code follows syntax rules, but is written by humans
• It is repetitive and predictable, similar to natural language [Hindle et al. 2012]

i/j/k as variables in for/while

```java
for (int i=0; i<len; i++) {
    // use i
    code = code*31 + text[i];
}
```

Repeating `println` calls

```java
System.out.println("token start offset: " + offsetAtt.startOffset());
System.out.println(" token end offset: " + offsetAtt.endOffset());
```

[All code samples are taken from lucene-3.6.2]
NLP Applications for Software Code

- **Code token completion**
  - [Hindle et al., 2012; Han et al., 2009; Jacob and Tairas, 2010]

- **Analysis of identifier names in code (methods/classes/variables)**
  - [Lawrie et al., 2006; Binkley et al., 2011]

- **Mining software repositories**
  - [Gabel and Su, 2008; Linstead et al., 2009; Gruska et al., 2010; Allamanis and Sutton 2013]
/**
 * A Token is an occurrence of a term from the text of a field. It consists of a term's text, the start and end offset of the term in the text of the field, and a type string.
 * 
 * A Token can optionally have metadata (a.k.a. Payload) in the form of a variable length byte array. Use `{@link TermPositions#getPayloadLength()} and `{@link TermPositions#getPayload(byte[], int)}` to retrieve the payloads from the index.
 * 
 * Tokenizers and TokenFilters should try to re-use a Token instance when possible for best performance, by implementing the `{@link TokenStream#incrementToken()}` API.
 * 
 * @see org.apache.lucene.index.Payload
 */

public class Token extends TermAttributeImpl
 implements TypeAttribute, PositionIncrementAttribute,
 FlagsAttribute, OffsetAttribute, PayloadAttribute,
 PositionLengthAttribute
Predicting Code Comments

• In this work we apply language models to the task of predicting class comments
  – N-grams
  – LDA
  – Link-LDA

• Evaluation metric: how much typing can we save?
  – 26% - 47% !
Uses of Comment Prediction

• Prediction of comment words can be useful for a variety of linguistic tasks
  – Document summarization
  – Document expansion
  – Code categorization / clustering
  – Improved search over code bases
Data

• Source code from 9 open source JAVA projects
  – Ant, Batik, Cassandra, Log4j, Lucene, Maven, Minor-Third, Xalan and Xerces
  – 9019 source code files

• Document: source code including comments

```java
package org.apache.lucene.index;
import java.io.IOException;
import java.io.Closeable;
/**
 * Abstract class for enumerating terms.
 * Term enumerations are always ordered by
 * Term.compareTo(). Each term in
 * the enumeration is greater than all that precede it. */
public abstract class TermEnum implements Closeable {
  /**
   * Increments the enumeration to the next element. True if
   * one exists. */
  public abstract boolean next() throws IOException;
}
```
More Data

• We include a source of data with a varying amount of text versus code: StackOverflow
  – 200K posts tagged with the ‘JAVA’, including question and all answers

<table>
<thead>
<tr>
<th>How to exclude transitive dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>I use JavaMail in the same project with cxf. cxf brings an older version of JavaMail which does not suit me. How to excluded? I did so: compile (group: 'org.apache.cxf', name: 'cxf-rt-bindings-soap', version: apacheCfxVersion) { exclude module: 'geronimo-javamail_1.4_spec' } But it did not help. I find in the war WEB-INF \ lib \ geronimo-javamail_1.4_spec-1.6.jar</td>
</tr>
</tbody>
</table>
Models

• N-grams (n = 1, 2, 3)
  – Trained over code + text tokens
    \[ d = \{ w_i \}_{i=1}^{N} \]
  – Class comment predicted from the combined model
  – We use the *Berkeley Language Model* package
Topic Models: Training

- LDA (topics = 1, 5, 10, 20)
  - Trained over code + text tokens
    \[ d = \{w_i\}_{i=1}^{N} \]
  - Joint distribution
    \[
p(\theta, z, w|\alpha, \beta) = p(\theta|\alpha) \prod_w p(z|\theta)p(w|z, \beta)
    \]
Topic Models: Training

• Link-LDA (topics = 1, 5, 10, 20)
  – Trained over mixed-membership documents
    \[ d = \left( \{ w_{i}^{\text{code}} \}_{i=1}^{C_n}, \{ w_{i}^{\text{text}} \}_{i=1}^{T_n} \right) \]
  – Joint distribution
    \[
p(\theta, z, w|\alpha, \beta) = p(\theta|\alpha) \cdot \\
    \prod_{w^{\text{text}}} p(z^{\text{text}}|\theta)p(w^{\text{text}}|z^{\text{text}}, \beta) \cdot \\
    \prod_{w^{\text{code}}} p(z^{\text{code}}|\theta)p(w^{\text{code}}|z^{\text{code}}, \beta) \]

Topic Models: Testing

- LDA + Link-LDA

1) Estimate document topics
\[ p(\hat{\theta}, z^r | w^r, \alpha, \beta) \]

2) Infer probability of comment tokens
\[ p(w^c | \hat{\theta}, \beta) \]
Evaluation Metric

- How much typing can we save? (comment completion)

Train a named-entity extractor

1) Rank dictionary of comment tokens by probability
2) Is the next token in the top 2?
3) If not – filter dictionary by next character
Three Training Scenarios

- **IN**: Comments are generated in the middle of project development
  - Learn from same-project data
- **OUT**: Comments are generated at the beginning of project development
  - Learn from other/related projects
- **SO**: No documented code is available
  - Learn from textual data source that combines text with code segments
### Main Results

<table>
<thead>
<tr>
<th>Data</th>
<th>3-gram</th>
<th>LDA</th>
<th>Link-LDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>47.1</td>
<td>34.20</td>
<td>35.81</td>
</tr>
<tr>
<td>OUT</td>
<td>32.96</td>
<td>26.86</td>
<td>28.03</td>
</tr>
<tr>
<td>SO</td>
<td>34.56</td>
<td>27.8</td>
<td>28.12</td>
</tr>
</tbody>
</table>

1) **IN > OUT**: in-project data improves predictions
2) **N-gram > topic-models**: sequential prediction
3) **Link-LDA > LDA**: distinguishing text from code improves predictions
4) **SO > OUT**: Training on more English text is useful
Motivation for a Hybrid Model

- Avg. words per project better predicted by each model

<table>
<thead>
<tr>
<th>Data</th>
<th>3-gram</th>
<th>Link-LDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{IN}</td>
<td>2778.35</td>
<td>574.34</td>
</tr>
<tr>
<td>\textit{OUT}</td>
<td>1865.67</td>
<td>670.34</td>
</tr>
<tr>
<td>\textit{SO}</td>
<td>1898.43</td>
<td>638.55</td>
</tr>
</tbody>
</table>

- Sample comment:

  \textit{IN 3-gram} Train a named-\textit{entity} extractor

  \textit{IN link-LDA} Train a named-\textit{entity} extractor
Contributions

✓ Application of language models to software code
✓ Task: Predicting class comments
✓ Evaluation metric: How much typing can we save?
  • Almost half!
✓ Prediction is improved by In-project data, when available
  • Distinguishing code from text tokens
  • Training on more English text
✓ Could be further improved using a hybrid model