

Natural Language Models for Predicting Programming Comments

Dana Movshovitz-Attias, William Cohen

Aug 5, 2013

ACL 2013



School of
Computer
Science



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

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

Repeating **println** calls

```
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]

Code Comments are also Repetitive

```
/**
```

```
 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
```

Provides a high level description of the code

Includes examples and description of use-cases

Refers to specific class names/ methods/variables

Includes both structured and unstructured element references

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

```
package org.apache.lucene.index;
import java.io.IOException;
import java.io.Closeable;
/** Abstract class for enumerating terms.
 * <p>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;
}
```

Class
Comment

● Code Tokens
● Text Tokens

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

How to exclude transitive dependency

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

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 [Pauls and Klein, 2011] with Kneser-Ney smoothing [Kneser-Ney, 1995]

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

$$d = \left(\{w_i^{code}\}_{i=1}^{C_n}, \{w_i^{text}\}_{i=1}^{T_n} \right)$$

- Joint distribution

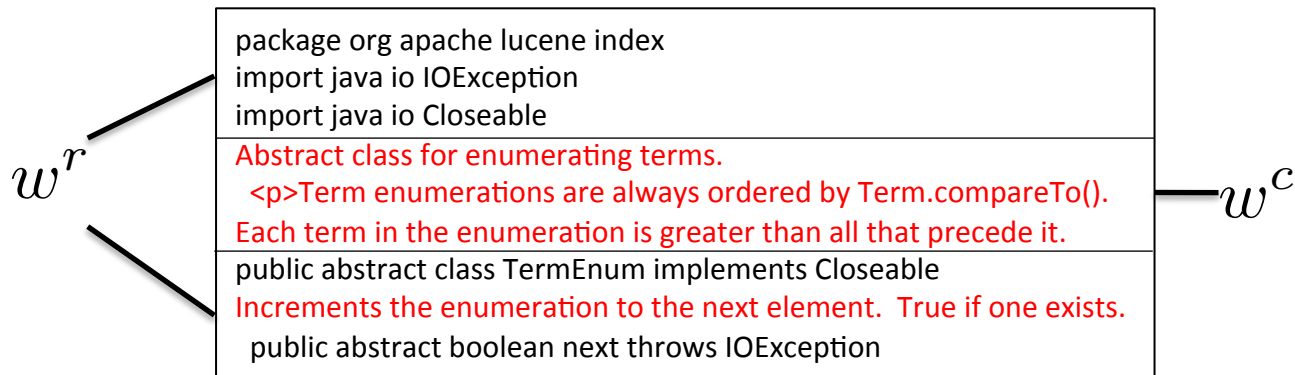
$$p(\theta, z, w | \alpha, \beta) = p(\theta | \alpha) \cdot$$

$$\prod_{w^{text}} p(z^{text} | \theta) p(w^{text} | z^{text}, \beta) \cdot$$

$$\prod_{w^{code}} p(z^{code} | \theta) p(w^{code} | z^{code}, \beta)$$

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

Data	3-gram	LDA	Link-LDA
<i>IN</i>	47.1	34.20	35.81
<i>OUT</i>	32.96	26.86	28.03
<i>SO</i>	34.56	27.8	28.12

- 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

Data	3-gram	Link-LDA
<i>IN</i>	2778.35	574.34
<i>OUT</i>	1865.67	670.34
<i>SO</i>	1898.43	638.55

- Sample comment:

IN 3-gram Train a named-entity extractorr

IN link-LDA Train a named-entity extractor

Contributions

Thanks!

- ✓ 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

dma@cs.cmu.edu

www.cs.cmu.edu/~dmovshov/