Impact of Evaluation Methodologies on Code Summarization

Pengyu Nie, Jiyang Zhang, Junyi Jessy Li, Raymond J. Mooney, Milos Gligoric

THE UNIVERSITY OF TEXAS AT AUSTIN

Introduction

Temporal relations are not explicitly modeled in the evaluation of machine learning (ML) models for code summarization. This may lead to evaluations inconsistent with the intended use cases.

Our contributions

- Study the evaluation methodologies in prior work: mixed-project and cross-project
- Propose a time-aware methodology: time-segmented
- Empirically experiment the three methodologies

Use Cases

We argue that evaluation methodology should be designed according to the intended use case of the ML model. Results obtained in batch-mode could be very different from those obtained in continuous-mode.

Evaluation Methodologies

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>mixed-project</td>
<td>in-project</td>
</tr>
<tr>
<td>cross-project</td>
<td>cross-project</td>
</tr>
<tr>
<td>time-segmented</td>
<td>continuous-mode</td>
</tr>
</tbody>
</table>

- does not consider software evolution (temporal relations among samples)
- only happen once in the lifecycle of a project
- train on past samples and use on new samples
- may be more practical in the context of continuously developing software

Experiments

- We run several existing ML models using different methodologies to understand their impact on automatic metrics, which are commonly used to judge the performance of models. We focus on the two most studied code summarization tasks: comment generation and method naming. For each task, we select several well-studied, representative, publicly available models, and use automatic metrics that are frequently reported in prior work.

- We collect a dataset of (code, comment) samples with timestamps from open-source Java projects on GitHub. We split the dataset to get training (Train), validation (Val), and standard test (Test) sets for each methodology (MP = mixed-project, CP = cross-project, T = time-segmented), and a common test (TestC) set for each pair of methodologies.

Finding 1: different methodologies may lead to conflicting evaluation results. For example, Code2Vec is better than Transformer under the mixed-project and time-segmented methodologies, but is worse under the cross-project methodology.

Finding 2: absolute values for automatic metrics vary widely across the three methodologies. Results under the mixed-project methodology are inflated, and results under the cross-project methodology may be an under-estimation of the more realistic continuous-mode use case.

Conclusion

- We need to more diligently choose evaluation methodology and report results of ML models according to the intended use cases.
- Time-segmented evaluation methodology should be adopted in the evaluation of ML models for code summarization.

data and code: github.com/EngineeringSoftware/time-segmented-evaluation
preprint: arxiv.org/abs/2108.09619
Pengyu Nie pynie@utexas.edu
The evaluation methodologies of prior work on developing ML models for code summarization do not consider the timestamps of code and comments, which may lead to misunderstanding if a model might be useful once adopted. In this work, we investigate a **time-aware** evaluation methodology for code summarization and empirically study the impact of different methodologies.

- Study two evaluation methodologies used in prior work: mixed-project and cross-project
- Propose a time-aware evaluation methodology: time-segmented
- Experiment several ML models using the methodologies

**Objectives**

<table>
<thead>
<tr>
<th>Task</th>
<th>comment generation</th>
<th>method naming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Models</td>
<td>DeepComHybrid</td>
<td>Transformer</td>
</tr>
<tr>
<td></td>
<td>Hu et al. EM'20</td>
<td>Ahmed et al. ACL'20</td>
</tr>
<tr>
<td></td>
<td>Code2Seq</td>
<td>Code2Vec</td>
</tr>
<tr>
<td></td>
<td>Ahmed et al. ICLR'19</td>
<td>Ahmed et al. ICLR'19</td>
</tr>
<tr>
<td>Metrics</td>
<td>BLEU</td>
<td>Precision</td>
</tr>
<tr>
<td></td>
<td>METEOR</td>
<td>Recall</td>
</tr>
<tr>
<td></td>
<td>ROUGE-L</td>
<td>F1</td>
</tr>
<tr>
<td></td>
<td>EM (exact match)</td>
<td>EM (exact match)</td>
</tr>
</tbody>
</table>

**Introduction**

There has been a growing interest in developing machine learning (ML) models for code summarization tasks, e.g., comment generation and method naming. Despite substantial increase in the effectiveness of ML models, the evaluation methodologies, i.e., the way people split datasets into training, validation, and test sets, were not well studied. Specifically, no prior work on code summarization considered the timestamps of code and comments during evaluation. This may lead to evaluations that are inconsistent with the intended use cases. In this work, we introduce the **time-segmented evaluation methodology**, which is novel to the code summarization research community, and compare it with the mixed-project and cross-project methodologies that have been commonly used.

**Use Cases**

We argue that evaluation methodology should be designed according to the intended use case of the ML model. We defined the **in-project and cross-project batch-mode** use cases (see paper for definition) which can be evaluated by the mix-project and cross-project methodologies, but they do not consider software evolution. As such, we define a more practical **continuous-mode** use case which can be evaluated using the time-segmented methodology: training the model with code available at a timestamp, and using the model on new code after that timestamp.

- does not consider software evolution
- only happen once in the lifecycle of a project
- train on past samples and use on new samples
- may be more practical in the context of continuously developing software