Heterogeneous Metric Learning with Content-based Regularization for Software Artifact Retrieval

Liang Wu¹, Liang Du¹, Bo Liu², Guandong Xu³, Yong Ge⁴, Yanjie Fu⁵, Jianhui Li¹, Yuanchun Zhou¹ and Hui Xiong⁵

Chinese Academy of Sciences¹, NEC Labs China², University of Technology Sydney³
University of North Carolina at Charlotte⁴, Rutgers University⁵
What is Software Artifact Retrieval

- Software Artifact Retrieval (SAR) is also called Software Traceability.
  - Given a text query, we want to find the corresponding code files.
  - Programs are transformed into plain text.
    - Identifier names, types, program comments and specifications are extracted to form a document.
Why we need SAR

- SAR is a fundamentally important task in software engineering.
  - Requirement traceability, concept/bug location, IT compliance, system reuse and so on.

- Example:

<table>
<thead>
<tr>
<th>A bug is filed</th>
<th>Source codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries written in natural languages</td>
<td>Files written in programming languages</td>
</tr>
</tbody>
</table>
Software Artifact Retrieval

• **Existing methods:**
  – Vector space model
  – Topic modeling
  – Manually compiled rules

• **Drawbacks:**
  – By collapsing source codes into a bag of words, structural information gets lost.
  – Rules depend on specific applications and need manual work.
Our approach

• Discovering new features to better represent code programs

• Proposing learning methods to allow similarity comparison between heterogeneous data
Our approach: new features

- As most source codes can be formalized as a tree structure, e.g., the whole program is the root node and its sub modules are first layer children. We collect these nodes to build the candidate set of our code snippet features.

- We also extract the relationships between functions and classes as code relationship features, including reference, implementation, inheritance.

(a) The tree structure of the example program, where each node representing the corresponding line of source code

(b) The source code and the extracted features
Content similarity extraction

- The content of code features is useful for the distance metric learning. Thus, we propose to use a data matrix $R$ to save this content-based similarity between heterogeneous features.
- The entry is '1' if the code feature contains a word.
Our approach: heterogeneous distance metric learning

- Goal: infer semantic of code features by mining the relationships between code and text, thus to allow direct comparison between code and text.

- query x:

- program y:

- To calculate similarity between x and y, we need to find heterogeneous distance metrics U and V:
  Our objective is to get \(|xU - yV|\)
Heterogeneous distance metric learning

- Use cost function to force matched query and code are closer to each other in the new space.

- Graph Laplacian regularization is adopted to smooth the mapping

- Code feature should be similar to words it contains

- Use L2 norm to avoid overfitting
Heterogeneous distance metric learning

- **Given:**
  
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>number of training code files and queries</td>
</tr>
<tr>
<td>$d^x$</td>
<td>dimensionality of textual feature space</td>
</tr>
<tr>
<td>$d^y$</td>
<td>dimensionality of code feature space</td>
</tr>
<tr>
<td>$\lambda_{1\ldots 3}$</td>
<td>regularization parameters</td>
</tr>
<tr>
<td>$X$</td>
<td>$d^x \times m$ matrix of words extracted from code</td>
</tr>
<tr>
<td>$Y$</td>
<td>$d^y \times m$ matrix of code features</td>
</tr>
<tr>
<td>$U$</td>
<td>$d^x \times k$ transformation matrix text features</td>
</tr>
<tr>
<td>$V$</td>
<td>$d^y \times k$ transformation matrix code features</td>
</tr>
<tr>
<td>$R$</td>
<td>$d^x \times d^y$ matrix of similarity between features</td>
</tr>
</tbody>
</table>

- **Optimization objective:**

  $$
  \arg \min_{U,V} \quad \lambda_1 \epsilon_{pull} + \lambda_2 g + \lambda_3 c + r
  $$

  $$
  \epsilon_{pull}(U, V) = \frac{1}{2} \| X^T U - Y^T V \|_F^2 \\
  g(U, V) = \frac{1}{2} tr(\bar{O} L^T O^T) \\
  c(U, V) = \frac{1}{2} \| UV^T - R \|_F^2
  $$
Heterogeneous distance metric learning

• Optimization objective:

\[
g(U, V) = \frac{1}{2} tr(O \tilde{L}^T O^T)
\]

\[
O = (U^T X, V^T Y)
\]

\[
\tilde{L} = I - D^{-\frac{1}{2}} WD^{-\frac{1}{2}}
\]

\[
w_{ij} = \begin{cases} 
1, & l_i = l_j \land i \neq j; \\
0, & \text{otherwise}
\end{cases}
\]

\[
d_{ii} = \sum_j^{m+m} w_{ij}
\]

\[
\tilde{L} = \begin{pmatrix}
\tilde{L}^{xx} & \tilde{L}^{xy} \\
\tilde{L}^{yx} & \tilde{L}^{yy}
\end{pmatrix}
\]
Heterogeneous distance metric learning

- Optimization: \[ \arg \min_{U,V} \lambda_1 \epsilon_{pull} + \lambda_2 g + \lambda_3 c + r \]

**Algorithm 2 Iterative Optimization for HMLCR**

**Input:** The data matrices: \( X, Y \); The similarity matrices: \( W, R \); The parameters, \( \lambda_{1...3} \); The learning rate \( \eta \); The maximal number of iterations \( MaxIter \).

**Output:** The transformation matrices \( U \) and \( V \).

1: Generate \( U \) and \( V \)
2: for \( i = 1 \) to \( MaxIter \) do
3: \( U \leftarrow U + \eta \frac{\partial \text{Loss}}{\partial U} \)
4: \( V \leftarrow V + \eta \frac{\partial \text{Loss}}{\partial V} \)
5: if convergence then
6: \( \text{break} \)
7: end if
8: end for
9: return \( U \) and \( V \)

\[
\frac{\partial \text{Loss}}{\partial U} = \lambda_1 X(X^T U - Y^T V) + \lambda_2 (X \bar{L}^x X^T U + X \bar{L}^{xy} Y^T V) + \lambda_3 (UV^T - R)V + U
\]

\[
\frac{\partial \text{Loss}}{\partial V} = \lambda_1 Y(Y^T V - X^T U) + \lambda_2 (Y \bar{L}^{yx} X^T U + Y \bar{L}^y Y^T V) + \lambda_3 (U V^T - R)^T U + V
\]
Experiments

• Dataset:
  – Eclipse
    • query: [Bug 5138] - [typing] Double-click-drag to select multiple words doesn't work
    • code: eclipse/jdt/internal/ui/text/java/JavadocDoubleClickStrategy.java
eclipse/jdt/internal/ui/text/java/JavaStringDoubleClickSelector.java

  • Mainly written in Java.
  • The project contains approximately 7,000 classes with about 89,000 methods in about 2.4 million lines of code (MLOC).

  – Filezilla
    • change logs are used instead.
    • The project is written in C and is much smaller than Eclipse, with about 8,012 methods in 410 KLOC.
Experiments

• Baselines:
  – COS: Cosine similarity between words and codes.

  – LM: This method adopts language modeling to calculate the similarity between the textual representation of code and queries.

  – LSI: The latent semantic indexing method that first compresses the textual representation of code and queries and then calculate their similarity.

  – CFA: Cross-modal Factor Analysis model, which was proposed to discover the associations between the feature space of different media.

  – CFA+CR: In this method, we adopt the content-based constraint to regularize the training process of CFA.
Experiments

- Results:

<table>
<thead>
<tr>
<th>TABLE II. THE PRECISION @ TOP N RESULTS OF ECLIPSE</th>
<th>TABLE III. THE PRECISION @ TOP N RESULTS OF FILEZILLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>P@1</td>
<td>P@2</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>COS</td>
<td>0.016</td>
</tr>
<tr>
<td>LM</td>
<td>0.0164</td>
</tr>
<tr>
<td>LSI</td>
<td>0.0092</td>
</tr>
<tr>
<td>CFA</td>
<td>0.0221</td>
</tr>
<tr>
<td>CFA+CR</td>
<td>0.0246</td>
</tr>
<tr>
<td>HMLCR</td>
<td><strong>0.027</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE IV. THE RECALL @ TOP N RESULTS OF ECLIPSE</th>
<th>TABLE V. THE RECALL @ TOP N RESULTS OF FILEZILLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>R@1</td>
<td>R@3</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>COS</td>
<td>0.0128</td>
</tr>
<tr>
<td>LM</td>
<td>0.0164</td>
</tr>
<tr>
<td>LSI</td>
<td>0.0092</td>
</tr>
<tr>
<td>CFA</td>
<td>0.0168</td>
</tr>
<tr>
<td>CFA+CR</td>
<td>0.0176</td>
</tr>
<tr>
<td>HMLCR</td>
<td><strong>0.0217</strong></td>
</tr>
</tbody>
</table>
Experiments

• Results:

<table>
<thead>
<tr>
<th></th>
<th>nDCG @ n=2</th>
<th>nDCG @ n=4</th>
<th>nDCG @ n=10</th>
<th>nDCG @ n=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>COS</td>
<td>0.0256</td>
<td>0.0388</td>
<td>0.0506</td>
<td>0.0526</td>
</tr>
<tr>
<td>LM</td>
<td>0.0164</td>
<td>0.0299</td>
<td>0.0415</td>
<td>0.0515</td>
</tr>
<tr>
<td>LSI</td>
<td>0.0183</td>
<td>0.0287</td>
<td>0.0339</td>
<td>0.0339</td>
</tr>
<tr>
<td>CFA</td>
<td>0.0344</td>
<td>0.048</td>
<td>0.0734</td>
<td>0.0865</td>
</tr>
<tr>
<td>CFA+CR</td>
<td>0.0356</td>
<td>0.052</td>
<td>0.0751</td>
<td>0.0887</td>
</tr>
<tr>
<td>HMLCR</td>
<td><strong>0.0442</strong></td>
<td><strong>0.0589</strong></td>
<td><strong>0.0804</strong></td>
<td><strong>0.0947</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>nDCG @ n=2</th>
<th>nDCG @ n=4</th>
<th>nDCG @ n=10</th>
<th>nDCG @ n=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>COS</td>
<td>0.0171</td>
<td>0.022</td>
<td>0.0242</td>
<td>0.0252</td>
</tr>
<tr>
<td>LM</td>
<td>0.0153</td>
<td>0.0207</td>
<td>0.0586</td>
<td>0.0765</td>
</tr>
<tr>
<td>LSI</td>
<td>0.0153</td>
<td>0.0226</td>
<td>0.0333</td>
<td>0.0403</td>
</tr>
<tr>
<td>CFA</td>
<td>0.0887</td>
<td>0.0935</td>
<td>0.1482</td>
<td>0.1665</td>
</tr>
<tr>
<td>CFA+CR</td>
<td>0.1491</td>
<td><strong>0.1644</strong></td>
<td>0.203</td>
<td>0.2347</td>
</tr>
<tr>
<td>HMLCR</td>
<td><strong>0.1506</strong></td>
<td>0.1598</td>
<td><strong>0.2035</strong></td>
<td><strong>0.2351</strong></td>
</tr>
</tbody>
</table>
Conclusions and future work

• To further improve code artifact retrieval:
  – We propose new features to encode more information.
  – We put forward a heterogeneous distance metric learning approach to map code features and words into a unified space.

• Things to do in the future:
  – Calculate similarity between code features
  – Scalability
Thanks