Detecting Duplicate Bug Reports Using Character N-Gram Based Features

Ashish Sureka and Pankaj Jalote

Software Engineering Group
Indraprastha Institute of Information Technology, Delhi (IIIT-D), India

The authors acknowledge SIGSE Computer Society of India (CSI) for providing conference registration and travel support.
Presentation Outline

- **Background Information**
  - Defect tracking system and bug triaging
  - Duplicate bug report annotation

- **Problem Description**
  - Research motivation
  - Technical challenges and issues
  - Illustrative examples of duplicate bug reports from Eclipse project
  - Contributions and main idea

- **Proposed Solution**
  - Character-level N-gram model
  - Examples illustrating advantages of Character-level N-gram model
  - Solution architecture

- **Performance Study/Empirical Evaluation**
  - Evaluation dataset and performance metrics
  - Empirical results and insights
  - Limitations of the proposed solution

- **Related Work**
  - Previous approaches

- **Conclusions and Future Work**

30 November – 03 December 2010, at Sydney, Australia
(Sureka et al, 2010), Proceedings of IEEE APSEC 2010
Bug Triaging Tasks [Illustrative List]

13,016 bug reports were filed from Jun-04 to Jun-05 between the release of Eclipse Platform version 3.0 and 3.1: averaging 37 reports per day, with a maximum of 220 reports in a single day [Anvik et al., 2006]

In 2005, for Mozilla project, almost 300 bugs filed every day needed triaging [Forrest et al., 2009]

Does the bug report contains all the required information? [Quality Determination]

Is it unique or duplicate of an existing bug report? [Duplicate Detection]

Who is the expert to fix this bug? [Expert Identification]
Duplicate Bug Reports

Two bug reports are said to be duplicates if they describe the same issue or problem and thereby have the same solution to fix the issue.

Percentage of duplicate bug reports can be up-to 25-30% [Anvik et al., 2006] [Jalbert et al., 2008][Hiew et al, 2003]
Research Motivation and Technical Challenges

Research Motivation

- Studies show that duplicate bugs is a major concern to project teams
- Duplicate identification is time consuming and requires knowledge of the system

Research Aim

- To investigate text analytics based solution for automatic duplicate bug report identification

Technical challenges and issues

- Bug reports are expressed in natural language text. Natural language is vast and ambiguous
- Huge number of problem reports in large software setting makes the task daunting
- Presence of poorly expressed bug reports (missing information, noisy text) poses challenges
### Illustrative Examples of Duplicate Bug Reports from Eclipse Project

<table>
<thead>
<tr>
<th>BUID ID</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>211356</td>
<td>Clipboard example crashes</td>
</tr>
<tr>
<td>210894</td>
<td>Examples that use StyledText crash on startup</td>
</tr>
<tr>
<td>212404</td>
<td>ArrayIndexOutOfBoundsException in SystemHostPool#getHosts()</td>
</tr>
<tr>
<td>206742</td>
<td>NPE in SystemRegistry.getHostsBySystemType()</td>
</tr>
<tr>
<td>212148</td>
<td>NPE in getMarkerItem</td>
</tr>
<tr>
<td>212002</td>
<td>NPE in CachedMarkerBuilder.getMarkerItem</td>
</tr>
<tr>
<td>211649</td>
<td>No Border in Win32</td>
</tr>
<tr>
<td>182797</td>
<td>CDT.BORDER does not make effect for Win32</td>
</tr>
<tr>
<td>294093</td>
<td>Out of memory error opening large files in Eclipse Text Editor</td>
</tr>
<tr>
<td>75086</td>
<td>Request for a memory efficient way of loading large documents (in the mega bytes range)</td>
</tr>
<tr>
<td>23193</td>
<td>Improve performance editing large .java files</td>
</tr>
<tr>
<td>295158</td>
<td>Canceling unresponsive jobs</td>
</tr>
<tr>
<td>123075</td>
<td>Allow unconditionally killing the job that refuses to cancel</td>
</tr>
<tr>
<td>208750</td>
<td>Background Jobs should have a kill option</td>
</tr>
<tr>
<td>272481</td>
<td>Ability to kill tasks</td>
</tr>
</tbody>
</table>
Contributions and Main Idea

Research Contributions

- A novel method for identifying duplicate bug reports using a character-level n-gram model for text similarity matching.
- An empirical evaluation of the proposed approach on real-world and publicly available dataset from a popular open-source project

Solution Approach (Main idea and Key-points)

- Apply a character-level n-gram based similarity computation
- Previous approaches are word-based and we propose a character-based model
- Advantages over word-based models: robustness to noisy text & language independence
  - Ability to match concepts from system messages
  - Ability to extract super-word features
  - Ability to handle misspelled words
  - Ability to match short-forms with their expanded-form
  - Ability to match term variations to a common root
  - Ability to match hyphenated phrases
Character-Level N-Gram Model

- N-gram means a subsequence of N contiguous items within a sequence of items
- Word n-grams represent a subsequence of words
- Character n-grams represent a subsequence of characters
- String: Software Engineering Data Mining
  - Word-level bi-grams (N=2): Software Engineering, Engineering Data, Data Mining
  - Word-level tri-grams (N=3): Software Engineering Data, Engineering Data Mining
- String: Software
  - Character-level bi-grams (N=2): So, of, ft, tw, wa, ar, re
  - Character-level tri-grams (N=3): Sof, oft, ftw, twa, war, are
- Character-level n-gram representation has certain distinct advantages which align with the task of duplicate bug report detection
## Ability to Match Concepts from System Messages

- Source code segments, system error messages [not natural language]
- `StringIndexOutOfBoundsException = String, IndexOutOfBounds, Exception` [concepts]
- Duplicate bug report: `StringIndexOutOfBoundsException` (the word Exception is not mentioned)
- Similar semantic interpretations but are lexically different
- Two terms share several character n-grams

<table>
<thead>
<tr>
<th>Concept in one Bug Report</th>
<th>Similar Concept in duplicate Bug Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewerContributions</td>
<td>Contributions</td>
</tr>
<tr>
<td>Add Java exception</td>
<td>AddExceptionDialog</td>
</tr>
<tr>
<td><code>StorageDocumentProvider#setDocumentContent</code></td>
<td><code>StorageDocumentProvider</code></td>
</tr>
<tr>
<td>Launch Configuration Type page of Property Sheet</td>
<td><code>LaunchConfigurationTypePropertyPage</code></td>
</tr>
<tr>
<td><code>IMethodBinding</code></td>
<td>method binding</td>
</tr>
<tr>
<td><code>IDebugViewAdapter</code></td>
<td>AbstractDebugView, IDebugView</td>
</tr>
<tr>
<td><code>ILabelDecorator</code></td>
<td>label decorators</td>
</tr>
<tr>
<td>Out of memory</td>
<td>OutOfMemoryError</td>
</tr>
<tr>
<td><code>TimeOutException</code></td>
<td><code>org.eclipse.jdi.TimeOutException</code></td>
</tr>
</tbody>
</table>
Ability to Extract Super-Word Features

- ing to: switching to and changing to
- error wh: error while and error when
- ate getter: create getter and generate getter

Handling Misspelled Words

- dinamically and dynamically
- bugzilla and bugzila

Ability to Match Short-Forms with their Expanded-Form

- (nav, navigator), (synch, synchronizing), (config, configuration)
- (temp, temporary), (multi, multiple), (anon, anonymous), (doc, documentation)

Ability to Match Term Variations to a Common Root

- comput: computing, computation, computed

Ability to Match Hyphenated Phrases

- (drag-and-drop, drag-drop, drag and drop), (plug-in, plugin, plug in), (Ctrl-F7, Ctrl F7)
- (read-only, read only), (out of synch, out-of-synch)
Solution Architecture

New Bug Report

- Metadata
- Title
- Description

Feature Extraction and Document Representation (FEDR)

- Character N-Gram Feature Extraction
- Bag of Character N-Grams + Description

Bug Repository

Pre-defined Threshold
(Learnt from experience, historical data)

Similarity Computation

Top-N Similar Bug Reports

<table>
<thead>
<tr>
<th>Bug Report</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Triager

Search Results
Experimental Evaluation

- Bug reports for several open source projects such as Mozilla Firefox and Eclipse are available in public domain.

- **Evaluation dataset**: 213000 Eclipse bugs (XML format) [Largest evaluation dataset in literature on duplicate detection]

- Data Source: MSR (Mining Software Repositories) Mining Challenge 2008 website

- The dataset contains the ground-truth (pre-annotated having bug-reports tagged as duplicate by the Triager)

- **Performance metrics**: Recall rate (RR) = percentage of duplicates for which the master bug-report is found within the top-N search results

- Effective dataset size = **205242** (7758 bug reports unable to retrieve)

- Number of bug reports marked as duplicates : 27036 (27036/205242 = 13.17%)

- Number of different Products (Platform, JDT, Equinox, Web Tools, EPP and Java Server Faces) = **77**

- Number of different Components (Resources, Compare, UI, Build, Doc, Debug, SWT, IDE, CVS and Server) = **482**

- Number of version (1.1.4, 2.0, 3.3, 1.0.0 Release, Ganymade and 4.0 Beta-1) = **165**
### Percentage of Duplicate Reports belonging to same/different product/component

<table>
<thead>
<tr>
<th>Product</th>
<th>Component</th>
<th>Records</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same</td>
<td>Same</td>
<td>19903</td>
<td>73.62%</td>
</tr>
<tr>
<td>Same</td>
<td>Different</td>
<td>4056</td>
<td>15.00%</td>
</tr>
<tr>
<td>Different</td>
<td>Same</td>
<td>1209</td>
<td>4.48%</td>
</tr>
<tr>
<td>Different</td>
<td>Different</td>
<td>1868</td>
<td>6.90%</td>
</tr>
</tbody>
</table>

More than 1/4th of the duplicate bug reports do not belong to the same Product and Component.

Compare the query bug report with all the bug reports.

Do not apply filter based on the Product and Component attribute.
Empirical Results

Randomly selected 1100 duplicate bug reports

Recall rate (based on T-T + T-D + D-T Similarity)

Similarity score of 100 to 1000 results in Top 100 rank
Average difference between the bug ids of the master and duplicate report is less than 3138 for 50% of the bug reports and 15335/21513 for 75%/80% of the bug reports in the dataset. 5% of the duplicate bug reports, the distance between the duplicate and master report is more than 68624.

40% of the data points has the value of variable T-T between 50 and 1290. Recall rate of 40.22% for Top-10 results and 61.94% for Top-50.

Recall rate for 2270 bug reports with Title-Title score greater than 50 based on Title-Title similarity.
Previous Approaches

Hiew et al. (2003): Grouping similar reports in the repository and deriving a centroid of the clustered reports, comparing an incoming report (represented as a document vector) to the centroids of bug report

Runeson et al. (2007): Apply the standard vector-space model, representing a bug report as a Weighted vector of words, use similarity measures such as Cosine, Dice and Jaccard

Jalbert et al. (2008): bag of word approach, use cosine based similarity measure

Wang et al. (2008): execution information (execution trace and list of executed methods) + natural language information

Kemp et al. (2009): natural language vector cosine techniques, machine learning and neural networks

Sun et al. (2010): discriminative model based approach (SVM learning algorithm)
Summary and Conclusions

- Presents an approach to compute text similarity between two bug reports
- To assist a Triager in the task of duplicate bug report detection
- Central idea: Application of character n-grams as low-level features
- Advantages: robustness to noisy text and language independence
- Evaluation dataset: bug database consisting of more than 200 thousand bug reports
- Recall rate for the Top 50 results is 33.92% (no predefined threshold)
- Recall rate for the Top 50 results is 61.94% (predefined threshold for similarity score = 50)
Thank You