April 2023

Automated Techniques to Identify Most Relevant Duplicates for Bug Deduplication

Aman Singh

Vidhi Gupta

Richa Gupta

Vineet Jain

Subham Mishra

See next page for additional authors

Inventor(s)
Aman Singh, Vidhi Gupta, Richa Gupta, Vineet Jain, Subham Mishra, and Babu Prasad Elumalai

Automated Techniques to Identify Most Relevant Duplicates for Bug Deduplication

ABSTRACT

During bug deduplication, selecting top bugs based on scores from a binary classification model does not work well if the model tends to return lower scores. Further, the selected top candidate bugs are not ranked based on relevance, which means that there is no mechanism to mark a bug as a duplicate without verifying each candidate. This disclosure describes automated techniques to identify ancestor bugs for a newly reported bug, to retrieve the top candidates for duplicate bugs, and rank the candidates based on a scoring function. The techniques are robust to pairwise matches between a newly reported bug and its ancestor bug failing to meet threshold scores. Rather, by relying on comparisons across the pool of bugs - open bugs as well as prior identified duplicates - along with transitive properties, the techniques automatically identify the ancestor bug in such situations. A scoring function is described that utilizes features such as number of duplicates attached to a bug, number of updates, score, number of incorrect duplicate predictions that the bug was part of, the number of affected users, etc. to rank the identified bugs.

KEYWORDS

- Machine learning
- Binary classification
- Bug deduplication
- Bug retrieval
- Bug triaging
- Software development
- Pairwise matching

BACKGROUND

Bug deduplication is a well-known problem in software development. Binary classification techniques are widely used to solve the problem of bug deduplication. Selecting top K bugs based on classification scores that meet a threshold may not work well if the classification model tends to return lower scores. Further, the selected top K candidates are not ranked based on relevance, thus there is no mechanism to mark a bug as a duplicate without verifying each candidate. Thus, current methods for identifying top candidates of duplicates are not efficient.

DESCRIPTION

This disclosure describes automated techniques to identify ancestor bugs for a newly reported bug, to retrieve the top candidates for duplicate bugs, and rank the candidates based on a scoring function. The techniques eliminate the need to perform a manual search for duplicate bugs across a pool of bugs. This allows the bug triager to only investigate real bugs rather than iterating over duplicate bugs. The techniques are robust to pairwise matches between a newly reported bug and its ancestor bug failing to meet threshold scores. Rather, by relying on comparisons across the pool of bugs - open bugs as well as prior identified duplicates - along with transitive properties, the techniques automatically identify the ancestor bug in such situations. The techniques leverage existing binary classification models trained on historical bugs to perform comparisons. The binary classification model M may be a logistic regression model trained on a historical dataset of bugs to predict if a pair of source and target bugs are duplicates. Features available in bug metadata can be used to train the model.

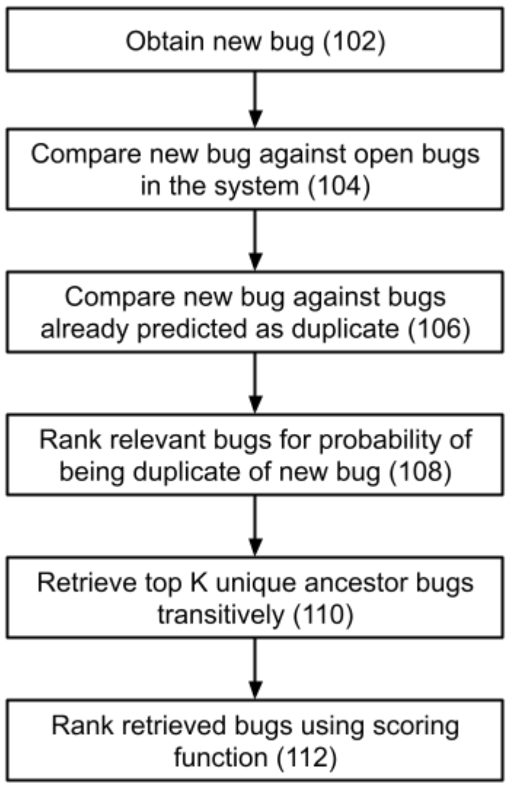
Obtain new bug (102)
Compare new bug against open bugs in the system (104)
Compare new bug against bugs already predicted as duplicate (106)
Rank relevant bugs for probability of being duplicate of new bug (108)
Retrieve top K unique ancestor bugs transitively (110)
Rank retrieved bugs using scoring function (112)

Fig. 1: Identifying most relevant duplicates for bug deduplication

Fig. 1 illustrates an example method to automatically retrieve and rank the top *k* candidate of duplicate bugs. A new bug is obtained (102), e.g., from bug reports. The new bug is compared against the currently open bugs (104). Open bugs, in this context, are all bugs that have not been marked as duplicates. The new bug is also compared against bugs that have already been marked as duplicates (106).

For example, consider a new bug within the system, *b_new*. This bug is first paired against the other open bugs to generate a prediction of whether it is a duplicate of any of them. For example, the system may currently have *b_y* open bugs, for *y* in *[1, M]*. In this case, *b_new* is compared against the *b_y* open bugs. Further, there are additional *a_x* bugs, for *x* in *[1, N]*. These are bugs that have already been identified as being duplicates of other open bugs. In the method illustrated in Fig. 1, *b_new* is compared on a pairwise basis with each of these *a_x* bugs.

During each pairwise comparison, across *b_y* and *a_x*, a probability score is generated that indicates the probability of *b_new* being a duplicate of the bug it is being compared to (108). Using these scores, the top *k* unique ancestor bugs can be retrieved transitively.

For example, *b_new* may be classified as not being a duplicate of any of the currently open bugs *b_y*. However, *b_new* may get a high probability score of being a duplicate of a bug that is already identified as being a duplicate, say *a_4*. In this case, bug *a_4* has already been identified as being a duplicate of another bug, say *b_4*. In such a scenario, *b_4* is retrieved on a transitive basis - since *b_new* is classified as a duplicate of *a_4*, and *a_4* is classified as a duplicate of *b_4*, *b_4* is identified as an ancestor bug for *b_new*. In this instance, *b_new* has a low probability score of being a duplicate of *b_4*. However, since the probability of *b_new* being a duplicate of *a_4* is high, the ancestor bug *b_4* is retrieved on a transitive basis (110).

Fig. 2 illustrates pseudocode for an example function to identify top *K* duplicate bugs.

```
FUNCTION getTopKBugs:

let p = [p_1, p_2, ..., p_r] ordered list of relevant bugs
based on probability

let ancestor = [(a_1, b_1), (a_2, b_1) , ...., (a_N, b_M)]
denoting a map of ancestor and already duplicate predicted bug

topKBugs={}

for bug in p:
     if setsize(topKBugs) == K:
          break
     if bug in a:
          topKBugs.add(ancestor[bug])
     else:
          topKBugs.add(bug)
return topKBugs
```

Fig. 2: Pseudocode to identify top duplicate bugs

While a binary classification retrieval model uses bug metadata features like bug description, title, etc., such features are based on the initial state of a bug. The binary classification model does not take into account features that change over time, such as the number of duplicates attached to a canonical bug, the number of updates made on the bug as comments, etc.

Per techniques of this disclosure, the retrieved ancestor bugs are scored using a custom scoring function (112). For example, the scoring function can be:

$$F(D, U, score, X, upvotes) = ((D^2 + U) * score * max(1, upvotes))/max(1, X)^2$$

where:

- D is the number of duplicates attached to the bug at any instant
- U is the number of updates in the bug at any instant
- $score$ is the probability for prediction being duplicate between the source (b_new) and the target bug
- X is the number of incorrect predictions the bug was a part of
- $upvotes$ is the number of upvotes on the bug affecting users

The scoring function can be chosen or modified as appropriate for the particular use case. For example, in case the individual parameters are large enough, the above function can be modified as:

$$F(D, U, score, X, upvotes)= log(D^2 + U) + log(score) + log(max(1, upvotes)) - 2log(max(1, X))$$

The retrieved bugs are ranked in descending fashion on the basis of their score in the custom scoring function. The process can be performed for new bugs that are to be processed.

The described bug deduplication techniques can be used in a bug triager to identify the most relevant duplicate bugs. Bug management tools can also utilize the identified duplicates to surface relevant information on a user interface. The techniques can be used by any bug management tool or technical infrastructure used for debugging.

CONCLUSION

This disclosure describes automated techniques to identify ancestor bugs for a newly reported bug, to retrieve the top candidates for duplicate bugs, and rank the candidates based on a scoring function. The techniques are robust to pairwise matches between a newly reported bug and its ancestor bug failing to meet threshold scores. Rather, by relying on comparisons across the pool of bugs - open bugs as well as prior identified duplicates - along with transitive properties, the techniques automatically identify the ancestor bug in such situations. A scoring function is described that utilizes features such as number of duplicates attached to a bug, number of updates, score, number of incorrect duplicate predictions that the bug was part of, the number of affected users, etc. to rank the identified bugs.

REFERENCES

1. “Binary classification - Wikipedia” available online at https://en.wikipedia.org/wiki/Binary_classification accessed April 10, 2023.
2. Karasov, Nikolay, Aleksandr Khvorov, Roman Vasiliev, Yaroslav Golubev, and Timofey Bryksin. "Aggregation of Stack Trace Similarities for Crash Report Deduplication." *arXiv preprint arXiv:2205.00212* (2022).