October 2021

Bug Management Using Machine Learning

Megha Jindal

Richa Gupta

Subham Mishra

Carlos Arguelles

Babu Prasad Elumalai

Bug Management Using Machine Learning

ABSTRACT

Automated tests of software can often independently log different bugs for the same underlying problem (root cause). Manually identifying duplicate bugs is a source of toil for engineers. A related problem of bug management is that of bug routing, e.g., determining the right team or person to route a bug report to for the purposes of debugging. This disclosure describes techniques for bug deduplication and bug routing based on machine learning (ML). Per the techniques, a binary machine classification model is trained to aggregate bugs with a common root cause. Bugs in a class of bugs with a common root cause are deduplicated, e.g., represented by just one of the multiple bugs in the class. Further, a multi-class ML model is trained to predict the right team for handling a new (incoming) bug.

KEYWORDS

- Bug deduplication
- Bug routing
- Bug allocation
- Bug management
- Software testing
- Software debugging
- Bug features
- Machine learning

BACKGROUND

Automated tests of software can often independently log different bugs (or symptoms) for the same underlying problem or root cause. Manually identifying duplicate bugs, e.g., bugs that arise from the same root cause, is a source of toil for engineers.

A related problem of bug management is that of bug routing, e.g., determining the right team or person to route a bug report to for the purposes of debugging. In complex software development environments, a bug can get bounced amongst many developers before it reaches the right person who can take corrective action. Such bouncing around of a bug results in routing latency - the delay before a bug is corrected which can potentially block critical releases. It is also wasteful of the time of each developer that expends effort examining it only to realize that the bug is not theirs to fix.

Existing techniques for bug deduplication require a user to manually mark bugs as duplicates, which only adds to developer toil. Automatic techniques to deduplicate bugs use basic distance or term frequency/ inverse document frequency (TF-IDF) semantics. These can be error-prone and carry the risk that a bug reaches production code due to incorrectly being marked as a duplicate of an existing bug. Such techniques have no mechanism to measure or to fine-tune performance.

DESCRIPTION

This disclosure describes techniques for bug deduplication and bug routing based on machine learning (ML). Per the techniques, a binary machine classification model is trained to aggregate bugs with a common root cause. Bugs in a class of bugs with a common root cause are deduplicated, such that they are represented by just one of the multiple bugs in the class. Further, a multi-class ML model is trained to predict the right team for handling a new (incoming) bug.

Bug deduplication

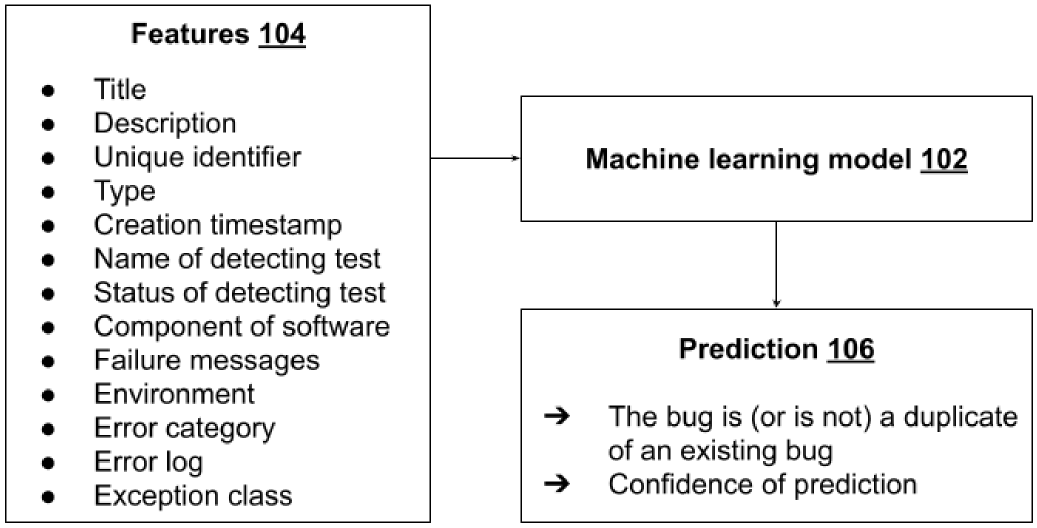
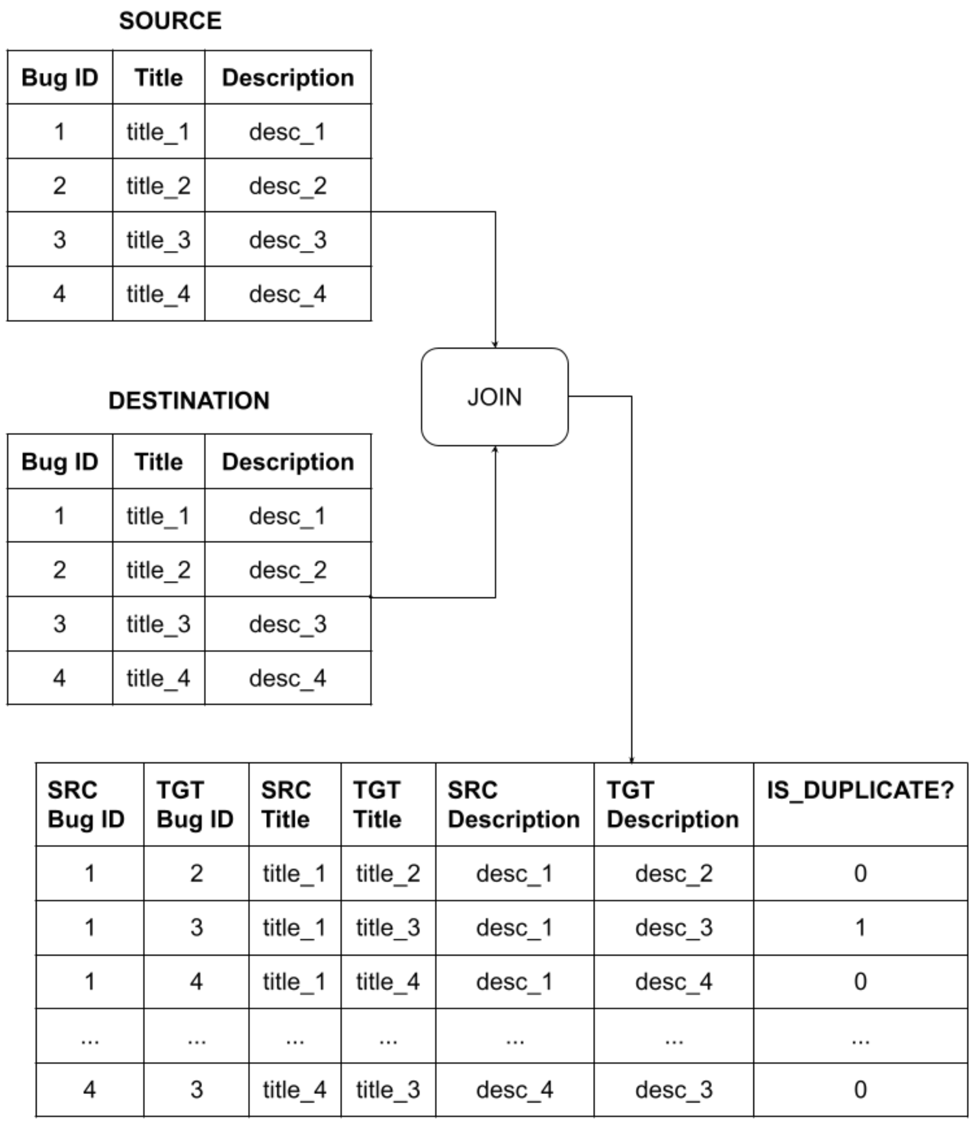


Fig. 1: Bug deduplication

Fig. 1 illustrates bug deduplication. A single bug entity comprises various attributes of the bug such as title; description; unique identifier; bug type; creation timestamp; name of detecting test; status of detecting test; software component where the bug originated; failure messages; environment; error category; error logs; exception class; etc. As illustrated, these are features (104) provided to the ML model (102). The ML model then generates a prediction (106) for the bug and a confidence for the prediction. The bug deduplication model can be a binary classification model that predicts if a new (incoming) bug is a duplicate of an existing open bug in the database. A duplicate bug is one that shares a root cause with an existing bug in the bug database. When a duplicate bug is identified it can be closed automatically, or, alternatively, a comment can be inserted linking it to an existing bug in the bug database.



SOURCE

Bug ID	Title	Description
1	title_1	desc_1
2	title_2	desc_2
3	title_3	desc_3
4	title_4	desc_4

DESTINATION

Bug ID	Title	Description
1	title_1	desc_1
2	title_2	desc_2
3	title_3	desc_3
4	title_4	desc_4

SRC Bug ID	TGT Bug ID	SRC Title	TGT Title	SRC Description	TGT Description	IS_DUPLICATE?
1	2	title_1	title_2	desc_1	desc_2	0
1	3	title_1	title_3	desc_1	desc_3	1
1	4	title_1	title_4	desc_1	desc_4	0
...	...	...	...	...	...	...
4	3	title_4	title_3	desc_4	desc_3	0

Fig. 2: Data generation pipeline for training

Fig. 2 illustrates a pipeline for generating training data for the bug deduplication ML model. Previously resolved bugs are grouped into (source, destination) pairs along with their features. The manually determined status of a pair is used to mark the ground truth (output label) of the pair, e.g., whether the constituent bugs of the pair are duplicates or not. For example, in Fig. 2, the constituents of bug-pairs (1,2), (1,4), and (4,3) are not duplicates, whereas the constituents of bug-pair (1,3) are duplicates. Duplicate bugs serve as positive training examples, while non-duplicate bugs serve as negative training examples. Effectively, the data generation pipeline uses features extracted from bug entities existing in the database to form their cartesian products, which are then provided to the ML model as training data. The pipeline also labels or identifies the status of the bugs.

During training, the data generation (or preparation) strategy enables the ML bug deduplication model to identify correlations in the features of a pair of resolved bugs. The model learns the correlations between the input features and the output label (which is indicative of whether the input bug-pair constitutes a duplicate or not). During inference, the cartesian product of a new (incoming) bug is computed with existing open bugs and fed to the trained model for prediction.

Bug routing

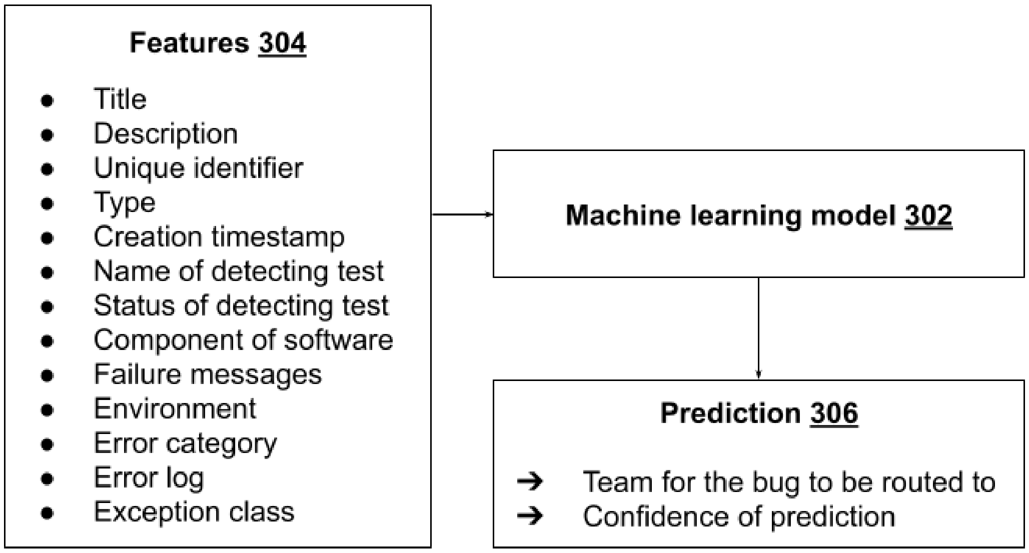


Fig. 3: Bug Routing

Fig. 3 illustrates bug routing. A single bug entity comprises bug attributes such as title, description, unique identifier, bug type, creation timestamp, last comment, etc., as well as signals identified in a test failure, e.g., test name, test status, environment name, error category, exception class, etc. As illustrated, these features (304) are provided as input to a trained multi-class machine-learning and classification model (302). The model generates a prediction (306) for the bug and a confidence for the prediction.

The output label generated by the bug-routing model is a prediction of the unique team identifier of the team the bug is optimally assigned to. For effective model training, e.g., balanced label distribution, the team identifier is advantageously used as output label (rather than names of individual developers). Heuristics can be used to generate high fidelity, ground truth, output labels that incorporate insights gathered from historically resolved bugs and from teams handling the test failures.

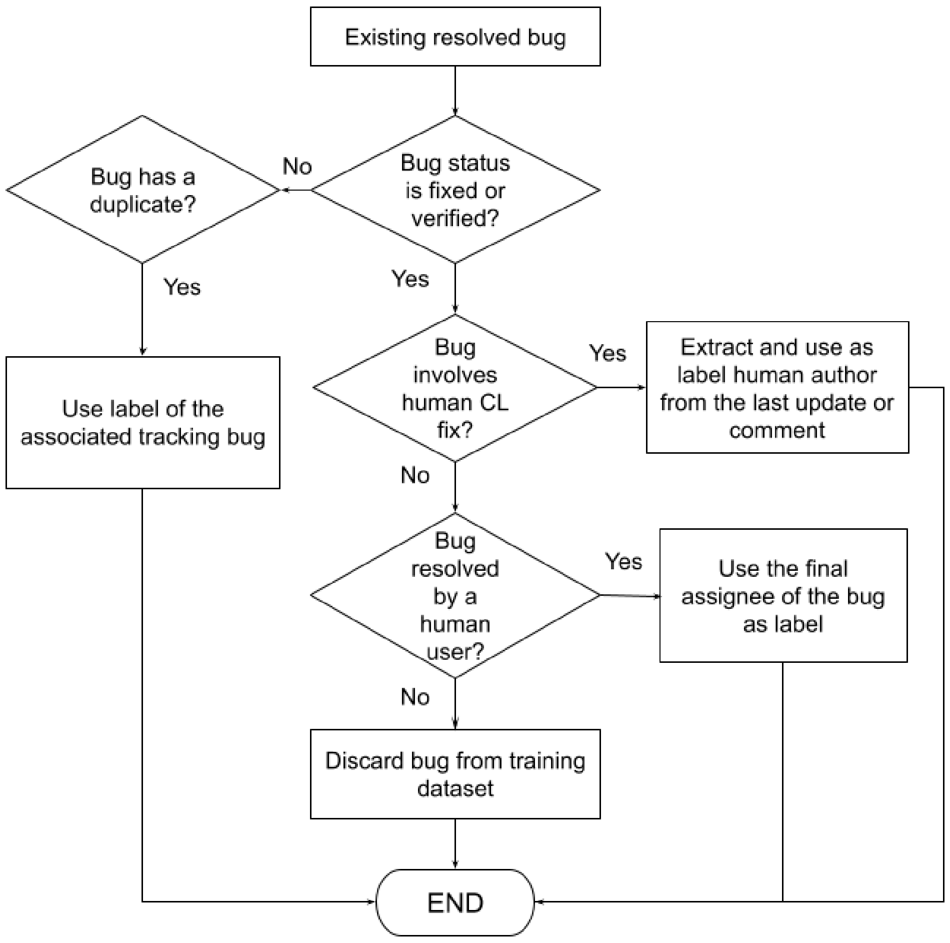

Existing resolved bug
Bug status is fixed or verified?
No
Bug has a duplicate?
Yes
Yes
Use label of the associated tracking bug
Bug involves human CL fix?
Yes
Extract and use as label human author from the last update or comment
No
Bug resolved by a human user?
Yes
Use the final assignee of the bug as label
No
Discard bug from training dataset
END

Fig. 4: Selecting resolved bugs to be included in the training set for the bug-routing model

Fig. 4 illustrates an example method to select resolved bugs to be included in the training set used to train the bug-routing ML model. Generally, fixed bugs that have been resolved by a human user (or that are associated with a code change with a human author) can be added to the training set. In such cases, the team that the bug is assigned to (the ground truth label) is extracted based on updates to the code repository. If there is a code fix from a human author, the team that the author belongs to is identified. If no such code fix exists, it is determined whether the bug was resolved by a human user, and the team of that human user is identified. An unfixed bug with a duplicate is assigned to the team of the duplicate bug. Bugs that don’t fall under the above classifications, e.g., bugs that were resolved not by a human user but by a machine user, bugs associated with code that was automatically pushed into the repository, or whose change-list author is a machine user, etc., are discarded. The flowchart of Fig. 4 is also referred to as a labeling strategy since it labels the ground truth for the purposes of training and performance tuning of the bug-routing ML model.

Similar to bug deduplication, the data generation pipeline extracts features from the bugs. During training, the labeling procedure (Fig. 4) is used in the data generation pipeline to obtain the corresponding output labels. The data source used for training includes resolved bugs in a database of bugs. The features and the labels extracted during the data preparation phase enable the trained bug routing model to identify and capture different patterns in the resolved bugs. The model learns and establishes relationships between the input bug patterns and the output label - the appropriate team to handle the bug.

During inference, a new (incoming) bug is processed using the data generation pipeline to extract its features. The processed bug is fed to the trained bug routing model which provides a prediction of the team to which the bug is to be routed.

To ensure that the bug deduplication and bug routing models adapt to the latest trends, the models can be periodically re-trained. The inference workflow can be set up to either run on a real-time basis (instantaneous prediction) or in batch fashion (scheduled prediction). In scenarios where the bug duplication and routing patterns are specific to a particular team or a product, customized models can be trained and used.

CONCLUSION

This disclosure describes techniques for bug deduplication and bug routing based on machine learning (ML). Per the techniques, a binary machine classification model is trained to aggregate bugs with a common root cause. Bugs in a class of bugs with a common root cause are deduplicated, e.g., represented by just one of the multiple bugs in the class. Further, a multi-class ML model is trained to predict the right team for handling a new (incoming) bug.