Triaging Software Bugs[1]

Background

It is not unusual for large software projects to track hundreds of thousands of bugs over time. The Mozilla defect tracking dataset, for example, contains over 150,000 bugs reported. Bugs cause software to produce incorrect or unexpected results, and behave in unintended ways. The annual cost of software bugs is estimated at $59.5 billion[2].

Bugs are assigned to developers through a process known as *bug triaging.* Bug triaging involves reviewing bugs with the goal of prioritizing development work towards fixing the underlying defects[3]. Some bugs are prioritized for immediate attention while others are carried over to future releases of the software. While prioritization is done using several factors, the success of the triaging process depends on whether the bugs prioritized for fixing are actually fixed or not.

Bug triaging is labor-intensive and time consuming if done manually. A better understanding of which bugs get fixed can inform the design of improved triaging policies for bug management as well as help developers be more productive and efficient in spending their time on bugs. It can also help set user expectations when they first report a bug. Bugs that are highly unlikely to be fixed can be flagged or just closed and users advised of appropriate workarounds.

Bug Reports

When a software bug is reported (by developers, QA engineers, product managers, customer support, or even end users) it is assigned to a developer to be fixed[4]. Once the bug is assigned, the developer will work on it and will add commentary as he/she makes progress. Other comments can be logged during this time as well, from design discussions, QA notes, software reviews, etc. Over its lifetime, a bug will transition through various states and can even move between different developers

---

[1] This case was prepared by Mr. Sandeep Sikka in collaboration with Dr. Dorit Nevo from the Lally School of Management. Mr. Sikka is a data scientist and software engineer with over 15 years of experience. He holds an M. S. in Computer Science from Washington University and a B. Tech in Computer Science from Indian Institute of Technology. Dr. Nevo is an Associate Professor of Information Systems and director of the MS program in Business Analytics. We also thank Mr. Majeed Simaan from the Lally School of Managemetn for his help with this case.

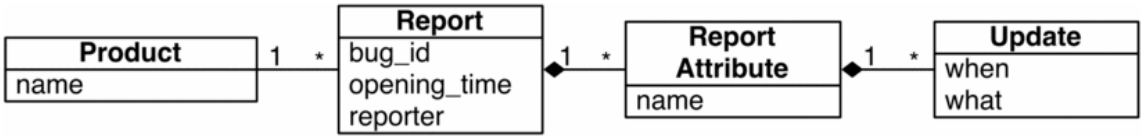
[2] P Bhattacharya and I Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging", *Software Maintenance* (ICSM) 2010 (ieeexplore.ieee.org)

[3] P J Guo, T Zimmermann, N Nagappan and B Murphy. "Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows", *Proceedings of the 32th International Conference on Software Engineering* (ICSE 2010), Cape Town, South Africa, May 2010

[4] An example of a bug report for Mozilla: https://bugzilla.mozilla.org/show_bug.cgi?id=100009

or departments (i.e. get reassigned) until it is resolved. Naturally, this process generates much data that can be captured and analyzed to gain insights on bug resolution.

In this case we will use The Eclipse and Mozilla Defect Tracking Dataset[5] (https://github.com/ansymo/msr2013-bug_dataset), which contains bug reports for Eclipse and Mozilla. Given the limited time frame, we will focus on the Eclipse subset, which includes the following four products: Eclipse Platform, which supports other development tools, the Java Development Tools (JDT), C/C++ Development Tools (CDT) and Plug-in Development Environment (PDE). The file structure within this dataset is shown below:



Source: Lamkanfi et al. (2013); Note that * means a one-to-many relationship

That is, for each product there is a list of bug reports (with reporter and opening time) and a set of report attributes. The following attributes are included in the dataset:

- **priority**: indicates how soon the bug should be fixed. Varies between P1 (highest priority) and P5 (lowest priority).
- **severity**: an indication of the impact of the bug on the software system. Values include: trivial, minor, normal, major, critical and blocker.
- **product**: the specific software application the bug is related to.
- **component**: the relevant subsystem of the product for the reported bug (can be more than one).
- **bug_status**: the current state of a bug. Values include: unconfirmed, new, assigned, reopened, ready, resolved, verified.
- **resolution**: what happened to the bug. Values include: fixed, invalid, wontfix, duplicate, worksforme, incomplete.
- **assigned_to**: an identifier field for the developer who got assigned the bug.
- **cc**: literally a cc field, indicating users who are interested in the progress of this bug.
- s**hort_desc**: a one-line summary describing the bug.
- **version**: the version of the product the bug was found in.
- **op_sys**: the operating system against which the bug is reported.

[5] A Lamkanfi, J Perez and S Demeyer, "The Eclipse and Mozilla Defect Tracking Dataset: a Genuine Dataset for Mining Bug Information", *MSR '13: Proceedings of the 10th Working Conference on Mining Software Repositories,* May 18-–19, 2013. For the original data please visit: https://github.com/ansymo/msr2013-bug_dataset

Your task

This link contains the dataset for bugs reported for the Eclipse browser to be used in this competition:
https://www.dropbox.com/s/zcp1u69qwiub880/Eclipse.zip?dl=0

The zipped directory contains twelve csv files corresponding to the above listed attributes. Each file contains a list of updates and changes that have been performed during the bug's lifetime. Each update is indicated using a UNIX timestamp (when), the new value of the attribute of the bug report (what), the reporter (who), and the bug (id). Two report attributes are unchangeable: the reporter and the opening time of the reported bug. They are included in the 'reports.csv' file.

Note that the file titled "cc.csv" needs to be reformatted as some of the records shifted.

Your task is to explore the Eclipse defect tracking dataset and to develop insight into the factors that affect the likelihood of a bug being fixed. With this understanding, you are asked to develop a prediction model for which bugs would be fixed. You can use any software and modeling approach to develop this prediction. *Please work only on the Eclipse dataset and refrain from using any other external sources such as Websites, blogs, articles, etc.*

To get you started, below are some examples of factors you may wish to consider in your model (these are not readily available in the data, you need to compute them):

1. How does the past success rate of a bug assignee impact the likelihood of a bug being fixed?
2. What influence does the reputation of a bug reporter and the success rate of their past bug reports have on the likelihood of a bug being fixed?
3. How do bug report edits and the editors making those edits influence whether a bug is fixed or not?
4. What impact do bug reassignments have on the likelihood of a bug being fixed?
5. Do bug re-openings influence the bug fix likelihood?
6. How does the time a bug has been open influence bug fix likelihood?
7. How does the software module that a bug gets assigned to impact the likelihood of a bug being fixed?
8. Do social relationships between the various bug participants on current and past bugs influence the likelihood of a bug being fixed?
9. How does the age of the software version the bug is being reported on impact the bug fix likelihood?

10. Does the nature of the bug (e.g.: a) user interface, b) end user environment concerns, c) network flakiness) influence the likelihood of a bug being fixed?

Deliverables

You are expected to provide the following:

(1) Model predictions on hold-out data set: Run your model on a hold-out dataset and submit a file containing bug IDs and you prediction for each bug.
(2) Presentation: a 10 minutes presentation of your exploratory work and insights leading to the creation of your model. Your presentations should include: (1) explanation of the problem and dataset, (2) discussion of the exploratory work and creation of prediction variables, (3) explanation of the logic of solution approach.