Automated Bug Triaging Using Machine Learning

Sabancı Üniversitesi
FACULTY OF ENGINEERING AND NATURAL SCIENCES

Student(s)

Erdem Bozkurt
Emre Özdinçer
Oğuz Özsaygın
Ege Yapıcıoğlu

Faculty Member(s)

Berrin Yanıkoğlu
Cemal Yılmaz

Company Advisor(s)

Serter Bekler

Siemens
SIEMENS

ABSTRACT

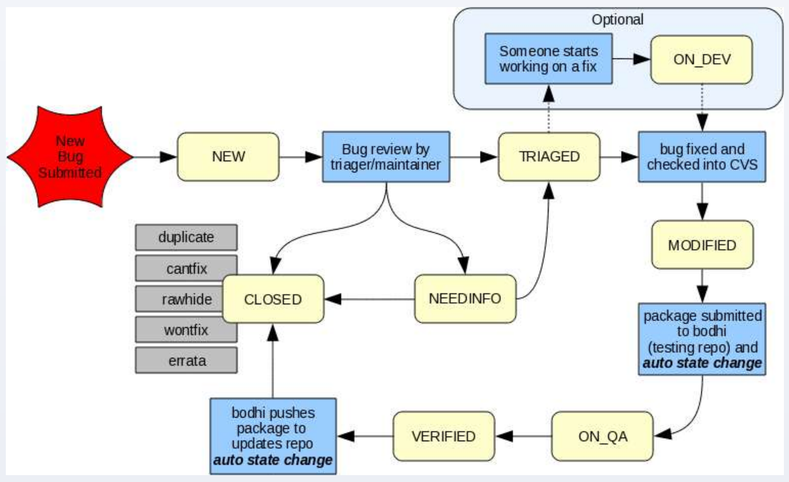

Figure 1, Bug Triaging Lifecycle Example

Developers and testers encounter with many challenges in software development lifecycles. During the testing processes, several bugs and defects of a product are detected by testers. Each bug is submitted to a bug tracking system with its relevant information such as summary, severity and priority. This project aims to simplify and shorten bug triaging process by finding the duplicate bugs and grouping them together.

OBJECTIVES

- To implement a machine learning system that successfully identifies bugs as duplicate and attaches it to a previous bug, therefore saving development time
- To deploy this system in Siemens' development environment

PROJECT DETAILS (AUTOMATED DUPLICATE BUG DETECTION)

We used Mozilla's Bugzilla repository as our data source. All our approaches were done with Python, and we also used libraries such as Scikit-learn and NLTK for text processing and machine learning.We downloaded all bugs which were created in the last 10 years which in total accounted for 952,361 bugs. We decided to focus on top 5 products (Core, Firefox, Firefox OS, Toolkit, Thunderbird) because they contained almost half of the bugs. Our first step was to cluster the duplicates we had in those bugs product by product. For this we used bug_id and dupe_of fields, since if a bug had dupe_of field not empty, it contained the id of one of it's duplicates. We used this approach for all bugs we had and the ones created in 2017.

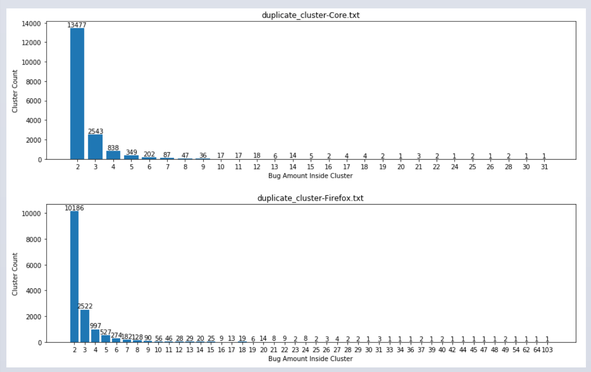

Figure 2, Duplicate clusters for top 2 products, Core and Firefox

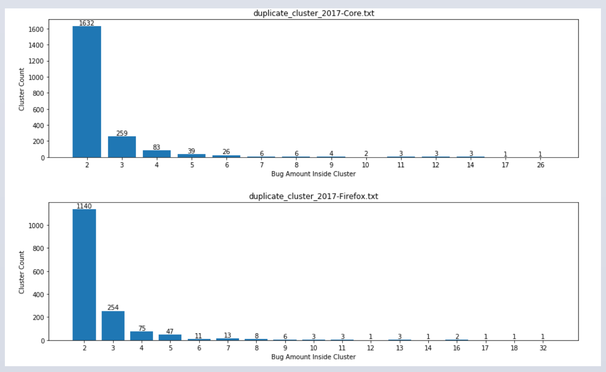

Figure 3, Duplicate clusters for top 2 products, Core and Firefox (only 2017 data)

After the clustering step, we did use bag of words approach and found cosine similarities of bugs in top 5 products. In this step we didn't use any preprocessing methods. We also selected same amount of random bugs and also calculated their cosine similarities to see if we can use cosine similarities to identify duplicates.

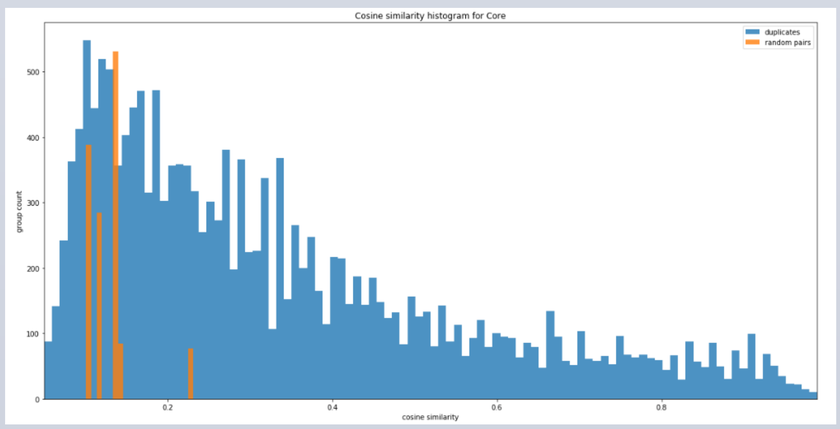

Figure 4, Initial cosine similarity histogram for Core

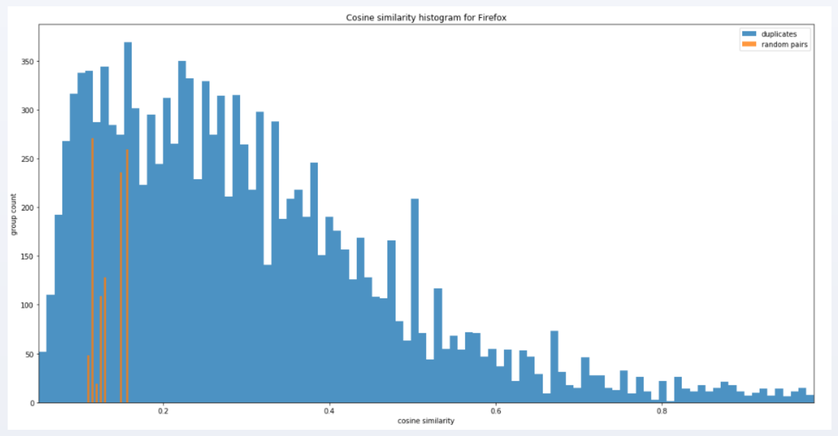

Figure 5, Initial cosine similarity histogram for Firefox

After calculating cosine similarities and seeing them in a histogram, we also tried an approach with k-means algorithm for Firefox. In this step we also did some preprocessing such as removing stop words and we also used TfIdfVectorizer (tf-idf instead of bag of words). Although in some cases this approach didn't gave bad results, it wasn't reliable. Also we had to give cluster count, to get a result.

Therefore we abandoned k-means approach and returned back to calculating cosine similarities for clustering. This time we did calculate cosine similarities in a different way. We ensured that only duplicates from same clusters were compared, and we also took random bugs to have a non duplicate group to create a baseline for comparison. Also we had preprocessing such as removing stop words and stemming with NLTK library. We also had to use 2017 data as base, add some bugs from 2007-2017 data, and remove some from 2017 data. We had to do this because for clustering bugs we used bug_id field, but we either didn't have some of the bugs with those id's as downloaded, and when we removed those cluster had 1 item only.

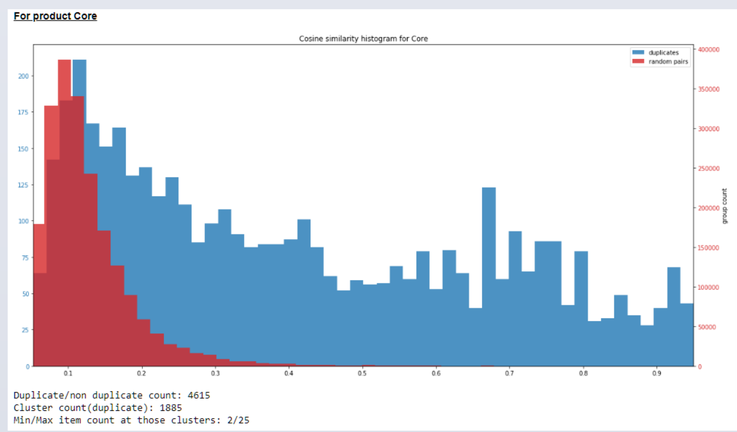

Figure 6, Cosine similarity histogram with preprocessing and adjusting pairs for product Core

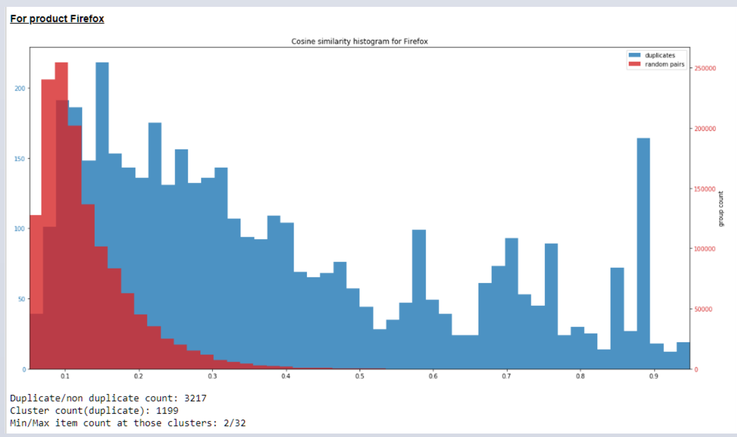

Figure 7, Cosine similarity histogram with preprocessing and adjusting pairs for product Firefox

Conclusion:

With our final approach we had managed to find a pattern such that if a bug pair had cosine similarity score ≥ 0.3 we can say they are duplicates with very high probability. If we also include component comparison, this also enhances that probability.

REFERENCES

[1] Bugzilla Main Page, https://bugzilla.mozilla.org, last accessed 2018/02/01.

[2] Scikit-learn: Unsupervised learning, http://scikit-learn.org/stable/unsupervised_learning.html, last accessed 2018/05/09

[3] NTLK Natural Language Processing Tool Kit, https://www.nltk.org/ , last accessed 2018/05/01