Developing 3 Products to Help Linux Kernel Project Bug Report Classification

Berke Ucvet
Istanbul, Turkey
berkeucvet@sabanciuniv.edu

Okan Sarp Kaya
Istanbul, Turkey
okansarp@sabanciuniv.edu

Yigit Tamer
Istanbul, Turkey
yigittamer@sabanciuniv.edu

I. Introduction

The Linux kernel is a free and open-source, monolithic, modular, multitasking, Unix-like operating system kernel. Even though lots of developers keep working on this project constantly, just like any software, this project also has many reported bugs. Bug reports are essential for any large software development process since end-users can encounter extreme edge cases which the developers may not foresee.

In this report, we are going to explain the products we developed in order to solve some of the main problems of linux kernel project bug reports.

The first problem with bug reports is that they are made by humans and they may have misinformation inside. For example, a linux kernel project bug report includes fields like product and component since there are around 120 components and around 20 products in the project and it would be very difficult to find and reproduce bugs without knowing the product and component in which the bug exists. If this product or component information is wrong, it would create problems for the developer that will try to fix the issue.

Our solution to this wrong product information being filed in the bug report problem is implementing a machine learning model that can predict the product and the component of a bug report by checking the summary field of the bug report.

Another problem about bug reports is that there are lots of duplicate bug reports. Usually, regular people file bug reports before even checking if a similar bug report exists. Therefore, even in the dataset we used to create everything that is explained in this report has lots of bug reports that are labeled as “duplicate”.

We developed another algorithm that checks a given bug report summary in order to see if this bug report is a duplicate of another bug report. This algorithm allowed us to detect duplicate reports even before they got filed.

The last problem of these bug reports that we addressed is the “will_not_fix” labeled ones. A bug report may be labeled as “will_not_fix” if the developer that examines this bug report is sure that this is not a bug that needs to be fixed or that this is not even a bug but a feature. There are many bug reports that are labeled this way. This means that there are lots of developers who work on this project selflessly and waste time with these bug reports that are not even a bug.

In order to handle this problem, we developed another algorithm that can identify a bug report that has the possibility to be a “will_not_fix” labeled one by checking its summary. This way, we will be able to prevent people from filing bug reports for features or bugs that are not going to be fixed in the near future.

As explained above, we ended up with 4 end-products that can be developed even further to increase their accuracy and reliability. These products are “Product Classifier”, “Component Classifier”, “Duplicate bug report identifier” and “WILL_NOT_FIX bug report identifier”. All these products are implemented using Python and several ML libraries.

II. Approach

For all of the products we designed, we used the Google Sentence Encoder from TensorFlow library in order to embed the summaries of the bug reports into vectors in a bag of word format. After embedding the summaries of around 27000 cleaned bug reports, we used different techniques to analyze the dataset and to implement the algorithms.

In product and component estimator we designed k-means algorithm and gave the algorithm summary of bug reports as an input (with google sentence encoder). After several tries we realized data was not well distributed and in order to solve it we used bugzillas advanced search queries in order to balance the data set for each feature datapoint as we can see below. The final dataset was pretty good and this solved the over-fitting problem.

```
print("Homogeneity: %0.3f" % metrics.homogeneity_score(train.Product, kmeans.labels_))
print("Completeness: %0.3f" % metrics.completeness_score(train.Product, kmeans.labels_))
print("V-measure: %0.3f" % metrics.v_measure_score(train.Product, kmeans.labels_))
print("Adjusted Rand-Index: %.3f"
      % metrics.adjusted_rand_score(train.Product, kmeans.labels_))
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(train_embeddings, train.Product, sample_size=1000))

#display(pd.crosstab(train['Clusters'], train['Product']))
```

```
Homogeneity: 1.000
Completeness: 1.000
V-measure: 1.000
Adjusted Rand-Index: 1.000
Silhouette Coefficient: 1.000
```

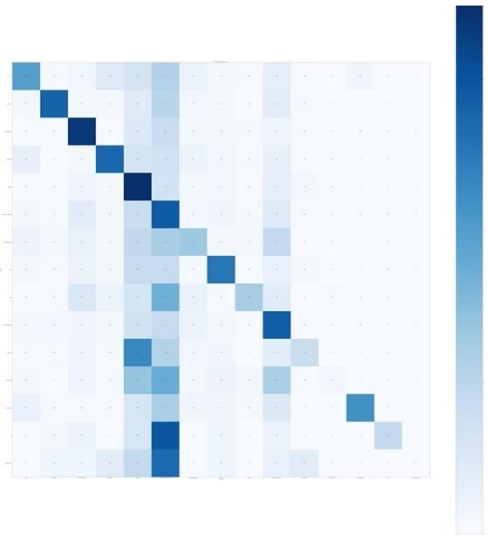
For duplicate identifier, we compare the vector representation of the input, which is the summary of a new bug report, with the dataset and if cosine similarity of the input and one of the entries in the dataset is higher than a certain threshold (we decided on 0.80 to be the threshold value after some brute force optimization) we considered the new input bug report as a duplicate one.

For WILL_NOT_FIX identifier, we compare the vector representation of the input, which is the summary of a new bug report, with the dataset again but this time dataset contains only “WILL_NOT_FIX” labeled bug reports and if cosine similarity of the input and one of the entries in the dataset is higher than a certain threshold (we decided on 0.80 to be the threshold value after some brute force

optimization) we considered the new input bug report as a "WILL_NOT_FIX" one.

III. Preprocessing

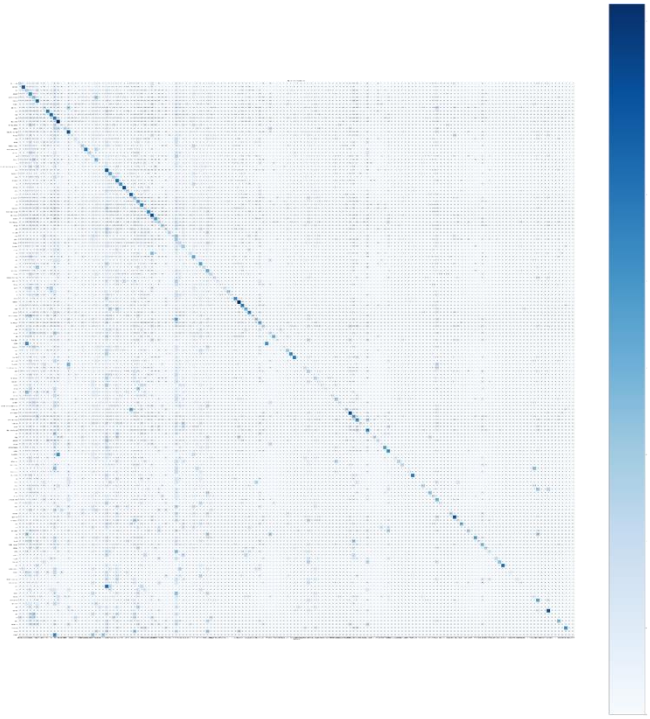
In order to increase accuracy we removed components and products labeled as "Other" since it is not possible to find any pattern between them.



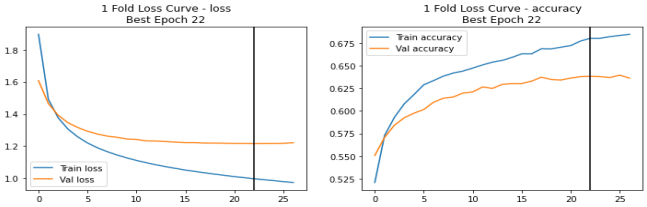
Another thing we did was removing columns that result in high false positive rates after we analyzed the confusion matrix you can see above as we can see "platform specific hardware" results in a lot of false positives after we remove it from the dataset our accuracy increased around %10.

For the component prediction, the problem with K-Means Clustering was the fact that there were way too many clusters (around 120). We decided that lowering the number of clusters will increase the accuracy of the model. We analyzed the dataset again and we decided to remove 20 components that have an insignificant number of entries. This method also increases the accuracy of the predictor by around %10.

As it can be seen in the figure, analyzing the confusion matrix of the component classifier was a bit more complicated. Therefore, we decided to leave the bias of the data as it was and focused more on merging components and removing some of them.

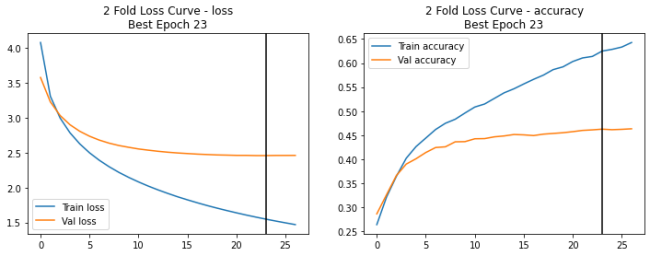


Our aim for the product classification task was to have an accuracy around 0.7. Since a random guess among 18 products (the number of products we had after the initial preprocessing) would have an accuracy of 0.05, this accuracy would be just enough for our case. We ended with an accuracy of ~0.64 for the product classification and since this number is less than our target, we kept on trying to improve the accuracy.



As it can be seen in the figures the initial results we had was a test accuracy of 0.6364. After removing 4 products with the least amount of entries the accuracy number increased to ~0.67. Still, not above our target but close enough for the scope of this project.

For the component classification task, our target was a bit lower with an expected accuracy of around 0.6. The reason behind this is the fact that a random guess among 121 products would have an accuracy of 0.00826. Increasing this accuracy by around 80 times was enough for our scope. We ended up with a test accuracy of 0.4633 after the initial preprocessing and just the product classification task, we kept on trying to improve.

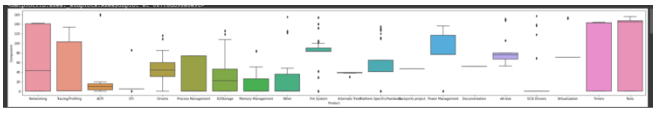


As it can be seen from the figure, working on this dataset with the same algorithm that we used for product classification was harder than we expected because the difference between training accuracy and validation accuracy was larger. This means that the trained model may not be an ideal one. We did not change the algorithm behind our model but we removed 40 components with the least amount of entries. This improved the accuracy of the model to around 0.49.

IV. FUTURE WORK

The main drawback of the models is mediocre accuracy score in order to improve it several methods can be implemented.

First, enlarging the data set with detailed description of bug report can help since we extract additional features like created date and modified date we can use them to calculate priority of the bug since important bugs can't be ignored for long. In addition to that we can add description text to the NLP algorithm to increase accuracy.



We didn't have time to research whether there is a true relationship between certain products and components since our dataset might be missbalenced for certain combinations but enlarging the dataset could help this one too since according to correlation adding bias will help to increase accuracy.

Since linux kernel bug reports contains a lot of technical terms(processor names, computer models, component names, memory locations) our normalization methods did not worked since generic lemmatization, stemming, stop word removal libraries can not handle this types of words finding or creating tailor made text normalization libraries definitely will help to finding accurate results.

V. CONCLUSION

As it is explained in the report, Linux Kernel Project can benefit the developed tools quite a lot. To conclude what we have done, we developed a classifier to predict the product and component information of a given bug report in order to check if the specified product and component information is correct, we also developed another classifier to predict if a given bug report is a duplicate of an already existing one, and lastly, we developed another classifier to predict if a given bug report is a candidate to be labeled as "WILL_NOT_FIX". These tools will help the developers that work on this project to fix these bugs by allowing them to not waste time with bug reports with misinformation.