filtering so that we could build a predictor using bug attributes only. Therefore, we decided to set the number of bugs in the first step at least 10000 denoted as constant value(C_v). We generated five different training sets each of which contains different number of bugs that will be fed into the models. To make the same increase in each set, we chose increment value(I_v) as I_v = (15513-10000)/5 and found as 1103. Thus, the first group should be $C_v + 1* I_v$, second group should be $C_v + 2* I_v$, and so on. Approximately 1% of the group is taken as test set, and its size is fixed at 100. The train and test groups are shown in Table IV.

TABLE IV. EXPERIMENTAL GROUPS

Group number	*Train instance count*	*Test instance count*
1	11003	100
2	12106	100
3	13209	100
4	14312	100
5	15413	100

B. Algorithms

The first model that use bug attributes only is trained with Random Forest Classifier. Random Forest is a collection of the decision trees by using the subset of the features and data to get the best result. The predictions are done by using these decision trees, their results are aggregated, and the majority class is selected as a final class [5]. A single decision tree can overfit the data, but Random Forest combines the results to overcome the problem of overfitting.

In the second model, for textual similarity search, we used the Lucene developed the by the Apache Foundation, which is a high-performance, text search engine library written in Java [12]. It provides powerful, ranked, field based, memory efficient search capabilities. It is a code library, and can be combined with different applications by using the API (Application Program Interface). It supports rich, custom and multi field queries. Lucene scoring supports different similarities such as TF-IDF, and BM25. Instead of using classical Lucene TF-IDF, we used the BM25 Similarity. BM25 is based on the TF and IDF. BM25 have some improvements than classical Lucene TF-IDF such as term frequency saturation, tuning the influence of the document length, etc. [21]. We used Lucene Multi Field Query Parser to make search on multiple fields because we have bug description and summary data. The fields are multiplied by the boost factor when calculating Lucene score, we took same weights on these fields. After applying second model, we have normalized score values. Thus we define threshold value as T_v to consider the bugs that have at least similarity score T_v , based on the study in [23].

The third model, our proposed approach, consists of two steps, essentially combining the first and second models. After the textual similarity calculation using Lucene, the selected bugs based on the similarity score are used as the input for the next step. As stated in the paper [23] using only textual attributes makes predictions accuracy closer to the original ones. Therefore, we also believe that this step should filter the bugs that are closest to the bug selected from the test set. This process needs to considered carefully, because the textual similarity calculation may lower the number of the bugs that are going to determine the training set size for the second step. We focus on the recent bugs with the similarity score ranging from 0 to 0.9, and then build the second model with the filtered bugs' attributes only, using Random Forest classifier.

V. EVALUATION CRITERIA

In our dataset, we have four bug categories, namely *very slow*, *slow*, *fast*, and *very fast*. For each category, we calculate Precision and Recall as follows: First we calculate the confusion matrix corresponding to each category, as shown below. These metrics are for category *very fast* denoted as *vf*, others are like.

- T_{pvf} : number of true positives for *vf* are the number of *vf* instances that are predicted as *vf*
- $T_{nvf:}$ number of true negatives for *vf* are the number of non-vf instances that are predicted as *non-vf*
- $F_{pvf:}$ number of false positives for *vf* are all *non-vf* instances that are predicted as *vf*
- F_{nvf} : number of false negatives for *vf* are all *vf* instances that are predicted as *non-vf*

For Class *vf*, Precision(P) is then calculated as in the (1) and Recall(R) in the (2). The same calculation was done for other bug categories. Furthermore, we define *Total Accuracy* of our proposed models over all categories, based on the (3). In this equation, T_a corresponds to the total accuracy of a model, whereas T_c is the total correctly classified instances and T_t is the total number of predictions.

$$Pvf = \frac{T_{pvf}}{T_{pvf}+F_{pvf}} \quad (1)$$

$$Rvf = \frac{T_{pvf}}{T_{pvf}+F_{nvf}} \quad (2)$$

$$T_a = \frac{T_c}{T_t} \quad (3)$$

In addition to these evaluation criteria, for Model II, we calculated the accuracy of the model with respect to the changing values of a threshold. This threshold, whose value range from 0 to 0.9 with an increase of 0.1, corresponds to the normalized Lucene score. A similar approach was proposed in [23], in which the authors report that higher Lucene score represents more similarity among the bugs. We would also like to investigate this in our dataset, by changing the score and calculating the accuracy of Model II in every iteration.

Besides these, we applied the Kruskal-Wallis H-test [11] on these observations. This test is used to check whether of two or more models have statistically different performance values. If two groups' medians are likely the same, we could say that, it cannot be claimed that they are different at all, and conclude that their distributions may be similar. On the other hand, for any level of significance level α, if the computed statistic of Kruskal-Wallis is greater than 0.05, we reject the null hypothesis, and accept that two groups are statistically significant from each other in terms of medians.