A Study on Refactoring Technique Frequency for SATD Comment Removals and Refactoring Prediction Modeling

Julia Okvath
Rochester Institute of Technology
jso5289@rit.edu

Praveen Chandrasekaran
Rochester Institute of Technology
pc2846@rit.edu

Mayank Singh
Rochester Institute of Technology
ms6674@rit.edu

Nandhini Shankar Lakshman
Rochester Institute of Technology
nl7222@g.rit.edu

I. INTRODUCTION

Most of the time, software developers are pushed by project managers in organizations to develop a new product or feature that is ready to be introduced into the market. The pressure builds up because of the deadline that is already aligned with key stakeholders or sometimes with the customers. During these pressurized situations, it is prevalent for the software experts to engineer a project with some shortcuts to meet short term goals. In this way, developers and product managers are bound to compromise the code quality which might lead to problems in the future. However, these problems may not make the released product unreliable to the customer, making the maintenance convoluted for the organization. Specifically, software developed in a sub-optimal way that contains artifacts that ought to be addressed in the future is called Technical Debt or TD. When the software development team makes a deliberate decision to compromise on code quality, it is typical for the developers to self admit TD using source code comments, this is known as Self-Admitted Technical Debt or SATD [1].

Technical debts can also be due to many other reasons such as, lack of understanding of the consequences of TDs, lack of code cohesion, lack of test-driven development. These TDs are often removed by refactoring the specific code. For example, the Cardinal Rule of Software Evolution is refactoring, which Martin Fowler defines as *"a change made to the internal structure of the software to make it easier to understand and cheaper to modify without changing its observable behavior"* [2]. However, the function of refactoring has evolved to become a critical factor in agile methodologies and is one of the commonly used approaches to remove SATD [3]. There are multiple reasons to refactor the code, some of them are, the existence of duplicate code, poor class cohesion, long loops, overloaded class etc.

Previous studies have shown refactoring activities have significantly higher odds to co-occur with the removal of SATD comments [4]. Another study takes this work further and creates a model which recommends a refactoring activity for a given SATD comment [5]. A limitation to this work is the assumption that a given SATD comment correlates with the refactoring done in the same commit. This assumption can lead to an improper refactoring technique to be recommended. Another limitation identified is the use of a neural network to predict appropriate refactors. Neural networks are known as black boxes - it is not clear how the given output relates to the inputs. This can be problematic in software development since it limits understanding of solutions between developers.

To address the previous stated limitations, we propose a solution that starts with a frequency analysis between refactoring activity and SATD removal. We will determine the types of refactoring that occur frequently with the removal of a given SATD comment. Using this knowledge, we will choose simplistic ML models keeping model's interpretability in mind and train the ML models to predict the appropriate refactoring given a SATD comment. Following is an example to give more clarity on what is intended to be done. Our trained ML model will take the SATD comments as inputs, an example input can be like,'FIXME: use analysis name'. This comment has a "FIXME" tag and the input comment will be broken down to process and the ML model will give an appropriate recommendation, for the above comment the expected recommendation would be "Rename Method".The research questions to be answered by these activities are as follows:

- **RQ1:** Which specific refactoring activities occur most frequently with the removal of SATD comments?
- **RQ2:** Can we implement an ML solution to identify the appropriate refactoring activity for a given SATD comment to be removed?

II. STUDY DESIGN

A. Dataset Descriptions

There are 3 datasets available for the initial study design and experiments. The first dataset is comprised of github commit

IDs, associated project IDs, SATD comments that existed in the code prior to the commit (a.k.a. V1 comment), the comment that existed in the code after the commit (a.k.a. V2 comment), and the defined SATD commit type (i.e. whether the commit resulted in an SATD comment removal or not). The second dataset consists of github commit IDs and the associated type of refactoring(s) which occurred for a given commit. The third dataset contains the projects IDs, project names, and associated github URLs to each repository.

B. Approach Overview

Figure 1 demonstrates a high level summary of the approach taken for RQ2. The first step is querying the 3 datasets from an SQL database and then merging the 3 extracted tables using the common commit IDs and project IDs in the respective datasets.

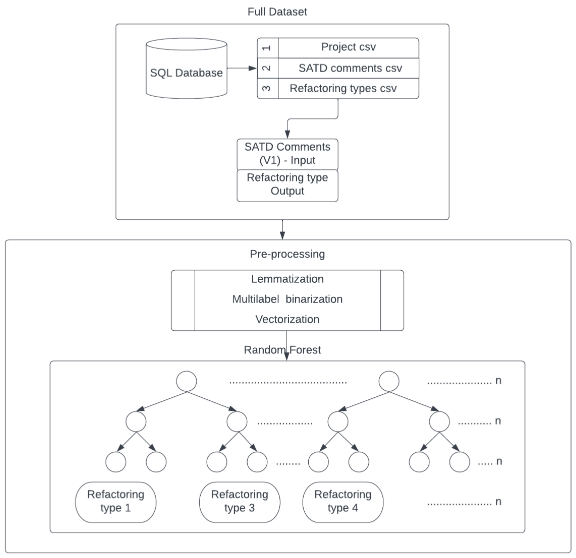


Fig. 1. Whitebox flowchart for RQ2

With this merged dataset we can now consider using the V1 comments as an input to the model and have the output be the predicted refactoring types. The first step in this effort is to only consider a subset of commits that correspond to SATD removals. Next the remaining V1 comments must be tokenized/vectorized in preparation for input to the ML model. StopWords were used to eliminate the redundant elements like punctuation. In-order to work with multiple classes (refactoring types) they are converted into unique columns and are encoded as 0 or 1 respectively. Different pre-processing methods were used to improve the performance of the model. 3 vectorizers were used for pre-processing: countvectorizer, tf-IDF and fastText.

These vectorizers were used to fit the V1 comments column to deal with the frequently occurring important words. Once complete, the prepared dataset will be split for training and testing. Finally the model will be evaluated on the test dataset.

III. Experiments

RQ1: Which specific refactoring activities occur most frequently with the removal of SATD comments?

A bar graph was created to examine the frequencies of refactoring types where SATD was removed. From figure 2, it is evident that the refactoring types that occur most frequently with the removal of SATD commits are 'Rename Variable', 'Rename Method' and 'Change Variable Type'.

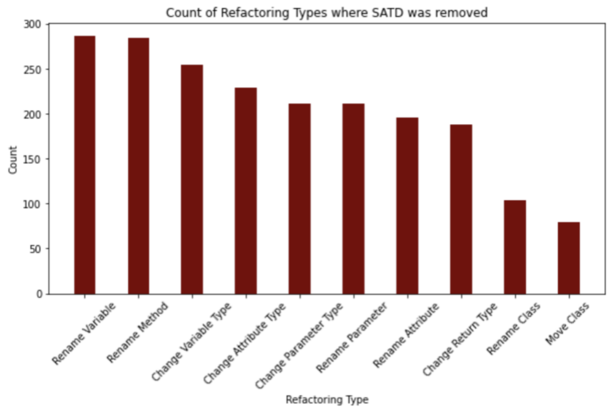
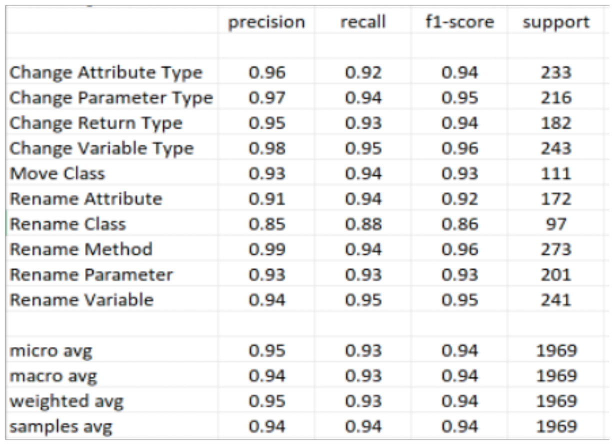


Fig. 2. Refactoring type count for commits where SATD was removed.

RQ2: Can we implement an ML solution to identify the appropriate refactoring activity for a given SATD comment to be removed?

All models' performances will be measured using well known metrics such as precision, recall, and F-measure.

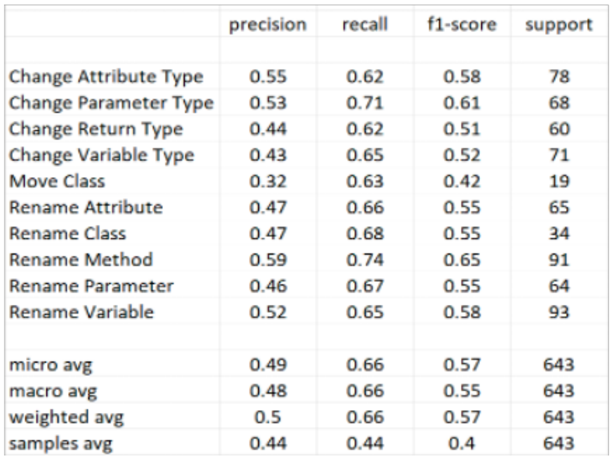
Two models were investigated during the preliminary analysis. These are the Random Forest Classifier and Logistic Regression Model.



	precision	recall	f1-score	support
Change Attribute Type	0.96	0.92	0.94	233
Change Parameter Type	0.97	0.94	0.95	216
Change Return Type	0.95	0.93	0.94	182
Change Variable Type	0.98	0.95	0.96	243
Move Class	0.93	0.94	0.93	111
Rename Attribute	0.91	0.94	0.92	172
Rename Class	0.85	0.88	0.86	97
Rename Method	0.99	0.94	0.96	273
Rename Parameter	0.93	0.93	0.93	201
Rename Variable	0.94	0.95	0.95	241
micro avg	0.95	0.93	0.94	1969
macro avg	0.94	0.93	0.94	1969
weighted avg	0.95	0.93	0.94	1969
samples avg	0.94	0.94	0.94	1969

Fig. 3. Random Forest Model's performance using tf-IDF vectorizer on the train dataset.

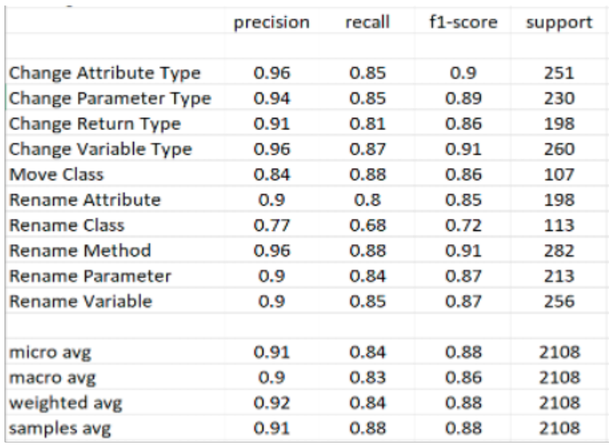
Figures 3 and 4 depict the performance metrics of Random Forest without hyperparameter tuning using the tf-IDF vector-



	precision	recall	f1-score	support
Change Attribute Type	0.55	0.62	0.58	78
Change Parameter Type	0.53	0.71	0.61	68
Change Return Type	0.44	0.62	0.51	60
Change Variable Type	0.43	0.65	0.52	71
Move Class	0.32	0.63	0.42	19
Rename Attribute	0.47	0.66	0.55	65
Rename Class	0.47	0.68	0.55	34
Rename Method	0.59	0.74	0.65	91
Rename Parameter	0.46	0.67	0.55	64
Rename Variable	0.52	0.65	0.58	93
micro avg	0.49	0.66	0.57	643
macro avg	0.48	0.66	0.55	643
weighted avg	0.5	0.66	0.57	643
samples avg	0.44	0.44	0.4	643

Fig. 4. Random Forest Model's performance using tf-IDF vectorizer on the test dataset.

izer. The metrics are shown for both train and test datasets. It is clear from the comparison of the weighted average that the model performs really good with the training data, which means the model has overfitted the training data and the model lacks generalization. We wanted to try improve the model performance by using different preprocessing methods.

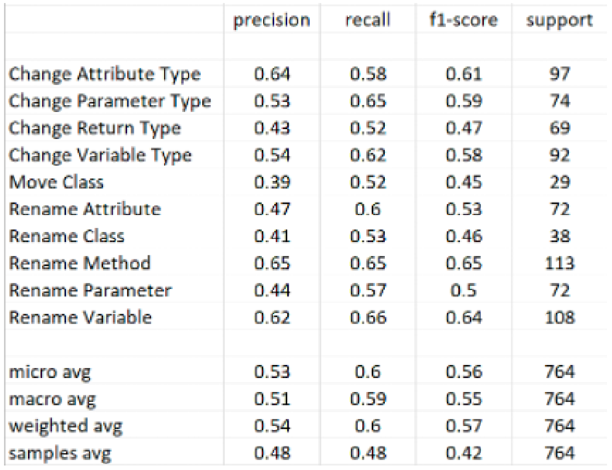


	precision	recall	f1-score	support
Change Attribute Type	0.96	0.85	0.9	251
Change Parameter Type	0.94	0.85	0.89	230
Change Return Type	0.91	0.81	0.86	198
Change Variable Type	0.96	0.87	0.91	260
Move Class	0.84	0.88	0.86	107
Rename Attribute	0.9	0.8	0.85	198
Rename Class	0.77	0.68	0.72	113
Rename Method	0.96	0.88	0.91	282
Rename Parameter	0.9	0.84	0.87	213
Rename Variable	0.9	0.85	0.87	256
micro avg	0.91	0.84	0.88	2108
macro avg	0.9	0.83	0.86	2108
weighted avg	0.92	0.84	0.88	2108
samples avg	0.91	0.88	0.88	2108

Fig. 5. Random Forest Model's performance using FastText vectorizer on the train dataset.

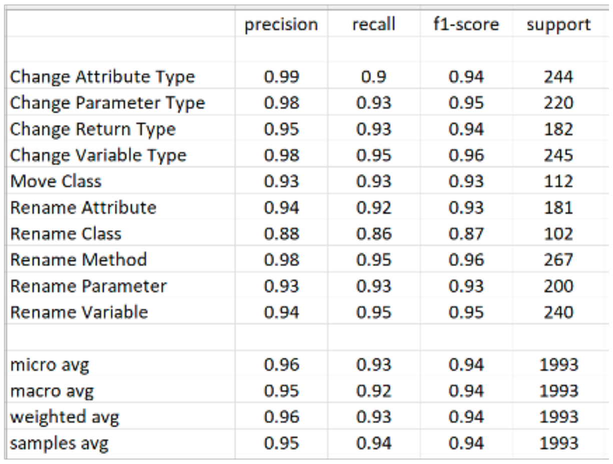
Figures 5 and 6 depict the results of the random forest model after FastText was used for preprocessing. Here the weighted average precision score has increased by 0.04 compared to the weighted average precision score using the TF-IDF vectorizer.

Figures 7 and 8 depict the results of the random forest model using the countvectorizer on the train dataset and the test dataset. The weighted average of the precision score has increased to 0.69 in figure 8 which is a 15



	precision	recall	f1-score	support
Change Attribute Type	0.64	0.58	0.61	97
Change Parameter Type	0.53	0.65	0.59	74
Change Return Type	0.43	0.52	0.47	69
Change Variable Type	0.54	0.62	0.58	92
Move Class	0.39	0.52	0.45	29
Rename Attribute	0.47	0.6	0.53	72
Rename Class	0.41	0.53	0.46	38
Rename Method	0.65	0.65	0.65	113
Rename Parameter	0.44	0.57	0.5	72
Rename Variable	0.62	0.66	0.64	108
micro avg	0.53	0.6	0.56	764
macro avg	0.51	0.59	0.55	764
weighted avg	0.54	0.6	0.57	764
samples avg	0.48	0.48	0.42	764

Fig. 6. Random Forest Model's performance using FastText vectorizer on the test dataset.



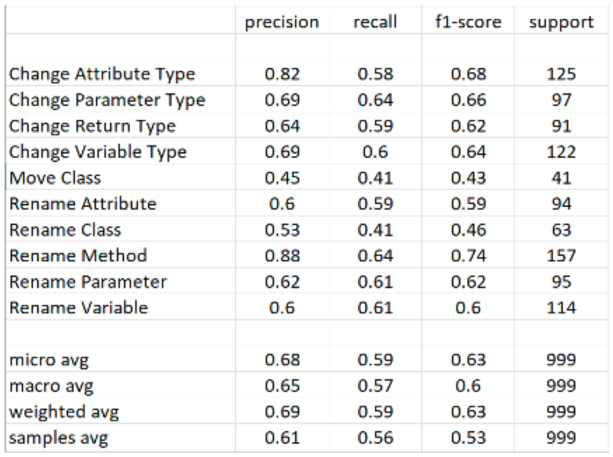
	precision	recall	f1-score	support
Change Attribute Type	0.99	0.9	0.94	244
Change Parameter Type	0.98	0.93	0.95	220
Change Return Type	0.95	0.93	0.94	182
Change Variable Type	0.98	0.95	0.96	245
Move Class	0.93	0.93	0.93	112
Rename Attribute	0.94	0.92	0.93	181
Rename Class	0.88	0.86	0.87	102
Rename Method	0.98	0.95	0.96	267
Rename Parameter	0.93	0.93	0.93	200
Rename Variable	0.94	0.95	0.95	240
micro avg	0.96	0.93	0.94	1993
macro avg	0.95	0.92	0.94	1993
weighted avg	0.96	0.93	0.94	1993
samples avg	0.95	0.94	0.94	1993

Fig. 7. Random Forest Model's performance using Count vectorizer on the train dataset.

Figures 9 and 10 depict the results of the logistic regression model's performance using the countvectorizer on the train dataset and the test dataset. It's evident from the weighted average scores of the precision value from figure 10 that there was no increase.

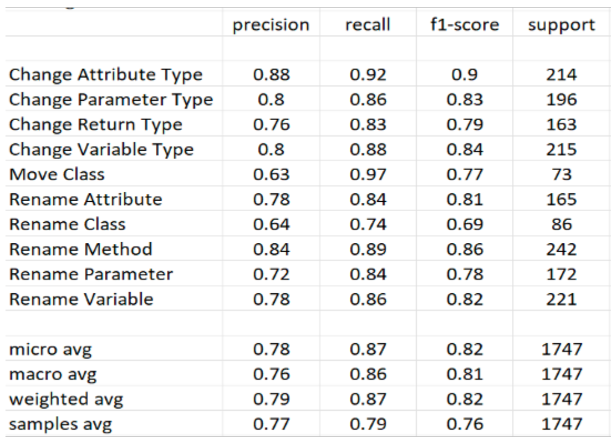
Figure 11 visualizes the weighted average of Precision, Recall and F1 scores of the Logistic Regression model and the Random Forest model. Precision as an appropriate metric for this project as this is a multiclass classification problem. It's evident that the precision and F1 scores are higher for the Random Forest , the recall value seems to be lower than the logistics regression model. From this observation, the Random Forest model was chosen as the final model based on the precision score.

Figure 12 depicts the precision, recall and f1 scores for the random forest model using count, tf-idf and fast text vectorizers. This was done before hyperparameter tuning.



	precision	recall	f1-score	support
Change Attribute Type	0.82	0.58	0.68	125
Change Parameter Type	0.69	0.64	0.66	97
Change Return Type	0.64	0.59	0.62	91
Change Variable Type	0.69	0.6	0.64	122
Move Class	0.45	0.41	0.43	41
Rename Attribute	0.6	0.59	0.59	94
Rename Class	0.53	0.41	0.46	63
Rename Method	0.88	0.64	0.74	157
Rename Parameter	0.62	0.61	0.62	95
Rename Variable	0.6	0.61	0.6	114
micro avg	0.68	0.59	0.63	999
macro avg	0.65	0.57	0.6	999
weighted avg	0.69	0.59	0.63	999
samples avg	0.61	0.56	0.53	999

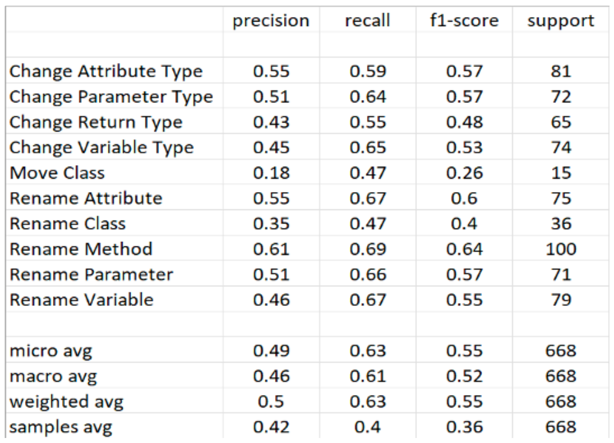
Fig. 8. Random Forest Model's performance using Count vectorizer on the test dataset.



	precision	recall	f1-score	support
Change Attribute Type	0.88	0.92	0.9	214
Change Parameter Type	0.8	0.86	0.83	196
Change Return Type	0.76	0.83	0.79	163
Change Variable Type	0.8	0.88	0.84	215
Move Class	0.63	0.97	0.77	73
Rename Attribute	0.78	0.84	0.81	165
Rename Class	0.64	0.74	0.69	86
Rename Method	0.84	0.89	0.86	242
Rename Parameter	0.72	0.84	0.78	172
Rename Variable	0.78	0.86	0.82	221
micro avg	0.78	0.87	0.82	1747
macro avg	0.76	0.86	0.81	1747
weighted avg	0.79	0.87	0.82	1747
samples avg	0.77	0.79	0.76	1747

Fig. 9. Logistic Regression Model's performance using Count vectorizer on the train dataset.

The Random Forest's weighted average precision scores are compared here. It is observed that the precision is high for the random forest model without hyperparameter tuning. Based on this, random forest was chosen without tuning. After learning to apply the above stated preprocessing methods like TF-IDF, CountVectorizer and fastText on the input V1 comments to predict/classify the 10 refactoring methods, we extended our exploration and wanted to see if we were able to make this classification system perform better by making the current problem easier. In-order to make the problem easy we reduced the number of refactoring methods/classes to 3 from 10. After conducting the same experiment (using above mentioned models and preprocessing steps) the precision, Recall, F1 score and accuracy were lower than the models that trained with the 10 refactoring classes. We decided to stick with the problem of classifying the 10 refactoring types based on the input SATD comment. The evolution of the project is clearly shown in the Jupyter notebook submited with this project.



	precision	recall	f1-score	support
Change Attribute Type	0.55	0.59	0.57	81
Change Parameter Type	0.51	0.64	0.57	72
Change Return Type	0.43	0.55	0.48	65
Change Variable Type	0.45	0.65	0.53	74
Move Class	0.18	0.47	0.26	15
Rename Attribute	0.55	0.67	0.6	75
Rename Class	0.35	0.47	0.4	36
Rename Method	0.61	0.69	0.64	100
Rename Parameter	0.51	0.66	0.57	71
Rename Variable	0.46	0.67	0.55	79
micro avg	0.49	0.63	0.55	668
macro avg	0.46	0.61	0.52	668
weighted avg	0.5	0.63	0.55	668
samples avg	0.42	0.4	0.36	668

Fig. 10. Logistic Regression Model's performance using Count vectorizer on the test dataset.

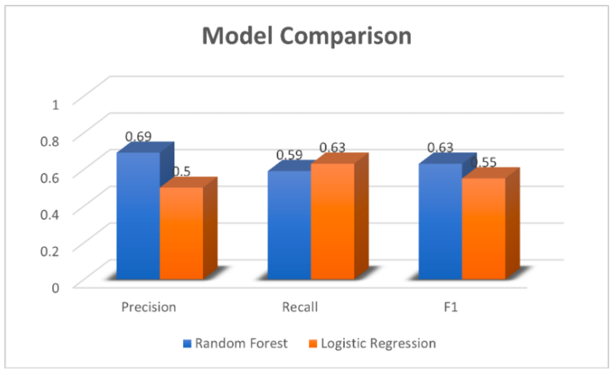


Fig. 11. Model Comparison

IV. Threats to Validity

A. Threats to Internal Validity

Since the goal of this project is to recommend the best type of refactoring method, it is seen as a classification problem. We have approached the problem with two classification models. First one being, Logistic regression and we thought that this would be an appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. As a second model, we went with the Random Forest Classifier. This model consists of multiple decision trees and operates as an ensemble. It uses bagging and features randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. The top advantage of choosing the above stated model is to interpret and understand the model. As an important step we have also compared their performances for evaluation purposes. Several NLP techniques are performed on the V1 comments in-order to extract the relevant features. The one-hot encoded

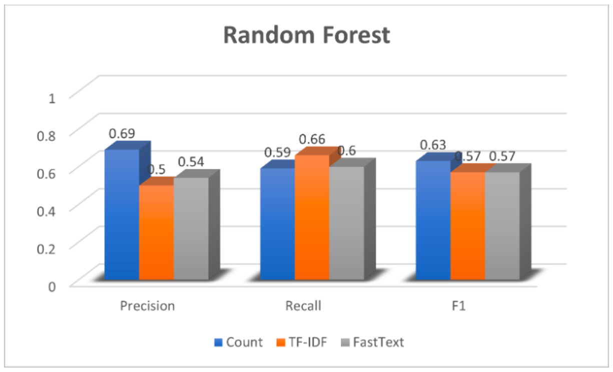


Fig. 12. Random Forest before and after tuning

version of the relevant features are supplied as inputs for both the models mentioned above. In-order to evaluate the models' performances we have selected the following metrics, Precision, Recall and F1 score. We believe that these metrics let us interpret and compare the performances of both models. However, we are going to focus on Precision as an important metric.

B. Threats to External Validity

The threat to external validity in our analysis is the uniqueness of our merged dataset. The collected GitHub commits and their SATD comments along with the associated refactoring types may not realistically represent the GitHub commits that are available on GitHub. To ensure the heterogeneity of the merged dataset, the dataset we have considered for this paper includes GitHub commits and their SATD comments along with the associated refactoring types. These projects are of different natures and characteristics and they were implemented by different software developer communities. Currently, we are targeting a subset of projects with 10 refactoring types. We need to extend our analysis on a larger and more diverse dataset that includes more refactoring types.

C. Threats to Construct Validity

Threats to construct relate to the relationship between the experiment settings and theory considered. One of the threats can be represented by grammatical mistakes in V1 comments of the dataset. This can impact the contextual information of the comment. We have attempted to mitigate this by using lemmatization and stop word removal during preprocessing. This will preserve context and reduce unnecessary verbiage, which would be beneficial for the model. Another threat is the choice of the evaluation metric. Since our aim is to recommend the most suitable refactoring types for the given SATD which should help the developers to resolve SATD quickly. Precision would be an ideal evaluation metric, because focusing on recommending correct refactoring over recommending more types with error will be more relevant, as it would not result in wasting developer's time in addressing the SATD.

V. Related Work

The prediction of refactoring types for the removal of an SATD comment is being investigated by several researchers. Zampetti et al. [5] have detailed the techniques that were used to automatically recommend SATD-removal strategies. Two pieces of information were combined: the SATD comment and the SATD-affected source code. The removals were recommended using CNN and RNN. Our paper is aimed at determining the appropriate types of refactoring that occur with the removal of a given SATD comment using Random Forest. Our work differs from the aforementioned work because we are only considering the SATD comment for the predictions we're using a machine learning approach instead of a deep learning approach.

VI. Conclusion

In this paper, we have attempted to recommend SATD removal strategies using a machine learning based approach. We have used two models: logistic regression and random forest and compared their performance for SATD removal. We have prioritized the precision metrics for the current problem and compared the two models based on that. We have found that random forest performed better than logistic regression with a precision score of 0.72 whereas it was 0.69 for the latter. Current work can further be improved. The dataset used is targeting a subset of projects with 10 refactoring types which can be further extended to larger and diverse datasets in terms of projects and refactoring types to improve the model performance . Also, source code can be considered along with the existing SATD comments. This would allow us to capture more information and produce better recommendations for removal.

References

[1] A. Potdar and E. Shihab, ”An Exploratory Study on Self-Admitted Technical Debt,” 2014 IEEE International Conference on Software Maintenance and Evolution, 2014, pp. 91-100, doi: 10.1109/ICSME.2014.31

[2] Fowler, M., 1999. *Refactoring: Improving the design of existing code.* Addison Wesley.

[3] S. McConnell, *Code Complete.* Redmond, Wash.: Microsoft Press, 1993.

[4] Iammarino, M., Zampetti, F., Aversano, L. and Di Penta, M., 2021. An empirical study on the co-occurrence between refactoring actions and Self-Admitted Technical Debt removal. Journal of Systems and Software, 178, p.110976.

[5] F. Zampetti, A. Serebrenik and M. Di Penta, ”Automatically Learning Patterns for Self-Admitted Technical Debt Removal,” 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 355-366, doi: 10.1109/SANER48275.2020.9054868.