Predicting Bug-Fixing Time with Machine Learning - A Collaborative Filtering Approach

Bruno Rafael de Oliveira Rodrigues
LAIS – Laboratory for Advanced Information Systems,
FUMEC University
Belo Horizonte, MG, Brazil
brunorodriguesti@yahoo.com.br

Fernando Silva Parreiras
LAIS – Laboratory for Advanced Information Systems,
FUMEC University
Belo Horizonte, MG, Brazil
fernando.parreiras@fumec.br

ABSTRACT

Predicting bug-fixing time helps software managers and teams prioritize tasks, allocations and costs in software projects. In literature, machine learning (ML) models have been proposed to predict bug-fixing time. One of features highlighted by studies is the reporter (the person who open the bug) has positive influence in the time to resolve a bug. In this way, this paper answers the following research question: *How does a collaborative filtering approach perform in predicting bug-fixing time compared to the supervised machine learning approaches?* In order to answer this question we performed an experiment using collaborative filtering approach to recommend the bugs that are fast to be resolved in two open software projects. We compare our proposed approach with the ML approach related to the literature. As a result, the collaborative filtering approach outperforms the supervised ML achieving an F-measure of 74% while the supervised ML achieved 66%. The collaborative filtering approach showed to be a new perspective to predict bug-fixing time in software projects focusing the prediction on the reporter.

KEYWORDS

Bug Fixing Time, Predict, Recommender System, Software Maintenance

1 INTRODUCTION

Estimating the time to fix a bug in software projects is an important part of the bug triage process. With the estimated time to fix a bug, the bug is then prioritized enabling the team to plan releases, and increase software quality and client satisfaction. Previous works have proposed models using Machine Learning (ML) techniques to predict bug-fixing time [1–4]. For instance, Zhang et al. [2] applied the KNN technique to three commercial projects of CA Technologies company. The work of Zhang et al. was replicated in the Mozilla Firefox project by Akbarinasaji et al. [3]. Habayeb et al. [4] propose a model based on Hidden Markov, which uses temporal sequences of the activities performed during the triage process.

The problem of predicting bug-fixing time can be addressed as a Recommender System (RS) problem, predicting the time to fix bugs and classifying whether these bugs can be fixed quickly or slowly, helping the software manager to prioritize the software activities [1]. Although the researches on predicting bug-fixing time presenting good results, we understand that by using a recommender approach, one can improve predictions.

The literature shows that the person who open the bug play the important role in the ML models. Therefore, we propose an approach based in RS technique, the collaborative filtering (CF), that we use the information about the reporter (person who open a bug) and the bug. This paper aims to answer the following research question: *How does a collaborative filtering approach perform in predicting bug-fixing time compared to the supervised machine learning approaches?* In answering this question, we proposed an approach can be applied to prioritizing bugs.

In order to validate our approach, we performed an experiment with data from the issue tracking system of two open source projects: Eclipse Platform and Netbeans. We compared the CF approach with the supervised ML approaches recurrent in literature. Our experiments show that CF approaches outperformed the supervised ML classifications. The SVD, KNNBaseline and SVD++ algorithms, which used CF, achieved an f-measure 74% of in the Eclipse and 69% in NetBeans projects. While for traditional ML, the best result was obtained by the Random Forest algorithm with an f-measure of 66% in the Eclipse project and 63% in the Netbeans project. This paper demonstrates that the CF approach is a viable approach for predicting bug-fixing time, enabling the determination of whether the bug will be fixed quickly or slowly. In the approach presented, the algorithms use fewer features and less computational power than traditional ML approaches.

This paper is organized as follows: Section 2 presents the background about recommender systems; Section 3 presents the related work; Section 4 describes the proposed model to predict bug-fixing time using the collaborative filtering approach; Section 5 outlines the research methodology used to answer our research questions; Section 6 presents the results obtained in this study and discusses the results obtained; Section 7 details the threats to the validity of the research; and Section 8 presents the conclusions.

2 RECOMMENDER SYSTEMS

RS produce individual recommendations for users, filtering large amounts of information into personalized recommendations for users. These recommendations can be classified in explicit or implicit feedback [5]. In explicit feedback, the users express their opinion about an item through ratings or a scale, therefore the feedback can either be positive or negative. In implicit feedback, the system itself takes the behavior of the user as input and the feedback is always positive [5]. Next, we present the concepts behind the CF.

2.1 Collaborative Filtering

The main idea of CF recommendation systems is that people have similar tastes. For this, algorithms and methods are used to find user preferences based on numerical ratings. Ratings in collaborative filtering can be scalar, binary or unary. Scalar ratings are numerical

ratings such as 1-5 or ordinal ratings such as strongly agree, agree, neutral, disagree, strongly disagree. [6].

The items and user ratings are used to generate a user-item matrix. With this information, it is possible to predict and recommend items to users. The recommendation can be based on the similarity between users (user-user) or items (item-item) [7]. For this purpose, machine learning techniques have been applied to increase recommendation quality.

3 RELATED WORK

Predicting bug-fix times with ML is a recurring software engineering problem in literature. Zhang et al. [2] applied the Markov-based method to predict the number of bugs, the Monte Carlo-based method to predict the total time required to fix a given number of bugs, and the KNN technique to determine the effort required for a new bug through the similarity of bugs, on three commercial projects of CA Technologies. The kNN-based method achieved an average weighted F-measure of 72.45% in predicting the time to fix each individual bug.

The Zhang et al. study was replicated by Akbarinasaji et al. [3], which used Mozilla Firefox as the dataset. The result of the time required to fix a particular bug using the KNN approach was an F-measure from 56% to 74%, with an average of 62.7%.

Habayeb et al. [4] propose an approach to predicting bug-fixing time based on the Hidden Markov Model (HMM), which allows work with stochastic processes. They used the Mozilla Firefox project as dataset. The data of bug report activities were transformed into temporal activities. The temporal sequences of activities related to resolved bugs were used to train the HMM, where the training was separated by the classification of bug-resolution time into slow or fast. To classify the bugs, they calculated the median number of days to fix the bugs whereby if the time to fix a bug was less than the median, the bug was classified as fast, otherwise it was classified as slow. The model was trained with 60% of bug reports and 40% was used in testing. After they trained the model by year, from 2006 to 2014, the test was performed with the current year.

In contrast to the works cited above, the present paper proposes predicting bug-fixing time in software projects using a ML from a perspective of recommender system. Although, unlike previous studies, we not only propose an approach based on the characteristics of a bug or on the bug activity sequence, but also one in which the bug reporter is at the center of the approach. In the literature, the reporter is a feature that has increasing the performances in reported models, but the focus of research until now has been only on bug report similarity and the reporter is used as feature of the models. We understand that the similarity of reporters might make the models more accurate. Thus, differently to other works, this study proposes the use of a recommendation approach using CF to predict the time to fix a bug.

4 PROPOSED APPROACH

Unlike supervised ML present in literature to predict bug-fixing time which seek by similarity between the bug reports, our CF approach focus on the similarity between the reporters. In other words, our approach recommends whether the bug may be fixed fast or slow, although the similarity between the reporters based on the history of bug reports by reporters. Our proposed approach uses only the initial attributes of the bug when they are created, these attributes are: severity, priority, identification of product, component, operational system and platform. We concatenate this information, forming a name of a bug. For example: "normalP216409Windows XPPC", next, we encode the "names of bug" to numerical value in order to improve the performance of the approach. Thus, we use the similarity between the bug reports regarding a reporter to predict the time to fix a bug. This concept is like predict a rating of a movie in a recommender system.

Predicting bug-fixing time is a problem addressed in literature from two angles: classification or regression [8]. In this study, we address this problem as a classification problem. We therefore use the approaches proposed by Giger [1], also followed by Zhang [2] and Habayeb [4], in which the bugs are classified as *fast* or *slow* according to the median resolution time. If the time to resolve the bug was less than median it is considered *fast*, so one may prioritize this bug. Otherwise, it is considered *slow* may be postponed.

To apply our approach, we used the data available on the bug tracking system. After extracting the data, we performed the following steps:

(1) Calculate the time in days (last modification - data creation)
(2) Remove outliers. (75% quartile and bugs resolved in the same day)
(3) Classify the bugs as fast or slow according to the median
(4) Discretize the data
(5) Calculate the similarity between the reporters and bug (collaborative filtering)
(6) Classify the bug as fast or slow with the proposed approach

The proposed approach can be used to prioritize the bugs recommending which bugs may be fixed *fast*.

5 EXPERIMENT

In order to carry out the experiment, we performed the following steps:

- Select the dataset
- Select the attributes
- Prepare the environment
- Select the ML algorithms
- Remove the outliers
- Classify the data
- Balance the dataset
- Clean the data
- Execute the ML approach
- Execute the Collaborative Filtering approach

These steps are described in further detail below.

5.1 Select the dataset and their attributes

We first obtained the Eclipse and Netbeans datasets, available on MSR 2011 [9]. Eclipse and Netbeans are two popular open-source projects of integrated development environments (IDE). These datasets contain bug reports stored in MySQL format. Both the Eclipse and Netbeans projects use Bugzilla as their bug issue tracking system, thus, the structure of these datasets are similar. We then

explored the database, selecting attributes from the bug reports for our experiments and exporting them in CSV format. We selected those attributes from Bugzilla which were initially filled when the person opened a bug. We intend to predict the time to fix a bug at the beginning of the triage. The attributes used in our experiment are listed in table 1.

Table 1: Attributes selected from dataset

Attribute	Description
bug_id	the bug identification
assigned_to	The developer that worked on the bug
bug_severity	the bug severity
priority	the priority
bug_status	status
creation_ts	date when the bug was reported
short_desc	the title of the bug
comments	the bug description
op_sys	the operational system where the bugs occurred
rep_platform	the platform
reporter	the person who reported the bug
number of CC	Users who are interested in the progress of this bug
number of attachments	number of attachments added by users to a bug report
resolution	the status of resolution
lastdiffed	the date of the last modification of the bug report
estimated_time	the estimated time to resolve the bug
product_id	the products
component_id	the component
days	the lastiffed - creation_ts in days
year_creation	the year when the bug was reported
year_lastdiffed	the year when the bug was resolved
mouth_creation	the month when the bug was reported
mouth_lastdiffed	the month when the bug was resolved

5.2 Environment and Machine Learning Algorithms Selected

We used Google Colab to carry out the experiment. Google Colab is an environment prepared to execute data analysis directly in a browser [1]. We selected the machine learning algorithms present in the predicting bug-fixing time literature. The ML algorithms selected can be seen in the table 2. The data exploration was performed with Python (version 3.6.9) and the libraries NumPy (version 1.18.5), Pandas (version 1.0.5) and Matplotlib (version 3.2.2). These Python libraries are used to understand the data, remove the outliers and discretize the data. The code and dataset used in this experiment are available in: https://github.com/brunorodriguesti/Experiment_CF_ML_Bug_Fixing_Time.

[1]https://colab.research.google.com/

Table 2: ML algorithms by Studies

Algorithm	Studies
Decision Tree	[10] [11] [12] [13] [14] [1] [15] [16] [17] [2] [18] [19] [20] [21] [22] [23]
Naive Bayes	[10] [24] [11] [12] [14] [15] [16] [25] [26] [27] [20] [28] [21] [29] [22]
KNN	[30] [14] [31] [2] [18] [20] [32] [4] [3] [22] [33]
SVM	[13] [14] [16] [34] [35] [36] [29] [22] [23] [33]
Random Forest	[17] [37] [19] [38] [20] [39] [29] [22] [23]

5.3 Outliers

In our experiment, we only considered the bugs that had already been resolved, thus we used bugs with a status resolution of fixed. The bug-fix time was calculated in days. For example, if the bugs were resolved in the same day, they have zero days of resolution.

In treating the outliers, we removed the 75% quartile and the bugs whose resolution were equal to zero days. The quality of the data and the accuracy of the models was improved by removing the outliers [25]. On analysing our dataset, we noticed that the bug reports with zero day resolutions were not always bugs. These reports are tests of environment or bugs resolved prior to reporting, in other words, they were reported only for registration purposes. We remove those bugs over 75% quartile from the dataset as they were outliers that do not represent the reality of all types of bugs.

5.4 Classification and balancing of dataset

In order to classify the bugs as fast or slow, the approach proposed by [1] was used. The data was divided as fast and slow according to the formula 1. The threshold was calculate by median of days, the Eclipse Platform dataset was 17 days and 6 days that of Netbeans. Classification of the Eclipse project bugs resulted in 12353 fast and 12353 slow, whilst that of the NetBeans project produced both 1534 fast and slow bugs.

$$BugClass = \begin{cases} Fast & days\ resolution <= median \\ Slow & days\ resolution > median \end{cases} \quad (1)$$

5.5 Experiment Execution

After removing the outliers, the attributes from table 1 were used with the supervised ML algorithms listed in table 2. We use the following features in the supervised machine learning approach: *the reporter, the first developer, the priority, the severity, the product, the component, the month of creation, the year of creation, the number of cc and the number of attachment, the description of summary and the first comment.* The priority and severity attributes were transformed into categorical features. We joined the content of the description of the summary and description of the first comment of bug reports in one field. For this field, we extracted the bag of words and treated the bug report texts following the steps:

(1) convert all words to lower case
(2) remove links and html tags

(3) remove numbers and special characters
(4) remove stop words
(5) stem words

We build a document-terms matrix with the frequency of terms, removing one percent of the terms infrequently. Next, we use the term frequency (TF) and term frequency-inverse document frequency (TFIDF) in order to create features for classification algorithms.

In our experiment, the behaviour of algorithms without the textual features was first tested and then both the features and textual elements (summary and comment) with the goal of classifying the bug report as fast or slow.

To run the proposed collaborative approach, we used the Surprise library [2] which is a Python library built to facilitate the build of recommender systems. Surprise only works with explicit rating data therefore the data needed to be adapted to work with explicit ratings. A scale of 1 to 5 was used to classify the bug resolution as fast or slow according to table 3. Bugs with a classification from 1 to 3 on the scale were determined as fast. We tested the performance of the algorithms available in the Suprise library (SVD, KNNBaseline, SVD++, BaselineOnly, KNNWithMeans, KNNWithZScore, SlopeOne, CoClustering, NMF, KNNBasic, NormalPredictor). In this approach, we use only the following features: *severity, priority, product identification, component identification, operational system and platform.*

Table 3: Bugs Classified with Surprise Library

Rating	Time in day
1	1 day to fix
2	greater than 1 and less or equal than 25% of percentile
3	greater than 25% and less or equal to 50% of percentile
4	greater than 50% and less or equal 75% of percentile
5	greater than 75%

5.6 Evaluation Metrics

To evaluate the performance of the models, we used the precision, recall and f-measure. The tests were performed with 10-fold cross-validation.

To assess classification approaches, we can use table 4. The metrics used to evaluate the techniques are the same as those used in an Information Retrieval context, such as Precision, Recall, F-measure, Accuracy [40], Receiver Operator Characteristic (ROC), and Area under the Curve (AUC). In this paper we use the Precision, Recall and F-measure in order to evaluation the models, the definition of these metrics are present in this section.

Precision, also called Confidence, is the ratio of predicted positive cases that are correctly true-positives, while the Recall or Sensitivity is the ratio of true-positive cases that are correctly predicted positive [42]. The combination of precision and recall can be formulated by the F-measure, also called F-Score. The F-measure is the harmonic mean between the precision and the recall [6]. The formulas for Precision, Recall and F-measure are presented in 2, 3, and 4, respectively.

[2]http://surpriselib.com/

-	Recommended	Not Recommended
Preferred	True-Positive (tp)	False-Negative (fn)
Not Preferred	False-Positive (fp)	True-Negative (tn)

Table 4: Classification of the possible results of an item recommendation to a user [41]

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F - measure = 2\frac{Precision.Recall}{Precision + Recall} \tag{4}$$

5.7 Dataset

In order to evaluate our model,an experiment was performed using bug tracking data from the following systems: Eclipse and Netbeans available from MSR 2011 [9]. The data is available in MySQL Database format. The table 5 shows data from Eclipse and Netbeans bug reports.

Table 5: Dataset Characteristics

Dataset	Bugs	Period
Eclipse	87,696	2001-10-10 at 2010-06-26
Netbeans	12,091	2009-11-07 at 2010-06-29

In this experiment, only bug reports with the a resolution status of "FIXED" was used. There were 37,261 bugs in Eclipse and 4747 bugs in Netbeans. After removing the outliers, there were 24,863 bugs in Eclipse and 4747 in Netbeans. In categorising the bugs as either fast or slow, there were 12,510 fast and 12,353 slow bugs in Eclipse, and 1699 fast and 1534 slow in Netbeans. We classify the dataset as fast and slow with the same number of bugs. The Eclipse dataset has a total of 24,706, therefore 12,353 were classified as fast and the same number as slow. In Netbeans, the experiment was run with 3068 bugs, of which 1534 were classified as fast and the same number as slow.

In order to execute the collaborative filtering using the Surprise library, the bugs were classified on a scale of 1 to 5, where the bug was considered fast when it was equal or less than 3 in the scale, as explained in section 5. The table 6 shows the distribution of bugs along this scale.

Classification	Eclipse	Netbeans
1	2,360	413
2	3,969	577
3	6,024	544
4	6,295	836
5	6,058	698

Table 6: Distribution of amount of bugs in scale of 1 to 5

Finally, the Eclipse project has 1,881 bugs classified according to the scale and 3,090 reporters. In Netbeans, there are 1207 bugs classified according to the scale and 624 reporters.

6 RESULTS AND DISCUSSION

In this section the results of the experiment are presented. First, the results of predicting bug-fixing time using supervised ML to classify the bug reports as fast or slow are presented. Second, the results using the collaborative filtering approach are detailed. In the supervised ML approach, the experiment was binned with the attributes without the of bug report summaries and comments and next all attributes described in table 1 were used. These results can be seen in the tables 7, 8, 9 and 10.

Table 7: Results of Supervised ML Approach without Textual Features - Eclipse Plataform

Algorithm	Precision	Recall	F-measure
Random Forest	67%	66%	66%
Decision Tree	63%	60%	61%
KNN	61%	59%	60%
SVM	40%	46%	46%
Naive Bayes	57%	26%	36%

Table 8: Results of Supervised ML Approach with all Features - Eclipse Plataform

Algorithm	Precision	Recall	F-measure
Random Forest	63%	65%	64%
KNN	61%	59%	60%
Decision Tree	58%	58%	58%
SVM	50%	44%	46%
Naive Bayes	58%	26%	36%

Table 9: Results of Supervised ML Approach without Textual Features - NetBeans

Algorithm	Precision	Recall	F-measure
Random Forest	64%	63%	63%
Decision Tree	59%	59%	59%
KNN	58%	57%	58%
SVM	44%	55%	46%
Naive Bayes	71%	21%	33%

Table 10: Results of Supervised ML Approach with Textual Features - NetBeans

Algorithm	Precision	Recall	F-measure
Random Forest	61%	65%	63%
SVM	63%	57%	58%
Decision Tree	55%	57%	56%
Naive Bayes	57%	52%	55%
KNN	56%	54%	55%

The Random Forest algorithm produced the best results amongst all tests presented in this work. In literature, Random Forest has performed well in predicting bug-fixing time [17, 23, 38, 39]. The Random Forest is an ensemble learning method based on multiple decision trees. The ensemble methods apply multiple learning algorithms in order to classify the data [43].

It is possible to see in tables 7, 8, 9 10 that the results were best when summaries and comments were not used. Despite the studies showing that textual features increase the accuracy of predictions, in the present work, textual features did not increase the performance when used in conjunction with other features. The exception was the SVM algorithm, which used all features (textual and not textual) in NetBeans project.

The tables 11 and 12 show the results using the algorithms available in the Surprise library. For this approach, only the attributes cited in 5 were used, thus ignoring textual attributes like summary and comment in this approach.

Table 11: Results of Collaborative Filtering Approach Eclipse Plataform

Algorithm	Precision	Recall	F-measure
SVD	75%	73%	74%
SVD++	77%	72%	74%
KNNBaseline	72%	74%	73%
BaselineOnly	72%	75%	73%
KNNWithMeans	63%	79%	70%
KNNWithZScore	63%	79%	70%
SlopeOne	63%	80%	70%
CoClustering	63%	80%	70%
KNNBasic	59%	82%	69%
NMF	59%	82%	69%
NormalPredictor	68%	70%	69%

Table 12: Results of Collaborative Filtering Approach NetBeans

Algorithm	Precision	Recall	F-measure
SVD	59%	83%	69%
KNNBaseline	60%	83%	69%
SVD++	60%	81%	69%
KNNWithMeans	60%	81%	69%
KNNWithZScore	61%	81%	69%
SlopeOne	61%	80%	69%
BaselineOnly	56%	88%	68%
NMF	58%	83%	68%
CoClustering	59%	81%	68%
KNNBasic	56%	85%	67%
NormalPredictor	69%	66%	67%

From the results presented in this work, one can see that the collaborative filtering approaches outperformed the supervised ML

approaches in predicting the bug-fixing time in this experiment. Unlike supervised ML approaches, collaborative filtering focuses on the similarity of users. In our study, we considered the reporter to be the user in our approach. Although performing collaborative filtering has shown better compared to the supervised machine learning in our experiment. Our work shows that has increased performance compared to supervised machine learning. In other words, we can see it is possible to use the collaborative filtering approach to predict bug-fixing time focus on the reporter presenting good performances without information about the developer. We understand that collaborative filtering is a promising approach when focusing on the recommendation of the reporter. In our proposed model, it is possible prioritize the bugs that will be fixed fast before they be assigned to a developer.

The Singular Value Decomposition(SVD), KNNBaseline and SVD++ algorithms presented the best results in all tests using the collaborative filtering approach in this work. The SVD is a technique that identifies latent semantic factors in information retrieval, automatically deriving semantic "concepts" from a low dimensional space. The SVD++ is an improved version of SVD that considers implicit feedback [7]. The SVD++ produced good results in this experiment, but the time required to train the model is more expensive than conventional SVD. The KNNBaseline algorithm, unlike conventional KNN, computes the baseline, in other words it balances the scores when there are very high or low ratings [44]. Like Habayeb et al. [4] we do not use textual features such as the summary and comments of bug reports. Textual features are computationally expensive [4] and the supervised ML approaches used in this study did not demonstrate that textual features increase the performance of predicting bug-fixing time, so in our proposed approach summaries or comments are not used as features. A further advantage of the collaborative filtering approach is that few features are required to make the predictions, in other words, only the initial attributes available in the bug tracking system before triage and described in 5 are needed. As with any collaborative filtering approach, the proposed approach of this study may suffer from issues such as cold start, scalability and sparsity [45].

This study clearly shows that it is viable to recommend whether the bug will be fixed quickly or slowly based on the reporter. Studies have shown that the reporter has a positive influence on predicting bug-fixing time [1, 2, 4, 32, 46–48]. The collaborative filtering approach can therefore recommend the bug-fixing time based on the reporter, similar to what a recommender system does for the user of a system. Reporters have different levels of experience assigning bugs. The more experienced a reporter, the better their assignment to the developer fixing the bug, which could take the time to fix the bug more predictable.

Besides the reporter, the designated developer also influences the time to fix the bug [49]. However, in our experiment, in the collaborative filtering approach, when we used the developer as input of models instead of the reporter, performing the model decreases. For instance, predicting using the developer in SVD, SVD++ and Baseline Only the average F-measure is 45% in the Eclipse project. In the NetBeans project, using the developer as input, the f-measure is 63% to SVD, SVD++ and Baseline Only. The collaborative filtering approach enables the prediction of the bug-fixing time before being assigned to a developer. As shown in the results, we understand it is possible to build a prediction bug-fixing time model using collaborative filtering focus on the reporter rather than on the developer. One advantage of using the collaborative filtering approach instead of supervised machine learning is that we can use fewer features and use initial fields available in a bug report before it is assigned. Another advantage of our proposed approach is regarding the amount of data used in training and testing the approaches. Our approach has presented a good performance on both datasets. We used 113904 bugs reported in the Eclipse and only 3068 in the NetBeans dataset in order to train and test the approaches. We can note that performing supervised machine learning decreased when the dataset contains fewer data when using the collaborative filtering approach, the performance remained stable.

7 THREATS TO VALIDITY

Internal validity: Despite the increasing use of deep-learning approaches in predicting bug-fixing time, which has been promising, our experiment focused only on supervised ML approaches. The goal of our experiment was to validate the recommender system approach in predicting bug-fixing time and its ability to recommend those bugs that can be fixed quickly. In other words, the model presented in this work can be used with deep learning algorithms and machine learning, as well as other recommender systems. The hyperparameters of algorithms were not tuned, instead the default hyperparameters of libraries were employed. Tests were performed to tune the hyperparameters, however the performance did not increase as expected.

External validity: Only two datasets were used. These datasets are generally used in the prediction of bug-fixing time research. We can however generalize our results. Eclipse and Netbeans are two popular open-source projects which have their own peculiarities arising from their community and processes. Collaborative filtering approaches may be improved with user information such as genre, age, geographic position and others. In our dataset, personal information was not included about the reporter, which could have improved our model. Reporter information like company or client position, time as system user, and other personal information could be used in the model to verify the performance of collaborative filtering in predicting the bug-fixing time.

Construct validity: in our experiment, we calculated the time to fix a bug in days. This assumption may not have captured the real time to fix a bug. Also, we remove the bug reports that have taken less than one day to be fixed and the outliers are a great 75% quartile. We based the equation to calculate the bug-fixing time and to the decision regarding removing the outliers based on the current literature.

Conclusion validity: despite the results of our proposed approach overcome slightly the supervised machine learning approach, we can not claim that our proposal is better. The results achieved by random forest are close to the proposed approach. We can consider that the proposed approach has the potential to be improved through a new perspective (collaborative filtering).

8 CONCLUSION

Literature has shown that a reporter is a relevant feature of ML models when predicting bug-fixing time. CF provides recommendations focusing on the user. Predicting bug-fixing time can be seen as a recommender system, which recommends those bugs that can be resolved quickly. The aim of this paper was to answer the following research question: *How does a collaborative filtering approach perform in predicting bug-fixing time compared to the supervised machine learning approaches?* In order to answer this question, an experiment was carried out in which the recurrent ML approach reported in literature was selected and used to classify reported bugs as fast or slow, according to the characteristics of the bug report. CF algorithms were then selected to recommend whether the bug would be fixed quickly or slowly in line with their characteristics and the similarity between the person who opened the bug, similar to how a movie recommender system works.

The results of this study show that the collaborative filtering approach is viable in predicting bug-fixing time. The algorithms used in this experiment outperformed the supervised ML approach. The SVD, KNNBaseline and SVD++ algorithms produced an F-measure of 74%, whilst the most successful ML algorithm, Random Forest, achieved an F-measure of 66% F-measure. In collaborative filtering, only six features are required to achieve a greater performance, compared to ten features in supervised ML. In applying collaborative filtering, one can therefore predict bug-fixing time with less data initially obtained from bug reports during triage.

In our experiment, we use the default configurations of Surprise Library. In this way, the similarity metric is the Mean Squared Difference (MSD). For future works, it is interesting to verify the performance of the techniques using other metrics like cosine similarity and Pearson.

The approach presented in this study does not necessarily require the use of the algorithms presented in this paper. Despite we use a specific library to run the collaborative filtering, we understand that other techniques can be apply. Other algorithms can be applied such as several deep learning algorithms which have shown great promise in predicting bug-fixing time. In the future, we believe it would be beneficial to perform a study using deep learning algorithms in a collaborative filtering approach to predict bug-fixing time. It would also be interesting to apply the proposed approach to a proprietary project that contained information about the person who opened the bug, such as position in company, time on the project, time in the company and other characteristics that would improve the collaborative filtering approach. A hybrid model using collaborative and content based filtering could also be tested to try and improve the model. The type of bug affects bug-fixing time as well, eg security bugs tend to be given higher priority and the time to fix security bugs is shorter, but also less variable and thus more predictable (ie you have to fix them). The type of bug should be taken into account in future work.

REFERENCES

[1] Emanuel Giger, Martin Pinzger, and Harald Gall. Predicting the fix time of bugs. In *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*, RSSE '10, pages 52–56, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-974-9. doi: 10.1145/1808920.1808933. URL http://doi.acm.org/10.1145/1808920.1808933.

[2] Hongyu Zhang, Liang Gong, and Steve Versteeg. Predicting bug-fixing time: An empirical study of commercial software projects. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1042–1051, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-3076-3. URL http://dl.acm.org/citation.cfm?id=2486788.2486931.

[3] Shirin Akbarinasaji, Bora Caglayan, and Ayse Bener. Predicting bug-fixing time: A replication study using an open source software project. *Journal of Systems and Software*, 136:173 – 186, 2018. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2017.02.021. URL http://www.sciencedirect.com/science/article/pii/S0164121217300365.

[4] M. Habayeb, S. S. Murtaza, A. Miranskyy, and A. B. Bener. On the use of hidden markov model to predict the time to fix bugs. *IEEE Transactions on Software Engineering*, pages 1–1, 2017. ISSN 0098-5589. doi: 10.1109/TSE.2017.2757480.

[5] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, July 2013. doi: 10.1016/j.knosys.2013.03.012. URL https://doi.org/10.1016/j.knosys.2013.03.012.

[6] Charu C. Aggarwal. *Recommender Systems*. Springer International Publishing, 2016. doi: 10.1007/978-3-319-29659-3. URL https://doi.org/10.1007/978-3-319-29659-3.

[7] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender Systems: Introduction and Challenges. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender Systems Handbook*, pages 1–34. Springer US, Boston, MA, 2015. ISBN 978-1-4899-7637-6. doi: 10.1007/978-1-4899-7637-6_1. URL https://doi.org/10.1007/978-1-4899-7637-6_1.

[8] Y. Lee, S. Lee, C. Lee, I. Yeom, and H. Woo. Continual prediction of bug-fix time using deep learning-based activity stream embedding. *IEEE Access*, 8:10503–10515, 2020.

[9] Adrian Schröter. Msr challenge 2011: Eclipse, netbeans, firefox, and chrome. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 227–229, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0574-7. doi: 10.1145/1985441.1985478. URL http://doi.acm.org/10.1145/1985441.1985478.

[10] Rattikorn Hewett and Aniruddha Kulkarni. Alternative approach to utilize software defect reports. In *15th International Conference on Software Engineering and Data Engineering (SEDE-2006), July 6-8, 2006, Omni Los Angeles Hotel at California Plaza, Los Angeles, California, USA, Proceedings*, pages 57–62, 01 2006.

[11] Rattikorn Hewett, Aniruddha Kulkarni, Catherine Stringfellow, and Anneliese Amschler Andrews. Software defect data and predictability for testing schedules. In *Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2006), San Francisco, CA, USA, July 5-7, 2006*, pages 499–504, 01 2006.

[12] Lucas D. Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 29–, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 978-0-7695-2950-9. doi: 10.1109/MSR.2007.25. URL http://dx.doi.org/10.1109/MSR.2007.25.

[13] Syed Nadeem Ahsan, Javed Ferzund, and Franz Wotawa. Program file bug fix effort estimation using machine learning methods for oss. In *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*, pages 129–134, Boston, Massachusetts, USA, 2009. Knowledge Systems Institute Graduate School.

[14] Rattikorn Hewett and Phongphun Kijsanayothin. On modeling software defect repair time. *Empirical Software Engineering*, 14(2):165, April 2009. ISSN 1382-3256, 1573-7616. doi: 10.1007/s10664-008-9064-x. URL https://link.springer.com/article/10.1007/s10664-008-9064-x.

[15] G. Bougie, C. Treude, D.M. German, and M. Storey. A comparative exploration of freebsd bug lifetimes. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 106–109, 2010. doi: 10.1109/MSR.2010.5463291.

[16] Zhimin He, Fengdi Shu, Ye Yang, Wen Zhang, and Qing Wang. Data unpredictability in software defect-fixing effort prediction. In *2010 10th International Conference on Quality Software*, pages 220–226. IEEE, July 2010. ISBN 978-1-4244-8078-4. doi: 10.1109/QSIC.2010.40. URL http://ieeexplore.ieee.org/document/5562962/.

[17] Nguyen Duc Anh, Daniela S. Cruzes, Reidar Conradi, and Claudia Ayala. Empirical validation of human factors in predicting issue lead time in open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pages 13:1–13:10, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0709-3. doi: 10.1145/2020390.2020403. URL http://doi.acm.org/10.1145/2020390.2020403.

[18] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. An approach to the discovery of accurate and expressive fix-time prediction models. In *Enterprise Information Systems*, Lecture Notes in Business Information Processing, pages 108–128. Springer, Cham, April 2014. ISBN 978-3-319-22347-6 978-3-319-22348-3. doi: 10.1007/978-3-319-22348-3_7. URL https://link.springer.com/chapter/10.1007/978-3-319-22348-3_7.

[19] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. A framework for the discovery of predictive fix-time models. In *Proceedings of the 16th International Conference on Enterprise Information Systems - Volume 1*, ICEIS 2014, pages 99–108, Portugal, 2014. SCITEPRESS - Science and Technology Publications, Lda. ISBN 978-989-758-027-7. doi: 10.5220/0004897400990108. URL http://dx.doi.org/10.5220/0004897400990108.

[20] Dietmar Pfahl, Siim Karus, and Myroslava Stavnycha. Improving expert prediction of issue resolution time. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, EASE '16, pages 42:1–42:6, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3691-8. doi: 10.1145/2915970.2916004. URL http://doi.acm.org/10.1145/2915970.2916004.

[21] Maggie Hamill and Katerina Goseva-Popstojanova. Analyzing and predicting effort associated with finding and fixing software faults. *Information and Software Technology*, 87:1 – 18, 2017. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2017.01.002. URL http://www.sciencedirect.com/science/article/pii/S0950584917300290.

[22] Ali Dehghan, Kelly Blincoe, and Daniela Damian. A hybrid model for task completion effort estimation. In *Proceedings of the 2Nd International Workshop on Software Analytics*, SWAN 2016, pages 22–28, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4395-4. doi: 10.1145/2989238.2989242. URL http://doi.acm.org/10.1145/2989238.2989242.

[23] Rucha Sawarkar, Naresh Kumar Nagwani, and Sanjay Kumar. Predicting Bug Estimation Time for Newly Reported Bug Using Machine Learning Algorithms. In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, pages 1–4, March 2019. doi: 10.1109/I2CT45611.2019.9033749.

[24] Qinbao Song, M. Shepperd, M. Cartwright, and C. Mair. Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, 32(2):69–82, February 2006. ISSN 0098-5589. doi: 10.1109/TSE.2006.1599417. URL http://ieeexplore.ieee.org/document/1599417/.

[25] A. Lamkanfi and S. Demeyer. Filtering bug reports for fix-time analysis. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 379–384, March 2012. doi: 10.1109/CSMR.2012.47.

[26] W. Abdelmoez, M. Kholief, and F. M. Elsalmy. Bug fix-time prediction model using na #x00ef;ve bayes classifier. In *2012 22nd International Conference on Computer Theory and Applications (ICCTA)*, pages 167–172, October 2012. doi: 10.1109/ICCTA.2012.6523564.

[27] W. AbdelMoez, Mohamed Kholief, and Fayrouz M. Elsalmy. Improving bug fix-time prediction model by filtering out outliers. In *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, pages 359–364. IEEE, May 2013. ISBN 978-1-4673-5613-8 978-1-4673-5612-1 978-1-4673-5611-4. doi: 10.1109/TAEECE.2013.6557301. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6557301.

[28] Pranjal Ambardekar, Anagha Jamthe, and Mandar Chincholkar. Predicting defect resolution time using cosine similarity. In *2017 International Conference on Data and Software Engineering (ICoDSE)*, pages 1–6. IEEE, November 2017. ISBN 978-1-5386-1449-5. doi: 10.1109/ICODSE.2017.8285884. URL http://ieeexplore.ieee.org/document/8285884/.

[29] Chuanqi Wang, Yanhui Li, and Baowen Xu. How many versions does a bug live in? an empirical study on text features for bug lifecycle prediction. In *The 30th International Conference on Software Engineering and Knowledge Engineering, Hotel Pullman, Redwood City, California, USA, July 1-3, 2018*, pages 415–461. KSI Research Inc. and Knowledge Systems Institute Graduate School, 07 2018. doi: 10.18293/SEKE2018-176.

[30] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. How long will it take to fix this bug? In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 1–, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 978-0-7695-2950-9. doi: 10.1109/MSR.2007.13. URL http://dx.doi.org/10.1109/MSR.2007.13.

[31] Alaa Hassouna and Ladan Tahvildari. An effort prediction framework for software defect correction. *Information and Software Technology*, 52(2):197 – 209, 2010. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2009.10.003. URL http://www.sciencedirect.com/science/article/pii/S0950584909001748.

[32] Pranav Ramarao, K. Muthukumaran, Siddharth Dash, and N. L. Bhanu Murthy. Impact of bug reporter's reputation on bug-fix times. In *2016 International Conference on Information Systems Engineering (ICISE)*, pages 57–61. IEEE, April 2016. ISBN 978-1-5090-2287-8 978-1-5090-2288-5. doi: 10.1109/ICISE.2016.18. URL http://ieeexplore.ieee.org/document/7486274/.

[33] Meera Sharma, Madhu Kumari, and V. B. Singh. Multi-attribute dependent bug severity and fix time prediction modeling. *International Journal of System Assurance Engineering and Management*, 10(5):1328–1352, October 2019. ISSN 0976-4348. doi: 10.1007/s13198-019-00888-5. URL https://doi.org/10.1007/s13198-019-00888-5.

[34] Ferdian Thung. Automatic prediction of bug fixing effort measured by code churn size. In *Proceedings of the 5th International Workshop on Software Mining*, SoftwareMining 2016, pages 18–23, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4511-8. doi: 10.1145/2975961.2975964. URL http://doi.acm.org/10.1145/2975961.2975964.

[35] Pasquale Ardimento, Massimo Bilancia, and Stefano Monopoli. Predicting bug-fix time: Using standard versus topic-based text categorization techniques. In *Discovery Science*, pages 167–182. Springer, Cham, October 2016. URL https://link.springer.com/chapter/10.1007/978-3-319-46307-0_11. DOI: 10.1007/978-3-319-46307-0_11.

[36] Pasquale Ardimento and Andrea Dinapoli. Knowledge extraction from on-line open source bug tracking systems to predict bug-fixing time. In *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*, WIMS '17, pages 7:1–7:9, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5225-3. doi: 10.1145/3102254.3102275. URL http://doi.acm.org/10.1145/3102254.3102275.

[37] Lionel Marks, Ying Zou, and Ahmed E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pages 11:1–11:8, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0709-3. doi: 10.1145/2020390.2020401. URL http://doi.acm.org/10.1145/2020390.2020401.

[38] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. Using dynamic and contextual features to predict issue lifetime in github projects. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 291–302, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4186-8. doi: 10.1145/2901739.2901751. URL http://doi.acm.org/10.1145/2901739.2901751.

[39] George Kour, Shaul Strachan, and Raz Regev. Estimating handling time of software defects. *Conference: Fourth International Conference on Computer Science and Information Technology*, page 127– 140, 03 2017. doi: 10.5121/csit.2017.70413.

[40] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of Machine Learning and Data Mining*, pages 1056–1066. Springer US, 2017. doi: 10.1007/978-1-4899-7687-1_964. URL https://doi.org/10.1007/978-1-4899-7687-1_964.

[41] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10(100):2935–2962, 2009. URL http://jmlr.org/papers/v10/gunawardana09a.html.

[42] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2011. ISSN 2229-399X. doi: http://dx.doi.org/10.9735/2229-3981.

[43] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 1573-0565. doi: 10.1023/a:1010933404324. URL https://doi.org/10.1023/a:1010933404324.

[44] Yehuda Koren. Factor in the neighbors. *ACM Transactions on Knowledge Discovery from Data*, 4(1):1–24, January 2010. doi: 10.1145/1644873.1644874. URL https://doi.org/10.1145/1644873.1644874.

[45] Poonam B.Thorat, R. M. Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36, January 2015. doi: 10.5120/19308-0760. URL https://doi.org/10.5120/19308-0760.

[46] Shirin Akbarinasaji, Ayse Basar Bener, and Atakan Erdem. Measuring the principal of defect debt. In *Proceedings of the 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, RAISE '16, page 1–7. ACM Press, 2016. doi: 10.1145/2896995.2896999. URL https://doi.org/10.1145/2896995.2896999.

[47] Pieter Hooimeijer and Westley Weimer. Modeling bug report quality. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 34–43, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-882-4. doi: 10.1145/1321631.1321639. URL http://doi.acm.org/10.1145/1321631.1321639.

[48] Lotfi Ben Othmane, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, and Achim D. Brucker. Time for addressing software security issues: Prediction models and impacting factors. *Data Science and Engineering*, 2(2):107–124, June 2017. ISSN 2364-1185, 2364-1541. doi: 10.1007/s41019-016-0019-8. URL https://link.springer.com/article/10.1007/s41019-016-0019-8.

[49] Xinxi Wang and Ye Wang. Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 627–636, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3063-3. doi: 10.1145/2647868.2654940. URL http://doi.acm.org/10.1145/2647868.2654940.