Predicting Software Bugs using Machine Learning algorithm

Jay Chetankumar Patel
Pursuing Master of Computer Science
University of Ottawa
Ottawa, Canada
jpate186@uottawa.ca - 300288159

Honey Jigarkumar Patel
Pursuing Master of Computer Science
University of Ottawa
Ottawa, Canada
hpate142@uottawa.ca - 300252223

***Abstract*—Since the field of software engineering is expanding; it is vital to prevent the software from being faulty. Though software bug prediction has many benefits, like increasing reliability and quality, it is arduous to construct a model to predict software bugs. In this project, we apply machine learning algorithms, namely: Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), Support Vector Machine (SVM), K-nearest Neighbors (KNN), and Naïve Bayes (NB), to make predictions of bugs in software. Also, a comparative analysis of these algorithms is done based on their accuracy. We have used the PROMISE Software Engineering Repository Datasets namely JM1, KC1, and PC1. We got 88% accuracy for predicting bugs in JM1, 90% in KC1, and 95% in PC1 using different algorithms.**

Keywords—Software engineering, Machine Learning, Software bugs prediction, Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), K-nearest Neighbors (KNN), Naïve Bayes (NB), Logistic Regression (LR).

I. Introduction

Software Engineering is one of the industries with the fastest recent growth, yet software fault prediction has been a critical issue for a long time. Making decisions with this level of uncertainty can create an urgent problem. Software quality prediction calls for exact technologies and expert knowledge. In traditional approaches, detecting software faults inside software requires a prior understanding of errors or a faulty module. However, with upcoming technologies, the software can predict software errors using Machine Learning techniques.

Also, predicting software bugs is a crucial step in creating software. Identifying defective modules before the software is deployed increases user satisfaction and enhances code quality.

In this project, we employ Machine Learning approaches to develop predictive models to detect faults in software. For this, we used datasets related to software engineering and built models for fault prediction using it. Most approaches to identifying the defects construct their model using traditional hand-crafted features, such as Halstead's software volume metric and McCabe's cyclomatic complexity metrics.

The predictions made by this model will give an initial idea about the software's defects, enabling the software to employ an approach to continue its projected tasks with overcoming the error.

These predictive models can serve as valuable tools for predicting software errors using historical data from software development metrics. Such a feature will allow the software to operate more effectively by reducing the faults, time, and cost.

We have implemented Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), K-nearest Neighbors (KNN), Naïve Bayes (NB), and Logistic Regression (LR) machine learning algorithms to predict the software bugs. Then we compared the best algorithm with specific performance metrics.

The remainder of the project report is structured as follows. Section II summarizes the related work in the same domain. Followed section III talks about the implementation. In the implementation, we describe the research questions, the dataset used, the analysis procedure and metric, the results, and its threats to validity. Section IV describes the empirical evaluation. Lessons learned while doing the project are stated in Section V. Lastly, Section VI provides the conclusion of the work.

II. Related Work

Research paper [1] describes the use of XGBoost (a tree-based algorithm) to detect the possibility of a flaw. The main idea behind their research is to deduct the features which do not contribute to the prediction. Using different Machine Learning (ML) techniques (such as Decision trees, Support Vector Machines, Logistic Regression, Naive Bayes, K-nearest Neighbors, and other ML methods), they have concluded that features might vary according to the given project, and it is possible to find a powerful model with the help of just a few parts. There are about 20 different attributes used, which were obtained from a Java code. Technical Approach

The studies in [2] analyzed and compared three supervised machine learning approaches, Naïve Bayes, Decision Tree, and Artificial Neural Network, to predict the defects in the software. They collected experimental results using recall, precision, F-measure, and RMSE. Results analyzed that the Decision Tree shows better performance over other implemented algorithms and the existing methods like Auto Regression and the Known Power Model.

To forecast the defective modules, the authors of [3] suggested using a linear auto-regression (AR) technique. Based on previous data on the program's cumulative defects, the study forecasts future software problems. The study also assessed and contrasted the AR model and employed the Root Mean Square Error (RMSE) metric with the Known Power Model (POWM). This work was compared by the research [2].

The researchers in [4] uniquely found a way to use Deep Learning in order to measure functional code-similarity. A model was created by encoding the data flow and control flow for a semantic representation that can be used to train a Deep Neural Network model. That model learns from the features to perform binary classification.

In the research paper [5], they suggested a deep learning-based method in this study to forecast the quantity of software module flaws. The suggested technique performs better than the Support Vector Regression (SVR), Fuzzy Support Vector Regression (FSVR), and Decision Tree Regression (DTR) in terms of performance. On two well-known datasets, the suggested strategy dramatically reduces the mean square error and boosts the squared correlation coefficient compared to existing techniques.

In work [6], the researchers have conducted a comparative study on several machine learning algorithms such as Artificial Neural Networks, Naïve Bayes, Linear Classifiers, Decision Trees, and Particle Swarm Optimization. The study provided significant findings, including that the linear classifier outperforms other algorithms regarding defect prediction accuracy. DT, BL, ANN, SVM, RBL, and EA are the most important techniques in software defect prediction. Metrics for use in software defect prediction studies include Line of Code (LOC) metrics, cohesion, coupling, and inheritance, as well as other metrics.

The research work [10] focused on applying semi-supervised learning to predict software faults. They mainly deduced that current supervised learning algorithms could be replaced by trust fitting. In terms of dimensional reduction, when random forest and the semi-supervised learning algorithms were applied to the same dataset, the semi-supervised learning showed better results.

III. Machine Learning Algorithms

As mentioned above that, we have used six machine-learning algorithms to train the model on the KC1 and JM1 datasets. Let us briefly understand the machine learning algorithms that are implemented:

Random Forest (RF):

- Random forest is mainly used in classification problems due to its ability to perform better in that case. However, it can also be applied to regression problems. In this algorithm, a tree is built to classify using the majority votes.
- So, in this algorithm, firstly, x number of arbitrary points are taken from the dataset. This is then used to create individual decision trees for each record. Later on, depending on the problem, averaging or majority voting is applied. Fig. 1 shows the working of the Random Forest algorithm.
- It has diversity because each tree has different features and is independent of the other.

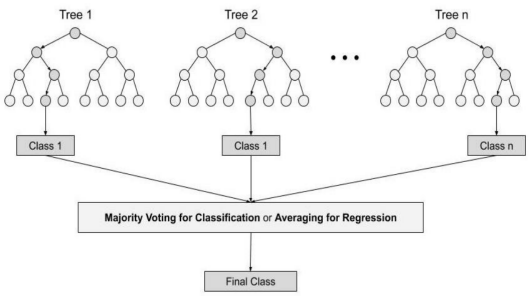

Figure 1. Random Forest [11]

Decision Tree (DT):

- A decision tree algorithm is also a supervised machine learning algorithm. In this algorithm, every time the data is split, either 'yes' or 'no.' And the node in the tree can be a decision node or the leaf.
- Now, let us consider an example to understand the case where we want to predict a person's fitness. So, the parent node will ask about the age. If it's less than the required, it goes into the 'yes' condition, 'no' otherwise. Figure 2 represents this example of the decision tree.
- This example is of the classification trees. A decision tree can also work for continuous data points.
- So, a parent node is created in a decision tree algorithm. Then, the entropy is calculated for each feature. This helps to select the attribute with maximum information gain. Do the same process for all other remaining features.

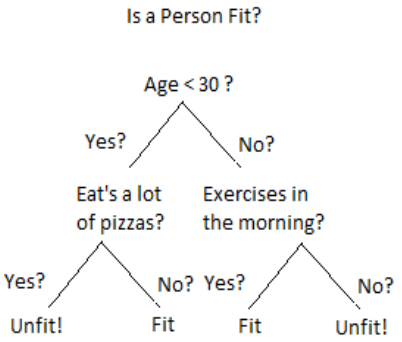

Figure 2. Decision Tree [12]

Support Vector Machine (SVM):

- A Support Vector Machine is also a supervised machine learning technique used for both classification and regression problems. But, it does very well for classification questions.
- The main idea behind this algorithm is to divide the data points using a hyperplane in multi-dimensional space. The hyperplane is drawn such that it creates the best separation between the classes.
- For instance, if we have two variables, x1 and x2, which are independent of each other, those variables are dependents for a variable, as shown in fig. 3. The hyperplane is a line separating two classes, namely red and blue (here).
- But, there can be a problem when the problem is non-linear. To do so, SVM kernels are used. They convert the lower dimensional sample space to higher dimensional sample space to make a separation.

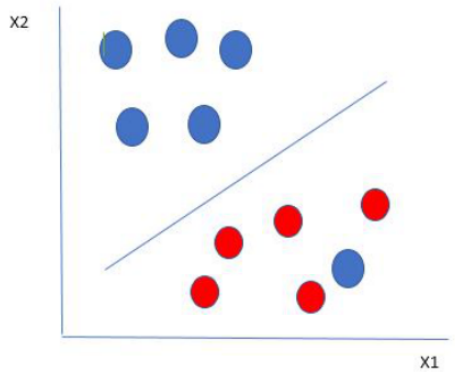

Figure 3. Support Vector Machine [13]

K-Nearest Neighbour (KNN):

- This algorithm does the same thing as previously described algorithms, i.e., using supervised learning to solve regression and classification problems. Here, 'K' is the total count of the nearest neighbours to a new data point that has to be classified.
- KNN can also be referred to as a distance-based algorithm because it finds the distance between the neighbouring data and the calculating variable to purify the closest neighbours.
- To quickly understand how it works, Fig. 4 shows how the five nearest neighbours are found to classify an unknown object as either a 'cat' or a 'dog.'

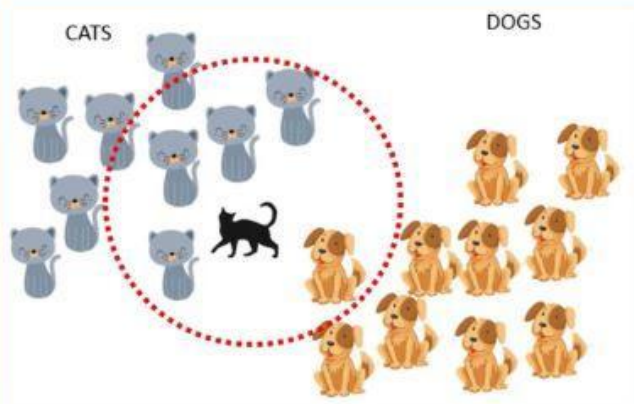

Figure 4. K-Nearest Neighbour [14]

Naïve Bayes (NB):

- Naïve Bayes is a supervised machine learning algorithm that is inspired by Bayes' theorem. It is mostly employed in text categorization with a large training dataset.
- It aids in the development of rapid machine-learning models capable of making accurate predictions. Being a probabilistic model, it makes predictions based on the likelihood that an object will occur.
- The following figure represents the formula for Bayes' theorem [17]. Where P(A|B) is the probability of A given B and P(B|A) represents the probability of B given A.
- There are three types of Naïve Bayes models, namely: Gaussian, Multinomial, and Bernoulli. In this project work, we have used the Gaussian model.
- Here we assume that the attributes are normally distributed. So, the predictors take continuous values when the Gaussian distribution is followed.

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

Figure 5. Naive Bayes equation

Logistic Regression (LR):

- An illustration of supervised learning is logistic regression. It is used to determine or forecast the likelihood that a binary event will happen.
- Binary categorization refers to the fact that there are only two answers to this question: either they are infected or they are not.
- Let us consider an example of logistic regression where we want to check whether a person is sick or not. That can be verified by considering the factors such as body temperature, cough, headache, and many others.
- In this method, the logistic function is used (or one can say it is a sigmoid function). Figure 6 shows the graph for the logistic function.

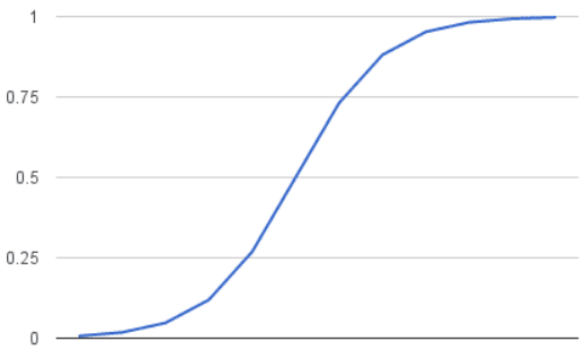

Figure 6. Logistic Function [18]

IV. IMPLEMENTATION

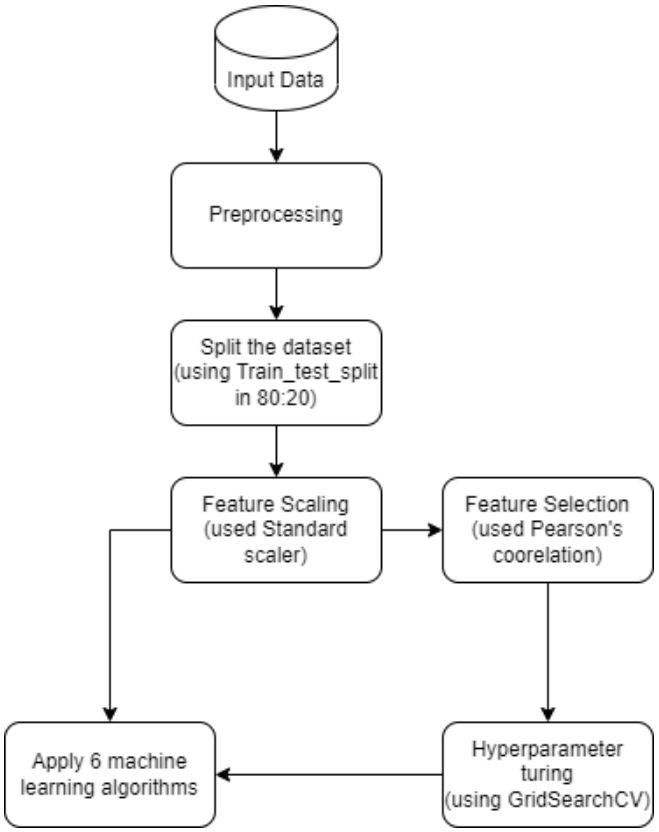

Figure 7. Implementation Flowchart

As the above flowchart denotes, firstly we performed preprocessing on all three datasets, then split the train data and test data with a ratio of 80:20. After that feature scaling is done using Standard scalar, and then to check the initial accuracy we applied six algorithms. Later, feature selection and hyperparameter tuning are done and algorithms are trained again, which gives improved accuracy.

Implementation Process – Preprocessing

For dataset *'JM1'*: Firstly we checked for missing values, and then for NaN values, but none were present. Later, we found that some entries have question marks i.e. "?" as their value. We changed such values to NaN values, and then replaced them with the median of the respective feature. After that, the last problem was some features were of object datatype, which machine learning models will not accept. Hence, we converted such features to the 'float64' data type. Finally, we got an accurate dataset, which was split with an 80:20 ratio, and since only the numeric data is present, we did feature scaling using a standard scaler.

For dataset *'KC1'*: This dataset is quite simple, we checked for missing values, and then for NaN values, but the entries were accurate. Later train data and test data are divided with a ratio of 80:20 and feature scaling is done using a standard scaler on numeric data.

For dataset *'PC1'*: Starting with the basic step, we checked for missing values and NaN values, but the entries were accurate. Dataset is then divided into train data and test data. Lastly, a standard scaler is applied for scaling the numeric data.

Implementation Process – Feature Selection:

- Feature selection is required before training the machine learning algorithm because it can reduce the computational cost and can improve the accuracy in some cases [19].
- In the statistical-based feature selection method, we check the relationship between the input attribute and the target attribute for all the inputs.
- However, depending on the data, we have to choose the type of statistical-based feature selection technique.
- Here, we have used Pearson's Correlation Coefficient which can find the relationship between the attributes.
- If the value of Pearson's Correlation Coefficient is 1, then it is highly correlated. 0 represents there is no presence of correlation and negative value states there is inverse proportion.
- In code, we have first calculated the Absolute Value of Correlation for each feature with other features. On getting any two features that are equally correlated with the target label ('defects'), we will remove one from them. We got eight such pairs of features that were equally correlated and on removing them we were left with 12 features in X.

Implementation Process – Hyperparameter Tuning:

- Finding the optimal combination of hyperparameters to enhance the performance of the model is known as hyperparameter tuning [20].
- It operates by doing several repetitions within a continuous training phase. Every attempt entails the full operation of the training data with the values of the selected hyperparameters set within the predetermined bounds.
- Once this procedure is complete, we will have the set of hyperparameter values that the model needs to perform at its finest.
- We have used the Grid Search method (GridSearchCV) to do hyperparameter tuning on the dataset in this project. For this, we made a dictionary of parameters that we wanted to tune and passed it to the GridSearchCV function, which will give output in the form of a table showing the optimal set of parameters for each algorithm. Later, these parameters are used while training the algorithms.
- Following are the features that we tunned across different algorithms which are shown in Table 1:

Table 1. Parameters for hyperparameter tuning

Algorithms	**Parameters for Hyperparameter Tuning**
Decision Tree	criterion, max_depth
Random Forest	n_estimators, criterion, max_depth
Logistic Regression	C, penalty, max_iter
Support Vector Machine	C
K-Nearest Neighbour	n_neighbors, weights
Naive Bayes	var_smoothing

Implementation Process – Training Algorithms:

- We have trained the model using K-fold cross-validation where K=10, here the dataset is divided into 10 folds, out of which 9 folds are used for training the model and 1 is used for testing the model. Thus it ensures that every observation from the dataset appears in the train-set and test-set, and the possibility of a lucky split is eliminated which results in better accuracy. Moreover, it improves the model prediction even when we don't have large data to use other methods. Then six algorithms mentioned above are trained and cross_val_score is used to get the accuracy of models. Also, the classification report, confusion matric, and F1_score are shown.

V. Empirical Evaluation

A. Research Questions

RQ1: What techniques we are using for feature reduction?
Answer: Refer to 'Implementation Process – Feature Selection'

RQ2: Does the best Machine Learning algorithm change according to the dataset?
Answer: Yes. (Refer to 'Results and discussion', 'Lesson Learn', and 'conclusion')

RQ3: What are the most important features for Software Bugs Prediction?
Answer: We have done feature ranking in the code in order to get the most important feature. Feature ranking is done by using the Information Gain measure. It uses 'mutual_info_classif', which will estimate the mutual information for a discrete target variable. This will give a ranking for different features. After then, we displayed the features with the highest to lowest ranking, in which 'ev(g)' has the highest ranking of 0.26 and 'l' has the lowest ranking of 0.0015. We have also shown the correlation matrix to display which feature is most correlated with the target label.

RQ4: What are the other datasets used for Software Bugs Prediction?
Answer: Many similar datasets are available on PROMISE Software Engineering Repository Datasets, for example, 'CM1', 'KC2', ' DATATRIEVE Transition', etc.

B. Description of the Dataset

We have used the two datasets which are freely available in the PROMISE Software Engineering Repository data [7]. The title of our three datasets are named below:

- JM1/software defect prediction [9]
- KC1/software defect prediction [8]
- PC1/software defect prediction [23]

In both datasets, there are the same 22 attributes. From 22 different attributes, one attribute is dependent on the other 21 characteristics because it has the outcome (meaning whether it has flaws or not). Later on, depending on the results from the above datasets, we might add a few more datasets for implementation.

Based on the arguments McCabe and Halstead gave, this dataset uses them as some of the attributes. For example, the line count of code, cyclomatic complexity, and others are counted using McCabe's method. At the same time, Halstead has several unique operators, a number of unique operands, and other attributes based on it. Table 2 represents the list of the metrics.

The cyclomatic complexity is from McCabe's method representing the maximum value or the mean over all the class methods. The line count of the code is the total number of lines in the program. The amount to which a flowgraph may be "reduced" by breaking it down into all of its "D-structured primes" is known as essential complexity. Design complexity is the CC of the module flow graph.

Table 2. List of the Attributes

Sr. No.	**Attribute name**
1	Line count of code
2	Cyclomatic complexity (CC)
3	Essential complexity
4	Design complexity
5	Total operators and operands
6	Volume
7	Program Length
8	Difficulty
9	Intelligence
10	Effort
11	b: numeric
12	Time Estimator
13	Line count
14	Count of lines of comments
15	Count of blank lines
16	IO Code and Comment
17	Unique operators
18	Unique operands
19	Total operators

20	Total operands
21	Branch count
22	Defects

C. *Analysis Procedure and Metric*

We employed a set of well-known metrics to determine the effectiveness of applying Machine Learning (ML) algorithms in Software Bug Prediction. All those metrics are described below:

Accuracy:

- As shown in Eq. 1, the ratio of all the accurate outputs over the total number of cases looked at.
- Where TP represents True-Positive, TN is True-Negative, FP is False-Positive, and FN is False-Negative for all the equations.
- If the accuracy value is 1, then the model is the most accurate.

$$\text{Accuracy} = (TP + TN)/(TP + TN + FP + FN) \quad (1)$$

Precision:

- The number of real true positives divided by the total of positive predictions is how precision is determined.
- The following equation shows the formula for Precision:

$$\text{Precision} = TP / (TP + FP) \quad (2)$$

Recall:

- If we take the ratio of true positives and that to the total number of positives, then we get the recall.
- Equation 3 shows the formula for it:

$$\text{Recall} = TP / (TP + FN) \quad (3)$$

F - score:

- The total efficacy of the classification approach is determined by the F-score, which takes P and R into account.
- The weighted harmonic mean of recall and precision is used to calculate the F-measure. In equation 4, P is Precision and R is Recall measuring the F – score.

$$F - \text{score} = (2*P*R) / (P + R) \quad (4)$$

Confusion Matrix:

- Confusion matrix is used to measure the performance of the machine learning classification tasks.
- Fig. 8 illustrates a confusion matrix where the rows show the predicted values, and the columns represent the actual values.
- It provides the count of True Positive, False Positive, False Negative, and True Negative when testing the algorithm.

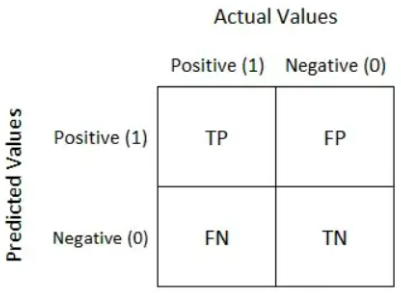

Figure 8. Confusion Matrix

D. *Results and Discussion*

For dataset '*JM1*':

- Following table shows the accuracy of six algorithms before and after applying feature selection and hyperparameter tuning:

	Accuracy_befr_HPT	Accuracy_wth_HPT_FS
Decision_Tree	72.724141	88.152287
Random_Forest	79.789143	86.661959
Logistic_Regression	78.558547	82.897533
SVM	72.249762	86.588016
KNN	79.458227	82.675183
Naive_Bayes	80.395676	87.854670

Figure 9. Accuracy for JM1 dataset

- As shown in the figure. 9, initially Navie Bayes performed well with an accuracy of 80.39%, followed by Random Forest, KNN, and Logistic Regression with 79.78%, 79.45%, and 78.55% of accuracy, respectively. However, after hyperparameter tuning and feature selection, accuracy improves dramatically for the **Decision Tree** from 72.72% to 88.15%, and it is the best-performing algorithm for dataset ***JM1***. Also, Naive Bayes has an improved accuracy of 87.85%, followed by Random Forest and SVM with an accuracy of 86.66% and 86.58%, respectively.
- Figure 10, shows accuracy, F1 score, Confusion Matrix, and Classification report for the Decision Tree Classifier.

```
Accuracy by cross_val_score fuction :  88.15228656889752
F1 score :  0.0

Confusion Matrix :-
 [[1756    0]
 [ 421    0]]

Classification Report :
              precision    recall  f1-score   support

       False       0.81      1.00      0.89      1756
        True       0.00      0.00      0.00       421

    accuracy                           0.81      2177
   macro avg       0.40      0.50      0.45      2177
weighted avg       0.65      0.81      0.72      2177
```

Figure 10. Evaluation metrics for DT

For dataset '*KC1*':

- The accuracy of six algorithms before and after applying feature selection and hyperparameter tuning is denoted in the table below:

	Accuracy_befr_HPT	Accuracy_wth_HPT_FS
Decision_Tree	78.286617	89.542541
Random_Forest	82.696005	90.544121
Logistic_Regression	84.165425	90.971564
SVM	80.514331	88.542541
KNN	82.932747	88.543444
Naive_Bayes	82.175807	89.593320

Figure 11. Accuracy for KC1 dataset

- Figure. 11 depicts that initially, Logistic Regression outperforms all the algorithms with an accuracy of 84.16%, followed by KNN, Random Forest, and Naive Bayes with 82.93%, 82.69%, and 82.17% of accuracy, respectively. After applying hyperparameter tuning and feature selection, still, Logistic Regression performs best with an accuracy of 90.97% which is however improved substantially from initial accuracy. Also, Random forest has similar accuracy of 90.54%, followed by Decision Tree and Naive Bayes with a similar accuracy of 89%. Here, **Logistic Regression** is the best-performing algorithm.
- Figure 12, shows accuracy, F1 score, Confusion Matrix, and Classification report for the Logistic Regression Algorithm.

```
Accuracy by cross_val_score fuction :  90.97156398104266
F1 score :  0.27586206896551724

Confusion Matrix :-
 [[347  10]
 [ 53  12]]

Classification Report :-
              precision    recall  f1-score   support

       False       0.87      0.97      0.92       357
        True       0.55      0.18      0.28        65

    accuracy                           0.85       422
   macro avg       0.71      0.58      0.60       422
weighted avg       0.82      0.85      0.82       422
```

Figure 12. Evaluation Metrics for LR

For dataset '*PC1*':

- Table below states the accuracy before and after applying feature selection and hyperparameter tuning for six algorithms:

	Accuracy_befr_HPT	Accuracy_wth_HPT_FS
Decision_Tree	89.546274	95.157330
Random_Forest	92.968878	95.257330
Logistic_Regression	93.147420	93.976249
SVM	85.117936	94.957330
KNN	93.149058	94.857330
Naive_Bayes	89.090090	93.967240

Figure 13. Accuracy for PC1 dataset

- From the figure. 13, it is visible that initially, both Logistic Regression and KNN perform equally well across all the algorithms with an accuracy of 93.14%, followed by Random Forest, and Decision Tree with 92.96% and 89.54% of accuracy, respectively. However, after applying hyperparameter tuning and feature selection, the Random forest performs best with an accuracy of 95.25% which is improved from the initial accuracy. Then, the Decision tree also performs well with an accuracy of 90.15%, followed by SVM and KNN with a similar accuracy of 94%. Here, almost all algorithm performs quite well, but **Random Forest** is the best-performing algorithm.
- Figure 14, which shows accuracy, F1 score, Confusion Matrix, and Classification report for the Random Forest Classifier.

```
Accuracy by cross_val_score fuction :  95.25733005733007
F1 score :  0.11764705882352941

Confusion Matrix :-
 [[206   1]
 [ 14   1]]

Classification Report :
              precision    recall  f1-score   support

       False       0.94      1.00      0.96       207
        True       0.50      0.07      0.12        15

    accuracy                           0.93       222
   macro avg       0.72      0.53      0.54       222
weighted avg       0.91      0.93      0.91       222
```

Figure 14. Evaluation Metrics for RF

E. Threats to Validity

As we will mention later, the investigation described in this report contains flaws that could jeopardize the validity of our findings. We start by outlining the internal threats to validity, followed by external threats.

Internal Validity:

- In internal validity, the threat arises when one is unaware of the situation's extraneous or confounding factors influencing the investigated aspect.
- Let us consider an example to understand the above statement. If we want to verify if the consumption of coffee increases memory power, then we take two groups of students of the same age bracket. The first group (the treatment one) is given coffee in the morning time, while the other group (the control one) is assigned a cup of water to drink before conducting sessions. The results showed that coffee improves memory.
- This example threatens internal validity because of a time difference while conducting experiments. This way, we can deduce that results are changed because the time to conduct an experiment is not similar [16].
- However, we have trained the models for all three datasets with the same feature selection method and hyperparameter tuning but the best-performing algorithms might change if there are changes in the correlation of features or in the values of features or in the parameters that we tune.

External Validity:

- External validity basically means to generalize the solution that is being proposed.
- A particular case study can face an external threat to validity, but the solution to that situation is carrying several case studies [15].
- To understand the definition more clearly, consider that we have around 5000 male students studying at the University of Ottawa. From that, we conduct an experiment with approximately 500 students who are Indian international students. We want to see which is their favourite outdoor sport. The experiment depicted that nearly 60% of the students love cricket.
- However, this cannot deduce that majority of the total students love cricket as an outdoor sport. Because let's assume the majority of the students are from North America, and they like football more than cricket. So, this led to external validity because our solution in this experiment was not generalized.
- Our code is flexible to use directly on datasets similar to JM1, KC1, and PC1. For, datasets of a different kind, they might need to add some additional steps in preprocessing other than what we have done. For instance, these datasets do not have any categorical data, hence there was no need for encoding categorical data to numerical data. However, if the dataset has categorical data then the user will have to do its encoding before employing our models.

VI. Lessons Learned

- Firstly, we discovered that not every algorithm will necessarily perform well on the same dataset. Every algorithm has unique characteristics and qualities, hence their performance will vary from one to another dataset. For example, here on dataset JM1, the Decision Tree works very well with an accuracy of 88.15% whereas on the same dataset KNN has lower accuracy of 82.67%.
- Secondly, by using the same algorithms across different datasets, we were able to conclude that the performance of the same algorithm will vary as per different datasets, even if those datasets are of the same kind. For instance, Logistic Regression performs moderately on the JM1 dataset with 82.89% accuracy and performs well on the KC1 dataset with 90.97% accuracy, whereas with the PC1 dataset it performs best with an accuracy of 93.97%.
- We also observed that various criteria will affect how well an algorithm performs. Simply said, employing alternative parameters or methods can improve or decrease the accuracy of a given algorithm. To cite, on dataset JM1 we tried to use SMOTE to improve the accuracy of algorithms, however, on employing SMOTE, the accuracy of all algorithms dropped drastically to 40% or 80%. While with hyperparameter tuning, accuracy actually improved.
- Lastly, we realized that accuracy is not the only metric used to evaluate the models. Other metrics like precision, recall, F1 score, confusion matrix, classification report, etc also exist.

VII. Conclusion

- Finally, in this project, we propose models which make predictions across different software-related datasets, where prediction is done with 88.15% accuracy in the JM1 dataset, 90.97% accuracy in the KC1, and 95.25% accuracy in the PC1 dataset.
- Also, a comparative analysis of these algorithms across different datasets is presented which concludes that the Decision Tree algorithm works best on the JM1 dataset, the Logistic Regression algorithm works well on the KC1 dataset, and the Random Forest algorithm works best on the PC1 dataset.

Table 3. Best algorithm across different datasets

Dataset	**Algorithm**	**Accuracy**
JM1	Decision Tree algorithm	88.15%
KC1	Logistic Regression algorithm	90.95%
PC1	Random Forest algorithm	95.25%

VIII. Code link (Github)

- https://github.com/jay-patel-07/Predicting-Software-Bugs-using-ML-algorithms

References

[1] Esteves, G., Figueiredo, E., Veloso, A. et al. Understanding machine learning software defect predictions. Autom Softw Eng 27, 369–392 (2020). https://doi.org/10.1007/s10515-020-00277-4

[2] Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. International Journal of Advanced Computer Science and Applications, 9(2).

[3] A. Sheta and D. Rine, "Modeling Incremental Faults of Software Testing Process Using AR Models," the Proceeding of 4th International Multi-Conferences on Computer Science and Information Technology (CSIT 2006), Amman, Jordan. Vol. 3. 2006.

[4] G. Zhao, J. Huang, Deepsim: Deep learning code functional similarity, in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018, ACM, New York, NY, USA, 2018, pp. 141–151. doi:10.1145/3236024.3236068.

[5] Qiao, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning-based software defect prediction. Neurocomputing, 385, 100-110.

[6] Singh, Praman Deep, and Anuradha Chug. "Software defect prediction analysis using machine learning algorithms." 7th International Conference on Cloud Computing, Data Science & EngineeringConfluence, IEEE, 2017.

[7] PROMISE Software Engineering Repository: http://promise.site.uottawa.ca/SERepository/datasets-page.html

[8] JM1/software defect prediction: http://promise.site.uottawa.ca/SERepository/datasets/jm1.arff

[9] KC1/software defect prediction: http://promise.site.uottawa.ca/SERepository/datasets/kc1.arff

[10] H. Lu, B. Cukic, and M. Culp, "Software defect prediction using semisupervised learning with dimension reduction," in Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on. IEEE, 2012, pp. 314–317.

[11] Understanding Random Forest: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[12] Decision Trees for Classification: A Machine Learning Algorithm: https://www.xoriant.com/blog/decision-trees-for-classification-a-machine-learning-algorithm#:~:text=Introduction%20Decision%20Trees%20are%20a,namely%20decision%20nodes%20and%20leaves.

[13] Support Vector Machine Algorithm: https://www.geeksforgeeks.org/support-vector-machine-algorithm/

[14] KNN - The Distance Based Machine Learning Algorithm: https://www.analyticsvidhya.com/blog/2021/05/knn-the-distance-based-machine-learning-algorithm/#:~:text=The%20abbreviation%20KNN%20stands%20for,by%20the%20symbol%20'K.'

[15] External validity: https://www.scribbr.com/methodology/external-validity/

[16] Internal validity: https://www.scribbr.com/methodology/internal-validity/

[17] Naïve Bayes Classifier: https://www.javatpoint.com/machine-learning-naive-bayes-classifier

[18] Logistic Regression for Machine Learning: https://machinelearningmastery.com/logistic-regression-for-machine-learning/

[19] Feature Selection: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

[20] Hyperparameter tuning in Python Complete Guide: https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide

[21] Hyperparameter tuning using Grid Search: https://medium.com/@jackstalfort/hyperparameter-tuning-using-grid-search-and-random-search-f8750a464b35

[22] Pearson Correlation: https://www.analyticsvidhya.com/blog/2021/01/beginners-guide-to-pearsons-correlation-coefficient/

[23] PC1/software defect prediction: http://promise.site.uottawa.ca/SERepository/datasets/pc1.arff