CLASSIFICATIONS OF VARIOUS CODE SMELLS USING MACHINE LEARNING TECHNIQUES

Submitted By

Harmanjot Singh
(Roll No. 202201013)
Shlok Aggarwal
(Roll No. 202201040)

MCA Final Year

Under the supervision of:

Dr. Palika Chopra
Assistant Professor

Dr. Nidhi Kalra
Assistant Professor

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

PATIALA – 147004

December 2023

Certificate

I hereby attest that the work presented in the thesis titled "*Detecting and characterizing various code smells using machine learning"* which was submitted to the Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala, in partial fulfillment of the requirements for the award of the degree of Master of Computer Application, is an authentic record of my own work completed under the guidance of Dr. Palika Chopra and Dr. Nidhi Kalra and refers other researcher's work which are duly listed in the reference section

This thesis has not been submitted for consideration for any other degree from this or any other university.

Shlok Aggarwal

Harmanjot Singh

That statement made by the candidate is correct and true to my knowledge, so I am guaranteeing it.

Dr. Palika Chopra

Assistant Professor,

Computer Science and Engineering Department

Dr. Nidhi Kalra

Assistant Professor,

Computer Science and Engineering Department

Acknowledgment

I would like to express my appreciation and deepest gratitude to my respected supervisors **Dr. Palika Chopra**, Assistant Professor and **Dr. Nidhi Kalra**, Assistant Professor, Computer Science and Engineering Department for guiding and encouraging me to accomplish this task successfully. I am very much thankful to my supervisors for her valuable times, patients, insightful recommendations and sharing experiences as well as knowledge with me which has made this journey very enjoyable and rewardable. It would not have been able to complete this research without their participation. We owe a huge debt of gratitude to our instructors for affording us the chance to contribute to this effort.

I am also very much thankful to **Dr. Shalini Batra**, Professor and Head of Computer Science and Engineering and **Dr. Karamjeet Singh**, MCA Coordinator, for the motivation and inspiring for the completion of project.

I am also very grateful to the entire faculty team and staff member of Computer Science and Engineering Department for their cooperation throughout for assistance.

I also would like to thank for their continuous support during this journey. Finally, I would like to really thank my family and friends for their wonderful support and encouragement, without them none of this work would have been accomplished.

Shlok Aggarwal

(**202201040**)

Harmanjot Singh

(202201013)

Abstract

These days, machine learning is incredibly effective and helpful for prediction. Various algorithms are available in machine learning based on the nature of the problem and the use cases. It falls under the category of artificial intelligence and computers. Its main focus is on how to use data and algorithms to imitate learning in a manner similar to that of "humans," with an emphasis on gradually increasing the system's accuracy. For the development of model that can predict and analysis of bad smell due to poor misguided programming specific coding structures or patterns that could point to the existence of a more serious issue some common types of code smells Long Method, Feature Envy, Duplicated Code, God Class. The identification of code smells can be a matter of personal judgment, and the process is often automated using tools called "linters" or "static analyzers." These technologies have the capability to detect potential problems and assist developers in upholding a uniform and uncluttered codebase. In addition, the use of code reviews and continuous integration procedures can assist in promptly identifying and resolving code smells during the first stages of software development.

In this research, we have applied different feature selection and extraction techniques on “Code Smell Detection” data set to select or extract top features, so that model can be trained using a different Rules and Rule Condition and other information to predict that following code give which type smell. This work utilized many prediction models in machine learning, including SVM, Decision tree, linear regression, Logistic regression, and AdaBoost, Bagging. These models were used as random forest regressors and classifiers, and their performance was evaluated. This study used various feature selection and extraction strategies.

The investigation revealed that Selection Techniques achieved an accuracy of up to 86% on K-Fold trained data set when used with Ensembles Algorithms. Similarly, resulted in Simple Algorithms achieving an accuracy of up to 98% on K-Fold trained data set. Machine learning algorithms exhibit improved performance when trained on K-Fold cross-validated datasets.

Table of Contents

Certificate....2
Acknowledgment....3
Abstract....4
Table of Contents....5
1. Introduction....7
  1.1 Introduction to Code Smell....7
  1.2 Introduction of machine learning....9
    1.2.1 Working of Machine Learning....10
    1.2.2 Need of Machine Learning....11
    1.2.3 Use cases of Machine Learning....12
  1.3 Code Smell with machine learning....13
  1.4 Preliminaries and Background....14
    1.4.1 Feature Selection Techniques....14
    1.4.2 Simple Algorithms....15
    1.4.3 Ensemble Algorithms....18
  1.5 Functional Requirements:....21
    1.5.1 Data Input:....21
    1.5.2 Smell Detection:....21
    1.5.3 Scalability:....22
    1.5.4 Machine Learning Model:....22
  1.6 Non - Functional Requirements:....22
    1.6.1 Performance:....22
    1.6.2 Accuracy:....22
    1.6.3 Usability:....22
    1.6.4 Resource Utilization:....22
  1.7 Work Breakdown Structure:....23
  1.8 Cost Analysis:....25
2. Literature Survey....26

3. Purpose of Study ... 39
4. Implementation & Results ... 41
4.1 Tools Used ... 41
4.1.1 Hardware requirements ... 41
4.1.2 Software Requirement ... 41
4.2 Implementation ... 42
4.2.1 Dataset ... 42
4.2.2 Preprocessing ... 43
4.2.3 Feature Transformation ... 44
4.2.4 Feature Selection ... 45
4.3 Splitting Data & Testing Simple Models ... 46
4.3.1 Testing Ensemble Models ... 48
4.4 Data Visualization ... 52
4.5 Algorithm Results ... 54
4.5.1 Simple Algorithms on Selection Technique ... 55
4.5.2 Ensemble Algorithms on Selection Technique ... 55
5. Conclusion ... 56
6. References ... 57

1. Introduction

In this chapter, the basics of the Code Smell Detector have been discussed. Further, an introduction to machine learning, its different types of application, and use cases are also mentioned. It also includes machine learning models and techniques on Code Smell that are performed to predict or Analysis the result or cases.

1.1 Introduction to Code Smell

Code smell refers to specific attributes present in the source code of a software application that may suggest an underlying issue. Similar to how a foul scent suggests potential problems with food, code smells indicate potential flaws with the structure, design, or implementation of code. Detecting and resolving code smells is essential for preserving a robust and sustainable codebase. Typical examples of code smells include the following: duplicate code, dead code, long methods, long parameter list, comments. Code that is considered "smelly" might be inefficient, lacking in performance, intricate, and challenging to modify and maintain. Although code smells may not necessarily signify a notably severe issue, adhering to them frequently results in uncovering diminished code quality, loss of application resources, or even crucial security vulnerabilities contained inside the application's code. At minimum, it necessitates teams to conduct thorough examinations of the code and frequently uncovers crucial sections in the code that demand corrective action. Code smells arise from poor or erroneous programming practices. These anomalies in the application code can frequently be directly attributed to errors committed by the application programmer during the coding process. Code smells generally arise from a lack of adherence to essential coding standards. Alternatively, it indicates that the paperwork necessary to precisely establish the project's development norms and anticipated outcomes was deficient, incorrect, or absent. There are many situations that can cause code smells, such as improper dependencies between modules, an incorrect assignment of methods to classes, or needless duplication of code segments. Code that is particularly smelly can eventually cause profound performance problems and make business-critical applications difficult to maintain. Code smell is not a literal defect; it is probable that the code can still be compiled, and functions as intended. Code smells are clear signs of possible violations of code discipline and design principles. However, it is conceivable that the origin of a code smell could lead to subsequent problems and failures as time progresses.

Here are some common types of code smells:

Duplicate Code**:**

Symptom: Duplicated code segments found in various sections of the program.

Impact: Augments the level of maintenance exertion and the likelihood of producing software defects

Long Method**:**

Symptom: An extremely lengthy and complex function or technique.

Impact: Diminishes code clarity and affects learning and maintenance of the code.

Large Class**:**

Symptom: A class that has become excessively huge, containing numerous methods and attributes.

Impact: Diminishes the legibility and manageability of the code; violates the Single Responsibility Principle.

Feature Envy**:**

Symptom: An approach that appears to prioritize the data of another class over its own.

Impact: The code violates the principle of encapsulation, which indicates a potential weakness in the design.

1.2 Introduction of machine learning

Machine learning refers to the process of a machine acquiring knowledge and skills to solve practical problems in the present or real-world context. Machine learning encompasses several models and strategies that enhance a machine's ability to learn from input data, resulting in improved accuracy and more precise outcomes. The process is similar to how the human brain functions, gradually improving the accuracy of models through experiences and information [5]. Machine learning allows models to learn systematically from historical datasets without the need for daily intervention in our programs or algorithms. It 's and arm of "AI (Artificial Intelligence)" that is more to related to execution of algorithms that help out workstation to gather information and get trained from historical experiences and data to produce accurate results for the models.

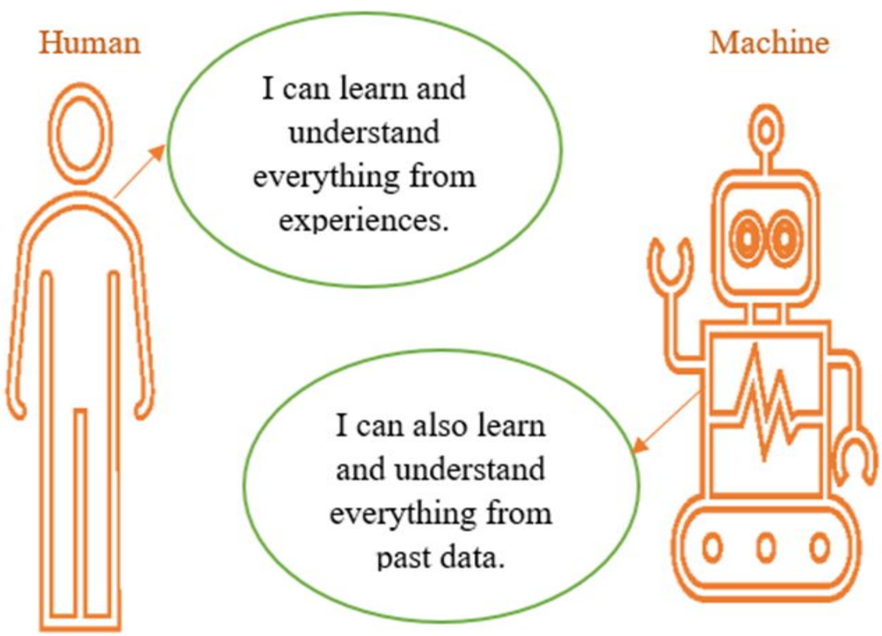


Figure 1: Machine learning description

Machine learning can be classified into three different kinds of algorithms:

. Supervised Learning

. Unsupervised Learning

. Reinforcement Learning

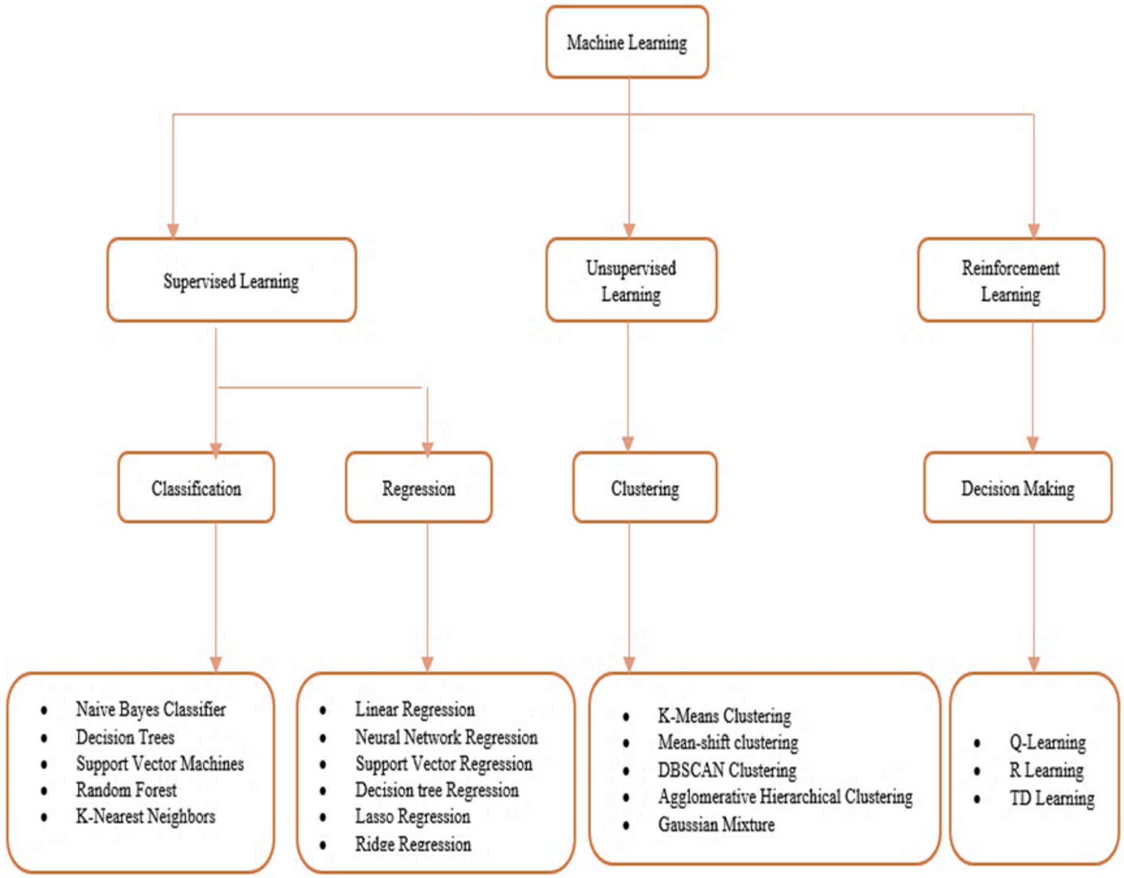


Figure 2: Types of machine learning

1.2.1 Working of Machine Learning

Machine learning algorithms are taught using historical data to create predictive models. When fresh data is inputted, these models are used to provide output predictions. The model undergoes training using previous or prior data, enabling it to forecast fresh input accurately and precisely.

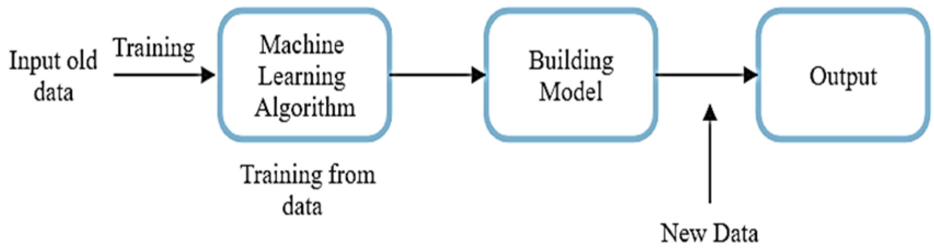


Figure 3: Working of machine learning

1.2.2 Need of Machine Learning

The usage of machine learning is increasing because there is an exponential increase in the data as well and it is not that easy for normal human beings to work on that much huge volume of data and analyses. Machine learning has models and techniques available. With their help it is easy to work on that much huge volume of data and perform the analysis of the output of that data. Machine learning helps us to work on difficult problems. This technology is driven based on input-data and used to find out different designs in a provided dataset. Also, it gets trained from history data provided to machine as input.

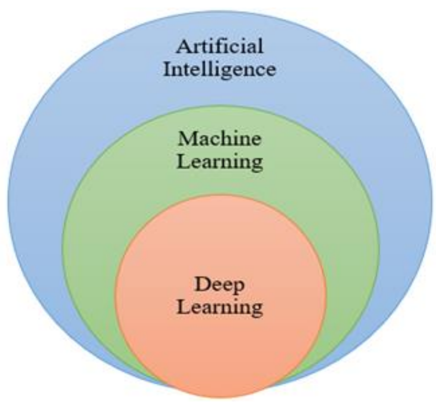


Figure 4: Family of machine learning

1.2.3 Use cases of Machine Learning

Machine learning can be applied to several use cases, such as tracking Uber rides. Machine learning is currently employed in various industries, including the healthcare sector, to facilitate medical diagnosis. Online transaction and payment fraud detection, automated identification and filtering of spam emails and malware. Autonomous vehicles, suggesting product recommendations based on clients' browsing history and search activity on the internet. The prediction of traffic patterns using historical data, the recognition of speech or conversion of audio to text, and the interpretation of images to make them understandable or legible in human language. Automatic language translation is a feature of machine learning that enables the conversion of text from one language to another without human intervention. It is anticipated that there will be an increase in the use of robots to perform tasks traditionally done by humans, resulting in cost reduction, improved quality, and faster problem resolution.

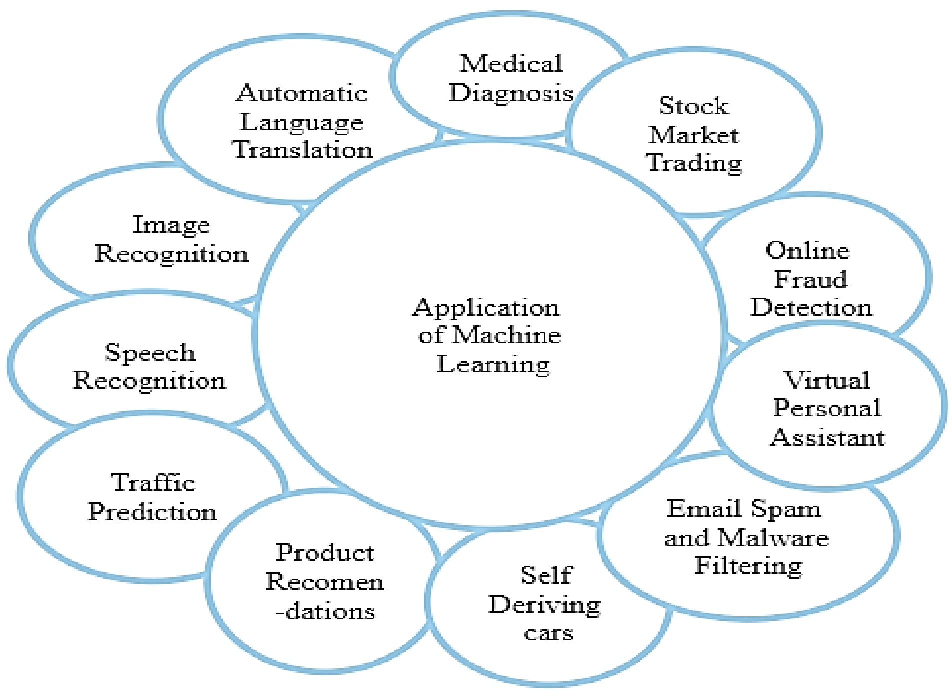


Figure 5: Applications of machine learning

Material & Methods

1.3 Code Smell with machine learning

Cases have been studied from distinct perspectives in recent times. There is an assumption that machine learning will be used to identify bad smells of Code Script. The approach presented in this study utilizes supervised machine learning methods to facilitate a learn-by-example approach for constructing code smell detection criteria. The majority of procedures include the capability to generate a confidence value, indicating the degree to which the results align with the acquired model. Certain algorithms additionally offer easily understandable guidelines that have been examined to determine which combination of measurements has the most impact on detecting and uncovering the inferred threshold values. To effectively use machine learning to the code smell detection problem, it is necessary to formalize the input and output of the learning algorithms. Additionally, careful selection of the data to be studied and the methods to be used in the experiments is required. Tempero et al. (2010) computed a comprehensive collection of object-oriented metrics on a diverse range of software systems, encompassing several elements of software design. Metrics are the factors that are independent in machine learning technique. A collection of code smells has been identified as the dependent variables. Each code scent has undergone manual evaluation and labeling of example occurrences to determine whether they are affected or not affected by the code smell. The process of selecting and labeling sample cases is crucial in machine learning approaches. Our methodology uses stratified random sampling to select example cases from many projects. This selection process is driven by the outcomes of a collection of pre-existing code smell detection tools and guidelines, referred to as Advisors.

Throughout this research, we used libraries of python and python itself, including sk learn, Pandas, NumPy, Machine Learning Algorithms like K-Nearest neighbors (KNN), Support vector machine (SVM), Logistic regression (LR), Decision tree (DT), Gaussian naïve bayes (GNB) and Ensemble Algorithms like Random Forest (RF), Bagging (BAG) and Adda Boosting (AB). For this investigation, we utilized the CodeSmellDescription Dataset and used a selection technique and to assess the accuracy of several algorithms. The study is structured into five distinct stages: data pre-processing, which involves cleansing, transformation, and reduction; encoding; feature selection or extraction; visualization; and prediction. We utilized many machine learning models and ensemble algorithms to train on the dataset and assessed the accuracy score. Given a dataset as input to any or all machine learning ensemble algorithms for the accuracy analysis in this work, we used the trained Normal and K-Fold dataset work on the 3-way efficient model (train-validation and test).

1.4 Preliminaries and Background

In this, we have mentioned machine learning selection techniques and different algorithms or models. The algorithms covered are from simple and ensemble part of machine learning.

1.4.1 Feature Selection Techniques

When constructing a model, the most crucial attributes are those that directly influence or provide advantages to the situation at hand. Feature selection is a strategy that involves eliminating redundant, superfluous, or distracting components from the initial collection in order to determine the most significant features. Feature selection is a technique used to limit the number of input variables in a model by picking just the most relevant data. This helps to prevent overfitting.

1. Information gain: - Information gain quantifies the extent to which entropy is decreased as a result of modifying the dataset. It is used as a technique for selecting features by computing the information gain of each variable in relation to the target variable.

2. Forward Feature Selection**:** – The decision to use Forward Selection is derived from an iterative process, starting with no initial model attributes. During each iteration, new features are continuously included into the models to improve their performance, until the addition of a new attribute no longer results in an increase in model performance.

3. Backward Feature Selection – When creating a machine learning model, the approach picks this feature. This is used to filter away characteristics that have no bearing on the output's dependent attribute or prediction.

4. Recursive Feature Elimination – This strategy uses a recursive greedy optimization approach, in which features or characteristics are chosen by iteratively selecting the smallest possible subset of features.

5. Random Forest Importance - The feature importance (variable importance) describes which features are relevant.

1.4.2 Simple Algorithms

An algorithm refers to the systematic procedures used to execute a certain set of tasks on input data obtained from a dataset, with the aim of predicting the corresponding output values. This section provides an explanation of the functioning of machine learning algorithms and ensemble algorithms.

Following are the machine learning algorithms:

1. K-Nearest Neighbor (KNN): - K-Nearest Neighbor is a key technique in supervised learning. Following the training process, this algorithm stores the dataset and subsequently uses it to categorize fresh data into a type that closely resembles the new data. This technique can effectively address both regression and classification problem statements. The letter 'K' represents the nearest neighbor count of a novel unidentified variable that must be forecasted or evaluated.

KNN Algorithm steps: -

Step1: - In KNN algorithm first step is load the test data as well as training.

Step2: - In the second step we choose nearest data points of k.

Step3: - Following are the points of test data.

3.1: -We calculate the distance between each row of training data and test data with the use of method is Euclidean.

3.2: - Then we sort in ascending order based on value of distance.

3.3: -From sorted array ten we will choose top k rows.

3.4: -Based on most frequent class of these rows then we will assign a class to test point.

Step4: -End with performance evaluation.

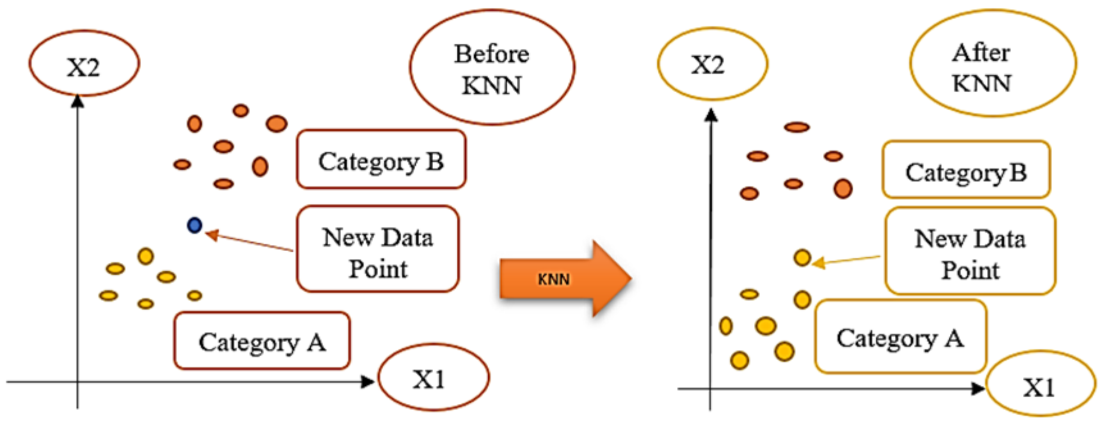


Figure 6: Working of K-Nearest (KNN)

2. Support Vector Machine (SVM): This is an agnostic clustering method that does not make any assumptions about the quantity or structure of the clusters in the data. This method is effective for small datasets. However, if your dataset is vast, such as when dealing with a complex problem, it is recommended to use principal components analysis.

3. Logistic Regression (LR): - It is a machine learning algorithm used for solving classification problems, and it is a predictive analytic tool that relies on the concept of probability. This is a widely recognized approach to supervised learning and is considered one of the most frequently used algorithms in machine learning. A set of

independent variables is used to predict the category dependent variable. This method forecasts the result of a categorical variable that is reliant on other factors.

Logistic Regression Algorithm Steps: -

Step1: -In this step we perform data pre-processing.

Step2: -In this step we do training set to fitting logistic regression.

Step3: - In this step we are predicting the result of the test.

Step4: -In this step we find the result of test accuracy.

Step5: -In this we visualize the result of test set.

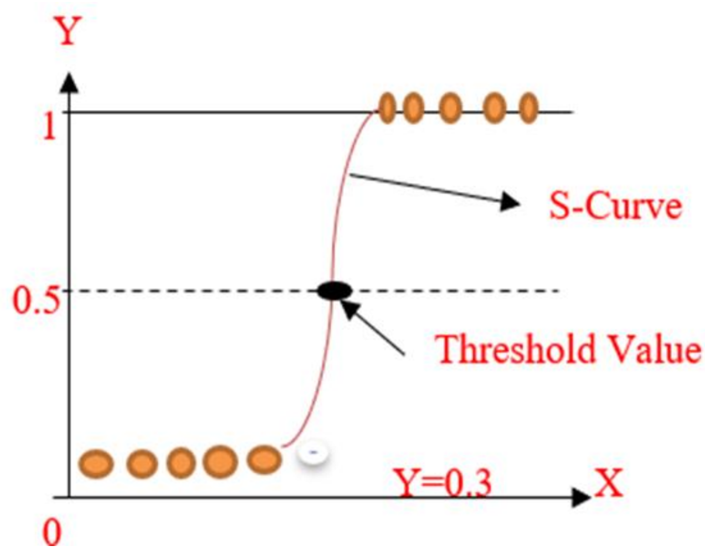


Figure 7: Working of Logistic Regression (LR)

4. Decision Tree (DT): - This approach has various applications, including classification and regression. Instead of using a decision tree, this approach utilizes a model that forecasts the value of an objective variable by considering the leaf nodes associated with class labels and the attributes they represent on the innermost node of the tree.

Decision Tree Algorithm steps: -

Step1: - In the first step start tree with the node of root says S, which contain the complete dataset.

Step2: - In the second step we find the best attribute in the dataset using attribute selection or extraction measure.

Step3: -In the third step we divide S into small subsets that have the best features or attributes for possible values.

Step4: -In this step we contain the best attributes then generate the decision tree node.

Step5: -With the help of sub-sets of the dataset created in step 3 then recursively make fresh decision.

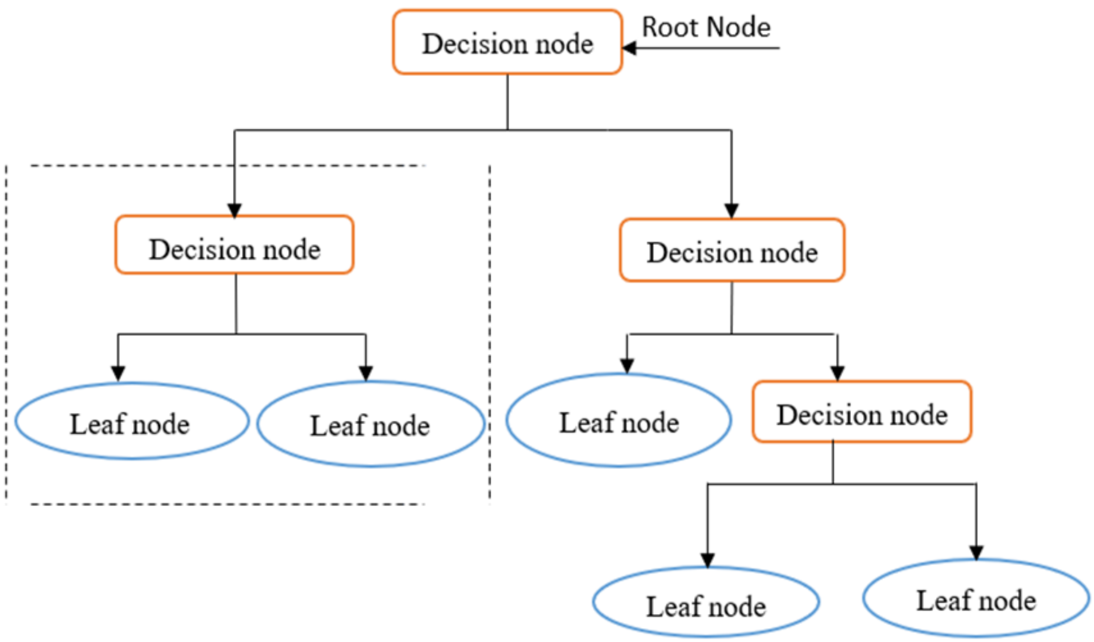


Figure 8: Working of Decision Tree (DT)

1.4.3 Ensemble Algorithms

Following are the ensemble algorithms:

1. Random Forest (RF): This is a classifier that uses decision trees on many subsets of the dataset to enhance the predicted accuracy of the data set. The random forest does not predict the output of at least one decision tree individually. Instead, it makes predictions by combining the forecasts from each tree and selecting the final prediction based on majority voting. Ensemble learning is used to do classification, regression, and other problems that involve a larger number of decision trees. In a random forest, the majority of trees make a decision on classification. The random forest is a classification system that comprises many decision trees.

Random Forest (RF) algorithm steps: -

Step1: -In this step from given dataset begin with the selection of random samples.

Step2: -Decide a decision tree. Then it will get prediction as result.

Step3: -For each predicted result as output, voting will be done.

Step4: -Decide final prediction outcomes as the highest voted prediction result.

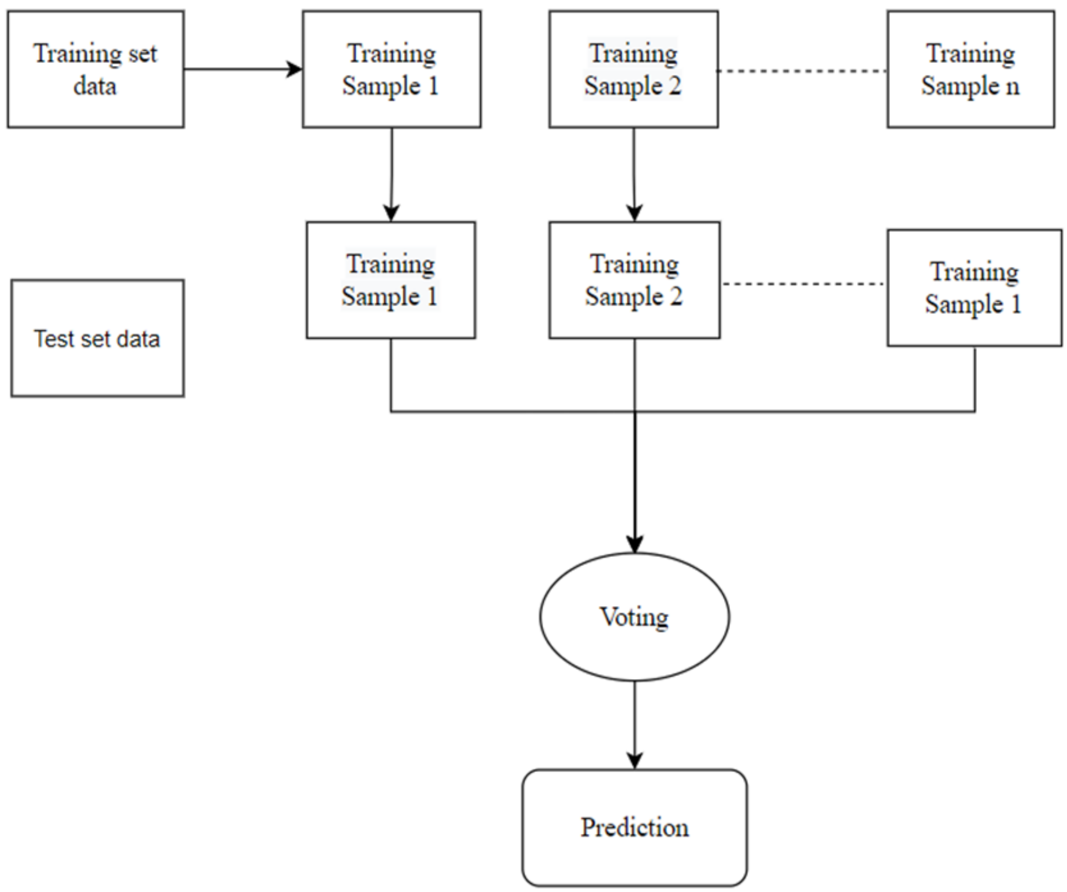


Figure 9: Working of random forest (RF)

2. Ada Bosting (AB): - The machine learning technique called Ada-Boost, often referred to as Adaptive Boosting, is commonly classified as an Ensemble Method. A commonly

used technique in AdaBoost uses a decision tree with a single level, indicating that it only has one branch. The term used to refer to these trees is Decision Stumps. Ada-Boost is the preferred approach for enhancing the performance of machine learning algorithms. Most efficient when instructing children with lower academic abilities. These models exhibit a classification accuracy slightly higher than random chance. AdaBoost is particularly well-suited for one-level decision trees, making them the most frequently used approach.

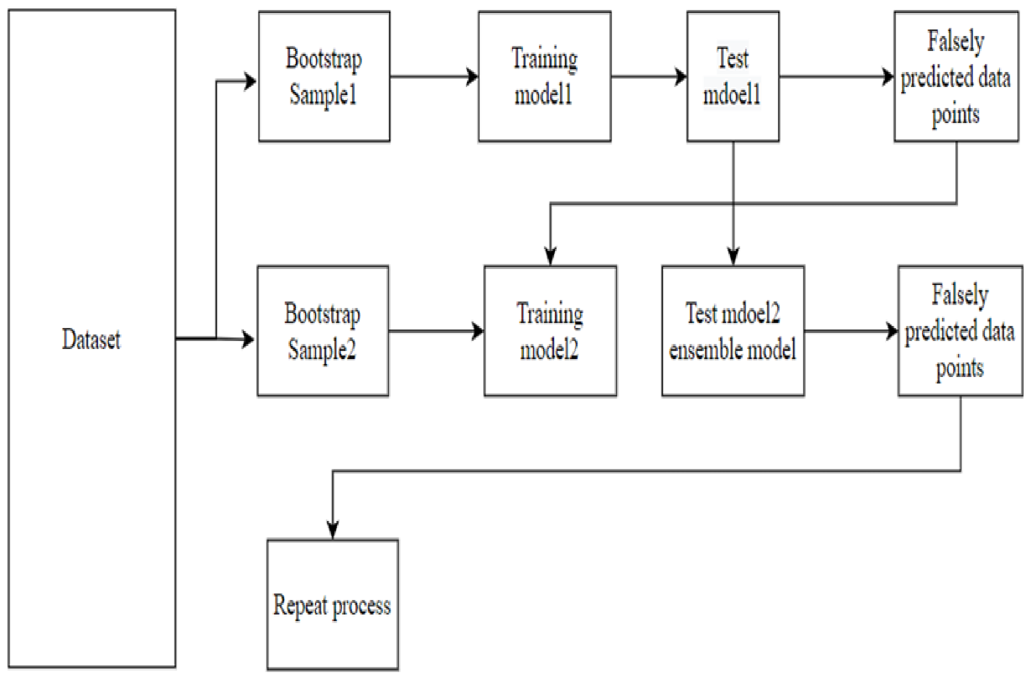


Figure 10: Working of Boosting (AB)

4. Bagging (BAG): - Bagging, also known as Bootstrap aggregation, is an ensemble machine learning technique that aims to enhance the accuracy and performance of machine learning algorithms. It is applied to handle bias-variance trade-offs and helps to minimize the variance of a prediction model. Bagging is a technique used to prevent overfitting in regression and classification models, as well as decision tree approaches.

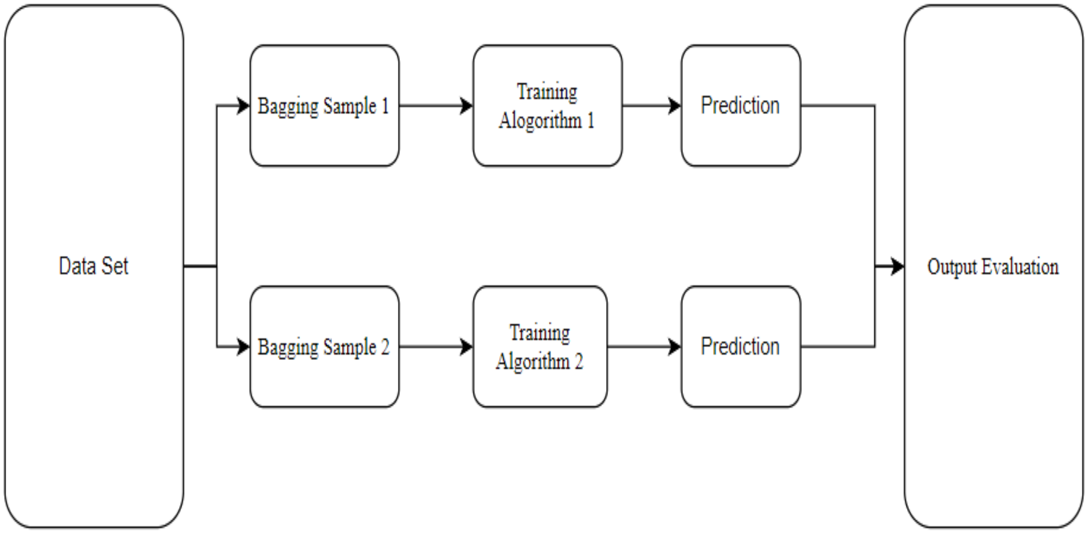


Figure 11: Working of Bagging (BAG)

1.5 Functional Requirements:

1.5.1 Data Input:

The input of codebase datasets, which comprise a range of structures, should be supported by the system. The model ought to be capable of managing a range of codebase dataset sizes, encompassing small-scale projects and large-scale applications.

1.5.2 Smell Detection:

The system should have a ability to detect a range of code smells but not limited to:

- Duplicate Code
- Long method/functions
- Large classes
- Complex Conditions
- Feature Envy
- God Classes

1.5.3 Scalability:

The system ought to be able to effectively manage a big codebase. It ought to provide parallel processing to enable quicker code analysis.

1.5.4 Machine Learning Model:

It details machine learning algorithms or algorithms that are used to detect code smells. The system ought to enable the model to be started with fresh datasets in order to gradually increase its accuracy.

1.6 Non - Functional Requirements:

1.6.1 Performance:

Even for huge codebases, the code smell detection model's response time should fall within a certain range, and it should process and analyze large codebases effectively.

1.6.2 Accuracy:

The system needs to be proficient in accurately identifying various kinds of code smells. It should obtain detection results with a low false positive and false negative rate.

1.6.3 Usability:

Provide documentation that explains the model's operation and the correct fundamentals of dataset creation for the training model.

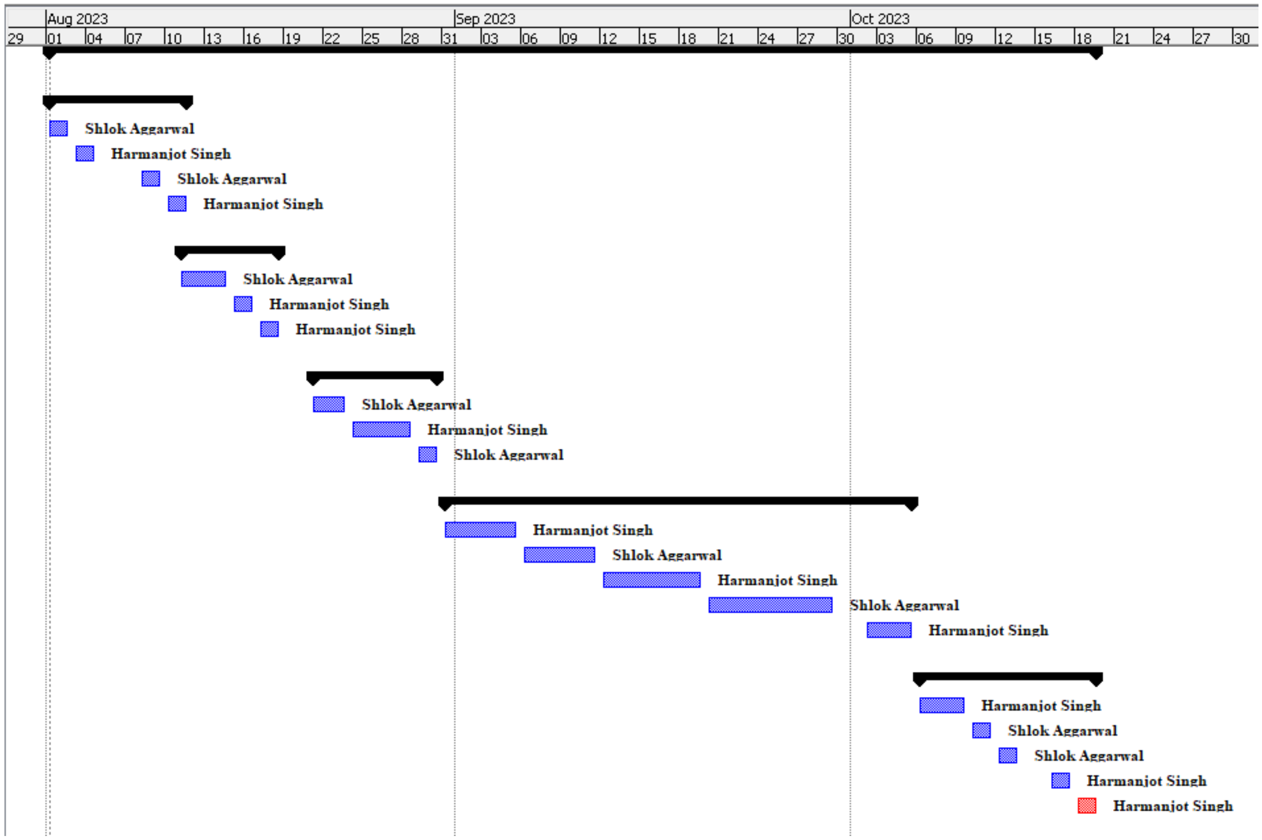
1.6.4 Resource Utilization:

Memory and processing power should be used as efficiently as possible to ensure that the processes don't waste any resources.

1.7 Work Breakdown Structure:

	Name	Duration	Start	Finish	Predecessors	Resource Names
1	**Code Smell Detection**	**58 days?**	**8/1/23 8:00 AM**	**10/19/23 5:00 PM**		
2	**Planning**	**9 days?**	**8/1/23 8:00 AM**	**8/11/23 5:00 PM**		
3	Estimate Size	2 days?	8/1/23 8:00 AM	8/2/23 5:00 PM		ShlokAggarwal
4	Estimate Time	2 days?	8/3/23 8:00 AM	8/4/23 5:00 PM		Harmanjot Singh
5	Estimate Cost	2 days?	8/8/23 8:00 AM	8/9/23 5:00 PM		ShlokAggarwal
6	Estimate Effort	2 days?	8/10/23 8:00 AM	8/11/23 5:00 PM		Harmanjot Singh
7	**Requirements Gather...**	**6 days?**	**8/11/23 8:00 AM**	**8/18/23 5:00 PM**		
8	FunctionalRequirements	2 days?	8/11/23 8:00 AM	8/14/23 5:00 PM		ShlokAggarwal
9	Data Requirements	2 days?	8/15/23 8:00 AM	8/16/23 5:00 PM		Harmanjot Singh
10	Gathering Pojects Details	2 days?	8/17/23 8:00 AM	8/18/23 5:00 PM		Harmanjot Singh
11	**Design**	**8 days?**	**8/21/23 8:00 AM**	**8/30/23 5:00 PM**		
12	Model Selection	3 days?	8/21/23 8:00 AM	8/23/23 5:00 PM		ShlokAggarwal
13	Data Schema	3 days?	8/24/23 8:00 AM	8/28/23 5:00 PM		Harmanjot Singh
14	Language Selection	2 days?	8/29/23 8:00 AM	8/30/23 5:00 PM		ShlokAggarwal
15	**Coding**	**26 days?**	**8/31/23 8:00 AM**	**10/5/23 5:00 PM**		
16	Data Collection	4 days?	8/31/23 8:00 AM	9/5/23 5:00 PM		Harmanjot Singh
17	Feature Extraction	4 days?	9/6/23 8:00 AM	9/11/23 5:00 PM		ShlokAggarwal
18	Data Preprocessing	6 days?	9/12/23 8:00 AM	9/19/23 5:00 PM		Harmanjot Singh
19	ModelTraining	8 days?	9/20/23 8:00 AM	9/29/23 5:00 PM		ShlokAggarwal
20	Post-Processing	4 days?	10/2/23 8:00 AM	10/5/23 5:00 PM		Harmanjot Singh
21	**Testing**	**10 days?**	**10/6/23 8:00 AM**	**10/19/23 5:00 PM**		
22	Data Collection	2 days?	10/6/23 8:00 AM	10/9/23 5:00 PM		Harmanjot Singh
23	Feature Extraction	2 days?	10/10/23 8:00 AM	10/11/23 5:00 PM		ShlokAggarwal
24	Data Preprocessing	2 days?	10/12/23 8:00 AM	10/13/23 5:00 PM		ShlokAggarwal
25	ModelTraining	2 days?	10/16/23 8:00 AM	10/17/23 5:00 PM		Harmanjot Singh
26	Post-Processing	2 days?	10/18/23 8:00 AM	10/19/23 5:00 PM		Harmanjot Singh

Code Smell - page1



Aug 2023
Sep 2023
Oct 2023
29
01
04
07
10
13
16
19
22
25
28
31
03
06
09
12
15
18
21
24
27
30
03
06
09
12
15
18
21
24
27
30
Shlok Aggarwal
Harmanjot Singh
Shlok Aggarwal
Harmanjot Singh
Shlok Aggarwal
Harmanjot Singh
Harmanjot Singh
Shlok Aggarwal
Harmanjot Singh
Shlok Aggarwal
Harmanjot Singh
Shlok Aggarwal
Harmanjot Singh
Shlok Aggarwal
Harmanjot Singh
Harmanjot Singh
Shlok Aggarwal
Shlok Aggarwal
Harmanjot Singh
Harmanjot Singh

1.8 Cost Analysis:

Given that all of the software utilized is publicly available and open-source, an overall cost study is not considered required.

The minimum system requirements for Matplotlib and Code Smell Tools are as follows:

Matplot:

- Matplotlib runs well on Windows or Linux(minimum hardware requirements Intel Core i5 or equivalent , RAM: 4 GB,Disk Space: 15 GB)
- For setting up Matplot Library ,need Python version at least 3.6 or later , Matplotlib can be used with both 32-bit and 64-bit versions of Python

Programming Mistake Detector (PMD):

- To run Eclipse-pmd need Eclipse 2022-12 or later Java 17 or later
- Eclipse system requirements Memory: 4 GB Graphics Card: NVIDIA GeForce GTX 680,CPU: Intel Core i5-3570,File Size: 30 GB
- he PMD plugin for Eclipse can use up to 12 GB of RAM

CheckStyle: an open source development tool

Jedodrant:

- Install the Plug-in Development Kit Eclipse plug-in, you need,Java SE 6 or later,Eclipse version 3.6.2 or late
- The minimum memory requirement for Jedodrant is 4 GB of RAM

2. Literature Survey

In this chapter, a summary of research work on various machine learning and ensemble learning approaches has been discussed.

Sr.no	Paper	Description	Advantages	Disadvantages
1	Caram, Frederico Luiz, et al. "Machine learning techniques for code smells detection: a systematic mapping study." *International Journal of Software Engineering and Knowledge Engineering* 29.02 (2019): 285-316. [1]	The study suggests a machine learning model for identifying code smell found in the literature, with the goal of identifying the approaches and procedures that are applied when using machine learning to identify code smells and the machine learning techniques that have been identification of code smells.	Boost the detection of code smell accuracy. Comparison with conventional rule-based techniques and empirical data indicating increased recall and precision. Scalability for big codebases.	Limited application of machine learning models may result from their poor generalization abilities across various codebases. In code smell datasets, class imbalance can lead to biased models and lower performance on minority classes.

2	Kokol, Peter, Marko Kokol, and Sašo Zagoranski. "Code smells: A synthetic narrative review." *arXiv preprint arXiv:2103.01088* (2021).[2]	This study triangulates the production of code smell literature using bibliometric and thematic analysis. 442 publications were found when the search term "code smells" was used in the Scopus (Elsevier, Netherlands) database. The	When developers recognize code smells as useful cues and educational opportunities, they may use them to improve code quality, foster teamwork, and increase the overall success of software projects.	Examination of how code smells can make it more difficult to read and edit code, which can obstruct software maintenance. Diminished ReadabilityTalk about how code smells can make code harder to read and make it harder to collaborate and share information. Increased Propensity to Bug examination of the relationship between the probability of adding bugs to the codebase and code smells.

3	Hadj-Kacem, Mouna, and Nadia Bouassida. "A Hybrid Approach To Detect Code Smells using Deep Learning." *ENASE*. 2018.[3]	This research is centered on reframing the software and improving it based on accuracy outcomes. This difficult research topic has drawn more attention thus far because of its significant impact on software maintenance.	The hybrid technique presents a valuable addition to the field of software quality assurance due to its synergistic advantages, which include enhanced accuracy, adaptability, decreased false positives, efficient handling of huge codebases, and connection with CI/CD pipelines.	Complex pattern and representation learning from data is a strong suit for deep learning models. In comparison to conventional techniques, the hybrid methodology can achieve higher recall and precision by integrating these models into the code smell detection process. As a result, there are fewer false positives and false negatives when identifying code smells.
4	Arcelli Fontana, Francesca, et al. "Comparing and experimenting machine learning techniques for code	This research evaluated sixteen machine-learning algorithms	The identification of J48 and Random Forest as top-performing	The study focuses on a predefined set of four code smells, potentially limiting the generalizability of

	smell detection." *Empirical Software Engineering* 21 (2016): 1143-1191.[4]	across seventy-four software systems, targeting four specific code smells. The study emphasizes the challenge of imbalanced data in the dataset and concludes that machine learning approaches can provide high accuracy requiring only a modest number of training examples.	algorithms provides practical guidance for practitioners seeking effective tools for code quality improvement. Moreover, the conclusion that high accuracy (>96%) can be achieved with a limited number of training examples contributes practical and resource-efficient recommendations for real-world implementations.	its findings to a broader spectrum of code issues. The evaluation is based on a dataset of seventy-four software systems, and the paper does not extensively discuss the diversity of these systems. A more in-depth examination of the characteristics of the evaluated systems could strengthen the external validity of the study.

5	Tiwari, Omkarendra, and Rushikesh K. Joshi. "Functionality based code smell detection and severity classification." *Proceedings of the 13th innovations in software engineering conference on formerly known as India software engineering conference*. 2020. [5]	This paper primarily addresses long method code smell; it encompasses all functionalities-based techniques. It groups of statements into possibilities for extract methods (or tasks).	The unique context in which code smells arise is taken into consideration by functionality-based code smell detection. This contextual awareness makes it possible to determine more precisely if a given code construct is a purposeful design decision or a real problem.	the drawbacks of studies on severity categorization and functionality-based code smell detection. Researchers and practitioners can better traverse the difficulties and work toward improving the efficacy of code smell detection and severity evaluation techniques by being aware of these shortcomings.
6	Kaur, Amandeep, Sushma Jain, and Shivani Goel. "SP-J48: a novel optimization and machine-learning-based approach for solving complex problems: special	The paper introduces a novel hybrid algorithm, SP-J48, that combines the Sandpiper Optimization Algorithm	This paper demonstrates an innovative approach to problem-solving, particularly in the domain of software	The paper primarily focuses on the proposed algorithm's performance without extensively discussing potential

	application in software engineering for detecting code smells." *Neural Computing and Applications* 32 (2020): 7009-7027. [6]	(SPOA) with the B-J48 pruned machine-learning approach for efficiently detecting code smells. The study highlights the limitations of traditional numerical methods and advocates for metaheuristic optimization algorithms, particularly population-based ones.	engineering for detecting code smells. The comparative evaluation against nine other optimization algorithms underscores the superior performance of SP-J48 in addressing complex problems.	drawbacks or challenges associated with its implementation. The specific application of the algorithm in detecting code smells might raise questions about its adaptability to different problem domains. Further research and empirical validation across diverse datasets and problem scenarios are needed to comprehensively assess the generalizability and robustness of SP-J48.

7	Paiva, Thanis, et al. "On the evaluation of code smells and detection tools." *Journal of Software Engineering Research and Development* 5.1 (2017): 1-28. [7]	The environment of code smells and the instruments used to identify them in software development are critically examined in this paper. The authors explore the difficulties of recognizing and resolving poor programming techniques that may result in software maintenance issues, with an emphasis on evaluating the efficacy of various code scent detection technologies.	By thoroughly evaluating the advantages and disadvantages of the current code scent detection methods, the essay on the evaluation of code smells and detection techniques provides insightful information on the subject of software engineering.	A significant drawback of the article is its exclusive emphasis on particular programming languages or tools, which may result in the neglect of the wider field of varied languages and tools employed in software development. The article might not sufficiently discuss how its conclusions can be applied to a variety of development settings.

8	Al-Shaaby, Ahmed, Hamoud Aljamaan, and Mohammad Alshayeb. "Bad smell detection using machine learning techniques: a systematic literature review." *Arabian Journal for Science and Engineering* 45 (2020): 2341-2369. [8]	It reviews and analyzes studies that employ machine learning techniques to detect code smells in software. The research explores the utilization of sixteen machine learning algorithms, revealing God Class and Long Method, Feature Envy, and Data Class as frequently detected code smells.	It provides a comprehensive overview of the current state of utilizing machine learning for code smell detection by analyzing seventeen primary studies. The emphasis on the negative impact of code smells on software quality underscores the significance of employing automated tools, with machine learning emerging as a promising solution. The findings, such as the prominence of support vector	The identified studies might not cover the entire landscape of research in this domain, potentially leading to a limited representation of machine learning techniques for code smell detection. Additionally, the paper acknowledges the need for more research in specific areas, such as ensemble learning techniques and multiclassificatio n, highlighting potential gaps in the current understanding of applying machine learning to code smell detection.

			machine techniques and the superior performance of J48 and Random Forest algorithms, offer practical guidance for future research directions.	
9	Di Nucci, Dario, et al. "Detecting code smells using machine learning techniques: are we there yet?." *2018 ieee 25th international conference on software analysis, evolution, and reengineering (saner)*. IEEE, 2018. [9]	This study questions the generalizability of the original findings, particularly due to limitations in dataset construction. The paper highlights concern about the imbalance in instances affected by code smells and non-smelly instances, along with a	It provides a critical examination of the application of machine learning (ML) techniques in code smell detection, particularly scrutinizing a large-scale study by Arcelli Fontana et al. This analysis contributes to a deeper understanding of the limitations and	It primarily focuses on the limitations of a particular study by Arcelli Fontana et al. While this provides valuable insights into the potential issues associated with dataset construction and metric distribution, the narrow focus might limit the broader applicability of the paper's

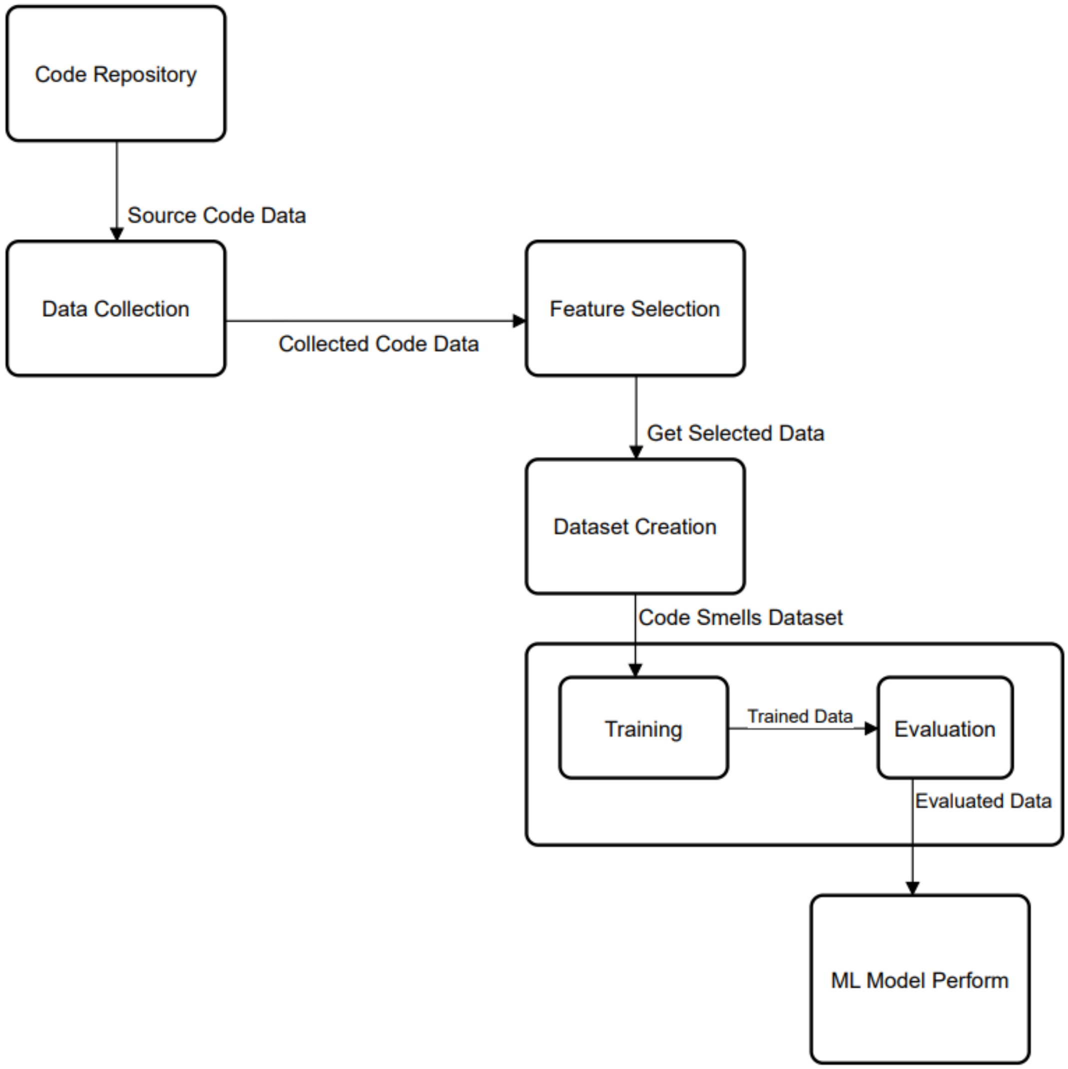
		distinct metric distribution, challenging the claim that the choice of ML algorithm may not significantly impact practical outcomes. Overall, the research emphasizes the need for a more realistic dataset.	challenges associated with existing methodologies, highlighting the need for more realistic datasets and nuanced interpretations of ML algorithm choices. The paper encourages a more thoughtful and context-specific approach to the utilization of ML in code smell detection.	conclusions. Additionally, the paper could benefit from a more extensive exploration of alternative approaches or solutions to address the identified limitations in ML-based code smell detection.
10	Zakeri-Nasrabadi, Morteza, et al. "A systematic literature review on the code smells datasets and validation mechanisms." *ACM Journal on Computing and Cultural*	It provides a thorough analysis of previous studies on code smells. It specifically focuses on the datasets and validation	This approach guarantees a thorough and organized analysis of the existing information, allowing academics and practitioners to	An inherent drawback could arise from relying on the existing literature, potentially leading to publication bias, particularly if there is

	Heritage (2023). [10]	mechanisms used to identify and analyze these issues related to code quality. The systematic literature study examines scholarly works to methodically collect and combine information on datasets used for code smell detection and the validation processes used to evaluate the efficiency of these detection approaches.	acquire a comprehensive understanding of the present state in this field. Additionally, it offers insights into potential gaps and opportunities for future investigations. This paper is an important resource for academics, professionals, and developers who want to improve their understanding of code smells and their validation. It aims to contribute to better software quality and maintainability.	insufficient study on specific parts of code smells datasets and validation techniques. Furthermore, the criteria established for the review may unintentionally omit pertinent studies that utilize distinct terminologies or methodology.

11	Bamizadeh, Lida, et al. "An Analytical Study of Code Smells." *Tehnički glasnik* 15.1 (2021): 121-126. [11]	This research examines typical code smells and their effect on software quality through a thorough analysis. The paper offers significant insights into how these code smells might potentially undermine the maintainability, extensibility, and overall resilience of software, based on practical evidence and case studies.	The research paper provides considerable benefits by undertaking a thorough examination of code smells, making a substantial contribution to the field of software engineering. The study methodically examines and classifies different forms of code smells, offering a complete framework for comprehending and detecting potential problems in software source code.	An inherent drawback exists in the applicability of the results. The study's emphasis on particular code smells or programming languages may limit the generalizability of its findings to a wider software development setting. Furthermore, the dynamic nature of software development methods and technology may make the findings of the study susceptible to being outdated in the future.

12	Pereira dos Reis, José, et al. "Code smells detection and visualization: a systematic literature review." *Archives of Computational Methods in Engineering* 29.1 (2022): 47-94. [12]	The article offers an in-depth review of current research and advancements in the domain of detecting and visually representing code smells in software systems. It rigorously evaluates different strategies, techniques, and tools used to identify code smells, providing insights into their strengths, limits, and relative effectiveness.	The study paper provides a thorough examination of the current body of literature on the detection and visualization of code smells. The paper presents a thorough and systematic analysis that brings together many methodologies, tools, and strategies used to identify and visualize code smells across time.	An inherent drawback exists in the reliance on the quality and extent of accessible literature. If there is a scarcity of research or a lack of consensus in specific domains of code smells detection and visualization, the systematic review may be limited in its ability to offer a thorough overview.

3. Purpose of Study



Block Diagram of Code Smell Detection on Machine Learning Model

Code Repository**:**

Represents the source code repository where the codebase is stored.

Data Collection**:**

Entails taking pertinent code metrics and features out of the source. Code complexity, code duplication, and other code smell indications could be included in this.

Feature Selection:

It takes significant characteristics out of the gathered information. In this stage, selection of features gather from the dataset to implement further processing on machine learning models.

Dataset Creation**:**

Assembles the dataset by fusing identified code smell occurrences with the features that were extracted. Next, the dataset is divided into testing and training sets.

Machine Learning Model:

It Trains a model on the labeled dataset using a machine learning algorithm. The model gains the ability to recognize patterns linked to code smells.

Training:

Using the training dataset, the model's parameters are tuned to reduce prediction errors and enhance its detection of code smells.

Evaluation:

The testing dataset is used to evaluate the trained model's performance in identifying code smells. This stage aids in confirming the efficacy of the mode

Objective

The objectives of our thesis are as follows:

1. To study and explore existing literature and available softwares in the field of code smell detection and prediction.
2. To design and develop machine learning algorithms to detect and characterize various types of code smells.
3. To verify and validate the proposed algorithm.

4. Implementation & Results

In this chapter, we have mentioned the Hardware and software requirements required to perform this research. Also, elaborated about the implementation of feature selection techniques as well as the implementation of the ML and ensemble algorithms on the best feature selected and we have validated and evaluated the output for accurate comparative analysis of algorithms.

4.1 Tools Used

4.1.1 Hardware requirements

- AMD based processor
- RAM
- SD

Hardware	Specifications
CPU	AMD A9-9420 RADEON R5, 5 COMPUTE CORES 2C +3G 3.00 GHz
RAM	4.0 GB (Gigabyte))
ROM	1TB (Terra Byte)

Table 1: Hardware Specifications.

4.1.2 Software Requirement

1. Jupyter Notebook: - In this website application, data scientists may exchange and create papers that incorporate live code, calculation output, equations, and visualizations as well as other multimedia resources and explanatory text all into single document

2. Python: - Python 3.11.1 Version used for these experiments and results.

Software Tools	**Specification**
Operating System	Windows 10 Pro
IDE	Python 3.11.1, Juypter Notebook

Programming Languages	Python
Dataset	Excel

4.2 Implementation

This section has the information about the dataset and preprocessing steps as well the information about the implementation.

4.2.1 Dataset

The dimension of the dataset is 49738 rows and 9 columns. Data set contains different attributes of Code Smells. Dataset was not normalized and required preprocessing as mentioned in next steps so that models are bias towards one class or algorithms.

The data set contains different parameters as mentioned below.

1. File - Name of the java class.
2. Priority – Priority level ranging f=from 1 to 5, 1 being highest priority and 5 being the lowest.
3. Line – Line number at which code smell is detected.
4. Rule Set – Heading of rule.
5. Rule – Sub rules under a particular rule Set.
6. Rule Condition – Description of rule.
7. Description - Suggestion on how to remove that particular code smell.

Sr. No.	Column Name	Non-Null Count	Data type
0	Problem	49748 non-null	object
1	Package	31472 non-null	object
2	File	49731 non-null	object
3	Priority	49731 non-null	float64
4	Line	49731 non-null	float64

5	Rule Set	49731 non-null	object
6	Rule	49731 non-null	object
7	Rule Condition	49268 non-null	object
8	Description	49731 non-null	object

Table 2: Description of data set attributes

4.2.2 Preprocessing

Dataset is imported in the juypter noted book of python environment for analysis. We have performed normalization on the imported dataset to remove the null values. Feature transformation is performed as machine learning models need all the required information passed as an input to be in numerical.

In this research Most of work on the dataset that has labeled one, or more columns and it can be in strings or numbers. Data is made understandable by labeling the training dataset into strings or words. Label Encoding is performed to transform other than numerical values into numerical labels. If labels are numeric, it is easy for machine learning models to decide how those labels must be operated. It is a very important step of pre-processing for the dataset that is structured in super-vised learning.

Following are the features used from data set for feature selection

	Problem	Package	File	Priority	Line	Rule set	Rule	Rule Condition	Description
0	1	(default)	src/Admin_Recruiter.java	1.0	20.0	Code Style	ClassNamingConventions	Configurable naming conventions for type decla...	The class name 'Admin_Recruiter' doesn't match...
1	2	(default)	src/Admin_Recruiter.java	3.0	20.0	Error Prone	MissingSerialVersionUID	Serializable classes should provide a serialVe...	Classes implementing Serializable should set a...
2	3	(default)	src/Admin_Recruiter.java	3.0	20.0	Code Style	NoPackage	Detects when a class, interface, enum or annot...	All classes, interfaces, enums and annotations...
3	4	(default)	src/Admin_Recruiter.java	3.0	22.0	Documentation	CommentRequired	Denotes whether javadoc (formal) comments are ...	Field comments are required

Figure: Reformatted data set

[10]

```
refined_data = data.dropna(subset=['priority', 'rule_set', 'rule','rule_condition','description'])
```

+ Code + Markdown

```
print('Number of Null values in Columns')
refined_data.isnull().sum()
```

✓ 0.0s

```
Number of Null values in Columns

problem           0
priority          0
rule_set          0
rule              0
rule_condition    0
description       0
dtype: int64
```

4.2.3 Feature Transformation

Feature Transformation is performed since Machine Learning model need all information passed as input to be in numerical form

Implementation of Label encoder to encoding categorical labels with numerical values are mentioned below to achieve this.

```
from sklearn.preprocessing import LabelEncoder
e=LabelEncoder()
```

✓ 0.0s

```
len(df1.columns)
```

✓ 0.0s

```
9
```

```
a=df1.columns.to_list()
a
```

✓ 0.0s

```
['Problem',
 'Package',
 'File',
 'Priority',
 'Line',
 'Rule set',
 'Rule',
 'Rule Condition',
 'Description']
```

```
for i in a:
    df1[i]=e.fit_transform(df1[i])
```

✓ 0.2s

```
df1
```

✓ 0.0s

	Problem	Package	File	Priority	Line	Rule set	Rule	Rule Condition	Description
0	0	0	284	0	19	1	32	48	4507
1	1	0	284	2	19	4	95	129	3966
2	2	0	284	2	19	1	104	56	3920
3	3	0	284	2	21	3	38	54	3919
4	4	0	284	2	21	2	127	147	2590
...	...	...	...	...	...	...	...	...	...
49733	16773	22	159	0	22	1	92	47	3475
49734	16774	16	166	3	4	1	145	123	3899
49735	16775	16	166	2	11	1	7	183	3988
49736	16776	16	166	2	13	3	38	54	2801
49737	16777	16	166	0	13	1	92	47	2811

49738 rows × 9 columns

4.2.4 Feature Selection

Attribute selection is performed to make sure that only the most important feature associated to Covid-19 dataset has been passed to machine learning models. This technique involves choosing the best features to get better classification results.

1.Selecting 3 attributes to apply feature selection technique are mentioned below.

```
df=df1.drop(["File","Package","Line"],axis=1)
```
✓ 0.0s

```
df
```
✓ 0.0s

	Problem	Priority	Rule set	Rule	Rule Condition	Description
0	0	0	1	32	48	4507
1	1	2	4	95	129	3966
2	2	2	1	104	56	3920
3	3	2	3	38	54	3919
4	4	2	2	127	147	2590
...	...	...	...	...	...	...
49733	16773	0	1	92	47	3475
49734	16774	3	1	145	123	3899
49735	16775	2	1	7	183	3988
49736	16776	2	3	38	54	2801
49737	16777	0	1	92	47	2811

49738 rows × 6 columns

2. Below is the implementation of Splitting the dataset into features and target variables

```
# Split dataset into features and target variables
feature_cols=["Problem","Priority","Rule set","Rule","Rule Condition"]
x=df3[feature_cols]
y=df3["Description"]
```
✓ 0.0s

+ Code + Markdown

4.3 Splitting Data & Testing Simple Models

To assess the performance of a model on unseen data, below is an implementation of simple models.

1. Splitting data for training and testing machine learning models.

```
from sklearn.model_selection import train_test_split , GridSearchCV
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state=123)
```
[132] ✓ 0.0s

Below is the implementation of simple algorithms on best features.

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn import tree
```
[133] ✓ 0.0s

```
from sklearn.ensemble import RandomForestClassifier
RFclassifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")

RFclassifier.fit(x_train, y_train)
y_pred3=RFclassifier.predict(x_test)
#accuracy_score(y_test,y_pred)
accuracy=metrics.accuracy_score(y_test, y_pred)*100
print("Random Forest Accuracy:",accuracy)
```
[38]

```
Random Forest Accuracy: 94.85
```

```
X = df3[feature_cols]
y = df3["Description"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=123)
classifiers = [
    SVC()
]

for classifier in classifiers:
    scores = cross_val_score(classifier, X_train, y_train, cv=5)
    print(f"{classifier.__class__.__name__} Accuracy: {scores.mean()*100:.2f}%")
```
[144] ✓ 0.8s

```
SVC Accuracy: 94.51%
```

```
dect = tree.DecisionTreeClassifier()
dect=dect.fit(x_train,y_train)
y_pred = dect.predict(x_test)
accuracy=metrics.accuracy_score(y_test, y_pred)*100
print("Decision Tree Accuracy:",accuracy)
```
[134] ✓ 0.0s

```
Decision Tree Accuracy: 99.99877
```

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

logreg.fit(x_train,y_train)
y_pred1=logreg.predict(x_test)
from sklearn.metrics import accuracy_score,confusion_matrix
y_pred1=logreg.predict(x_test)
accuracy_score(y_test,y_pred)
print("Logistic Regression  Accuracy:",metrics.accuracy_score(y_test, y_pred1)*100)
```
[138] ✓ 0.0s

```
Logistic Regression  Accuracy: 83.6104513064133
```

```
from sklearn.neighbors import KNeighborsClassifier

KNN_classifier= KNeighborsClassifier(n_neighbors=184)

KNN_classifier.fit(x_train, y_train)
y_pred2=KNN_classifier.predict(x_test)
accuracy=metrics.accuracy_score(y_test, y_pred)*100
print("Decision Tree Accuracy:",accuracy)
```
[140] ✓ 0.1s

```
Decision Tree Accuracy: 84.5
```

4.3.1 Testing Ensemble Models

Below is the implementation of Ensemble Learning algorithms on best Selected features.

```
feature_names = [ 'priority', 'rule_set', 'rule', 'rule_condition']
X = refined_data[feature_names]
y = refined_data.description
```
[135] ✓ 0.0s

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=100, test_size=0.3)
print(X_train.shape)
```
[136] ✓ 0.1s

```
(34487, 4)
```

```
from mlxtend.feature_selection import SequentialFeatureSelector as sf
```
[137] ✓ 0.0s

+ Code + Markdown

```
from sklearn.linear_model import LinearRegression
```
[138] ✓ 0.0s

```
lreg=LinearRegression()
```
[139] ✓ 0.0s

```
sf1=sf(lreg,k_features=4,forward=False,n_jobs=-1)
```
[140] ✓ 0.0s

```
sf1.fit(X,y)
```
[141] ✓ 0.9s

```
▸ SequentialFeatureSelector
▸ estimator: LinearRegression
    ▸ LinearRegression
```

```
features=list(sf1.k_feature_names_)
```
[142] ✓ 0.0s

```
print(features)
```
[143] ✓ 0.0s

```
['priority', 'rule_set', 'rule', 'rule_condition']
```

```
sf1
```
[144] ✓ 0.0s

Splitting data for training and testing an Ensemble learning models are mentioned below

```
sf1
```
[144] ✓ 0.0s

```
▸ SequentialFeatureSelector
▸ estimator: LinearRegression
    ▸ LinearRegression
```

```
feature_names = [ 'priority', 'rule_set', 'rule','rule_condition']
X = refined_data[feature_names]
y = refined_data.description
```
[145] ✓ 0.0s

```
X = X.dropna(subset=['priority'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,random_state=1)
```
[146] ✓ 0.0s

```
models = []
models.append(('AdaBoost', AdaBoostClassifier(random_state=1)))
models.append(('Bagging', BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=1), random_state=1)))
```

[148] ✓ 0.0s

```
warnings.filterwarnings("ignore", category=FutureWarning)
names = []
scores = []
models = {
    'AdaBoost': AdaBoostClassifier()
}

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scores = {name: model.fit(X_train, y_train).score(X_test, y_test) * 100 for name, model in models.items()}

tr_split = pd.DataFrame({'Name': list(scores.keys()), 'Score': list(scores.values())}).sort_values(by='Score', ascending=False)
print(tr_split)
models = {

    'Bagging': BaggingClassifier()
}

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scores = {name: model.fit(X_train, y_train).score(X_test, y_test) * 100 for name, model in models.items()}

tr_split = pd.DataFrame({'Name': list(scores.keys()), 'Score': list(scores.values())}).sort_values(by='Score', ascending=False)
print(tr_split)
```

157] ✓ 1.0s

```
       Name    Score
0  AdaBoost  89.7411
      Name   Score
0  Bagging  87.255
```

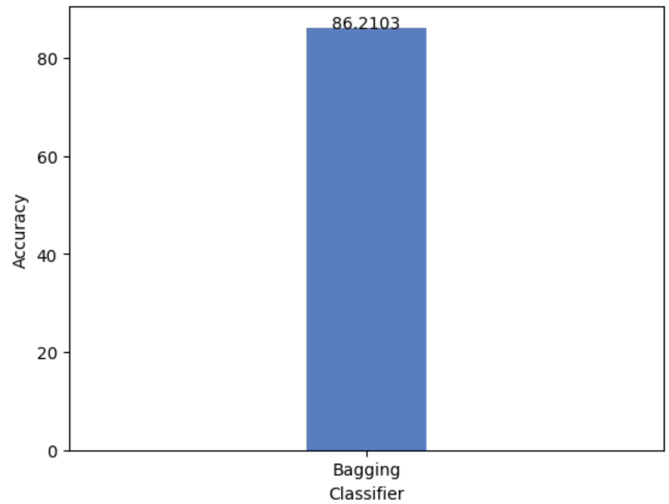
```
import warnings

warnings.filterwarnings("ignore", category=FutureWarning)
from sklearn.model_selection import KFold
names = []
scores = []
models = [('AdaBoost', AdaBoostClassifier())]
names, scores = zip(*[(name, cross_val_score(model, X, y, cv=KFold(n_splits=5, random_state=5,shuffle=True), scoring='accuracy').mean()*100) for name, model
kf_cross_val1 = pd.DataFrame({'Name': names, 'Score': scores})
print(kf_cross_val1)
names1 = []
scores1 = []
models1 = [('Bagging', BaggingClassifier())]
names1, scores1 = zip(*[(name, cross_val_score(model, X, y, cv=KFold(n_splits=5, random_state=5,shuffle=True), scoring='accuracy').mean()*100) for name, mod
kf_cross_val = pd.DataFrame({'Name': names1, 'Score': scores1})
print(kf_cross_val)
```

[158] ✓ 1.7s Pyth

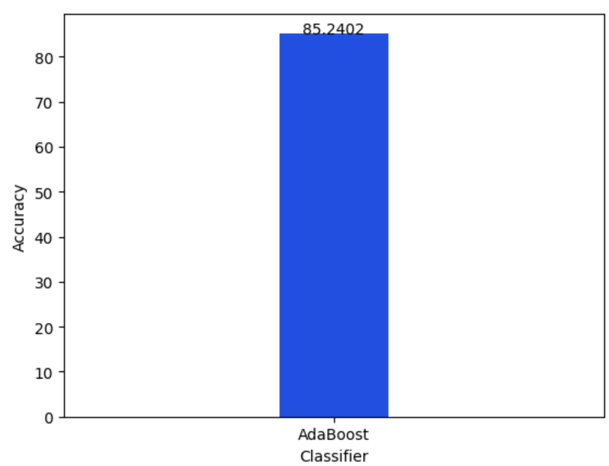
```
       Name    Score
0  AdaBoost  85.2402
      Name    Score
0  Bagging  86.2103
```

Plotting a bar graph on the basis of Ensemble Learning model's accuracy

```
axis = sns.barplot(x = 'Name', y = 'Score', data = kf_cross_val,width = 0.2, palette = 'muted')
axis.set(xlabel='Classifier', ylabel='Accuracy')
for p in axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.005, '{:1.4f}'.format(height), ha="center")
plt.show()
```
[196] 5.8s




```
axis = sns.barplot(x = 'Name', y = 'Score', data = kf_cross_val1,width = 0.2, palette = 'bright')
axis.set(xlabel='Classifier', ylabel='Accuracy')
for p in axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.005, '{:1.4f}'.format(height), ha="center")
plt.show()
```
[197] 0.5s



4.4 Data Visualization

Visualizing some of the attributes from the dataset through a pie chart through the following code.

1. Considering a Rule Set Attribute from the dataset to plot a pie chart

```
def data_to_dict(data):
    desc_dict = {}
    for row in data:
        desc = row[1]
        if desc in desc_dict:
            desc_dict[desc] += 1
        else:
            desc_dict[desc] = 1
    return desc_dict
def pie_chart(data):
    desc_dict = data_to_dict(data)
    labels = list(desc_dict.keys())
    sizes = list(desc_dict.values())
    fig, ax = plt.subplots()
    ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
    ax.axis('equal')
    plt.show()
df1["Rule set"] = df1["Rule set"].astype(str)
pie_chart(df1["Rule set"])
```

[154] ✓ 0.8s

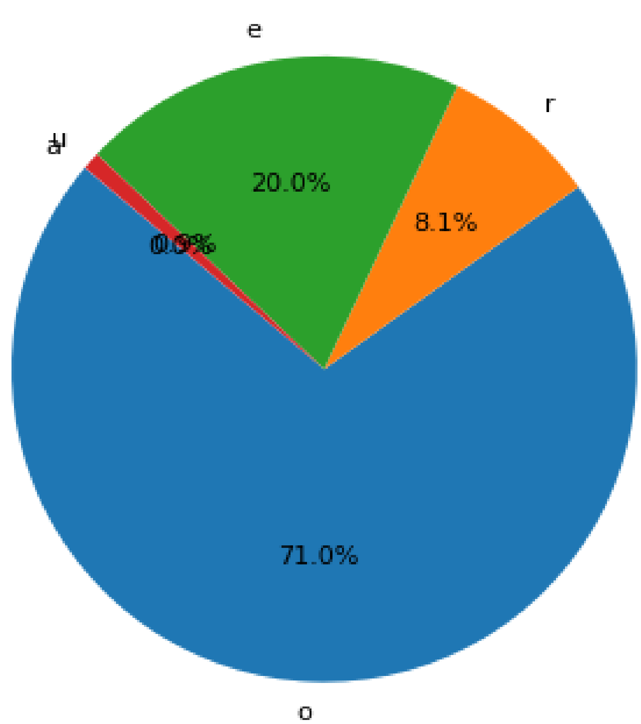


Figure 12: Pie Chart of Rule Set

2. Considering a Description Attribute from the dataset to plot a pie chart mentioned below.

```python
def data_to_dict(data):
    desc_dict = {}
    for row in data:
        desc = row[1]
        if desc in desc_dict:
            desc_dict[desc] += 1
        else:
            desc_dict[desc] = 1
    return desc_dict
def pie_chart(data):
    desc_dict = data_to_dict(data)
    labels = list(desc_dict.keys())
    sizes = list(desc_dict.values())
    fig, ax = plt.subplots()
    ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
    ax.axis('equal')
    plt.show()
df1["Description"] = df1["Description"].astype(str)
pie_chart(df1["Description"])
```

[115] ✓ 1.7s

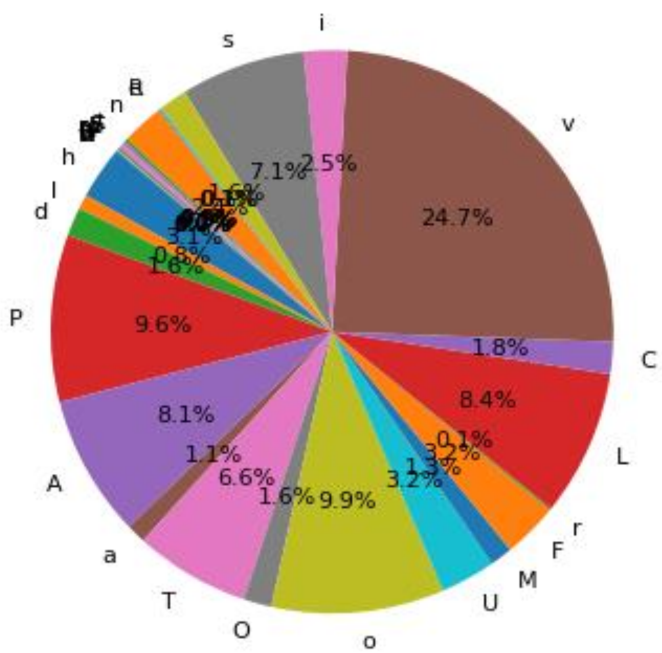


Figure 13: Pie Chart of Description

3. Considering a Rule Attribute from the dataset to plot a pie chart mention below

```python
def data_to_dict(data):
    desc_dict = {}
    for row in data:
        desc = row[1]
        if desc in desc_dict:
            desc_dict[desc] += 1
        else:
            desc_dict[desc] = 1
    return desc_dict
def pie_chart(data):
    desc_dict = data_to_dict(data)
    labels = list(desc_dict.keys())
    sizes = list(desc_dict.values())
    fig, ax = plt.subplots()
    ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
    ax.axis('equal')
    plt.show()
df1["Rule"] = df1["Rule"].astype(str)
pie_chart(df1["Rule"])
```

[116] ✓ 2.3s

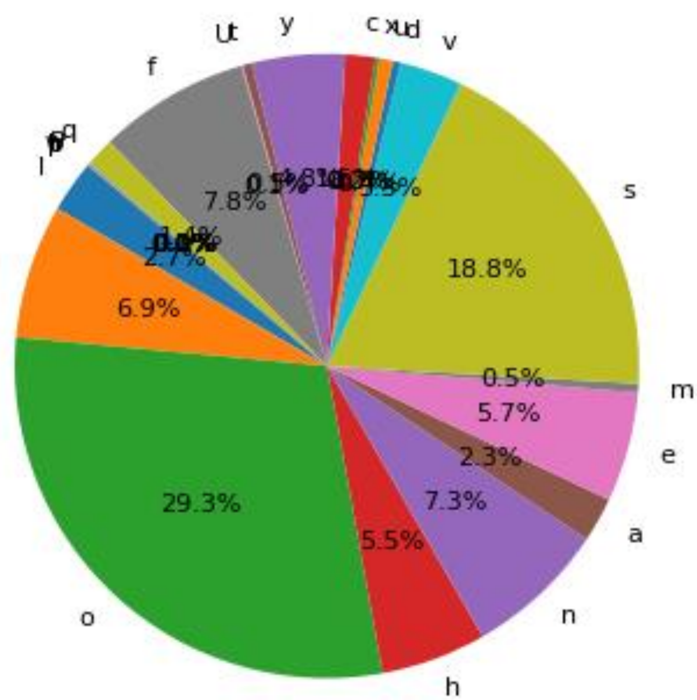


Figure 14: Pie Chart of Rule.

4.5 Algorithm Results

In this we implemented simple and ensemble algorithms to check out accuracy of different models.

4.5.1 Simple Algorithms on Selection Technique

Algorithms	SelectBest	
	Normal	Cross_Validation
	Accuracy	
KNN	**0.84**	**0.87**
SVC	**0.83**	**0.89**
LR	**0.96**	**0.75**
DT	**0.99**	**0.98**
RF	**0.94**	**0.95**

Table 3: Simple Algorithms Results on Feature Selection

4.5.2 Ensemble Algorithms on Selection Technique

.

Algorithms	SelectBest	
	Normal	Cross_Validation
	Accuracy	
Adaboost	**0.89**	**0.85**
Bagging	**0.87**	**0.86**

Table 4: Ensemble Algorithms Results on Feature Selection

5. Conclusion

In conclusion, the project on "Detecting and Characterizing Various Code Smells Using Machine Learning Techniques" has successfully leveraged the power of machine learning to create a comprehensive dataset by utilizing popular code analysis tools such as PMD, JDeodorant, and Checkstyle. By training both a simple model and an ensemble model on this dataset, the project has achieved efficient accuracy in identifying and characterizing various code smells. This achievement not only validates the effectiveness of the chosen machine learning techniques but also underscores the potential for automation in code smell detection. The use of multiple tools and the ensemble approach enhances the robustness of the model, contributing to a more reliable and comprehensive code smell detection system. The findings of this project have significant implications for software developers and maintainers, providing them with a valuable tool to enhance code quality and maintainability through the automated identification of code smells. Overall, the successful execution of this project demonstrates the promising intersection of machine learning and software engineering in addressing crucial aspects of code quality.

6. References

[1] F. L. Caram, B. R. D. O. Rodrigues, A. S. Campanelli, and F. S. Parreiras, "Machine Learning Techniques for Code Smells Detection: A Systematic Mapping Study," *International journal of software engineering and knowledge engineering*, vol. 29, no. 2, pp. 285–316, Feb. 2019, doi: 10.1142/S021819401950013X.

[2] P. Kokol, M. Kokol, and S. Zagoranski, "Code smells: A Synthetic Narrative Review," *Library Philosophy and Practice*, vol. 2020, Mar. 2021, Accessed: Dec. 18, 2023. [Online]. Available: https://arxiv.org/abs/2103.01088v1

[3] M. Hadj-Kacem and N. Bouassida, "A hybrid approach to detect code smells using deep learning," *ENASE 2018 - Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering*, vol. 2018-March, pp. 137–146, 2018, doi: 10.5220/0006709801370146.

[4] F. A. Fontana, M. V Mäntylä, M. Zanoni, and A. Marino, "Comparing and Experimenting Machine Learning Techniques for Code Smell Detection", doi: 10.1007/s10664-015-9378-4.

[5] O. Tiwari and R. K. Joshi, "Functionality Based Code Smell Detection and Severity Classification," *ACM International Conference Proceeding Series*, Feb. 2020, doi: 10.1145/3385032.3385048.

[6] A. Kaur, S. Jain, and S. Goel, "SP-J48: a novel optimization and machine-learning-based approach for solving complex problems: special application in software engineering for detecting code smells," *Neural Comput Appl*, vol. 32, no. 11, pp. 7009–7027, Jun. 2020, doi: 10.1007/S00521-019-04175-Z.

[7] T. Paiva, A. Damasceno, E. Figueiredo, and C. Sant'Anna, "On the evaluation of code smells and detection tools," *Journal of Software*

Engineering Research and Development 2017 5:1, vol. 5, no. 1, pp. 1–28, Oct. 2017, doi: 10.1186/S40411-017-0041-1.

[8] A. Al-Shaaby, · Hamoud Aljamaan, · Mohammad Alshayeb, M. Alshayeb, and H. Aljamaan, "Bad Smell Detection Using Machine Learning Techniques: A Systematic Literature Review," *Arab J Sci Eng*, vol. 45, no. 3, pp. 2341–2369, 2020, doi: 10.1007/s13369-019-04311-w.

[9] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia, "Detecting code smells using machine learning techniques: Are we there yet?," *25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2018 - Proceedings*, vol. 2018-March, pp. 612–621, Apr. 2018, doi: 10.1109/SANER.2018.8330266.

[10] M. Zakeri-Nasrabadi, S. Parsa, E. Esmaili, and F. Palomba, "A Systematic Literature Review on the Code Smells Datasets and Validation Mechanisms," *ACM Comput Surv*, vol. 55, no. 13 s, Jul. 2023, doi: 10.1145/3596908.

[11] L. Bamizadeh, B. Kumar, A. Kumar, and S. Shirwaikar, "An Analytical Study of Code Smells," *Tehnicki Glasnik*, vol. 15, no. 1, pp. 121–126, 2021, doi: 10.31803/TG-20210205095410.

[12] J. P. dos Reis, F. B. e Abreu, G. de F. Carneiro, and C. Anslow, "Code smells detection and visualization: A systematic literature review," Dec. 2020, doi: 10.1007/s11831-021-09566-x.