

IDENTIFICATION OF SOFTWARE CLONE FILES USING MACHINE LEARNING

Mrs. A. Swapna ¹ (Asst. Professor) and J. Nandini ², G. Shiva Chandra ³, G Raghu Ram ⁴,

G Nanditha Sridhar ⁵

Sreyas Institute of Engineering and Technology

ABSTRACT:

One technique to reduce the likelihood of introducing a bug is to identify code clone files. Refactoring, often known as removal, is the process of making software more understandable and maintainable. When additional clones are discovered in software programs, we need a method to help developers with refactored code and to enhance software quality. This method must tell developers which clone files require reworking. To provide developers with recommendations on which files require code refactoring, our research proposes a novel learning technique that automatically extracts features from the identified code clones and trains models to present a novel technique to enhance classification performance by transforming outliers into Unknown clone sets. The Eclipse dataset, which comprises 213 Java software files, was utilized for this project. Support Vector Machine (SVM), Random Forest, Bagging, and K-Nearest Neighbors (KNN) are the algorithms that are employed. We compare these four categorization models and recommend the model with the highest accuracy. Our tool can be developed and applied to reduce system bugs. By recognizing software clone files and eliminating duplication code, our study can enhance clone maintenance. Also, the possibility of bad design for a system, difficulty in a system improvement or modification, and introduction of a new bug can be decreased by identifying and refactoring clones.

KEYWORDS: Cloning, Refactoring, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision trees.

I.INTRODUCTION

Software clone files are nothing more than exact copies of the original code, without any changes made to enable flawless functionality. In software development, copying and pasting portions of source code is a typical task. However, there are a lot of disadvantages to this cloning. First of all, it is by no means a secure procedure; anyone may just obtain the code and abuse it. Second, it might be very challenging and time-consuming to spot bugs in the original code if they are present in all cloned files. Therefore, the discovery of code clones and the refactoring—or restructuring—of code clones without altering their original functionality—are the solutions to this cloning. Refactoring thereby lessens software cloning.

Refactoring has the following advantages:

1. Maintains functionality
2. Improves code readability
3. Reduce the intricacy of the code
4. Makes maintenance easier

Therefore, the goal of our project is to locate code clones and advise developers on what files should be refactored to reduce the amount of code errors and eliminate duplicate code. In the past, several scholars suggested classification-based methods to suggest clones suitable for refactoring based on advantages, disadvantages, and risks. These attempts, however, suffer from several drawbacks, including poor accuracy and lengthy clone file identification times. However, our project proposes a novel learning technique that trains models to automatically extract features from the discovered code clones and recommend to developers which files in the code require refactoring. We report a comprehensive comparative analysis and a model-based efficiency assessment of our proposed concept. It makes recommendations about whether or not to refactor a clone class. A novel approach is put forth to change clone outliers to the unknown class in order to increase the accuracy of the classification findings. To enhance the classification outcomes, we present a novel technique that transforms clone type outliers from the training categories into an Unknown clone.

II.LITERATURE SURVEY

Author Swati et al.[1] in A “Reliable Novel Approach of Bio-Image Processing—Age and Gender Prediction”presents the results on gender prediction and age estimation system based on convolutions neutral networks by extracting features from given input image.The proposed system can get accurate results by taking large sets of training data. The proposed method used ResNet architecture using facial points identification, to classify the age group and gender of the input subject.

Author Kanwal et al. [2] in “Historical perspective of code clone refactorings in evolving software” performed a systematic study on clone refactoring evolution and defined clone evolution patterns for studying refactorings in a formal notation.

Author Swati et al. [3] in “Deep Neural Network for Accurate Age Group Prediction through Pupil Using the Optimized UNet Model” performs the segmentation through UNet without using a dense layer to perform the segmentation and classification to predict accurate age group.

Author Swati et al. [4] in “Review on secure traditional and machine learning algorithms for age prediction using IRIS image”discussed one hundred and one papers in the literature with various image segmentation, feature extraction and classification of the iris. This paper summarizes publicly available standard databases and various evaluation parameters, i.e., accuracy, precision, recall, f-score, etc. The research community evaluated the age prediction through the iris-based state-of-the-art algorithms with secure prediction, i.e., TPR, TNR, FPR, FNR.

Author Swati et al. [6] in "A smart application to detect pupil for small dataset with low illumination." designed with 2 convolution layers and 3 dense layers. We examined the module with 5 datasets including 3 benchmark datasets, namely CASIA, UBIRIS, MMU, random dataset, and the live video. We calculated the FPR, FNR, Precision, Recall, and accuracy of each dataset. The calculated accuracy of CASIA using the proposed system is 82.8%, for UBIRIS is 86%, MMU is 84%, and the random dataset is 84%. On live video with low resolution, calculated accuracy is 72.4%. The proposed system achieved better accuracy compared to existing state-of-the-art systems.

Author Swati et al. [7] in "Robust deep learning technique: U-net architecture for pupil segmentation." proposed a novel CNN U-Net based model to perform the accurate segmentation of pupil. We experimented on the CASIA database and generated an accuracy of 90% in segmentation. We considered various parameters such as Accuracy, Loss, and Mean Square Error (MSE) to predict the efficiency of the model. The proposed system performed the segmentation of pupil from 512×512 images with MSE of 1.24.

Sheneamer et al. [8], proposed an approach to automatically suggest a treatment about refactoring of clones in the system. It suggests whether a clone class needs refactoring and what type of refactoring is needed by a clone. To improve accuracy of classification results, a new method is proposed to convert clone outliers to unknown class. Bagging, K-nearest Neighbors, forestPA and decision forest algorithm are used to train clone refactoring data. Experiments on six software systems are performed and results showed that proposed approach achieved better accuracy than proposed approaches i.e. 93% and 87% on two different subject systems. Our study is different from this because it provides analysis of clones refactoring at a broader level such as it provides metrics (e.g. ratio of removed clones through refactoring), as well as version wise analysis of clone refactoring through quantitative and qualitative analysis.

Volanschi et al. [9] presented an approach to manage clones either by clone removal through refactoring or providing clones to developers in a new look and feel so that they can decide about the clones according to their ease. They discussed the limitations of programming languages and existing clone management tools to help developers in refactoring clones. As all clones are not refactorable, providing new look and feel was helpful in deciding about clones. They developed a hybrid approach that combines refactoring and link editing to manipulate the benefits of both techniques. They also developed a prototype called Stereo based on the hybrid approach to help developers during clone maintenance. Evaluation of the hybrid approach revealed that in comparison to the existing approaches e.g. linked editors, this approach was better in expressiveness, scalability, and controllability.

Author Ruru et al. [10], proposed that when many clones are detected in software programs, not all clones are equally important to developers. To help developers refactor code and improve software quality, various tools were built to recommend clone-removal refactoring based on the past and the present information, such as the cohesion degree of individual clones or the co-evolution relations of clone peers. Given a set of software repositories, CREC first automatically extracts the clone groups historically refactored and those not refactored to construct the training set. CREC extracts 34 features to characterize the content and evolution behavior of individual clones, as well as the spatial, syntactical, and co-change relations of clone peers. The results show that our approach suggested refactoring with 83% and 76% F-scores in the within-project and cross-project settings.

Author Kodhai et al. [13], researchers focused on activities such as clone maintenance to assist the programmers. Refactoring is a well-known process to improve the maintainability of the software. This paper contributes to a more unified approach for the phases of clone maintenance with a focus on clone modification. This approach is implemented as an enhancement to the existing tool Clone Manager.

Author Higo et al. [14], proposed that in an actual software development process, code clones are introduced because of various reasons such as reusing code by 'copy-and-paste' and so on. Code clones are one of the factors that make software maintenance difficult. In this paper, we propose a method which removes code clones from object-oriented software by using existing refactoring patterns, especially Extract Method and Pull Up Method.

Gode et al. [15] studied refactoring of code clones from developers perspective. He manually analyzed how developers refactor cloned code. His main focus was on analyzing whether developers deliberately remove clones through software refactoring or the cloned code is accidentally removed i.e. during routine maintenance of software. He used source code change commits from version history to identify removed code fragments. He defined heuristics for detection of deliberate and accidental removal and also found discrepancies between cloned code and refactored code i.e. clones that are deliberately removed by developers are not reported in clone detection tools. *Extract method* refactoring is mostly used in deliberate clone removal.

Author Higo et al. [16], proposed that the presence of code clones is generally regarded as one factor that makes software maintenance more difficult. For example, if a code fragment with code clones is modified, it is necessary to consider whether each of the other code clones has to be modified as well. Removing code clones is one way of avoiding problems that arise due to the presence of code clones. This paper proposes a set of metrics that suggest how code clones can be refactored.

III. PROPOSED SYSTEM

To provide developers with advice on which files require code refactoring, our research proposes a novel learning technique that automatically extracts features from the discovered code clones and trains models in that manner. We report a comprehensive comparative analysis and a model-based efficiency assessment of our proposed concept. It makes suggestions about whether and what kind of refactoring a clone class requires. A novel approach is put forth to change clone outliers to the unknown class to increase the accuracy of the classification findings. We used four classification models KNN, SVM, Bagging, and Random Forest to obtain their relative performance and used to train clone refactoring data. Our approach has a high value in achieving high automated advising refactored clone accuracy. Time-consuming in identifying software clones is very less. It gives better accuracy than existing systems. We got an accuracy of 85% with Random Forest among the four classification models.


































































3.1 DATASET

The Eclipse dataset is a collection of data gathered from Google. There are 213 Java code files in the Eclipse dataset. The Java code files include AnnotableType.java, AbstractTypeDeclaration.java, Annotation.java, and so forth. Feature vectors are created from the Java code files. Models are created for training and testing using these feature vectors.

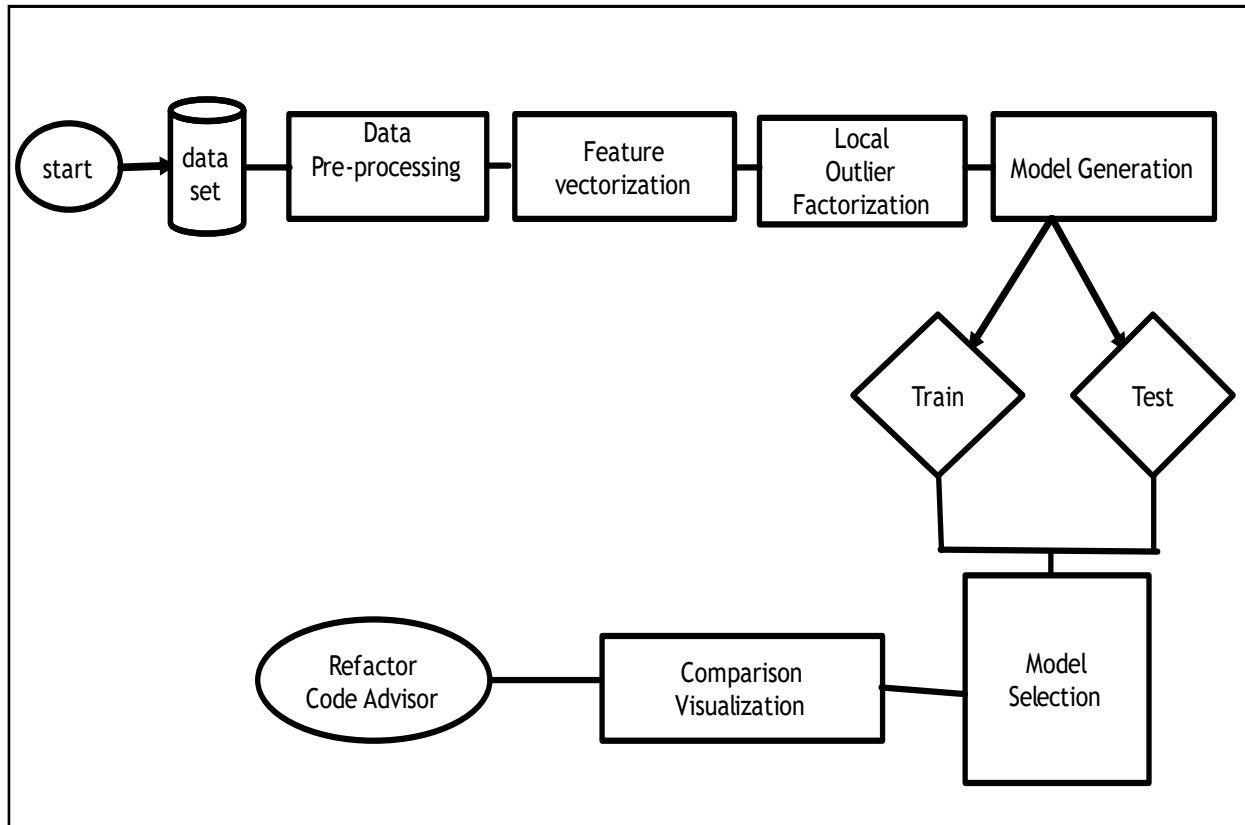
There are 213 files in total.

174 training records in total

39 test records in all.

 AbstractTypeDeclaration Java Source File 7.77 KB	 AnnotableType Java Source File 3.64 KB	 Annotation Java Source File 4.76 KB	 AnnotationBinding Java Source File 8.72 KB	 AnnotationProcessor Java Source File 2.87 KB
 AnnotationProcessorEnvironment Java Source File 6.37 KB	 AnnotationProcessorFactory Java Source File 4.75 KB	 AnnotationProcessorListener Java Source File 1.90 KB	 AnnotationProcessors Java Source File 4.28 KB	 AnnotationTypeDeclaration Java Source File 7.32 KB
 AnnotationTypeMemberDeclaration Java Source File 6.94 KB	 AnonymousClassDeclaration Java Source File 5.01 KB	 APTCompilationParticipant Java Source File 1.06 KB	 APTPlugin Java Source File 4.70 KB	 Archive Java Source File 4.54 KB
 ArchiveFileObject Java Source File 6.67 KB	 ArrayAccess Java Source File 6.94 KB	 ArrayCreation Java Source File 8.79 KB	 ArrayInitializer Java Source File 3.93 KB	 ArrayType Java Source File 12.2 KB
 AssertStatement Java Source File 6.92 KB	 Assignment Java Source File 12.1 KB	 AST Java Source File 126 KB	 ASTConverter Java Source File 278 KB	 ASTMatcher Java Source File 105 KB
 ASTNode Java Source File 101 KB	 ASTParser Java Source File 64.9 KB	 ASTRecoveryPropagator Java Source File 17.6 KB	 ASTRequestor Java Source File 4.73 KB	 ASTSyntaxErrorPropagator Java Source File 3.97 KB
 ASTVisitor Java Source File 82.2 KB	 BindingComparator Java Source File 15.1 KB	 BindingResolver Java Source File 35.9 KB	 Block Java Source File 4.05 KB	 BlockComment Java Source File 3.10 KB
 BodyDeclaration Java Source File 8.47 KB	 BooleanLiteral Java Source File 4.19 KB	 BreakStatement Java Source File 9.01 KB	 CastExpression Java Source File 6.52 KB	 CatchClause Java Source File 6.88 KB
 CharacterLiteral Java Source File 9.26 KB	 Checks Java Source File 19.9 KB	 ChildPropertyDescriptor Java Source File 3.31 KB	 ChildPropertyDescriptor Java Source File 3.96 KB	 ClassInstanceCreation Java Source File 17.4 KB
 Comment Java Source File 3.45 KB	 CompilationUnit Java Source File 40.8 KB	 CompilationUnitResolver Java Source File 54.0 KB	 ConditionalExpression Java Source File 9.17 KB	 ConstructorInvocation Java Source File 6.69 KB
 ContinueStatement Java Source File 4.53 KB	 CreationReference Java Source File 5.98 KB	 DefaultASTVisitor Java Source File 17.1 KB	 DefaultBindingResolver Java Source File 92.2 KB	 DefaultCommentMapper Java Source File 22.6 KB
 DefaultLocation Java Source File 4.41 KB	 DefaultValuePairBinding Java Source File 1.86 KB	 Dimension Java Source File 4.32 KB	 DocCommentParser Java Source File 29.2 KB	 DoStatement Java Source File 7.28 KB
 EclipseBatchRequestor Java Source File 2.61 KB	 EclipseCompiler Java Source File 8.42 KB	 EclipseCompilerImpl Java Source File 38.9 KB	 EclipseCompilerRequestor Java Source File 1.88 KB	 EclipseDiagnostic Java Source File 5.07 KB

3.2 SYSTEM ARCHITECTURE



Our project's system architecture outlines the system's whole flow. From the beginning to the finish of the flow, various modules are involved in this case. Uploading datasets, preprocessing data, feature vectorization, local outlier factor, using machine learning techniques, and identifying software clone files are among the modules.

Upload Dataset

We upload code clone files and repositories from Eclipse during this process. There are 213 Java code files in the Eclipse dataset. The Java code files include AnnotableType.java, AbstractTypeDeclaration.java, Annotation.java, and so forth. Data Preprocessing

Raw data is converted into a format that computers can understand and analyze through the process of data pre-processing. The algorithm can now easily interpret the data's features. All the files are processed by the application after the dataset is uploaded. It turns the text into tokens, eliminates all stop words, ignores punctuation, turns it into alphabets, and finally turns it back into a string.

Feature vectorization

A feature is a particular attribute or quality of an event that is being observed. They are frequently referred to as fields, attributes, variables, characteristics, or dimensions. Simply put, feature vectors are an array of all the features arranged in a particular manner. The object that the machine learning algorithms require is represented by an n-dimensional vector of numerical features. With the help of this module, we were able to transform code words into feature vectors using the Natural Language Toolkit (NLTK) and then normalize the feature vectors to between 0 and 1 by averaging their word counts. Among the features produced are zip, zip file, abort compilation, and abort exception. There are 4284 code words or features in total in the repository.

Local Outlier Factor (LOF)

An unsupervised technique for detecting anomalies is the Local Outlier Factor (LOF) algorithm, which determines the local density deviation of a particular data point about its neighbors. Samples with a significantly lower density than their neighbors are classified as outliers. This approach yields 1 if an attribute is an anomaly (not an outlier) and -1 if an attribute is not significant or an outlier. It is used to find key attributes/columns from feature vectors. Thus, we may extract significant properties from feature vectors using this technique.

Apply Machine learning Algorithms

In order to create a model, features will be translated into train and test records. Outliers will be used as an unknown class, and class labels will be assigned as 0 or 1 based on how similar the code modules are. One will be assigned if the code has a large number of comparable words; otherwise, zero will be assigned. Several machine learning techniques, including SVM, KNN, Bagging classifier, and Random Forest, will be used with this subject.

Identification of Software Clone Files

We can find and identify clone code files and suggest them to developers once we've built a machine-learning-trained model. The developer will then review the anticipated files that include clone code and restructure them.

3.3 ALGORITHMS USED

To find software clone files, a variety of machine learning algorithms are employed. In these cases, the model is trained using a dataset and then evaluated on new records to improve accuracy.

3.3.1 Random Forest Algorithm

An algorithm for supervised classification is the Random Forest algorithm. Its name, which implies to somehow construct a forest and make it random, makes it obvious. The amount of trees in a forest directly affects the outcomes it may provide; the more trees there are, the more precise the outcome. It is important to remember, nevertheless, that building a decision using an information gain or gain index strategy is not the same as building a forest. One tool for decision help is the decision tree. It illustrates the potential outcomes

with a graph that resembles a tree. The decision tree will create a set of rules if you feed it a training dataset that includes targets and features. Predictions can be made using these rules. Once our dataset has been divided into three categories, Random Forest assists in creating classes from the dataset. A random forest is a collection of clusters of decision trees. When you feed a decision tree a training dataset containing features and labels, the tree will create a set of rules that it will use to generate predictions.

3.3.2 K Nearest Neighbor Algorithm (KNN)

One of the most fundamental yet crucial classification methods in machine learning is K-Nearest Neighbors (KNN). It is heavily used in pattern recognition and falls within the supervised learning category. Since it is non-parametric—that is, it does not make any underlying assumptions about the distribution of data—it is extensively applicable in real-life circumstances (in contrast to other algorithms like GMM, which assume a Gaussian distribution of the given data). An attribute-based prior data set (also known as training data) is provided to us, allowing us to classify coordinates.

3.3.3 Support Vector Machine Algorithm (SVM)

A supervised machine learning approach called Support Vector Machine (SVM) can be applied to problems involving regression or classification. It is usually applied to categorization difficulties, though. In this approach, the value of each feature is represented by a specific coordinate, and each data item is plotted as a point in n-dimensional space (where n is the number of features you have). Next, classification is carried out by identifying the hyper-plane that best distinguishes the two classes (see the snapshot below for an example). In real-world applications, the SVM algorithm is carried out using a kernel. This SVM introduction does not cover linear algebra, which is required to transform the problem and learn the hyperplane in linear SVM. Rephrasing the linear SVM using the inner product of any two provided data instead of the observations themselves is a major idea. The total of each input value pair's multiplication is the inner product between two vectors.

3.3.4 Bagging Algorithm

One kind of ensemble machine learning technique called bagging combines the results from numerous learners to enhance performance. These algorithms work by dividing the training set into smaller groups and subjecting them to different machine-learning models. When the subsets return, the predictions from all of the models are combined to create an overall prediction for every instance in the training set. If a modified training data set is created using a bagging sampling approach (data sampled using replacement) or another method is used to train the bagging classifier, it can be dubbed an ensemble meta-estimator. A training set made up of duplicate or unique data sets can be produced by the bagging sampling technique. Another name for this sampling method is bootstrap aggregation. The final predictor, also known as a bagging classifier, averages (regression) or votes (classification) the predictions generated by each estimator/classifier. The amount of samples and/or features that must be taken into account while fitting each individual estimator can be customized while building each estimator.

IV. RESULTS

A GUI interface created using Tkinter, a Python framework, is visible in the front end. Several buttons on the site allow you to upload the dataset and take additional actions.

Figure 1 shows how to pick and upload the "Dataset" folder. Next, click the "Select Folder" button to load the dataset. Finally, wait a short while for the application to read all of the code files. The eclipse dataset is a collection of data gathered from Google. There are 213 Java code files in the Eclipse dataset. The Java code files include AnnotableType.java, AbstractTypeDeclaration.java, Annotation.java, and so forth.

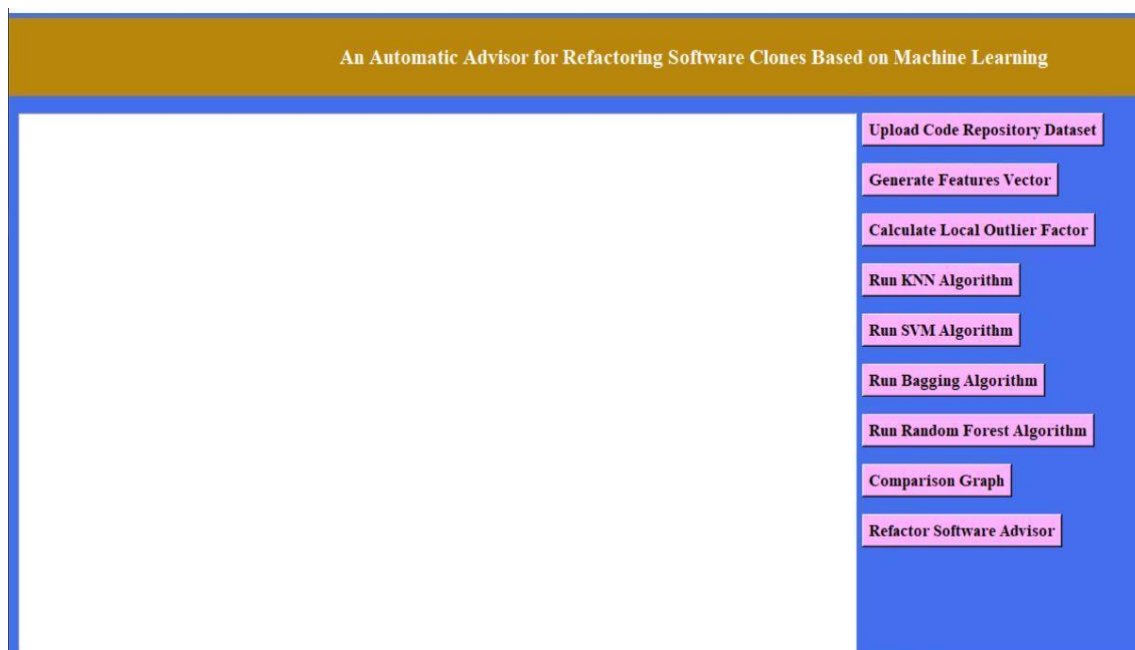


Fig 1. output interface

The application reads and processes each code file in the second stage of Figure 2. Code files like AbstractTypeDeclaration.java, AnnotableType.java, Annotation.java, and so forth are processed by the program. There are 213 total Java files in the dataset. To turn the above code into a vector, click the "Generate Features Vector" button.

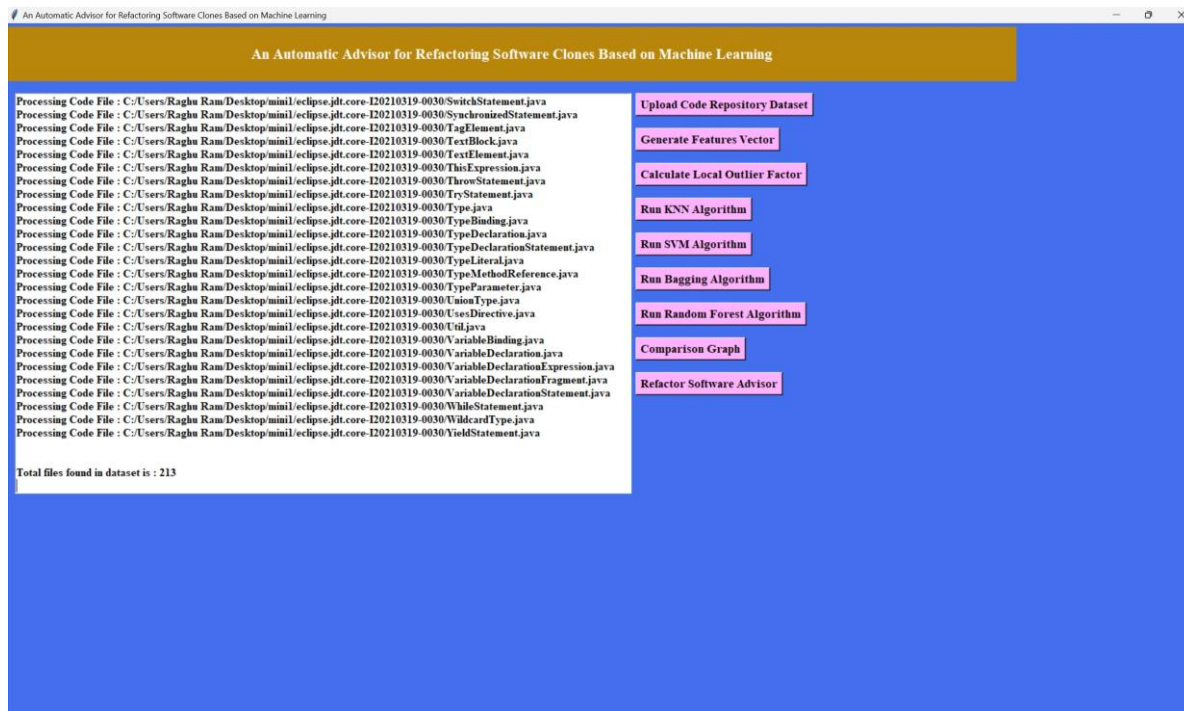


Fig 2. Data preprocessing

Figure 3 shows how all of the codes were transformed into vectors. Each word in the code was placed as a column header, and each word's count and average value were arranged in rows. At this point, the vector was complete. Among the features produced are zip, zip file, abort exception, and abort compilation. There are 4284 total code words or features in the repository. To eliminate unnecessary columns or characteristics, click the "Calculate Local Outlier Factor" button.

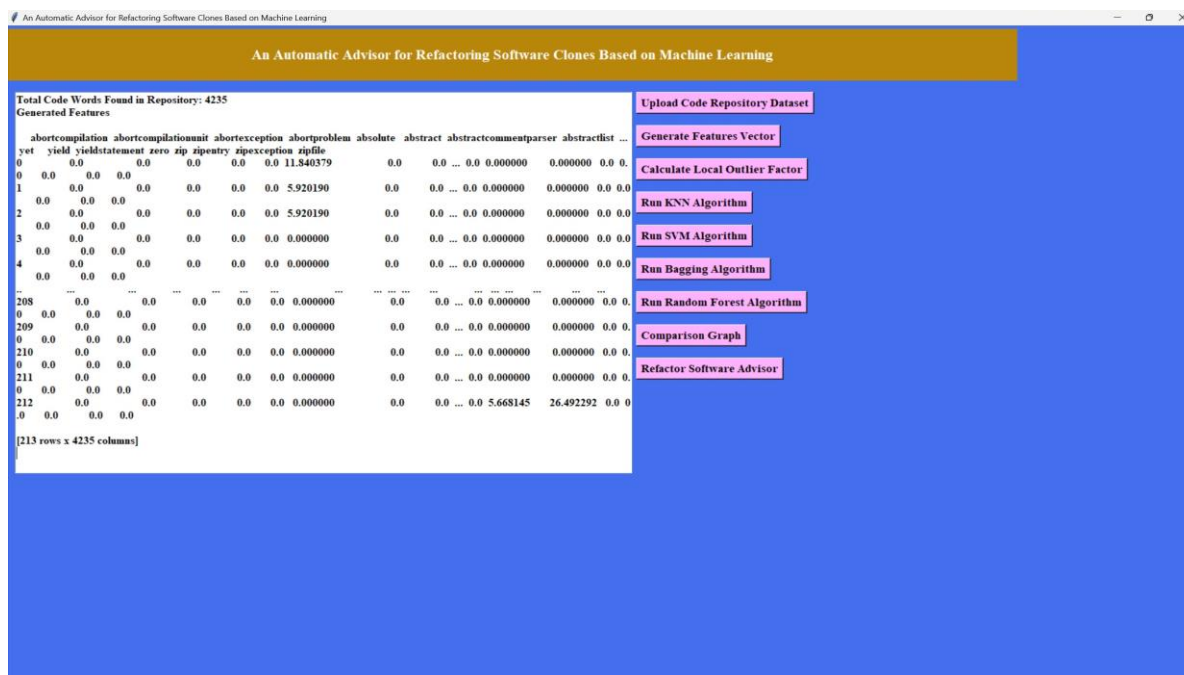


Fig 3. Feature extraction

If a column in the feature vector in Figure 4 has a value of 1, it is significant; if a column has a value of -1, it comprises unimportant attributes. All superfluous properties are removed from features before they are turned into train and test records. 44 test records and 174 training records are present.

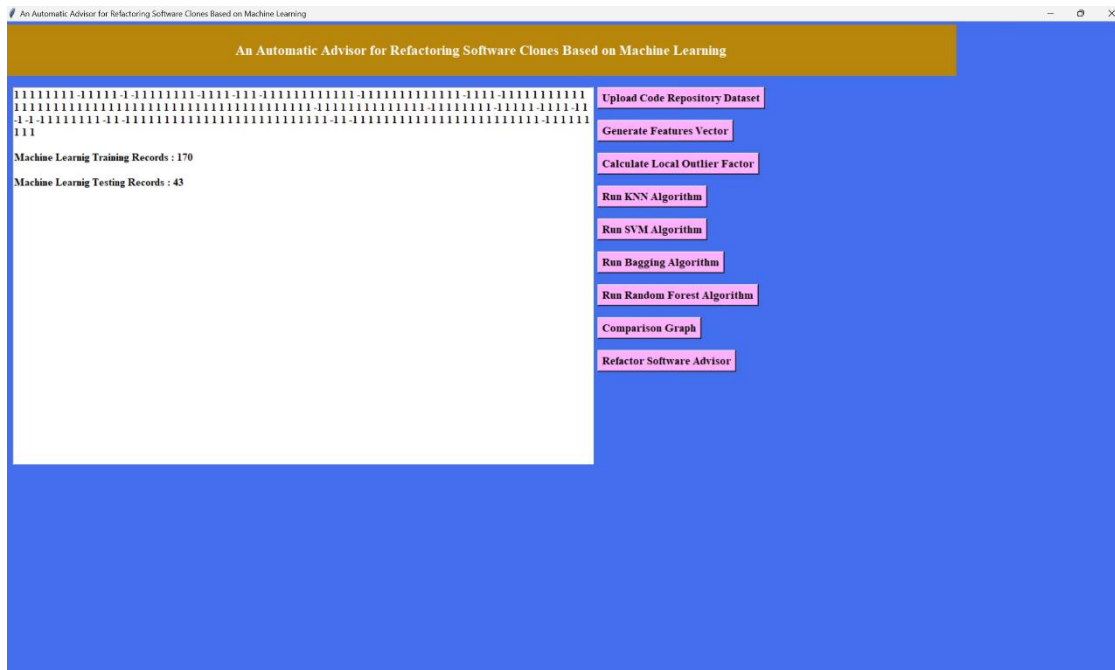


Fig 4. Local outlier factor

Now click on 'Run KNN Algorithm' and all other algorithm buttons to build a machine learning models.

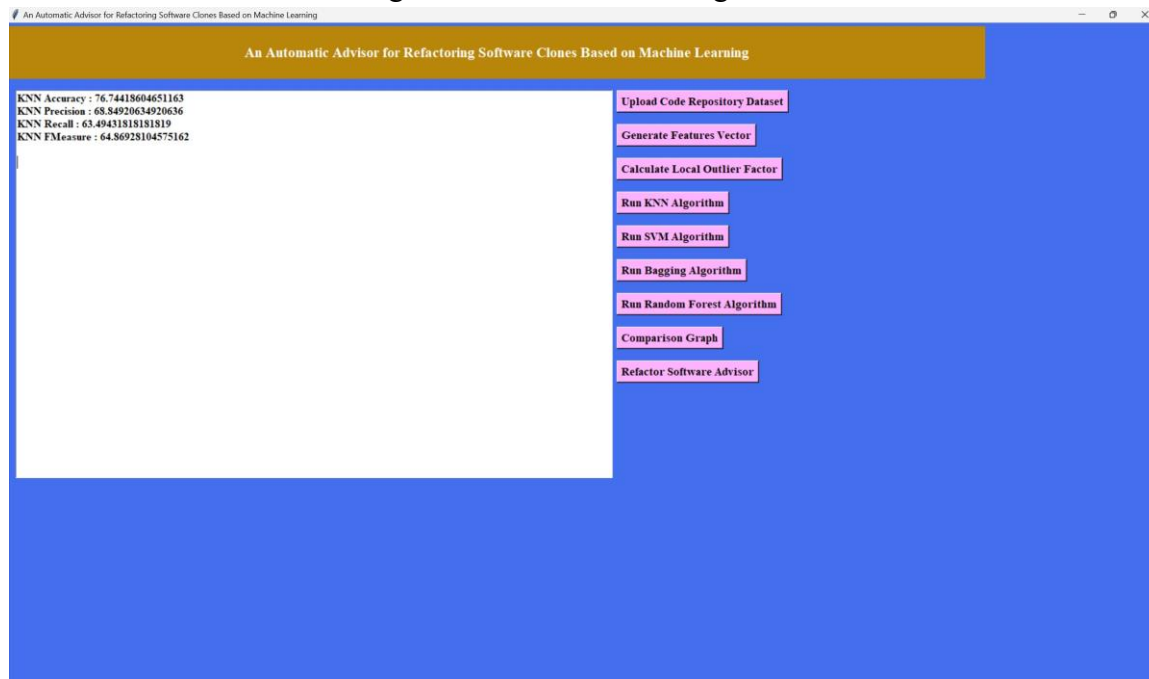


Fig 5. KNN Algorithm

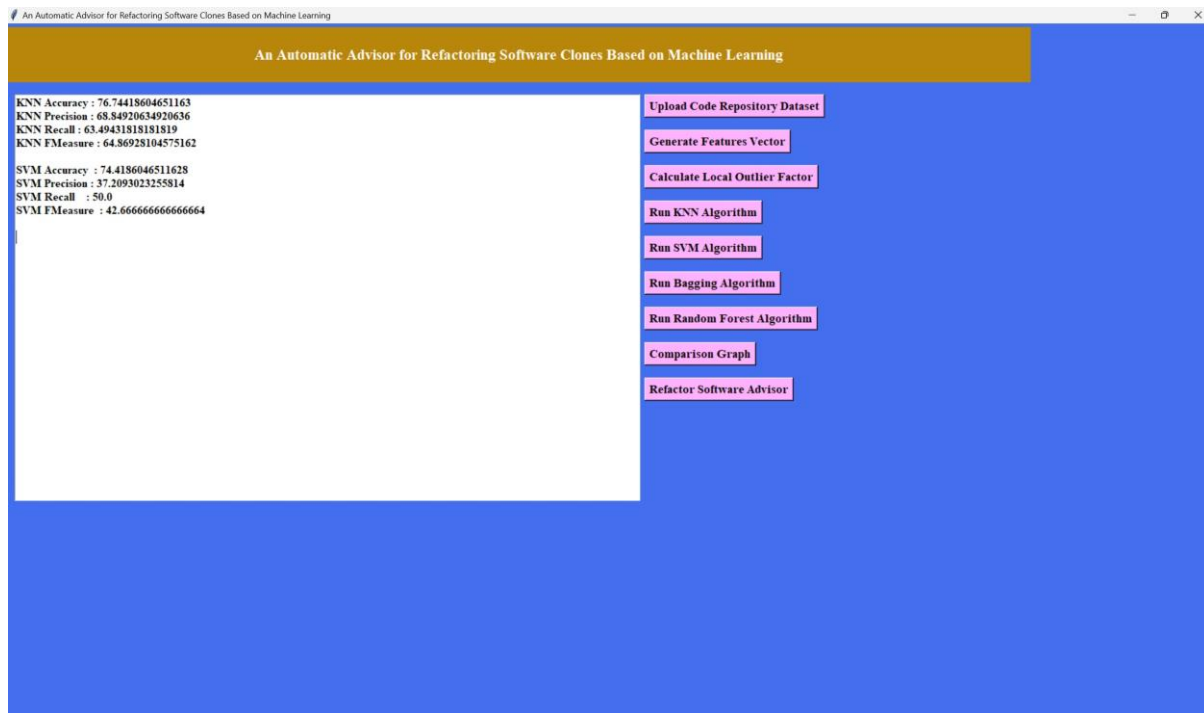


Fig 6. SVM Algorithm

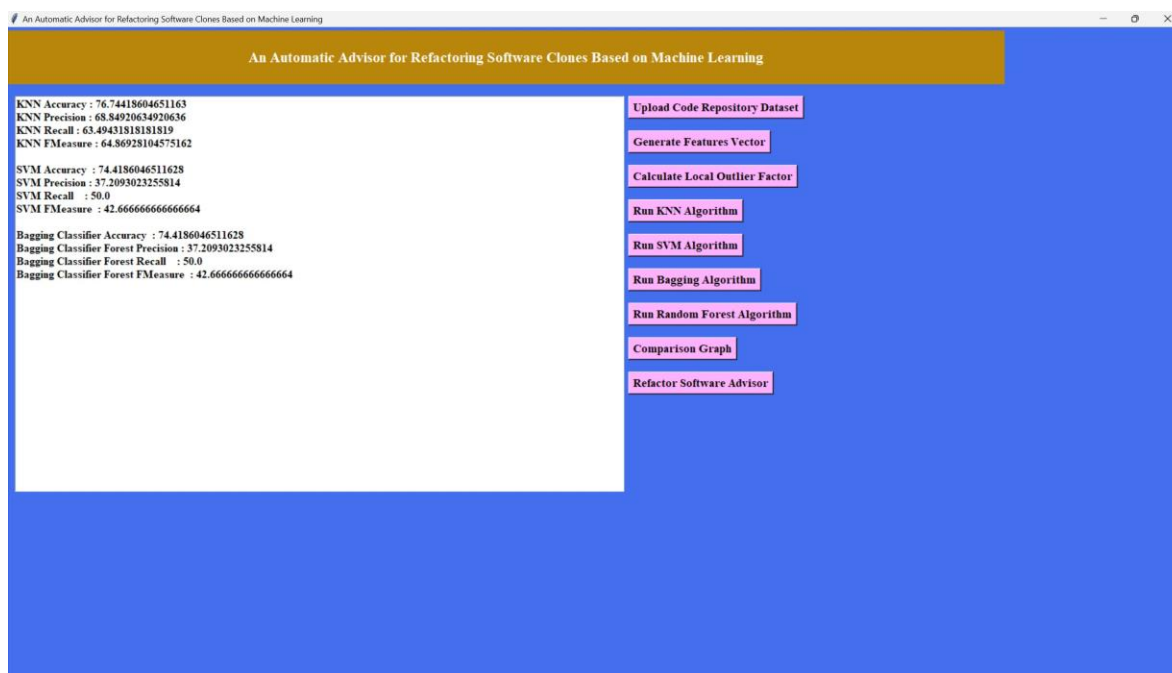


Fig 7. Bagging Algorithm

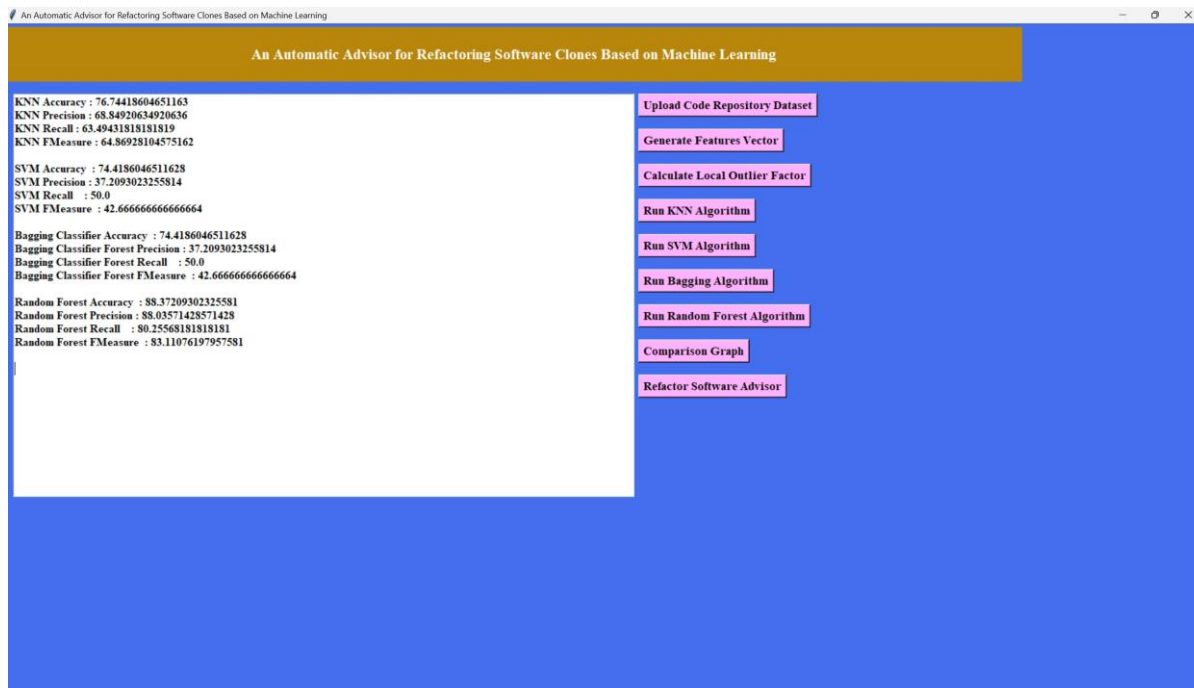


Fig 8. Random Forest Algorithm

After evaluating the machine learning algorithms KNN, SVM, Random Forest, and Bagging on the dataset, performance metrics like accuracy, precision, recall, and FMeasure were obtained. At last, the performance of each algorithm is displayed. Random Forest produces superior results in all algorithms, and machine learning models are now available. The "Refactor Software Advisor" button now displays all code names that need to be refactored.

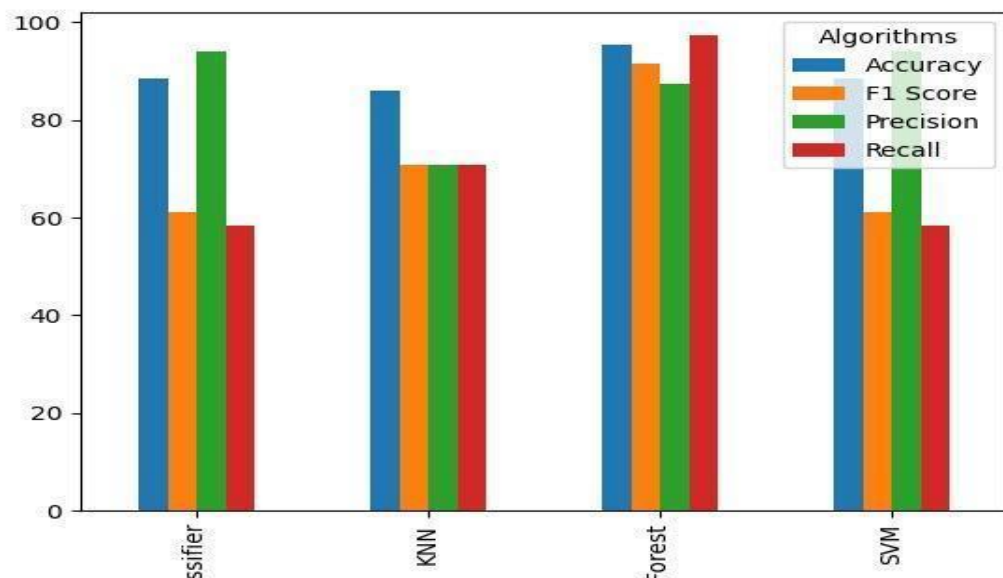


Fig 9. Comparison Graph

Figure 9 above illustrates the application of four distinct algorithms. Recall, f-measure, accuracy, and precision are the measures that are employed. Four algorithms are indicated on the x-axis, and values are indicated on

the y-axis. Every metric is represented by a different color, and while each algorithm displays a different value for each metric, overall random forest produced better results.

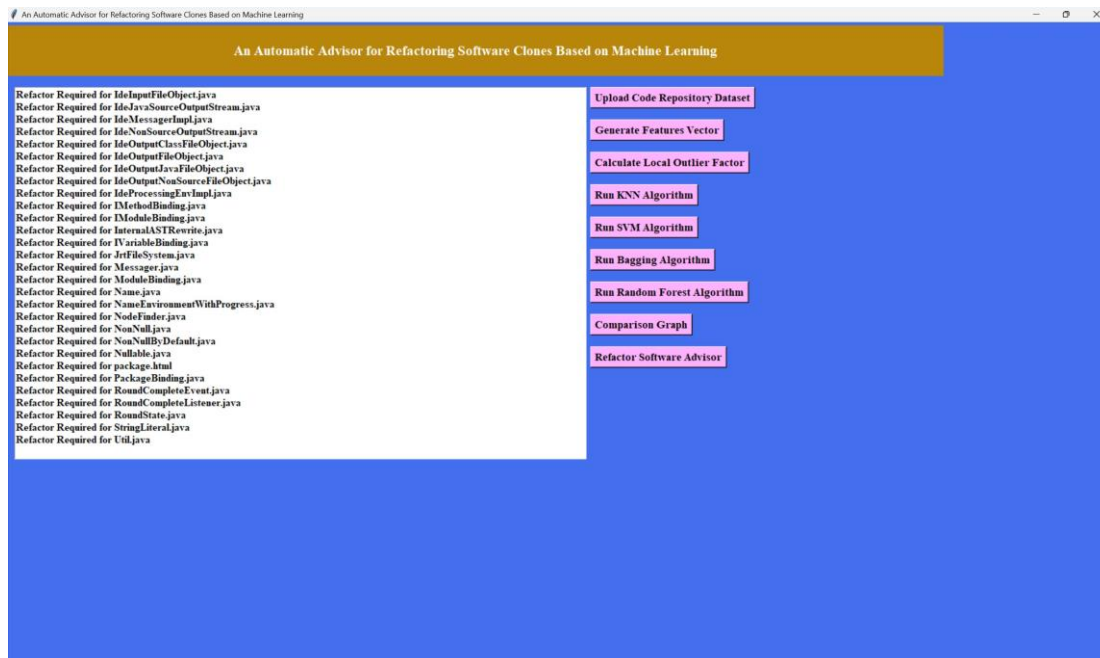


Fig 10. Refactor software advisor

Figure 10 shows that Random Forest produced better accuracy outcomes out of the four methods. Therefore, we decided to use the Random Forest technique to find duplicated files and advise engineers on which files require restructuring.

V.CONCLUSION

To provide developers with advice on which files require code refactoring, this project proposes a learning technique that automatically extracts features from the discovered code clones and trains the models. To enhance the classification outcomes, we present a novel technique that transforms clone-type outliers from the training categories into Unknown clones. We report a comprehensive comparison analysis and apply classification models to assess the effectiveness of our proposed concept. KNN, SVM, Bagging, and Random Forest were the four classification models we tested to determine how well they performed generally. The experimental findings indicate that our method is quite valuable for attaining high automated refactored clone accuracy advice. Using Random Forest, we achieved a 95% accuracy rate among the four classification models. However, in the future, we will be able to identify clones and simultaneously modify the code within them. Additional features, such a highlighter, can be added to further facilitate the identification of code clones. One feature we may include is the ability to remove code clones once they have been identified through code.

VI. REFERENCES

- [1] Swathi, A. et al. (2023). A Reliable Novel Approach of Bio-Image Processing—Age and Gender Prediction. In: Reddy, K.A., Devi, B.R., George, B., Raju, K.S., Sellathurai, M. (eds) Proceedings of Fourth International Conference on Computer and Communication Technologies. Lecture Notes in Networks and Systems, vol 606. Springer, Singapore. https://doi.org/10.1007/978-981-19-8563-8_31
- [2] kanwal J (2022) Historical perspective of code clone refactorings in evolving software. PLoS ONE <https://doi.org/10.1371/journal.pone.0277216>. 23.
- [3] Gowroju, Swathi, Sandeep Kumar, and Anshu Ghimire. "Deep Neural Network for Accurate Age Group Prediction through Pupil Using the Optimized UNet Model." Mathematical Problems in Engineering 2022 (2022).
- [4] Gowroju, Swathi, and Sandeep Kumar. "Review on secure traditional and machine learning algorithms for age prediction using IRIS image." Multimedia Tools and Applications 81, no. 24 (2022): 35503-35531.
- [5] Hu et al. Assessing Code Clone Harmfulness: Indicators, Factors, and Counter Measures. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE; 2021. p. 225–236.
- [6] Swathi, A. and Sandeep Kumar. "A smart application to detect pupil for small dataset with low illumination." Innovations in Systems and Software Engineering 17 (2021)
- [7] Gowroju, Swathi, and Sandeep Kumar. "Robust deep learning technique: U-net architecture for pupil segmentation." In 2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), pp. 0609-0613. IEEE, 2020.
- [8] Sheneamer An automatic advisor for refactoring software clones based on machine learning. IEEE access 2020. <https://doi.org/10.1109/ACCESS.2020.3006178>
- [9] Volanschi N. Stereo: editing clones refactored as code generators. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE; 2018. pp. 595–604.
- [10] R. Yue "Automatic clone recommendation for refactoring based on the present and the past," in Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME), Sep. 2018, pp. 115–126. 19.
- [11] Kanwal J., Structural clones an evolution perspective. In: Proceedings of the 12th International Workshop on Software Clones. IEEE; 2018.
- [12] Wang W, Recommending clones for refactoring using design, context, and history. In: 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE; 2014. p. 331–340.
- [13] Kodhai "Method-level code clone modification using refactoring techniques for clone maintenance," Adv. Comput. Int. J., vol. 4, no. 2, pp. 7–26, Mar. 2013.
- [14] K. Hotta, "Identifying, tailoring, and suggesting form template method refactoring opportunities with program dependence graph," in Proc. 16th Eur. Conf. Softw. Maintenance Reeng., Mar. 2012, pp. 53–62.
- [15] Gode N. Clone removal fact or fiction. In: Proceedings of the 4th International Workshop on Software Clones. ACM; 2010. p. 33–40.
- [16] Y. Higo "A metric-based approach to identifying refactoring opportunities for merging code clones in a java software system," J. Softw. Maintenance Evol. Res. Pract., vol. 20, no. 6, pp. 435–461, Nov. 2008. 31.
- [17] Krinke J. A study of consistent and inconsistent changes to code clones. In: Proceedings of 14th Working Conference on Reverse Engineering. IEEE; 2007. p. 170–178.

[18] Mondal M, Is cloned code really stable in Empirical Software Engineering 2007; 23(2):693–770.

[19] Mondal M A survey on clone refactoring and tracking journal of Systems and Software. 2007.

[20] Duala E, Tracking code clones in evolving software. In: Proceedings of the 29th international conference on Software Engineering. IEEE Computer Society; 2007. p. 158–167.