The Impact of Feature Importance Methods on the Interpretation of Defect Classifiers

Dissertation submitted in the partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

Kolaparthi Hyma	**19VV1A1229**
Gavidi Dhana Lakshmi	**19VV1A1218**
Shaik baji nagul shareef	**19VV1A1252**
Kotha Raghu Ram Gupta	**19VV1A1232**

Under the esteemed Guidance of

Dr.B.Tirimula Rao

Assistant professor & Head of The Department

Department of Information Technology

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTUGV - COLLEGE OF ENGINEERING VIZIANAGARAM(A)

VIZIANAGARAM – 535003, ANDHRA PRADESH, INDIA.

2019-2023

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTUGV COLLEGE OF ENGINEERING VIZIANAGARAM(A)

CERTIFICATE

This is to certify that dissertation entitled "The Impact of Feature Imporatance methods on the Interpretation of defect Classifiers" submitted by K. Hyma(19VV1A1229), G. Dhana Lakshmi(19VV1A1218), S.B.N. Shareef(19VV1A1252), K. Raghu Ram Gupta(19VV1A1232) in the partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY from Jawaharlal Nehru Technological University Gurajada Vizianagaram-College of Engineering Vizianagaram is a record of Bonafied work carried out by them under my guidance and supervision during the year 2022-2023. The result embedded in thisdissertation have not been submitted by any other university or Institution for the award of any degree.

Signature of the Supervisor

Dr. B. TIRIMULA RAO

Assistant professor & HOD

Dept. of Information Technology

JNTUGV-CEV.

Signature of the Head of the Department

Dr. B. TIRIMULA RAO

Assistant professor & HOD

Dept. of Information Technology

JNTUGV-CEV.

{External Examiner}

DECLARATION

We **K. Hyma(19VV1A1229), G. Dhana Lakshmi (19VV1A1218), S.B.N. Shareef(19VV1A1252), K. Raghu Ram Gupta(19VV1A1232)** declared that the dissertation report entitled ""The impact of feature importance methods on the interpretation of defect classifiers" is no more than 1,00,000 word in length including quotes and exclusive of tables, figures, bibliography and references. This dissertation contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated this dissertation in our own work.

Roll No	Name	Signature
19VV1A1229	K. Hyma	_____
19VV1A1218	G. Dhana Lakshmi	_____
19VV1A1252	S.B.N. Shareef	_____
19VV1A1232	K. Raghu Ram Gupta	_____

Date:

Place:

Subject Name: Project Stage - II **Subject Code:PR4204**

Academic Year: 2023 **Regulation: R19**

CO'S

Course Outcomes

Subject Code	Course Outcomes	
PR4107	CO1	Formulate solutions to computing problems using latest technologies and tools
	CO2	Work effectively in teams to design and implement solutions to computational problems and socially relevant issues
	CO3	Recognize the social and ethical responsibilities of a professional working in the discipline
	CO4	Apply advanced algorithmic and mathematical concepts to the design and analysis of software
	CO5	Devise a communication strategy (language, content and medium) to deliver messages according to the situation and need of audience.
	CO6	Deliver effective presentations, extemporaneous or impromptu oral presentations. Setting up Technical reports using technical tools.

CO-PO Mapping

Mapping of Course Outcomes (COs) with Program Outcomes (POs)

Course Outcomes		Program Outcomes(POs)														
		PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
-PR 4107	CO1	3	3	3	3	-	-	-	-	-	1	-	-	3	3	-
	CO2	3	3	3	2	3	-	-	-	3	3	3	-	3	3	3
	CO3	-	1	-	-	-	-	-	-	-	-	1	-	-	-	3
	CO4	3	3	3	3	2	-	-	-	-	2	1	-	3	3	-
	CO5	2	2	2	1	-	-	-	-	3	2	3	-	1	-	3
	CO6	2	3	3	3	3	-	-	-	3	3	3	-	1	-	3

Enter correlation levels 1,2 and 3 as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High) If there is no correlation, put "-"

1.
2.
3.
4.

Signature of the Students **Signature of the Guide** **Signature of the HOD**

ACKNOWLEDGEMENT

This report dissertation could not have been written without the support of our guide **Dr. B. TIRIMULA RAO, Assistant Professor & HOD. Information Technology Department** who not only served as our superior but also encouraged and challenged us throughout our academic program Our foremost thanks goes to him. Without him this dissertation would not have been possible. We appreciate his vast knowledge in many areas, and his insights, suggestions and guidance that helped to shape our research skills.

It is needed with a great sense of pleasure and immense sense of gratitude that we acknowledge the help of these individuals. We owe many thanks to many people who helped and supported us during the writing of this report.

We are thankful to our project coordinator **Dr. B. TIRIMULA RAO**, Assistant Professor. & HOD. Department of Information Technology, for his continuous support.

We express our sincere thanks to our respected **Dr. B. TIRIMULA RAO, Assistant Professor & Head of the Department Of Information Technology** of JNTUGV college of Engineering Vizianagaram for bet valuable suggestion and constant motivation that greatly helped us in successful completion of project We also take the privilege to express our heartfelt gratitude to **Prof. K. SRI KUMAR, Principal,** JNTUGV University college of Engineering Vizianagaram.

We are thankful to all faculty members for extending their kind cooperation and assistance. Finally, we are extremely thankful to our parents and friends for their constant moral support.

19VV1A1229	K. Hyma
19VV1A1218	G. Dhana Lakshmi
19VV1A1252	S.B.N. Shareef
19VV1A1232	K. Raghu Ram Gupta

ABSTRACT

The classifier specific (CS) and classifier agnostic (CA) feature importance methods are frequently used interchangeably to extract feature importance rankings from a defect classifier using prior research. But it is possible that will depend on various feature importance methods.

Despite using the same dataset and classifier, different feature importance methods compute different feature importance rankings. This interchangeable use of the feature is the result.

Instability in conclusions can result from importance methods if there is no strong agreement between them. Thus, in this project we assess the consistency of the feature importance rankings related to the investigated classifiers using a case study consisting of four widely used classifiers and a dataset from a software project.

In this project we will come to know: 1) whether the feature importance rankings calculated by the CA and CS methods strongly agree or not. 2) whether the calculated feature importance ranks by CA methods match or not and CS methods match or not.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction to Defect classifiers:

• Defect classifiers are machine learning models that are used to identify defects or bugs in software systems. The interpretation of defect classifiers is crucial for understanding the factors that contribute to defects and for improving software quality. Feature importance methods play an important role in the interpretation of defect classifiers, as they help to identify the most important features that contribute to the classification of defects.

• The impact of feature importance methods on the interpretation of defect classifiers is significant, as different methods can produce different results and lead to different conclusions. There are various feature importance methods they are classifier specific and classifier agnostic. The choice of feature importance method can have a significant impact on the interpretation of defect classifiers so it is important to know how the methods agree each other.

• When interpreting classifiers, prior studies typically employ a feature importance method to compute a ranking of feature importance. These feature importance ranks reflect the order in which the studied features contribute to the predictive capability of the studied classifier. These feature importance methods can be divided in two categories: classifier-specific (CS) and classifier-agnostic (CA) methods.

• A classifier-specific (CS) method makes use of a given classifier's internals to measure the degree to which each feature contributes to a classifier's predictions.

• A classifier-agnostic (CA) methods measure the contribution of each feature towards a

classifier's predictions. For instance, some CA methods measure the contribution of each feature by effecting changes to that particular feature in the dataset and observing its impact on the outcome. The primary advantage of CA methods is that they can be used for any classifier

• The interchangeable use of feature importance methods is acceptable only if the feature importance ranks computed by these methods do not differ from each other. Therefore, in order to determine the extent to which the importance ranks computed by different importance methods agree with each other, we conduct a case study on popularly used software defect dataset pc5 using classifiers from four different families. We compute the feature importance ranks using four CS and two CA methods on these datasets and classifiers. Finally, we compute Kendall's Tau, Kendall's W, and Top-k (k 1, 3) overlap to quantify the agreement between the computed feature importance ranks by the different studied feature importance methods for a given classifier and dataset.

• While Kendall's measures compute agreement across the different feature importance ranks, the Top-K overlap measure focuses on the top-k items of these rankings.

1.2 Types of feature importance methods:

1.2.1 •Classifier Specific method:

Classifier-specific feature importance methods are a type of feature importance method that is designed to be specific to a particular type of classifier or machine learning model. These methods are tailored to the characteristics of the classifier and the data used to train it, and they can provide more accurate and informative measures of feature importance than generic feature importance methods. Classifier-specific feature importance methods often take advantage of the specific properties of the classifier

CS methods:

-Logistic regression classifier

-Decision Tree classifier

-Random forest classifier

-Extreme Gradient boosting trees classifier.

Logistic regression classifier: Logistic regression is a statistical method used to

analyze and model the relationship between a dependent variable and one or more independent variables. It is a commonly used technique for binary classification problems, where the goal is to predict whether an observation belongs to one of two classes. The logistic regression model estimates the probability of an observation belonging to a particular class using a sigmoid function, which maps any real-valued input to a probability value between 0 and 1.

Decision Tree classifier: A decision tree classifier is a type of machine learning algorithm that is used for classification tasks. It works by recursively partitioning the data into subsets based on the values of features, until the subsets are homogenous or the stopping criteria is met. The final partitioning produces a tree-like structure where each internal node represents a decision based on a feature and each leaf node represents a class label. The decision tree algorithm is a top-down, greedy approach that starts with the entire dataset and recursively partitions the data into subsets based on the best feature to split on. The best feature is determined using a criterion such as information gain, which measures the reduction in uncertainty of the class labels after the split.

Random forest classifier: Random forest is a type of ensemble learning method that uses multiple decision trees to make predictions. It is a popular algorithm for classification, regression, and feature selection tasks. In a random forest, each decision tree is trained on a random subset of the training data, and a random subset of the features is considered for each split in the tree. This helps to reduce the risk of overfitting and improves the generalization performance of the model. To make a prediction with a random forest, each tree in the forest predicts the class label or value, and the final prediction is the majority vote (classification) or the average (regression) of the predictions from all the trees.

Extreme Gradient boosting trees classifier: Extreme Gradient Boosting (XGBoost) is a popular algorithm for gradient boosting trees, a method for ensemble learning in machine learning. It is a highly scalable and efficient algorithm that can handle both regression and classification problems. Like other boosting algorithms, XGBoost builds a sequence of decision trees, where each tree tries to correct the mistakes of the previous tree. However, XGBoost has several unique features that distinguish it from other boosting algorithms. One of the key features of XGBoost is its optimization objective, which includes both a loss function and a regularization term. The loss function measures the difference between the predicted and actual values, while the regularization term penalizes models

that are too complex and thus more likely to overfit. This helps to prevent overfitting and improve the generalization performance of the model. Another important feature of XGBoost is its use of gradient descent to optimize the objective function. Gradient descent is an optimization algorithm that iteratively updates the model parameters in the direction of the negative gradient of the objective function, which helps to find the optimal parameters that minimize the loss function. XGBoost also includes several other optimizations, such as approximate greedy algorithm to find the best split, parallel processing, and early stopping to prevent overfitting and reduce training time.

1.2.2 •Classifier Agnostic method:

Classifier agnostic feature importance methods are a type of feature importance method that can be used with any type of machine learning classifier, regardless of its specific properties or characteristics. These methods are not tailored to any specific classifier or model, but rather are designed to provide a general measure of feature importance that can be applied to a wide range of models. Classifier agnostic feature importance methods have the advantage of being applicable to a wide range of models and can provide a general measure of feature importance that can be used across different tasks and applications.
CA methods:
-permutation method
-SHAP method

Permutation method: Permutation method is a feature importance method that can be used to measure the importance of input features for machine learning models. This method involves randomly permuting the values of a single feature in the dataset and measuring the resulting decrease in model performance. The larger the decrease in performance, the more important the feature is considered to be. Permutation method is a simple and intuitive way to measure feature importance, and can be applied to a wide range of machine learning models, including regression and classification models. The method is easy to implement and does not require any assumptions about the data or the model. The permutation method can be used to obtain both global and local measures of feature importance. A global measure of feature importance can be obtained by permuting

each feature in turn and computing the decrease in model performance for each feature. The features that result in the largest decrease in performance are considered to be the most important. A local measure of feature importance can be obtained by permuting the feature values for a specific instance or observation in the dataset, and measuring the resulting decrease in model performance. This can help to identify which features are most important for a particular observation. One potential limitation of the permutation method is that it can be computationally expensive, especially for large datasets or complex models. Additionally, the method may not capture interactions between features, as the importance of a feature may depend on the values of other features in the dataset. Overall, the permutation method is a useful and flexible feature importance method that can provide insight into the factors that are most important for machine learning models.

SHAP method: SHAP (SHapley Additive exPlanations) is a feature importance method that can be used to explain the output of any machine learning model. The method is based on game theory and provides a way to allocate feature importance among the contributing features. The SHAP method provides both global and local measures of feature importance. A global measure of feature importance can be obtained by averaging the SHAP values across all instances in the dataset. The features with the largest average SHAP values are considered to be the most important. A local measure of feature importance can be obtained by computing the SHAP values for a specific instance or observation in the dataset. The SHAP values represent the contribution of each feature to the difference between the model output for the observation and the expected output for the dataset as a whole. The SHAP values are calculated using a modified version of the Shapley values from cooperative game theory. The Shapley value represents the marginal contribution of a feature to the output of the model, taking into account all possible subsets of features. The SHAP method adapts the Shapley value to the context of machine learning models, where the output is a function of many input features. The SHAP method has several advantages over other feature importance methods. It can handle complex models with non-linear interactions between features, and can provide intuitive explanations for the model output. The SHAP values can also be used to generate plots and visualizations that help to interpret the results. However, the SHAP method can be computationally expensive and may not scale well to large datasets or complex models. Additionally, the

method requires a good understanding of game theory and may be more difficult to implement than other feature importance methods. Overall, the SHAP method is a powerful and flexible feature importance method that can provide valuable insights into the factors that are most important for machine learning models.

1.3 Pros and Cons of Classifier specific and Classifier Agnostic feature importance methods:

1.3.1 pros of classifier-specific feature importance methods are:

1.They provide information specific to the model being used, which can be useful for improving the model's performance.
2.They can be computationally efficient, as they only need to consider the features used by the specific model.

1.3.2 cons of classifier-specific feature importance methods are:

1.They may not generalize well to other models or datasets.
2.They may be sensitive to the specific hyperparameters chosen for the model.

1.3.3 pros of classifier-agnostic feature importance methods are:

1.They are more likely to generalize to other models or datasets.
2.They can provide insight into which features are important for the classification task in general, rather than just for a specific model.

1.3.4 cons of classifier-agnostic feature importance methods are:

1.They can be computationally expensive, as they may need to evaluate many different models or perform feature selection.
2.They may not capture the unique characteristics of a specific model or dataset.

1.4 Motivation:

The impact of feature importance methods on the interpretation of defect classifiers project is motivated by the need to improve the interpretability and explainability of machine learning models used for defect prediction in software engineering. Defect prediction models are designed to predict whether a particular software artifact is likely to contain defects, based on a set of input features that capture various characteristics of the artifact. While these models can be highly accurate, they are often treated as black boxes, with little understanding of how they arrived at their predictions. This lack of interpretability and explainability can be a major barrier to adoption in practice, as stakeholders may be hesitant to trust the predictions of a model they do not fully understand. One approach to improving interpretability is to use feature importance methods, which aim to identify the input features that are most important for making predictions. By understanding which features are most influential, stakeholders can gain insights into the underlying factors that contribute to defects, and make informed decisions about how to improve software quality. However, there is currently a lack of consensus on which feature importance methods are most appropriate for defect prediction models, and how they should be used to guide interpretation. This project seeks to address this gap by evaluating the impact of different feature importance methods on the interpretation of defect classifiers, and providing guidance on best practices for their use. By improving the interpretability and explainability of defect prediction models, this project has the potential to increase the trust and adoption of these models in software engineering, and ultimately improve the quality and reliability of software systems.

1.5 Literature survey:

[1] Jiarpakdee, Tantithamthavorn, and Hassan (2019) conducted a study on the impact of correlated metrics on the interpretation of defect models. Their research was published in IEEE Transactions on Software Engineering. The authors investigated how the presence of correlated metrics in defect models affects the accuracy of the models and the interpretation of their results. Their study emphasizes the importance of considering the correlations among software metrics when building and interpreting defect prediction models. This re-

search could provide valuable insights to software developers and practitioners who rely on defect prediction models for software quality assurance.

[2] Mori and Uchihira (2019) provides a comprehensive review of existing literature and offers insights into various techniques for balancing the trade-off between accuracy and interpretability in software defect prediction. The findings of this study can be useful for researchers and practitioners in the field of software engineering who are interested in developing effective and interpretable prediction models.

[3] Lundberg and Lee(2017) Proposed SHAP (Shapley Additive Explanations),SHAP is a method for explaining the output of any machine learning model. It is based on the concept of Shapley values from cooperative game theory, which assigns a value to each feature by computing the average marginal contribution of the feature across all possible subsets of features. SHAP has been shown to be a consistent and theoretically sound method for explaining the output of black-box models.

[4] Breiman(2001) Proposed Random Forest (RF) feature importance,Random forest is an ensemble learning method that combines multiple decision trees to make predictions. RF feature importance is a method for estimating the importance of each feature in a random forest model. It is based on the decrease in accuracy of the model when a feature is randomly permuted, and is often used to rank features by importance.

[5]Zhou et al.(2019) Proposed Permutation Feature Importance (PFI) ,The authors proposed the use of PFI to measure the importance of each feature in a defect classification model. They compared the results of this method to other feature importance methods on four datasets.

[6] Zhang, Y., Yao, W., Zhang, Y., Chen, H.(2020) Proposed XGB tree classifier,XGB (Extreme Gradient Boosting) is a tree-based ensemble method that combines the strengths of gradient boosting and decision trees. It uses a greedy algorithm to build a series of weak decision trees and combines their predictions to make a final prediction. XGB is known for its high accuracy and efficiency.

[7] Hosmer, D. W., Lemeshow, S., Sturdivant, R. X.(2013),Proposed Logistic regression,Logistic regression is a statistical method that models the relationship between a binary dependent variable and one or more independent variables. It estimates the probability of the dependent variable taking a specific value (e.g., 0 or 1) based on the values of the independent variables.

[8] Japkowicz, N., Stephen, S.(2002) Proposed Scott-Knott ESD, Scott-Knott ESD (Effect Size Difference) is a non-parametric algorithm for ranking the performance of different classifiers. It works by recursively splitting the data into subsets based on their effect size difference, and then ranking the classifiers based on their performance on the subsets.

[9] F. E. Harrell Jr,(2015) Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis. Springer,Logistic regression,Logistic regression is a statistical method that models the relationship between a binary dependent variable and one or more independent variables. It estimates the probability of the dependent variable taking a specific value (e.g., 0 or 1) based on the values of the independent variables.

[10] C. Tantithamthavorn, "Scottknottesd(2006): The scott-knott effect size difference (esd) test," R package version, vol. 2.Scott-Knott ESD (Effect Size Difference) is a non-parametric algorithm for ranking the performance of different classifiers. It works by recursively splitting the data into subsets based on their effect size difference, and then ranking the classifiers based on their performance on the subsets.

Chapter 2

Requirements

2.1 Software Requirements:

Environment: Python

Tool: Jupyter notebook and Anaconda prompt

Operating system: Windows10

Edition: Windows 11 Home Single Language

Version: 22H2

Installed on: 9/1/2023

OS build: 22621.1265

Experience: Windows Feature Experience Pack 1000.22638.1000.0

2.2 Hardware Requirements:

Device name: LAPTOP-9LN74LEJ

Processor: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.40 GHz

RAM: 8.00 GB (7.65 GB usable)

Device ID: 1AF8E38C-65A5-4B68-B5CB-D67DA7046B52

Product ID: 00327-36273-91578-AAOEM

System type: 64-bit operating system, x64-based processor

Pen and touch: No pen or touch input is available for this display

Chapter 3

Design

3.1 Data Set:

PC5 dataset is a PROMISE dataset made publicly available in order to encourage repeatable , verifiable , refutable and improvable predictive models of software engineering.
PC5 is a NASA spacecraft instrument written in "C/C++".
PC5 dataset contains 39 attributes. In 38 are independent and 1 is dependent attribute.
The dataset contains more than 1711 records/rows
Source-PROMISE Repository Creators: NASA, then the NASA Metrics Data Program, (http://mdp.ivv.nasa.gov.)
We downloaded the dataset from the zenodo website.
link for dataset :- https://zenodo.org/record/268439

3.2 Methodology:

3.2.1 Data pre-processing:

Correlation analysis:

we perform correlation analysis to remove the highly correlated features among the independent features because the presence of the correlated features will yeilds unstable feature importance ranks. Redundancy analysis is used to remove the redundant features.

For the analysis we use AutoSpearman technique, because we can measure the non linear relationship and directional information about the relation between the features. And it provides information about both the strength and direction of the relationship between two variables.

In Spearman correlation ,the correlation coefficient ranges from -1 to 1

1 indicates a strong monotonic positive relationship

-1 indicates a strong monotonic negative relationship

0 indicates no monotonic relationship.

we compute correlation matrix and remove the highly correlated related features. Here we considered the features with correlation coefficient more than 0.8 as highly correlated features.

After correlation analysis of the data we got 14 features as non correlated features. Remaining 24 features has correlation among them so we removed the correlated features and performed further process on the 14 features.

3.2.2 Splitting the data:

Splitting the data refers to the process of dividing a dataset into two or more subsets, usually for the purpose of training and evaluating a machine learning model.

The most common way to split a dataset is to divide it into two subsets: a training set and a testing set. The training set is used to train the model, while the testing set is used to evaluate the performance of the model on new, unseen data.

There are also other variations of data splitting, such as k-fold cross-validation and leave-one-out cross-validation, which are used to obtain more accurate estimates of model performance by training and testing the model on different subsets of the data.

Overall, the process of splitting the data is an important step in the machine learning pipeline, as it helps ensure that the model is not overfitting to the training data and is able to generalize well to new, unseen data.

In our project we splitted the dataset into training and testing data by importing the train-test-split function from sklearn.model-selection module.

3.2.3 K-fold cross validation:

K-Fold Cross-Validation is a technique for evaluating the performance of machine learning models by dividing the dataset into k subsets or "folds". Each fold is used as a test set exactly once, while the k-1 remaining folds are used as training data. The average performance across all k iterations is used as the performance metric for the model. This method helps to overcome the problem of overfitting by training the model on different data each time and testing on a different set.

The k-fold cross-validation procedure divides a limited dataset into k nonoverlapping folds. Each of the k folds is given an opportunity to be used as a test set, and all other folds collectively are used as a training dataset. A total of k models are fit and evaluated on the k hold-out test sets and the mean performance is reported. Here the models that taken are

-logistic regression

-randomforest classifier

-Decision tree classifier

-xgb tree classifier

and we calculated the scores of the models and each model's scores mean and standard deviation.

The model with the highest mean accuracy score and the lowest standard deviation of the k-fold cross-validation scores is generally considered the best model to fit. The mean accuracy score indicates the overall performance of the model, and the low standard deviation shows that the model's performance is consistent across different folds. A model with high accuracy and low variance is considered a good generalizing model, as it performs well on both the training and validation datasets.

In the above models that we are taken the Random forest classifier is considered as good generalised model because it has high mean accuracy and low standard deviation compared to the remaining models.

3.2.4 Classifier construction and Hyperparameter tuning:

For Classifier construction and hyper parameter tuning we took
-randomforest classifier
-logistic regression
-Decision tree classifier
-xgb tree classifier.
Firstly, we imported classifiers from scikit library and then we fit the classifier to the training data using fit method.
fit(data) means that you are training a claiffier machine model on the trainig data which is stored in features and target variables.
The fit method adjusts the internal parameters of the model based on the trainig data ,so that the model can make accurate predictions on new data.
The input to the fit method are the trainig features and the corresponding target variables.
After trainig the model using the fit method ,we can use it to make predictions on new data using the predict method or evaluate its performance using metrics such as accuracy.
Now we perform the hyperparamater tuning.

The hyperparameters of the randomforest classifier are

• n-estimators: n-estimator is the hyperparameter that defines the number of trees to be used in the model.
• max-features:Max-features limits a count to select the maximum features in each tree.
• max-depth:max-depth determines the maximum number of splits each tree can take.
• max-leaf-nodes:It specifies the max division of nodes to be done.
• min-sample-split:min-sample-split determines the minimum number of decision tree observations in any given node in order to split.

The hyperparameters of the Dececision tree classifier are

• max-depth: The maximum depth of the decision tree. A higher max-depth can lead to overfitting, while a lower max-depth can lead to underfitting.

• min-samples-split: The minimum number of samples required to split a node. A lower min-samples-split can lead to overfitting, while a higher min-samples-split can lead to underfitting.
• min-samples-leaf: The minimum number of samples required to be at a leaf node. A lower min-samples-leaf can lead to overfitting, while a higher min-samples-leaf can lead to underfitting.
• max-features: The number of features to consider when splitting a node. None means to consider all features, while 'sqrt' and 'log2' mean to consider a square root and logarithm of the total number of features, respectively.
• criterion: The function to measure the quality of a split. 'gini' refers to the Gini impurity, while 'entropy' refers to the information gain.
• splitter: The strategy used to choose the split at each node. 'best' means to choose the best split, while 'random' means to choose a random split.

The hyperparameters of the xgb tree classifier are

• n-estimators: The number of trees in the ensemble. Higher values can lead to better performance but may increase training time.
• max-depth: The maximum depth of each tree in the ensemble. Higher values can lead to overfitting.
• learning-rate: The step size shrinkage used to prevent overfitting. Lower values can lead to better generalization but may increase training time.
• subsample: The fraction of samples to use for each tree. Values less than 1.0 can help prevent overfitting.
• colsample-bytree: The fraction of features to use for each tree. Values less than 1.0 can help prevent overfitting.
• gamma: The minimum loss reduction required to make a split. Higher values can lead to a more conservative model.
• min-child-weight: The minimum sum of instance weight needed in a child. Higher values can help prevent overfitting.

The hyperparameters of the logistic regression classifier are

• penalty: The regularization penalty to be applied. 'l1' refers to L1 regularization, 'l2' refers to L2 regularization, 'elasticnet' refers to a combination of L1 and L2 regularization, and 'none' refers to no regularization.
• C: The inverse regularization strength. Smaller values of C result in stronger regularization, while larger values result in weaker regularization.
• max-iter: The maximum number of iterations for the solver to converge.
To tune these hyperparameters we have different methods. They are
-RandomizedSearchCV
-GridSearchCV
We tune the hyperparameters by using the two methods and fit the best parameters and produce y-pred(predictions) and finally calculate the best parameter's fitting accuracy.

3.2.5 Confusion Matrix:

A confusion matrix is a table used to evaluate the performance of a classifier. It is used to compare the predicted class labels of a classifier to the true class labels of the data.
The confusion matrix is a 2x2 table with the following elements:

	Positive	Negative
Positive	TP	FP
Negative	FN	TN

confusion matrix table

True Positives (TP): The number of instances that were correctly classified as positive.
False Positives (FP): The number of instances that were incorrectly classified as positive.
False Negatives (FN): The number of instances that were incorrectly classified as negative.
True Negatives (TN): The number of instances that were correctly classified as negative.

Now by using the confusion matrix we caclculate the Recall, Precesion, F1 score and accuracy.

Recall- Recallmeasures the proportion of positive examples that were correctly identified as such.

A high recall value means that the model correctly identifies most of the positive examples.

Recall=TP/(TP+FN)

Precesion- Precision measures the proportion of positive predictions that are actually positive.high precision value means that the model makes very few false positive predictions that means less occurance of type 1 error

Precesion=TP/(TP+FP)

F1 score-The F1 score is a balance between recall and precision. It takes into account both the false positive and false negative rates. A high F1 score indicates a good balance between precision and recall.

F1 score=(2*Recall*Precesion)/(Recall+Precesion)

Accuracy-Accuracy measures the proportion of correct predictions made by the model

Accuracy=TP+TN/(TP+FP+TN+FN)

AUC-ROC curve:

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1.

The ROC curve is plot with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.

TPR =Recall

FPR=1-Specificity

Specificity=TN/TN+FP

IFA:

An initial false alarm refers to a false positive alarm or alert that is generated by a system or device, typically one designed to detect or alert in the case of an emergency or critical situation.

3.2.6 Computing feature importance scores:

Feature importance score is a metric used to determine the relative importance of each feature in a machine learning model. It helps to identify which features have the greatest

impact on the model's predictions and can help guide feature selection, engineering, and interpretation.
Computing feature importance scores is a common task in machine learning, as it can help in understanding the importance of different features in predicting the target variable.
Here are methods for computing feature importance scores:

Classifier Specific method:
Classifier-specific feature importance methods are a type of feature importance method that is designed to be specific to a particular type of classifier or machine learning model. These methods are tailored to the characteristics of the classifier and the data used to train it, and they can provide more accurate and informative measures of feature importance than generic feature importance methods.
Classifier-specific feature importance methods often take advantage of the specific properties of the classifier

CS methods:
-Logistic regression classifier
-Decision Tree classifier
-Random forest classifier
-Extreme Gradient boosting trees classifier.

In Classifier specific methodology, the feature importance of features is calculated using the classifier model which is used to fit the data. The above four models are used to fit the data and using each model or classifier separate feature importance lists are created.

Classifier Agnostic method:

Classifier agnostic feature importance methods are a type of feature importance method that can be used with any type of machine learning classifier, regardless of its specific properties or characteristics. These methods are not tailored to any specific classifier or model, but rather are designed to provide a general measure of feature importance that can be applied to a wide range of models.

Classifier agnostic feature importance methods have the advantage of being applicable to a wide range of models and can provide a general measure of feature importance that can be used across different tasks and applications.

CA methods:

-permutation method

-SHAP method

In Classifier agnostic methodology, the feature importance of features is calculated using the above two methods without any influence of the classifier model which is used to fit the data. And two separate CA importance scores lists are generated.one is permutation importance scores and other is shap importances.

Now by using this important scores we will generate the feature importance ranks lists.

3.2.7 Scott-Knott ESD:

The Scott-Knott Effect Size Difference (SK-ESD) test is a statistical method used to perform a post-hoc analysis on the results of a clustering or grouping algorithm. It is an extension of the Scott-Knott test that takes into account both the statistical significance and practical significance of the differences between the groups or clusters produced by the algorithm.

The SK-ESD test works by first sorting the groups or clusters based on their mean performance, as in the original Scott-Knott test. Then, a statistical test is applied to determine if there is a significant difference between the means of each adjacent group, as in the original test. However, in the SK-ESD test, an effect size measure is also calculated to determine the practical significance of the difference between the means.

After calculating the effect size between adjacent groups, the SK-ESD test identifies any groups that do not have a significant difference in mean performance and have an effect size smaller than a predetermined threshold. These groups are merged, and the test is repeated until no more groups can be merged.

The SK-ESD test is useful for comparing the performance of different clustering or grouping methods and determining if there are any significant and practically significant differences between the groups produced by these methods. It provides a more comprehensive analysis than the original Scott-Knott test, which only considers statistical significance.

In our project SK-ESD is used to compute the feature importance ranks using the feature importance scores.

To perform SK-ESD the mean AUC score of the classifier mode must be greater than 0.7 .In our project the mean AUC score of classifiers random forest, logistic regression and xgbtree are greater than 0.7 so we perform SK-ESD on those model fitted data but the decision tree classifier's mean AUC score is less than 0.7 so we will not perform SK-ESD on the decision tree model fitted data.

3.2.8 Computing feature importance ranks:

The importance rank of a feature reflects how much information it provides to the model for making accurate predictions. The more important a feature is, the more influential it is in determining the target variable. The ranking is usually expressed in descending order, with the most important feature ranked first.

Understanding the feature importance ranks can be helpful in several ways. It can guide feature selection and engineering efforts by identifying which features are most valuable and informative. It can also help interpret the model's predictions by revealing the underlying factors that contribute most strongly to the output. Additionally, feature importance

ranks can be used to compare the performance of different models or evaluate the stability of a model's results across different datasets.
Overall, feature importance ranks provide a useful way to assess the relative importance of features in a machine learning model, and can be a valuable tool for building more accurate and interpretable models.
In our project we compute the feature importance ranks for the features using the scott-knott ESD test and then one CS rank list and two CA rank lists are produced.

3.2.9 Kendall's tau:

Kendall's tau coefficient as a non-parametric rank correlation statistic to measure similarity between two rank lists.
Kendall's tau ranges from -1 to 1, with
-1 indicating perfect disagreement and
1 indicating perfect agreement.
The interpretation scheme proposed by Akoglu suggests that Kendall's tau agreement can be categorized as
weak if the absolute value of tau is less than or equal to 0.3,
moderate if it is between 0.3 and 0.6, and
strong if it is greater than 0.6.
This interpretation scheme is often used to evaluate the degree of agreement or similarity between two sets of rankings or preferences.
In our project we perform kendall's tau between the CA1 and CA2 rank lists, and among the CS and CA rank lists.

3.2.10 Kendall's w:

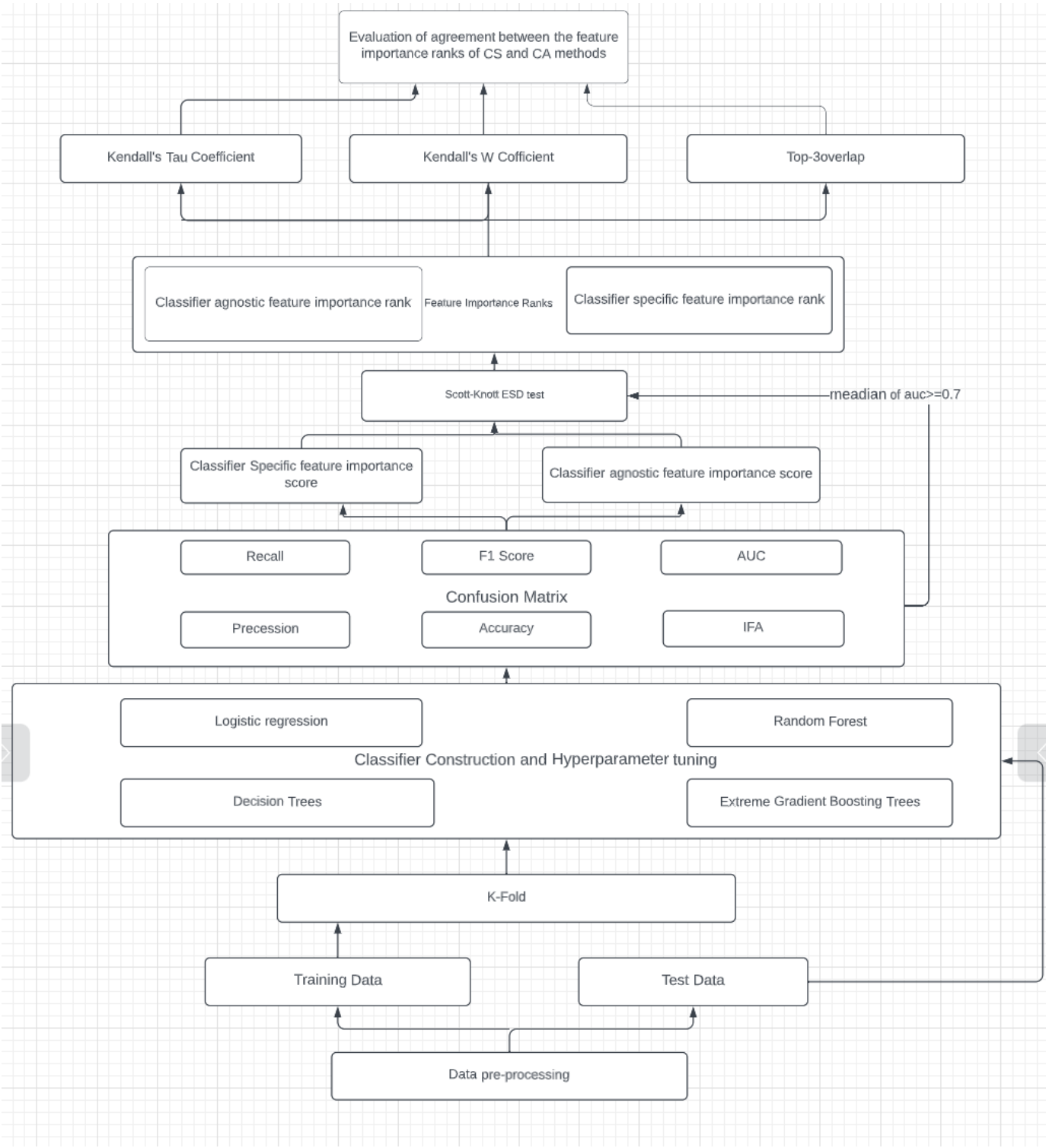
Kendall's W coefficient to measure the extent of agreement among multiple rank lists provided by different raters or classifiers.
Kendall's W ranges from 0 to 1, with
1 indicating perfect agreement and
0 indicating perfect disagreement.

In this case, Kendall's W is used to estimate the extent to which different feature importance ranks computed by CS methods agree across all the studied classifiers for a given dataset.
The interpretation scheme for Kendall's W is the same as that used for Kendall's tau, with values closer to 1 indicating strong agreement and values closer to 0 indicating weak agreement.
In our project we perform kendall's w among the ranks lists of the CS method for the three different classifiers.

3.2.11 Top-k overlap:

Top-k overlap metric, which measures the amount of overlap between features in the top-k ranks of multiple feature importance rank lists.
Mostly the value of k is taken as 3 and 1.
Top-3 overlap metric, which measures the amount of overlap between features in the top-3 ranks of multiple feature importance rank lists.
The Top-3 overlap metric does not consider the order of features in the top-3 ranks, only whether they appear in all of the lists.
The Top-3 overlap is adapted from the Jaccard Index for measuring similarity. The formula for computing the Top-3 overlap is shown, where k is set to 3 and n is the number of feature lists to compare.
The interpretation scheme for Top-3 overlap defines the degree of agreement between the feature lists.
The interpretation scheme categorizes Top-3 agreement as negligible if the overlap is between 0.00 and 0.25, small if the overlap is between 0.25 and 0.50, medium if the overlap is between 0.50 and 0.75, and large if the overlap is between 0.75 and 1.00.
This interpretation scheme is used to enable easier interpretation of the results obtained from the Top-3 overlap metric.
In our project we perform top-k overlap among the CA and CS rank lists.

3.3 Architecture:



Route map of the project

Chapter 4

Coding

4.1 Loading the dataset:

Loading a dataset involves reading the dataset file(s) from the storage device into the memory of the machine learning application or program. This is a necessary step before any further processing or analysis can be done on the data.

```
import numpy as np
import pandas as pd
```

```
data=pd.read_csv("C:/Users/hp/Downloads/pc5 data.csv.xls")
data
```

Figure 4.1: loading the data set

4.2 Correlation analysis implementation:

we perform correlation analysis to remove the highly correlated features among the independent features because the presence of the correlated features will yields unstable feature importance ranks. For the analysis we use AutoSpearman technique, we compute correlation matrix and remove the highly correlated related features. Here we considered the features with correlation coefficient more than 0.8 as highly correlated features.

```
corr_matrix = df.corr(method='spearman')
corr_matrix
```

```
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
to_drop = [column for column in upper.columns if any(upper[column] > 0.8)]
df = df.drop(df[to_drop], axis=1)
df
```

Figure 4.2: correlation analysis implementation

4.3 Splitting the data implementation:

Splitting the data refers to the process of dividing a dataset into two or more subsets, usually for the purpose of training and evaluating a machine learning model.The most common way to split a dataset is to divide it into two subsets: a training set and a testing set. The training set is used to train the model, while the testing set is used to evaluate the performance of the model on new, unseen data.

```
from sklearn.model_selection import train_test_split
```

```
X_sample = x
y_sample = y
X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample, test_size=0.2, random_state=42)
```

Figure 4.3: splitting the data implementation

4.4 K-fold cross validation implementation:

K-Fold Cross-Validation is a technique for evaluating the performance of machine learning models by dividing the dataset into k subsets or ”folds”. Each fold is used as a test set exactly once, while the k-1 remaining folds are used as training data. we calculated the scores of the models and each model’s scores mean and standard deviation using the K-fold technique.

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import statistics
```

```
import xgboost as xgb
```

```
cv = KFold(n_splits=10, random_state=1, shuffle=True)
```

```
model=xgb.XGBClassifier()
```

```
score = cross_val_score(model, x, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('score:',(score))
```

Figure 4.4: k-fold cross validation implementation

4.5 Classifier construction and Hyperparameter tuning implementation:

For Classifier construction and hyper parameter tuning we took

-randomforest classifier

-logistic regression

-Decision tree classifier

-xgb tree classifier.

Firstly, we imported classifiers from scikit library and then we fit the classifier to the training data using fit method. Here we are presenting the Classifier construction and hyperparameter tuning of XGB tree classifier.

```
import xgboost as xgb
```

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'subsample': [0.5, 0.75, 1.0],
    'colsample_bytree': [0.5, 0.75, 1.0],
    'gamma': [0, 0.1, 1],
    'min_child_weight': [1, 3, 5]
}
```

```
model=xgb.XGBClassifier()
search = RandomizedSearchCV(model, param_grid, cv=5, scoring='accuracy')
```

```
search.fit(X_train, y_train)
```

Figure 4.5: xgb hyperparameter tuning

Now we tuned the hyperparameters and fitted the best hypeparameters

```
best_params = search.best_params_
print("Best hyperparameters:", best_params)
```

```
xgb=xgb.XGBClassifier(**best_params)
xgb.fit(X_train, y_train)
```

Figure 4.6: fitting best xgb hyperparameters

After fitting the best hyerparameters we predict the y values and calculate the accuracy.

4.6 Confusion Matrix implementation:

A confusion matrix is a table used to evaluate the performance of a classifier. It is used to compare the predicted class labels of a classifier to the true class labels of the data.
Using confusion matrix we calculate Recall,Precesion,F1 score and accuracy.

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)
```

Figure 4.7: confusion matrix of y-test and y-pred

```
def calculate_metrics(cm):
    tp = cm[0][0]
    fp = cm[0][1]
    fn = cm[1][0]
    tn = cm[1][1]

    recall = tp / (tp + fn)
    precision = tp / (tp + fp)
    f1_score = 2 * (precision * recall) / (precision + recall)
    accuracy = (tp + tn) / (tp + fp + fn + tn)
    return recall, precision, f1_score, accuracy
```

```
recall, precision, f1_score, accuracy = calculate_metrics(cm)
print("Recall:", recall)
print("Precision:", precision)
print("F1 Score:", f1_score)
print("Accuracy:", accuracy)
```

Figure 4.8: performance metrics implementation

4.7 AUC-ROC curve implementation:

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1.

```
from sklearn import metrics
pred_prob1 = xgb.predict_proba(X_test)
```

```
from sklearn.metrics import roc_curve
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
```

```
import matplotlib.pyplot as plt
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='xgb tree')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.show();
```

Figure 4.9: AUC-ROC curve implementation

The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis

4.8 Computing feature importance scores:

We computed the feature importance scores using Classifier specific and Classifier agnostic methods.

```
importance_scores = xgb.feature_importances_
print(importance_scores)
```

Figure 4.10: Feature importance scores computation using xgb classifier (CS method)

```
from sklearn.datasets import make_classification
from sklearn.inspection import permutation_importance
results = permutation_importance(xgb, x, y, scoring='accuracy')
importance = results.importances_mean
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
```

Figure 4.11: Feature importance scores computation using permutation method

```
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(x)
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

```
shap_sum = np.abs(shap_values).mean(axis=0)
importance_df = pd.DataFrame([x.columns.tolist(), shap_sum.tolist()]).T
importance_df.columns = ['column_name', 'shap_importance']
importance_df1 = importance_df.sort_values('shap_importance', ascending=False)
importance_df1
```

Figure 4.12: Feature importance scores computation using SHAP method

4.9 Scott-Knott ESD implementation:

The Scott-Knott Effect Size Difference (SK-ESD) test is a statistical method used to perform a post-hoc analysis on the results of a clustering or grouping algorithm. It is an extension of the Scott-Knott test that takes into account both the statistical significance and practical significance of the differences between the groups or clusters produced by the algorithm.

We implemented the algorithm of the Scott-Knott Effect Size Difference (SK-ESD) test and computed the feature importance ranks for feature using feature importance scores.

```
def skesd_ranking(scores):

    if not scores:
        return {}
    sorted_scores = sorted(scores.values(), reverse=True)
    groups = [[score] for score in sorted_scores]
    min_ssd = float('inf')
    for k in range(2, len(sorted_scores)):
        new_groups = [[] for i in range(k)]
        for i, score in enumerate(sorted_scores):
            new_groups[i % k].append(score)
        group_means = [sum(group)/len(group) for group in new_groups]
        ssd = sum([sum([(score - mean)**2 for score in group]) for group, mean in zip(new_groups, group_means)])
        if ssd < min_ssd:
            min_ssd = ssd
            groups = new_groups
    ranked_groups = [sorted(group, reverse=True) for group in groups]
    ranked_groups.sort(key=lambda group: sum(group)/len(group), reverse=True)
    rank_dict = {}
    for i, group in enumerate(ranked_groups):
        for score in group:
            for feature, feature_score in scores.items():
                if feature_score == score:
                    rank_dict[feature] = i + 1

    return rank_dict
```

Figure 4.13: Scott-knott ESD implementation

To perform SK-ESD the mean AUC score of the classifier mode must be greater than 0.7 .In our project the mean AUC score of classifiers random forest, logistic regression and xgbtree are greater than 0.7 so we perform SK-ESD on those model fitted data but the

decision tree classifier's mean AUC score is less than 0.7 so we will not perform SK-ESD on the decision tree model fitted data.

4.10 Computing feature importance ranks:

By using the scott-knott ESD algorithm we compute the three different feature importance ranks .one is CS rank list and other two are CA rank lists.

```
import pandas as pd
score_dict ={'LOC_BLANK': 0.058978,'BRANCH_COUNT': 0.138004,'CALL_PAIRS': 0.051716,'LOC_CODE_AND_COMMENT': 0.077731,'LOC_CO
feature_ranks = skesd_ranking(score_dict)
feature_ranks
```

Figure 4.14: Computation of CS rank list

```
import pandas as pd
score_dict ={ 'LOC_BLANK': 0.041964,'BRANCH_COUNT': 0.019404,'CALL_PAIRS': 0.028404,'LOC_CODE_AND_COMMENT': 0.052718,'LOC_C
feature_ranks = skesd_ranking(score_dict)
feature_ranks
```

Figure 4.15: Computation of CA1 rank list

```
import pandas as pd
score_dict ={'LOC_BLANK': 0.226646, 'BRANCH_COUNT': 0.147758, 'CALL_PAIRS': 0.131575, 'LOC_CODE_AND_COMMENT': 0.244355, 'LOC_
feature_ranks = skesd_ranking(score_dict)
feature_ranks
```

Figure 4.16: Computation of CA2 rank list

4.11 Kendall's tau implementation:

We perform kendall'tau to see the agreement between the rank lists.

```
from scipy.stats import kendalltau
rank_list_1 = [5,1,9,4,13,8,3,12,2,3,6,7,10,11]
rank_list_2 = [1,11,8,9,12,6,2,7,13,9,5,3,10,4]
tau, p_value  = kendalltau(rank_list_1, rank_list_2)
print("Kendall's Tau: ", tau)
```

Figure 4.17: Kendall's tau for CS and CA1 ranks

```
from scipy.stats import kendalltau
rank_list_1 = [5,1,9,4,13,8,3,12,2,3,6,7,10,11]
rank_list_2 = [4,7,9,2,13,6,5,11,10,5,3,1,12,8]
tau, p_value = kendalltau(rank_list_1, rank_list_2)
print("Kendall's Tau: ", tau)
```

Figure 4.18: Kendall's tau for CS and CA2 ranks

```
from scipy.stats import kendalltau
rank_list_1 =[1,11,8,9,12,6,2,7,13,9,5,3,10,4]
rank_list_2 = [4,7,9,2,13,6,5,11,10,5,3,1,12,8]
tau, p_value = kendalltau(rank_list_1, rank_list_2)
print("Kendall's Tau: ", tau)
```

Figure 4.19: Kendall's tau for CA1 and CA2 ranks

4.12 Kendall's w implementation:

Kendall's W is used to estimate the extent to which different feature importance ranks computed by CS methods agree across all the studied classifiers for a given dataset.

```
import numpy as np
import scipy.stats as stats
rank_lists = [[6,2,8,10,7,5,4,13,9,10,12,1,11,3],
              [5,1,9,4,13,8,3,12,2,3,6,7,10,11],
             [10,8,5,4,7,11,1,6,3,4,2,12,13,9]]
rank_lists = np.array(rank_lists)
num_classifiers, num_features = rank_lists.shape
mean_rank = np.mean(rank_lists, axis=0)
ss_within = np.sum((rank_lists - mean_rank)**2, axis=1)
ss_total = np.sum((rank_lists - np.mean(rank_lists))**2)
W = 12 * ss_within / (num_features**2 * (num_classifiers**3 - num_classifiers))
print("Kendall's W: ", np.round(np.mean(W), 3))
```

Figure 4.20: Kendall's w for CS ranks imlementation

4.13 Top-k overlap implementation:

Top-k overlap metric, which measures the amount of overlap between features in the top-k ranks of multiple feature importance rank lists. Mostly the value of k is taken as 3 and 1.

This is the top-3 overlap of the three different CS feature importance rank lists. Similarly we performed the above implementation using random forest classifier, logistic regression and decision tree classifiers.

```python
def top_3_overlap(feature_lists):
    k = 3
    top_k_sets = [set(feature_list[:k]) for feature_list in feature_lists]
    intersection = set.intersection(*top_k_sets)
    union = set.union(*top_k_sets)
    top_3_overlap = len(intersection) / len(union)
    return top_3_overlap
```

```python
def top_3_agreement(top_3_overlap):
    if 0.00 <= top_3_overlap <= 0.25:
        return "negligible"
    elif 0.25 < top_3_overlap <= 0.50:
        return "small"
    elif 0.50 < top_3_overlap <= 0.75:
        return "medium"
    elif 0.75 < top_3_overlap <= 1.00:
        return "large"
```

```python
list1=[6,2,8,10,7,5,4,13,9,10,12,1,11,3]
list2=[5,1,9,4,13,8,3,12,2,3,6,7,10,11]
list3=[10,8,5,4,7,11,1,6,3,4,2,12,13,9]

feature_lists = [list1, list2, list3]
top_3_overlap_value = top_3_overlap(feature_lists)
top_3_agreement_value = top_3_agreement(top_3_overlap_value)
print(f"Top-3 overlap: {top_3_overlap_value}")
print(f"Top-3 agreement: {top_3_agreement_value}")
```

Figure 4.21: top-k overlap implementation

Chapter 5

Testing

5.1 Correlation analysis result:

After correlation analysis the highly correlated features are removed .so the 38 features are reduced to 14 features totally 24 features are removed.

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CYCLOMATIC_DENSITY	DE
0	3	11	2	1	4	0.19	
1	6	17	2	0	3	0.16	
2	0	11	5	3	6	0.19	
3	1	17	2	1	17	0.14	
4	7	17	4	0	7	0.23	
...	...	...	...	...	...	...	
1706	0	1	1	0	0	0.50	
1707	0	1	0	0	0	0.20	
1708	1	33	0	4	11	0.39	
1709	20	33	3	8	16	0.14	
1710	2	1	1	0	1	0.33	

1711 rows × 14 columns

Figure 5.1: Dataset after correlation analysis

5.2 Results of confusion matrix:

	Random forest	Logistic regression	Decision tree	XGB tree
True positive	215	225	219	216
False positive	23	13	19	22
True negative	34	16	21	41
False negative	71	89	84	64
Recall	0.751	0.716	0.722	0.771
Precision	0.903	0.945	0.920	0.907
F1 score	0.820	0.815	0.809	0.833
Accuracy	0.725	0.702	0.699	0.749

Figure 5.2: results of performance metrics of all classifiers

5.3 AUC-ROC Curve results:

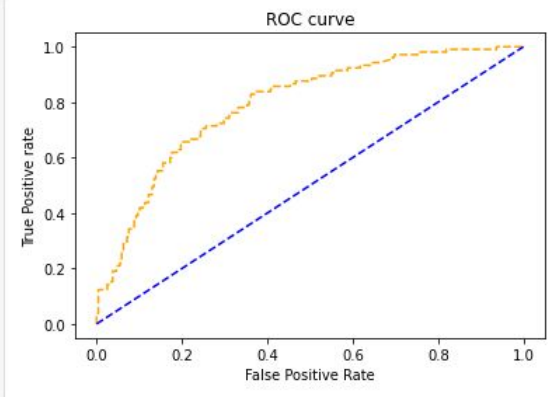


Figure 5.3: AUC-ROC curve of XGB Tree Classifier

The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis. Based on the roc curve we can observe that the predictions are in true positive region that means it has least occurance of type 1 error.

5.4 Feature importance rank lists results:

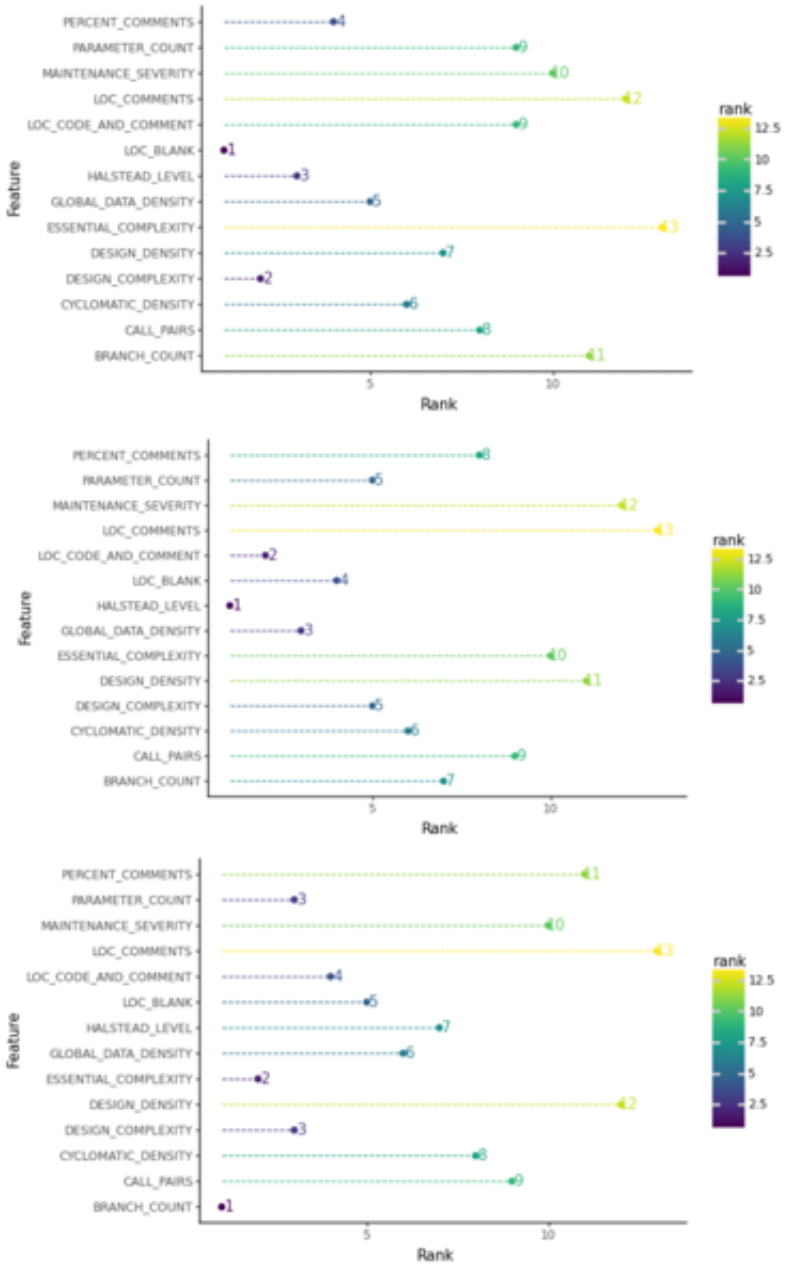


Figure 5.4: feature imortance ranks of cs and ca methods of xgb classifier.

These are the rank lists of different ca and cs methods.

5.5 Kendall's tau results:

Kendall's tau coefficient as a non-parametric rank correlation statistic to measure similarity between two rank lists.

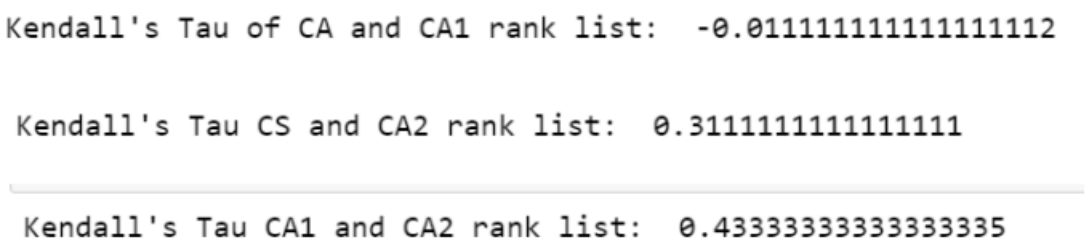


Figure 5.5: Kendall's Tau of rank lists

5.6 Kendall's w and Top-k overlap results:

Kendall's W coefficient to measure the extent of agreement among multiple rank lists provided by different raters or classifiers.
Kendall's W ranges from 0 to 1, with
1 indicating perfect agreement and
0 indicating perfect disagreement.
In this case, Kendall's W is used to estimate the extent to which different feature importance ranks computed by CS methods agree across all the studied classifiers for a given dataset.
Top-k overlap metric, which measures the amount of overlap between features in the top-k ranks of multiple feature importance rank lists. Mostly the value of k is taken as 3 and 1.

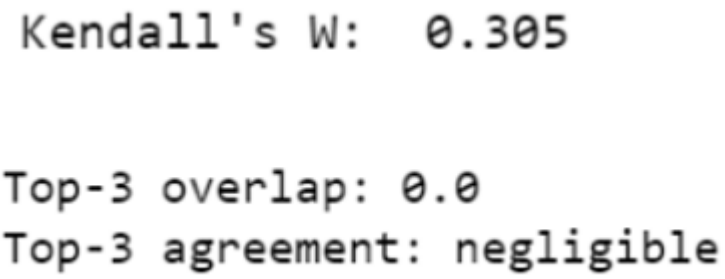


Figure 5.6: Kendall's w of rank lists and Top-3 overlap of CS rank lists

Chapter 6

Conclusion

6.1 Conclusion:

In order to extract insights from data, classifiers are used more and more. Insights are typically produced from the feature importance ranks that are determined by CS or CA computation techniques. Although, to obtain those insights, one can select between the CS and CA methods. Even for the same classifier, remains arbitrary.

It is rarely justified to choose the precise feature-important method. In Several earlier studies have employed feature importance methods, in other words interchangeably without any clear justification, despite the fact that different methods differ in how they calculate the feature importance rankings.

As a result, we wanted to determine how much the CS and CA-generated feature importance rankings agree each other.

From our project of The impact of feature importance methods on the interpretation of defect classifiers, we observe that while

the computed feature importance ranks by different CA methods agree with each other than

the computed feature importance ranks of CS method and CA methods.

The computed feature importance ranks of CS methods with different classifiers do not have a strong agreement.

Chapter 7

Bibliography

7.1 Bibliography

[1] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on the interpretation of defect models," IEEE Transactions on Software Engineering, 2019.

[2] T. Mori and N. Uchihira, "Balancing the trade-off between accuracy and interpretability in software defect prediction," Empirical Software Engineering, vol. 24, no. 2, pp. 779–825, 2019.

[3] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "The impact of using regression models to build defect classifiers," in 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017, pp. 135–145

[4] M. A. Hall, "Correlation-based feature selection for machine learning," 1999

[5] C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn, "Bias in random forest variable importance measures: Illustrations, sources and a solution," BMC bioinformatics, vol. 8, no. 1, p. 25, 2007

[6] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of modelagnostic techniques for defect prediction models," IEEE Transactions on Software Engineering, 2020.

[7] A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis, "Conditional variable importance for random forests," BMC bioinformatics, vol. 9, no. 1, p. 307, 2008.

[8] U. Gromping, "Variable importance assessment in regression: linear ¨ regression versus random forest," The American Statistician, vol. 63, no. 4, pp. 308–319, 2009.

[9] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "Autospearman: Automatically mitigating correlated software metrics for interpreting defect models," in 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE Computer Society, 2018, pp. 92–103.

[10] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," IEEE Transaction

[11] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press, 2015, pp. 789–800.

[12] S. Janitza, C. Strobl, and A.-L. Boulesteix, "An auc-based permutation variable importance measure for random forests," BMC bioinformatics, vol. 14, no. 1, p. 119, 2013.

[13] J. Jiarpakdee, "Towards a more reliable interpretation of defect models," in 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSECompanion). IEEE, 2019, pp. 210–213.

[14] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "The impact of automated feature selection techniques on the interpretation of defect models," Empirical Software Engineering, vol. 25, no. 5, pp. 3590– 3638, 2020.

[15] L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, pp. 5–32, 2001.

[16] o, A. Veloso, M. Viggiato, and N. Ziviani, "Understanding machine learning software defect predictions," Automated Software Engineering, vol. 27, no. 3, pp. 369–392, 2020.

[17] The scott-knott effect size difference (esd) test," R package version, vol. 2, 2016

[18] M. G. Kendall, "Rank correlation methods." 1948.

[19] C. Halimu, A. Kasem, and S. S. Newaz, "Empirical comparison of area under roc curve (auc) and mathew correlation coefficient (mcc) for evaluating machine learning algorithms on imbalanced datasets for binary classification," in Proceedings of the 3rd international conference on machine learning and soft computing, 2019, pp. 1–6