Download two datasets: JDT.csv and PDE.csv. They are bug datasets of two projects JDT and PDE. Two datasets having the same column names:

**file**: name of the class in this project. This column should be used as row names in R.

**bug**: number of post-release bugs found in that class. If bug = 0 then the class is considered defect free. Otherwise, it is defect prone.

**bf**: number of previous bug fixes on this class.

**loc**: number of lines of code of this class.

**hcm**: a measurement of the complexity of the changes applied to this class in its history.

In this assignment, you will compare logistic regression and decision tree on classification tasks. You also compare decision tree and linear regression on regression tasks. From previous assignments, we learn that using both **loc** and **bf** as input variables provide the best prediction. Therefore, we will use those two variables for all models in this assignment.

You should add a new column **defect** to each dataset. If a file has bugs (bug > 0) then **defect** is 1. Otherwise it is 0.

To answer the below questions, we need to set the directory where the “JDT.csv”, and “PDE.csv” files are located.

In my local machine, the code to set the working directory is

setwd(“C:/Users/cscha/OneDrive/Desktop/Software analytics”)

Now read the “JDT.csv” file by typing the following code

> data1 <- read.csv(“JDT.csv”)

Here, “data1” works as a variable to perform operations on the “JDT.csv” file

Now read the “PDE.csv” file by typing the following code

> data2 <- read.csv(“PDE.csv”)

Here, “data2” works as a variable to perform operations on the “PDE.csv” file

Q1 (6 pt) Build a logistic regression model M1 and a decision tree T1 to predict **defect** using both **loc** and **bf**. Because both models output as probabilities, you to find the optimal cutoff probability threshold.

a. Train M1 on JDT and also test it on JDT. Calculate recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9. Find the threshold with highest f-score. Do that similarly on PDE.

The ‘R’ code to build a logistic regression model M1 to predict defect using both loc and by training the JDT dataset and to calculate the recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9 along with threshold with highest f-score is as follows

> library(caret)

> data1 <- read.csv("JDT.csv")

> data1$defect <- as.factor(data1$defect)

> M1\_data1\_model <- train(defect ~ loc + bf, data = data1, method = "glm", family = "binomial")

> data1\_probability <- predict(M1\_data1\_model, data1, type = "prob")[, 2]

> cutoff\_sequence <- seq(0.1, 0.9, by = 0.1)

> data1\_metrics <- data.frame(Cutoff = numeric(), Recall = numeric(), Precision = numeric(), Accuracy = numeric(), F\_Score = numeric())

> for (cutoff in cutoff\_sequence) {

+ data1$Predicted\_Defect <- ifelse (data1\_probability > cutoff, 1, 0)

+ data1$Predicted\_Defect <- factor (data1$Predicted\_Defect, levels = levels(data1$defect))

+ resultant\_matrix\_data1 <- confusionMatrix (data = data1$Predicted\_Defect, reference = data1$defect)

+ data1\_recall <- resultant\_matrix\_data1$byClass["Recall"]

+ data1\_precision <- resultant\_matrix\_data1$byClass["Precision"]

+ data1\_accuracy <- sum(data1$Predicted\_Defect == data1$defect) / nrow(data1)

+ data1\_f\_score <- 2 \* (data1\_precision \* data1\_recall) / (data1\_precision + data1\_recall)

+ data1\_metrics <- rbind (data1\_metrics, data.frame(Cutoff = cutoff, Recall = data1\_recall, Precision = data1\_precision, Accuracy = data1\_accuracy, F\_Score = data1\_f\_score))

+ }

> highest\_threshold\_data1 <- data1\_metrics[which.max(data1\_metrics$F\_Score),]

> print(highest\_threshold\_data1)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to build a logistic regression model. Later loading the caret library we then consider the defect column which is newly created in the JDT dataset and we consider the defect factor to classify whether it is defective (1) or not(0). After that, we now create a logistic regression model named M1\_data1\_model where the goal is to predict the defect using loc and bf as predictor variables. Here in creating the logistic regression model we use the generalized linear model and implement a binomial family for the binary classification. Later we consider the predicted probabilities of the class (defect) on focusing each and every observation in the JDT.csv dataset. We store the respective values in a variable named data1\_probability. Later we mention the sequence of the probability cutoff threshold according to the given constraint. Here the sequence starts from 0.1 and ends at 0.9 by having the difference of 0.1. Later Performance metrics are initialized in an empty data frame named data1\_metrics to hold various cutoff levels. There are five columns: "Cutoff," "Recall," "Precision," "Accuracy," and "F-Score.". Now we iterate through a for loop for the below lines of code logic where at first, we classify each and every observation of the data1 dataset on the defect column to note whether the observation is defect or non-defect based on the condition where the predicted probability exceeds the cutoff threshold and store the values in a variable named data1$Predicted\_Defect. Later for accurate performance, we make sure that the level of defect and data1$Predicted\_Defect are matched. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix\_data1 to assess the generated model’s classification performance. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the metric with all calculated and extracted values to a data frame to the variable named data1\_metrices. Later we exit from the for loop and consider the highest F\_Score from the data1\_metrices and store the associated metrics of the highest F-Score in a variable called highest\_threshold\_data1 and print the required output.

The output:

Cutoff Recall Precision Accuracy F\_Score

Recall4 0.5 0.9683944 0.8558659 0.8455366 0.9086595

From the above output we state that a threshold with a value of 0.5 has the highest F\_Score of 0.9086595 of a logistic regression model named M1\_data1\_model which was created on given criteria.

The ‘R’ code to build a logistic regression model M1 to predict defect using both loc and by training the PDE dataset and to calculate the recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9 along with threshold with highest f-score is as follows

> data2 <- read.csv("PDE.csv")

> data2$defect <- as.factor(data2$defect)

> M1\_data2\_model <- train(defect ~ loc + bf, data = data2, method = "glm", family = "binomial")

> data2\_probability <- predict(M1\_data2\_model, data2, type = "prob")[, 2]

> cutoff\_sequence <- seq(0.1, 0.9, by = 0.1)

> data2\_metrics <- data.frame(Cutoff = numeric(), Recall = numeric(), Precision = numeric(), Accuracy = numeric(), F\_Score = numeric())

>

> for (cutoff in cutoff\_sequence) {

+ data2$Predicted\_Defect <- ifelse(data2\_probability > cutoff, 1, 0)

+ data2$Predicted\_Defect <- factor(data2$Predicted\_Defect, levels = levels(data2$defect))

+ resultant\_matrix\_data2 <- confusionMatrix(data = data2$Predicted\_Defect, reference = data2$defect)

+ data2\_recall <- resultant\_matrix\_data2$byClass["Recall"]

+ data2\_precision <- resultant\_matrix\_data2$byClass["Precision"]

+ data2\_accuracy <- sum(data2$Predicted\_Defect == data2$defect) / nrow(data2)

+ data2\_f\_score <- 2 \* (data2\_precision \* data2\_recall) / (data2\_precision + data2\_recall)

+ data2\_metrics <- rbind(data2\_metrics, data.frame(Cutoff = cutoff, Recall = data2\_recall, Precision = data2\_precision, Accuracy = data2\_accuracy, F\_Score = data2\_f\_score))

+ }

> highest\_threshold\_data2 <- data2\_metrics[which.max(data2\_metrics$F\_Score),]

> print(highest\_threshold\_data2)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to build a logistic regression model. Later loading the caret library we then consider the defect column which is newly created in the PDE dataset and we consider the defect factor to classify whether it is defective (1) or not(0). After that, we now create a logistic regression model named M1\_data2\_model where the goal is to predict the defect using loc and bf as predictor variables. Here in creating the logistic regression model we use the generalized linear model and implement a binomial family for the binary classification. Later we consider the predicted probabilities of the class (defect) on focusing each and every observation in the PDE.csv dataset. We store the respective values in a variable named data2\_probability. Later we mention the sequence of the probability cutoff threshold according to the given constraint. Here the sequence starts from 0.1 and ends at 0.9 by having the difference of 0.1. Later Performance metrics are initialized in an empty data frame named data2\_metrics to hold various cutoff levels. There are five columns: "Cutoff," "Recall," "Precision," "Accuracy," and "F-Score.". Now we iterate through a for loop for the below lines of code logic where at first, we classify each and every observation of the data2 dataset on the defect column to note whether the observation is defect or non-defect based on the condition where the predicted probability exceeds the cutoff threshold and store the values in a variable named data2$Predicted\_Defect. Later for accurate performance, we make sure that the level of defect and data2$Predicted\_Defect are matched. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix\_data2 to assess the generated model’s classification performance. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the metric with all calculated and extracted values to a data frame to the variable named data2\_metrices. Later we exit from the for loop and consider the highest F\_Score from the data2\_metrices and store the associated metrics of the highest F-Score in a variable called highest\_threshold\_data2 and print the required output.

The output:

Cutoff Recall Precision Accuracy F\_Score

Recall6 0.7 0.9953416 0.8691525 0.8670675 0.9279768

From the above output, we state that a threshold with a value of 0.7 has the highest F\_Score of 0.9279768 of a logistic regression model named M1\_data2\_model which was created on given criteria.

b) Train M1 on JDT and also test it on PDE. Calculate recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9. Find the threshold with highest f-score. Do that similarly with training on PDE and testing on JDT

The ‘R’ code to build a logistic regression model M1 to predict defect using both loc and by training the JDT dataset and testing on PDE to calculate the recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9 along with threshold with highest f-score is as follows

> library(caret)

> data1 <- read.csv("JDT.csv")

> data2 <- read.csv("PDE.csv")

> data1$defect <- as.factor(data1$defect)

> M1\_model\_train\_data1 <- train (defect ~ loc + bf, data = data1, method = "glm", family = "binomial")

> data2$defect <- as. factor(data2$defect)

> data2\_prob <- predict (M1\_model\_train\_data1, data2, type = "prob")[, 2]

> cutoff\_sequence <- seq (0.1, 0.9, by = 0.1)

> output\_metrics <- data.frame(Cutoff = numeric (), Recall = numeric(), Precision = numeric(), Accuracy = numeric(), F\_Score = numeric())

> for (cutoff in cutoff\_sequence) {

+

+ data2$Predicted\_Defect <- ifelse (data2\_prob > cutoff, 1, 0)

+ data2$Predicted\_Defect <- factor (data2$Predicted\_Defect, levels = levels(data2$defect))

+ resultant\_matrix <- confusionMatrix (data = data2$Predicted\_Defect, reference = data2$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum (data2$Predicted\_Defect == data2$defect) / nrow(data2)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+

+ output\_metrics <- rbind (output\_metrics, data.frame(Cutoff = cutoff, Recall = recall, Precision = precision, Accuracy = accuracy, F\_Score = f\_score))

+ }

> highest\_threshold <- output\_metrics[which.max(output\_metrics$F\_Score),]

> print(highest\_threshold)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to build a logistic regression model. Later loading the caret library we then consider the defect column which is newly created in the JDT dataset and we consider the defect factor to classify whether it is defective (1) or not(0). After that, we now create a logistic regression model named M1\_model\_train\_data1 by considering the JDT.csv where the goal is to predict the defect using loc and bf as predictor variables. Here in creating the logistic regression model we use the generalized linear model and implement a binomial family for the binary classification. Later we consider the predicted probabilities of the class (defect) on focusing each and every observation in the PDE.csv dataset. We store the respective values in a variable named data2\_prob. Later we mention the sequence of the probability cutoff threshold according to the given constraint. Here the sequence starts from 0.1 and ends at 0.9 by having the difference of 0.1. Later Performance metrics are initialized in an empty data frame named output\_metrics to hold various cutoff levels. There are five columns: "Cutoff," "Recall," "Precision," "Accuracy," and "F-Score.". Now we iterate through a for loop for the below lines of code logic where at first, we classify each and every observation of the PDE.csv dataset on the defect column to note whether the observation is defect or non-defect based on the condition where the predicted probability exceeds the cutoff threshold and store the values in a variable named data2$Predicted\_Defect. Later for accurate performance, we make sure that the level of defect and data2$Predicted\_Defect are matched. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the metric with all calculated and extracted values to a data frame to the variable named output\_metrices. Later we exit from the for loop and consider the highest F\_Score from the output\_metrics and store the associated metrics of the highest F-Score in a variable called highest\_threshold and print the required output.

The output:

Cutoff Recall Precision Accuracy F\_Score

Recall3 0.4 0.9953416 0.8703327 0.8684035 0.928649

From the above output, we state that a threshold with a value of 0.4 has the highest F\_Score of 0.928649 of a logistic regression model named M1\_model\_train\_data1 which was created on the given criteria.

The ‘R’ code to build a logistic regression model M1 to predict defect using both loc and by training the PDE dataset and testing on JDT to calculate the recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9 along with threshold with highest f-score is as follows:

> library(caret)

> data1 <- read.csv("JDT.csv")

> data2 <- read.csv("PDE.csv")

> data2$defect <- as.factor(data2$defect)

> M1\_model\_train\_data2 <- train(defect ~ loc + bf, data = data2, method = "glm", family = "binomial")

> data1$defect <- as.factor(data1$defect)

> data1\_prob <- predict(M1\_model\_train\_data2, data1, type = "prob")[, 2]

> cutoff\_sequence <- seq(0.1, 0.9, by = 0.1)

> output\_metrics <- data.frame(Cutoff = numeric(), Recall = numeric(), Precision = numeric(), Accuracy = numeric(), F\_Score = numeric())

> for (cutoff in cutoff\_sequence) {

+ data1$Predicted\_Defect <- ifelse(data1\_prob > cutoff, 1, 0)

+ data1$Predicted\_Defect <- factor(data1$Predicted\_Defect, levels = levels(data1$defect))

+ resultant\_matrix <- confusionMatrix(data = data1$Predicted\_Defect, reference = data1$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data1$Predicted\_Defect == data1$defect) / nrow(data1)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ output\_metrics <- rbind(output\_metrics, data.frame(Cutoff = cutoff, Recall = recall, Precision = precision, Accuracy = accuracy, F\_Score = f\_score))

+ }

> highest\_threshold <- output\_metrics[which.max(output\_metrics$F\_Score),]

> print(highest\_threshold)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to build a logistic regression model. Later loading the caret library we then consider the defect column which is newly created in the PDE dataset and we consider the defect factor to classify whether it is defective (1) or not(0). After that, we now create a logistic regression model named M1\_model\_train\_data2 by considering the PDE.csv where the goal is to predict the defect using loc and bf as predictor variables. Here in creating the logistic regression model we use the generalized linear model and implement a binomial family for the binary classification. Later we consider the predicted probabilities of the class (defect) on focusing each and every observation in the

JDT .csv dataset. We store the respective values in a variable named data1\_prob. Later we mention the sequence of the probability cutoff threshold according to the given constraint. Here the sequence starts from 0.1 and ends at 0.9 by having the difference of 0.1. Later Performance metrics are initialized in an empty data frame named output\_metrics to hold various cutoff levels. There are five columns: "Cutoff," "Recall," "Precision," "Accuracy," and "F-Score.". Now we iterate through a for loop for the below lines of code logic where at first, we classify each and every observation of the JDT.csv dataset on the defect column to note whether the observation is defect or non-defect based on the condition where the predicted probability exceeds the cutoff threshold and store the values in a variable named data1$Predicted\_Defect. Later for accurate performance, we make sure that the level of defect and data1$Predicted\_Defect are matched. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the metric with all calculated and extracted values to a data frame to the variable named output\_metrices. Later we exit from the for loop and consider the highest F\_Score from the output\_metrics and store the associated metrics of the highest F-Score in a variable called highest\_threshold and print the required output.

The output:

Cutoff Recall Precision Accuracy F\_Score

Recall7 0.8 0.9696587 0.8522222 0.8425276 0.9071555

From the above output, we state that a threshold with a value of 0.8 has the highest F\_Score of 0.9071555 of a logistic regression model named M1\_model\_train\_data2 which was created on the given criteria.

c) . Perform similarly in a) and b) with T1. Compare the results with M1

The ‘R’ code to build a decision tree T1 to predict defect using both loc and by training the JDT dataset and to calculate the recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9 along with threshold with highest f-score is as follows

> library(caret)

> data1 <- read.csv("JDT.csv")

> data1$defect <- as.factor(data1$defect)

> T1\_model\_data1 <- train(defect ~ loc + bf, data = data1, method = "rpart")

> cutoff\_sequence <- seq(0.1, 0.9, by = 0.1)

> output\_metrics <- data.frame(Cutoff = numeric(), Recall = numeric(), Precision = numeric(), Accuracy = numeric(), F\_Score = numeric())

> for (cutoff in cutoff\_sequence) {

+ data1\_prob <- predict(T1\_model\_data1, data1, type = "prob")[, 2]

+ data1$Predicted\_Defect <- ifelse(data1\_prob > cutoff, 1, 0)

+ data1$Predicted\_Defect <- factor(data1$Predicted\_Defect, levels = levels(data1$defect))

+ resultant\_matrix <- confusionMatrix(data = data1$Predicted\_Defect, reference = data1$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data1$Predicted\_Defect == data1$defect) / nrow(data1)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ output\_metrics <- rbind(output\_metrics, data.frame(Cutoff = cutoff, Recall = recall, Precision = precision, Accuracy = accuracy, F\_Score = f\_score))

+ }

> highest\_threshold <- output\_metrics[which.max(output\_metrics$F\_Score),]

> print(highest\_threshold)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to build a decision tree. Later loading the caret library we then consider the defect column which is newly created in the JDT dataset and we consider the defect factor to classify whether it is defective (1) or not(0). After that, we now create a decision tree named T1\_model\_data1 where the goal is to predict the defect using loc and bf as predictor variables. Here in creating the decision tree, we use the method as rpart which is known as Recursive Partitioning which is the starting point for creating machine learning algorithms such as decision tree. Later we mention the sequence of the probability cutoff threshold according to the given constraint. Here the sequence starts from 0.1 and ends at 0.9 by having the difference of 0.1. Later Performance metrics are initialized in an empty data frame named output\_metrics to hold various cutoff levels. There are five columns: "Cutoff," "Recall," "Precision," "Accuracy," and "F-Score.". Now we iterate through a for loop for the below lines of code logic where at first, we consider the predicted probabilities of the class (defect) on focusing each and every observation in the JDT.csv dataset. We store the respective values in a variable named data1\_prob and now we classify each and every observation of the data1 dataset on the defect column to note whether the observation is defect or non-defect based on the condition where the predicted probability exceeds the cutoff threshold and store the values in a variable named data1$Predicted\_Defect. Later for accurate performance, we make sure that the level of defect and data1$Predicted\_Defect are matched. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the metric with all calculated and extracted values to a data frame to the variable named output\_metrices. Later we exit from the for loop and consider the highest F\_Score from the output\_metrices and store the associated metrics of the highest F-Score in a variable called highest\_threshold and print the required output.

The output:

Cutoff Recall Precision Accuracy F\_Score

Recall1 0.2 0.9393173 0.8761792 0.8465396 0.9066504

From the above output, we state that a threshold with a value of 0.2 has the highest F\_Score of 0.9066504 of a decision tree named T1\_model\_data1 which was created on the given criteria.

By comparing the F\_Score the highest F\_Score was obtained in the case of training and testing the JDT set by building a logistic regression model rather than building a decision tree with a value of 0.9086595 and with the threshold of 0.5.

The ‘R’ code to build a decision tree T1 to predict defect using both loc and by training the PDE dataset and to calculate the recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9 along with threshold with highest f-score is as follows

> library(caret)

> data2 <- read.csv("PDE.csv")

> data2$defect <- as.factor(data2$defect)

> T1\_model\_train\_data2 <- train(defect ~ loc + bf, data = data2, method = "rpart")

> cutoff\_sequence <- seq(0.1, 0.9, by = 0.1)

> output\_metrics <- data.frame(Cutoff = numeric(), Recall = numeric(), Precision = numeric(), Accuracy = numeric(), F\_Score = numeric())

> for (cutoff in cutoff\_sequence) {

+ data2\_prob <- predict(T1\_model\_train\_data2, data2, type = "prob")[, 2]

+ data2$Predicted\_Defect <- ifelse(data2\_prob > cutoff, 1, 0)

+ data2$Predicted\_Defect <- factor(data2$Predicted\_Defect, levels = levels(data2$defect))

+ resultant\_matrix <- confusionMatrix(data = data2$Predicted\_Defect, reference = data2$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data2$Predicted\_Defect == data2$defect) / nrow(data2)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ output\_metrics <- rbind(output\_metrics, data.frame(Cutoff = cutoff, Recall = recall, Precision = precision, Accuracy = accuracy, F\_Score = f\_score))

+ }

> highest\_threshold <- output\_metrics[which.max(output\_metrics$F\_Score),]

> print(highest\_threshold)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to build a decision tree. Later loading the caret library we then consider the defect column which is newly created in the PDE dataset and we consider the defect factor to classify whether it is defective (1) or not(0). After that, we now create a decision tree named T1\_model\_train\_data2 where the goal is to predict the defect using loc and bf as predictor variables. Here in creating the decision tree, we use the method as rpart which is known as Recursive Partitioning which is the starting point for creating machine learning algorithms such as decision tree. Later we mention the sequence of the probability cutoff threshold according to the given constraint. Here the sequence starts from 0.1 and ends at 0.9 by having the difference of 0.1. Later Performance metrics are initialized in an empty data frame named output\_metrics to hold various cutoff levels. There are five columns: "Cutoff," "Recall," "Precision," "Accuracy," and "F-Score.". Now we iterate through a for loop for the below lines of code logic where at first, we consider the predicted probabilities of the class (defect) on focusing each and every observation in the PDE.csv dataset. We store the respective values in a variable named data2\_prob and now we classify each and every observation of the data2 dataset on the defect column to note whether the observation is defect or non-defect based on the condition where the predicted probability exceeds the cutoff threshold and store the values in a variable named data2$Predicted\_Defect. Later for accurate performance, we make sure that the level of defect and data2$Predicted\_Defect are matched. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the metric with all calculated and extracted values to a data frame to the variable named output\_metrices. Later we exit from the for loop and consider the highest F\_Score from the output\_metrices and store the associated metrics of the highest F-Score in a variable called highest\_threshold and print the required output.

The output:

Cutoff Recall Precision Accuracy F\_Score

Recall1 0.2 1 0.8603874 0.8603874 0.9249551

From the above output, we state that a threshold with a value of 0.2 has the highest F\_Score of 0.9249551 of a decision tree named T1\_model\_train\_data2 which was created on the given criteria.

By comparing the F\_Score the highest F\_Score was obtained in the case of training and testing the PDE dataset by building a logistic regression model rather than building a decision tree with a value of 0.9279768 and with the threshold of 0.7.

The ‘R’ code to build a decision tree T1 to predict defect using both loc and by training the JDT dataset and testing on PDE to calculate the recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9 along with threshold with highest f-score is as follows:

> library(caret)

> data1 <- read.csv("JDT.csv")

> data2 <- read.csv("PDE.csv")

> data1$defect <- as.factor(data1$defect)

> data2$defect <- as.factor(data2$defect)

> T1\_model\_train\_data1 <- train(defect ~ loc + bf, data = data1, method = "rpart")

> cutoff\_sequence <- seq(0.1, 0.9, by = 0.1)

> output\_metrics <- data.frame(Cutoff = numeric(), Recall = numeric(), Precision = numeric(), Accuracy = numeric(), F\_Score = numeric())

> for (cutoff in cutoff\_sequence) {

+ data2\_prob <- predict(T1\_model\_train\_data1, data2, type = "prob")[, 2]

+ data2$Predicted\_Defect <- ifelse(data2\_prob > cutoff, 1, 0)

+ data2$Predicted\_Defect <- factor(data2$Predicted\_Defect, levels = levels(data2$defect))

+ resultant\_matrix <- confusionMatrix(data = data2$Predicted\_Defect, reference = data2$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data2$Predicted\_Defect == data2$defect) / nrow(data2)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ output\_metrics <- rbind(output\_metrics, data.frame(Cutoff = cutoff, Recall = recall, Precision = precision, Accuracy = accuracy, F\_Score = f\_score))

+ }

> highest\_threshold <- output\_metrics[which.max(output\_metrics$F\_Score),]

> print(highest\_threshold)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to build a decision tree. Later loading the caret library we then consider the defect column which is newly created in the JDT dataset and we consider the defect factor to classify whether it is defective (1) or not(0).

After that, we now create a decision tree named T1\_model\_train\_data1 where the goal is to predict the defect using loc and bf as predictor variables. Here in creating the decision tree, we use the method as rpart which is known as Recursive Partitioning which is the starting point for creating machine learning algorithms such as decision tree. Later we mention the sequence of the probability cutoff threshold according to the given constraint. Here the sequence starts from 0.1 and ends at 0.9 by having the difference of 0.1. Later Performance metrics are initialized in an empty data frame named output\_metrics to hold various cutoff levels. There are five columns: "Cutoff," "Recall," "Precision," "Accuracy," and "F-Score.". Now we iterate through a for loop and we consider the predicted probabilities of the class (defect) on focusing each and every observation in the PDE.csv dataset. We store the respective values in a variable named data2\_prob later we classify each and every observation of the PDE.csv dataset on the defect column to note whether the observation is defect or non-defect based on the condition where the predicted probability exceeds the cutoff threshold and store the values in a variable named data2$Predicted\_Defect. Later for accurate performance, we make sure that the level of defect and data2$Predicted\_Defect are matched. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the metric with all calculated and extracted values to a data frame to the variable named output\_metrics. Later we exit from the for loop and consider the highest F\_Score from the output\_metrics and store the associated metrics of the highest F-Score in a variable called highest\_threshold and print the required output.

The output:

Cutoff Recall Precision Accuracy F\_Score

Recall1 0.2 0.992236 0.8705722 0.8663995 0.9274311

From the above output, we state that a threshold with a value of 0.2 has the highest F\_Score of 0.9274311 of a decision tree named T1\_model\_train\_data1 which was created on the given criteria.

By comparing the F\_Score the highest F\_Score was obtained in the case of training the JDT dataset and testing the PDE dataset by building a logistic regression model rather than building a decision tree with a value of 0.928649 and with a threshold of 0.4.

The ‘R’ code to build a decision tree T1 to predict defect using both loc and by training the PDE dataset and testing on JDT to calculate the recall, precision, accuracy, and f-score with multiple cutoff probability thresholds: 0.1 0.2 ... 0.9 along with threshold with highest f-score is as follows:

> library(caret)

> data1 <- read.csv("JDT.csv")

> data2 <- read.csv("PDE.csv")

> data1$defect <- as.factor(data1$defect)

> data2$defect <- as.factor(data2$defect)

> T1\_model\_train\_data2 <- train(defect ~ loc + bf, data = data2, method = "rpart")

> cutoff\_sequence <- seq(0.1, 0.9, by = 0.1)

> output\_metrics <- data.frame(Cutoff = numeric(), Recall = numeric(), Precision = numeric(), Accuracy = numeric(), F\_Score = numeric())

> for (cutoff in cutoff\_sequence) {

+ data1\_prob <- predict(T1\_model\_train\_data2, data1, type = "prob")[, 2]

+ data1$Predicted\_Defect <- ifelse(data1\_prob > cutoff, 1, 0)

+ data1$Predicted\_Defect <- factor(data1$Predicted\_Defect, levels = levels(data1$defect))

+ resultant\_matrix <- confusionMatrix(data = data1$Predicted\_Defect, reference = data1$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data1$Predicted\_Defect == data1$defect) / nrow(data1)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ output\_metrics <- rbind(output\_metrics, data.frame(Cutoff = cutoff, Recall = recall, Precision = precision, Accuracy = accuracy, F\_Score = f\_score))

+ }

> highest\_threshold <- output\_metrics[which.max(output\_metrics$F\_Score),]

> print(highest\_threshold)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to build a decision tree. Later loading the caret library we then consider the defect column which is newly created in the PDE dataset and we consider the defect factor to classify whether it is defective (1) or not(0).

After that, we now create a decision tree named T1\_model\_train\_data2 where the goal is to predict the defect using loc and bf as predictor variables. Here in creating the decision tree, we use the method as rpart which is known as Recursive Partitioning which is the starting point for creating machine learning algorithms such as decision tree. Later we mention the sequence of the probability cutoff threshold according to the given constraint. Here the sequence starts from 0.1 and ends at 0.9 by having the difference of 0.1. Later Performance metrics are initialized in an empty data frame named output\_metrics to hold various cutoff levels. There are five columns: "Cutoff," "Recall," "Precision," "Accuracy," and "F-Score.". Now we iterate through a for loop and we consider the predicted probabilities of the class (defect) on focusing each and every observation in the JDT.csv dataset. We store the respective values in a variable named data1\_prob later we classify each and every observation of the JDT.csv dataset on the defect column to note whether the observation is defect or non-defect based on the condition where the predicted probability exceeds the cutoff threshold and store the values in a variable named data1$Predicted\_Defect. Later for accurate performance, we make sure that the level of defect and data1$Predicted\_Defect are matched. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the metric with all calculated and extracted values to a data frame to the variable named output\_metrics. Later we exit from the for loop and consider the highest F\_Score from the output\_metrics and store the associated metrics of the highest F-Score in a variable called highest\_threshold and print the required output.

The output:

Cutoff Recall Precision Accuracy F\_Score

Recall1 0.2 1 0.7933801 0.7933801 0.8847875

From the above output, we state that a threshold with a value of 0.2 has the highest F\_Score of 0.8847875 of a decision tree named T1\_model\_train\_data2 which was created on the given criteria.

By comparing the F\_Score the highest F\_Score was obtained in the case of training the PDE dataset and testing the JDT dataset by building a logistic regression model rather than building a decision tree with a value of 0.9071555 and with a threshold of 0.8.

d) Perform cross-validation with M1 on JDT. That is, select randomly 50% of JDT for training M1 and test it with 50% remaining. Use the optimal threshold found on task a) to calculate the recall, precision, accuracy, and f-score. Repeat 30 times. Repeat this cross-validation with M1 on PDE.

The ‘R’ code to perform cross-validation with M1 on JDT. That is, select randomly 50% of JDT for training M1 and test it with 50% remaining. Use the optimal threshold found on task a) to calculate the recall, precision, accuracy, and f-score. Repeat 30 times is as follows:

> library(caret)

> data1 <- read.csv("JDT.csv")

> data1$defect <- as.factor(data1$defect)

> repetetion\_count <- 30

> recall\_vector <- numeric()

> precision\_vector <- numeric()

> accuracy\_vector <- numeric()

> f\_score\_vector <- numeric()

> M1\_data1\_model <- train(defect ~ loc + bf, data = data1, method = "glm", family = "binomial")

> optimal\_threshold <- 0.5

> for (i in 1:repetetion\_count) {

+ set.seed(i)

+ indices <- sample(1:nrow(data1), nrow(data1) / 2)

+ data1\_train <- data1[indices, ]

+ data1\_test <- data1[-indices, ]

+ data1\_test\_prob <- predict(M1\_data1\_model, data1\_test, type = "prob")[, 2]

+ data1\_test$Predicted\_Defect <- ifelse(data1\_test\_prob > optimal\_threshold, 1, 0)

+ data1\_test$Predicted\_Defect <- factor(data1\_test$Predicted\_Defect, levels = levels(data1\_test$defect))

+ resultant\_matrix <- confusionMatrix(data = data1\_test$Predicted\_Defect, reference = data1\_test$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data1\_test$Predicted\_Defect == data1\_test$defect) / nrow(data1\_test)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ recall\_vector <- c(recall\_vector, recall)

+ precision\_vector <- c(precision\_vector, precision)

+ accuracy\_vector <- c(accuracy\_vector, accuracy)

+ f\_score\_vector <- c(f\_score\_vector, f\_score)

+ }

> avg\_recall <- mean(recall\_vector)

> avg\_precision <- mean(precision\_vector)

> avg\_accuracy <- mean(accuracy\_vector)

> avg\_f\_score <- mean(f\_score\_vector)

> stddev\_recall <- sd(recall\_vector)

> stddev\_precision <- sd(precision\_vector)

> stddev\_accuracy <- sd(accuracy\_vector)

> stddev\_f\_score <- sd(f\_score\_vector)

> required\_output <- data.frame(

+ Metric = c("Recall", "Precision", "Accuracy", "F-Score"),

+ Average = c(avg\_recall, avg\_precision, avg\_accuracy, avg\_f\_score),

+ StandardDeviation = c(stddev\_recall, stddev\_precision, stddev\_accuracy, stddev\_f\_score)

+ )

> print(required\_output)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to perform cross-validation with M1 on the JDT.csv dataset. Later loading the caret library we then consider the defect column which is newly created in the JDT dataset and we consider the defect factor to classify whether it is defective (1) or not(0). Later we define the number of repetitions as 30 which was given as one of the constraints to evaluate the model's performance. Now we initialize the and create empty numeric vectors to store recall, precision, accuracy, and f\_score of the model respectively. Now we consider the logistic regression model named M1\_data1\_model which was trained on the JDT.csv dataset to predict defects using both loc and bf features from the JDT.csv dataset. Later we set the optimal threshold as 0.5 which was the threshold that has the highest f\_score which was obtained earlier by evaluating M1\_model which was trained and tested on the JDT.csv dataset. Now we iterate through a for-loop which iterates over a defined number of repetitions, and we set the random seed.

Later we randomly split the JDT dataset into training and testing which contains 50% of the data1 dataset for each iteration of a for-loop. Later we consider the trained model M1 to predict the probabilities of defects for the test set instances and the values are stored in a variable named data1\_test\_prob. Later, to classify flaws, we use the optimal threshold that has been specified. The instance is classed as having a defect (1) if the estimated likelihood of a defect exceeds the threshold; otherwise, it is labeled as not having a defect (0). After that, we convert the projected values into a factor in order to guarantee alignment with the real defect labels. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance which holds the values of Cutoff, Recall, Precision, Accuracy, and F\_Score. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the calculated values for the current cross-validation iteration to initialized vectors respectively. Now we exit from for-loop iteration and calculate the average and standard deviation of recall, precision, accuracy, and f\_score. Later we create a data frame called required\_output to display the mean and standard deviation of recall, accuracy, precision, and f\_score and we print the required\_output.

The output:

Metric Average StandardDeviation

1 Recall 0.9689093 0.006932561

2 Precision 0.8553723 0.014193163

3 Accuracy 0.8454910 0.012210904

4 F-Score 0.9085212 0.007772379

The ‘R’ code to perform cross-validation with M1 on PDE. That is, select randomly 50% of PDE for training M1 and test it with 50% remaining. Use the optimal threshold found on task a) to calculate the recall, precision, accuracy, and f-score. Repeat 30 times is as follows:

> library(caret)

> data2 <- read.csv("PDE.csv")

> data2$defect <- as.factor(data2$defect)

> repetition\_count <- 30

> recall\_vector <- numeric()

> precision\_vector <- numeric()

> accuracy\_vector <- numeric()

> f\_score\_vector <- numeric()

> M1\_data2\_model <- train(defect ~ loc + bf, data = data2, method = "glm", family = "binomial")

> optimal\_threshold <- 0.7

> for (i in 1:repetition\_count) {

+ set.seed(i)

+ indices <- sample(1:nrow(data2), nrow(data2) / 2)

+ data2\_train <- data2[indices, ]

+ data2\_test <- data2[-indices, ]

+ data2\_test\_prob <- predict(M1\_data2\_model, data2\_test, type = "prob")[, 2]

+ data2\_test$Predicted\_Defect <- ifelse(data2\_test\_prob > optimal\_threshold, 1, 0)

+ data2\_test$Predicted\_Defect <- factor(data2\_test$Predicted\_Defect, levels = levels(data2\_test$defect))

+ resultant\_matrix <- confusionMatrix(data = data2\_test$Predicted\_Defect, reference = data2\_test$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data2\_test$Predicted\_Defect == data2\_test$defect) / nrow(data2\_test)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ recall\_vector <- c(recall\_vector, recall)

+ precision\_vector <- c(precision\_vector, precision)

+ accuracy\_vector <- c(accuracy\_vector, accuracy)

+ f\_score\_vector <- c(f\_score\_vector, f\_score)

+ }

> avg\_recall <- mean(recall\_vector)

> avg\_precision <- mean(precision\_vector)

> avg\_accuracy <- mean(accuracy\_vector)

> avg\_f\_score <- mean(f\_score\_vector)

> stddev\_recall <- sd(recall\_vector)

> stddev\_precision <- sd(precision\_vector)

> stddev\_accuracy <- sd(accuracy\_vector)

> stddev\_f\_score <- sd(f\_score\_vector)

> required\_output <- data.frame(

+ Metric = c("Recall", "Precision", "Accuracy", "F-Score"),

+ Average = c(avg\_recall, avg\_precision, avg\_accuracy, avg\_f\_score),

+ StandardDeviation = c(stddev\_recall, stddev\_precision, stddev\_accuracy, stddev\_f\_score)

+ )

> print(required\_output)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to perform cross-validation with M1 on the PDE.csv dataset. Later loading the caret library we then consider the defect column which is newly created in the PDE dataset and we consider the defect factor to classify whether it is defective (1) or not(0). Later we define the number of repetitions as 30 which was given as one of the constraints to evaluate the model's performance. Now we initialize the and create empty numeric vectors to store recall, precision, accuracy, and f\_score of the model respectively. Now we consider the logistic regression model named M1\_data2\_model which was trained on the PDE.csv dataset to predict defects using both loc and bf features from the PDE.csv dataset. Later we set the optimal threshold as 0.7 which was the threshold that has the highest f\_score which was obtained earlier by evaluating M1\_model which was trained and tested on the PDE.csv dataset. Now we iterate through a for-loop which iterates over a defined number of repetitions, and we set the random seed.

Later we randomly split the PDE dataset into training and testing which contains 50% of the data1 dataset for each iteration of a for-loop. Later we consider the trained model M1 to predict the probabilities of defects for the test set instances and the values are stored in a variable named data2\_test\_prob. Later, to classify flaws, we use the optimal threshold that has been specified. The instance is classed as having a defect (1) if the estimated likelihood of a defect exceeds the threshold; otherwise, it is labeled as not having a defect (0). After that, we convert the projected values into a factor in order to guarantee alignment with the real defect labels. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance which holds the values of Cutoff, Recall, Precision, Accuracy, and F\_Score. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the calculated values for the current cross-validation iteration to initialized vectors respectively. Now we exit from for-loop iteration and calculate the average and standard deviation of recall, precision, accuracy, and f\_score. Later we create a data frame called required\_output to display the mean and standard deviation of recall, accuracy, precision, and f\_score and we print the required\_output.

The output:

Metric Average StandardDeviation

1 Recall 0.9961127 0.001576089

2 Precision 0.8679845 0.011104074

3 Accuracy 0.8665332 0.011156504

4 F-Score 0.9276088 0.006440494

e) Perform cross-validation with T1 on JDT and PDE similarly in d). Compare with M1 in task d).

The ‘R’ code to perform cross-validation with T1 on JDT. That is, select randomly 50% of JDT for training T1 and test it with 50% remaining. Use the optimal threshold found on task a) to calculate the recall, precision, accuracy, and f-score. Repeat 30 times is as follows:

> library(caret)

> data1 <- read.csv("JDT.csv")

> data1$defect <- as.factor(data1$defect)

> repetition\_count <- 30

> recall\_vector <- numeric()

> precision\_vector <- numeric()

> accuracy\_vector <- numeric()

> f\_score\_vector <- numeric()

> T1\_data1\_model <- train(defect ~ loc + bf, data = data1, method = "rpart")

> optimal\_threshold <- 0.2

> for (i in 1:repetition\_count) {

+ set.seed(i)

+ indices <- sample(1:nrow(data1), nrow(data1) / 2)

+ data1\_train <- data1[indices, ]

+ data1\_test <- data1[-indices, ]

+ data1\_test\_prob <- predict(T1\_data1\_model, data1\_test, type = "prob")[, 2]

+ data1\_test$Predicted\_Defect <- ifelse(data1\_test\_prob > optimal\_threshold, 1, 0)

+ data1\_test$Predicted\_Defect <- factor(data1\_test$Predicted\_Defect, levels = levels(data1\_test$defect))

+ resultant\_matrix <- confusionMatrix(data = data1\_test$Predicted\_Defect, reference = data1\_test$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data1\_test$Predicted\_Defect == data1\_test$defect) / nrow(data1\_test)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ recall\_vector <- c(recall\_vector, recall)

+ precision\_vector <- c(precision\_vector, precision)

+ accuracy\_vector <- c(accuracy\_vector, accuracy)

+ f\_score\_vector <- c(f\_score\_vector, f\_score)

+ }

> avg\_recall <- mean(recall\_vector)

> avg\_precision <- mean(precision\_vector)

> avg\_accuracy <- mean(accuracy\_vector)

> avg\_f\_score <- mean(f\_score\_vector)

> stddev\_recall <- sd(recall\_vector)

> stddev\_precision <- sd(precision\_vector)

> stddev\_accuracy <- sd(accuracy\_vector)

> stddev\_f\_score <- sd(f\_score\_vector)

> required\_output <- data.frame(

+ Metric = c("Recall", "Precision", "Accuracy", "F-Score"),

+ Average = c(avg\_recall, avg\_precision, avg\_accuracy, avg\_f\_score),

+ StandardDeviation = c(stddev\_recall, stddev\_precision, stddev\_accuracy, stddev\_f\_score)

+ )

> print(required\_output)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to perform cross-validation with T1 on the JDT.csv dataset. Later loading the caret library we then consider the defect column which is newly created in the JDT dataset and we consider the defect factor to classify whether it is defective (1) or not(0). Later we define the number of repetitions as 30 which was given as one of the constraints to evaluate the model's performance. Now we initialize the and create empty numeric vectors to store recall, precision, accuracy, and f\_score of the model respectively. Now we consider the decision tree named T1\_data1\_model which was trained on the JDT.csv dataset to predict defects using both loc and bf features from the JDT.csv dataset. Later we set the optimal threshold as 0.2 which was the threshold that has the highest f\_score which was obtained earlier by evaluating T1\_model which was trained and tested on the JDT.csv dataset. Now we iterate through a for-loop which iterates over a defined number of repetitions, and we set the random seed.

Later we randomly split the JDT dataset into training and testing which contains 50% of the data1 dataset for each iteration of a for-loop. Later we consider the trained model T1 to predict the probabilities of defects for the test set instances and the values are stored in a variable named data1\_test\_prob. Later, to classify flaws, we use the optimal threshold that has been specified. The instance is classed as having a defect (1) if the estimated likelihood of a defect exceeds the threshold; otherwise, it is labeled as not having a defect (0). After that, we convert the projected values into a factor in order to guarantee alignment with the real defect labels. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance which holds the values of Cutoff, Recall, Precision, Accuracy, and F\_Score. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the calculated values for the current cross-validation iteration to initialized vectors respectively. Now we exit from for-loop iteration and calculate the average and standard deviation of recall, precision, accuracy, and f\_score. Later we create a data frame called required\_output to display the mean and standard deviation of recall, accuracy, precision, and f\_score and we print the required\_output.

The output:

Metric Average StandardDeviation

1 Recall 0.9404144 0.009349213

2 Precision 0.8750067 0.012524950

3 Accuracy 0.8462926 0.010710119

4 F-Score 0.9064506 0.007103252

By looking at the above output and considering the output of cross-validation of the M1 model on the JDT dataset we can state that the average of f\_score is higher in the case of M1\_model with the value of 0.9085212.

The ‘R’ code to perform cross-validation with T1 on PDE. That is, select randomly 50% of PDE for training T1 and test it with 50% remaining. Use the optimal threshold found on task a) to calculate the recall, precision, accuracy, and f-score. Repeat 30 times is as follows:

> library(caret)

> data2 <- read.csv("PDE.csv")

> data2$defect <- as.factor(data2$defect)

> repetition\_count <- 30

> recall\_vector <- numeric()

> precision\_vector <- numeric()

> accuracy\_vector <- numeric()

> f\_score\_vector <- numeric()

> T1\_data2\_model <- train(defect ~ loc + bf, data = data2, method = "rpart")

> optimal\_threshold <- 0.2

> for (i in 1:repetition\_count) {

+ set.seed(i)

+ indices <- sample(1:nrow(data2), nrow(data2) / 2)

+ data2\_train <- data2[indices, ]

+ data2\_test <- data2[-indices, ]

+ data2\_test\_prob <- predict(T1\_data2\_model, data2\_test, type = "prob")[, 2]

+ data2\_test$Predicted\_Defect <- ifelse(data2\_test\_prob > optimal\_threshold, 1, 0)

+ data2\_test$Predicted\_Defect <- factor(data2\_test$Predicted\_Defect, levels = levels(data2\_test$defect))

+ resultant\_matrix <- confusionMatrix(data = data2\_test$Predicted\_Defect, reference = data2\_test$defect)

+ recall <- resultant\_matrix$byClass["Recall"]

+ precision <- resultant\_matrix$byClass["Precision"]

+ accuracy <- sum(data2\_test$Predicted\_Defect == data2\_test$defect) / nrow(data2\_test)

+ f\_score <- 2 \* (precision \* recall) / (precision + recall)

+ recall\_vector <- c(recall\_vector, recall)

+ precision\_vector <- c(precision\_vector, precision)

+ accuracy\_vector <- c(accuracy\_vector, accuracy)

+ f\_score\_vector <- c(f\_score\_vector, f\_score)

+ }

> avg\_recall <- mean(recall\_vector)

> avg\_precision <- mean(precision\_vector)

> avg\_accuracy <- mean(accuracy\_vector)

> avg\_f\_score <- mean(f\_score\_vector)

> stddev\_recall <- sd(recall\_vector)

> stddev\_precision <- sd(precision\_vector)

> stddev\_accuracy <- sd(accuracy\_vector)

> stddev\_f\_score <- sd(f\_score\_vector)

> required\_output <- data.frame(

+ Metric = c("Recall", "Precision", "Accuracy", "F-Score"),

+ Average = c(avg\_recall, avg\_precision, avg\_accuracy, avg\_f\_score),

+ StandardDeviation = c(stddev\_recall, stddev\_precision, stddev\_accuracy, stddev\_f\_score)

+ )

> print(required\_output)

In the above ‘R’ code at first, we load the library named caret because it is a part of machine learning analysis where we are going to perform cross-validation with T1 on the PDE.csv dataset. Later loading the caret library we then consider the defect column which is newly created in the PDE dataset and we consider the defect factor to classify whether it is defective (1) or not(0). Later we define the number of repetitions as 30 which was given as one of the constraints to evaluate the model's performance. Now we initialize the and create empty numeric vectors to store recall, precision, accuracy, and f\_score of the model respectively. Now we consider the decision tree named T1\_data2\_model which was trained on the PDE.csv dataset to predict defects using both loc and bf features from the JDT.csv dataset. Later we set the optimal threshold as 0.2 which was the threshold that has the highest f\_score which was obtained earlier by evaluating T1\_model which was trained and tested on the PDE.csv dataset. Now we iterate through a for-loop which iterates over a defined number of repetitions, and we set the random seed.

Later we randomly split the PDE dataset into training and testing which contains 50% of the data1 dataset for each iteration of a for-loop. Later we consider the trained model T1 to predict the probabilities of defects for the test set instances and the values are stored in a variable named data1\_test\_prob. Later, to classify flaws, we use the optimal threshold that has been specified. The instance is classed as having a defect (1) if the estimated likelihood of a defect exceeds the threshold; otherwise, it is labeled as not having a defect (0). After that, we convert the projected values into a factor in order to guarantee alignment with the real defect labels. Now we generate a confusion matrix and store the values in a variable named resultant\_matrix to assess the generated model’s classification performance which holds the values of Cutoff, Recall, Precision, Accuracy, and F\_Score. Later we extract the recall and precision from the confusion matrix. Now we calculate the accuracy by comparing the predicted values to the actual defect values. Later we calculate the f\_score which is determined by a formula that states that the values are harmonic mean of precision and recall. Now we append the calculated values for the current cross-validation iteration to initialized vectors respectively. Now we exit from for-loop iteration and calculate the average and standard deviation of recall, precision, accuracy, and f\_score. Later we create a data frame called required\_output to display the mean and standard deviation of recall, accuracy, precision, and f\_score and we print the required\_output.

The output:

Metric Average StandardDeviation

1 Recall 1.0000000 0.000000000

2 Precision 0.8588785 0.010618285

3 Accuracy 0.8588785 0.010618285

4 F-Score 0.9240485 0.006145901

By looking at the above output and considering the output of cross-validation of the M1 model on the PDE dataset we can state that the average of f\_score is higher in the case of M1\_model with the value of 0.9276088.

2.Build a linear regression model M2 and a decision tree T2 to predict **bug** using both **loc** and **bf**. Because both models output as the number of bugs, we measure the prediction effectiveness of a model by calculating the total number of actual bugs discovered in top-20% classes ranked based on the predicted number of bugs by that model on testing data. We will compare two models M2 and T2 based on that prediction effectiveness.

a. Perform cross-validation of M2 on JDT and PDE (the setting is similar in Q1).

The ‘R’ code to perform cross-validation of M2 on JDT.csv and PDE.csv is as follows:

> library(caret)

> required\_actual\_total\_bugs <- function(predicted\_data, actual\_values) {

+ n <- length(predicted\_data)

+ cutoff\_top\_20 <- round(0.2 \* n)

+ sorted\_indices <- order(predicted\_data, decreasing = TRUE)

+ indices\_top\_20 <- sorted\_indices[1:cutoff\_top\_20]

+ actual\_total\_bugs\_in\_top\_20 <- sum(actual\_values[indices\_top\_20])

+ return(actual\_total\_bugs\_in\_top\_20)

+ }

> top\_20\_required\_model <- function(predicted\_data, actual\_values) {

+ actual\_bugs <- sum(actual\_values)

+ actual\_total\_bugs\_in\_top\_20\_percent <- required\_actual\_total\_bugs(predicted\_data, actual\_values)

+ required\_effectiveness <- actual\_total\_bugs\_in\_top\_20\_percent / actual\_bugs

+ return(required\_effectiveness)

+ }

> required\_M2\_cross\_validation <- function(data) {

+ set.seed(450)

+ output\_data <- numeric(30)

+

+ for (i in 1:30) {

+ train\_indices <- createDataPartition(data$bug, p = 0.5, list = FALSE)

+ train\_data <- data[train\_indices, ]

+ test\_data <- data[-train\_indices, ]

+

+ M2\_model <- lm(bug ~ loc + bf, data = train\_data)

+ predicted\_data <- predict(M2\_model, newdata = test\_data)

+ required\_effectiveness <- top\_20\_required\_model(predicted\_data, test\_data$bug)

+ output\_data[i] <- required\_effectiveness

+ }

+

+ return(output\_data)

+ }

In the above code at first, we load the library named caret because it is a part of machine learning analysis where we are going to perform cross-validation with M2 on the JDT.csv and PDE.csv datasets. Later we create a function named required\_actual\_total\_bugs. This function calculates the total actual bugs found in the top 20% of predicted data. In this function, at first, we define the length of the predicted\_data and later we then calculate the 20% cutoff based on the length. Now we sort the predicted\_data in descending order and extract the top 20% of sorted indices. Now we compute the sum of actual values corresponding to the top 20% indices. We return the computed sum from this function. Later, we now define the other function named top\_20\_required\_model. This function calculates the model's effectiveness in predicting the top 20% of actual bugs by considering predicted data. Now we calculate the total actual bugs in the top 20% and effectiveness in terms of the ratio of actual bugs in the top 20% to total actual bugs. The effectiveness values are returned from this function. Now we generate another function named required\_M2\_cross\_validation to perform cross-validation from model M2. At first, we set the seed to 450 for reproducibility by referencing the last three digits of my R number. Later we initialize the empty numeric vector named output\_data to store the results. Now we iterate through a for-loop for 30 iterations and train\_indices were generated using createDataPartition to split the data into train\_data and test\_data. We generate a model named M2\_model which fits the linear regression model for the training data. Later we calculate the predicted values for test\_data and call the top\_20\_required\_model function to compute the effectiveness of the generated M2\_model. For each iteration, the generated results are stored in the output\_data vector. Now we exit from a for-loop.

To get the effectiveness and to perform the cross-validation of M2\_model on JDT.csv we implement the following ‘R’ code which refers to the above implemented functions.

> data1 <- read.csv("JDT.csv")

> output1 <- required\_M2\_cross\_validation(data1)

> print(output1)

[1] 0.6436782 0.7098446 0.7157895 0.7560976 0.6685083 0.6802326 0.6926829 0.7239819 0.6934673 0.6766169 0.7967914

[12] 0.6966292 0.6864865 0.6858639 0.7777778 0.6500000 0.7111111 0.7172775 0.7172775 0.7222222 0.6741573 0.7208122

[23] 0.7197802 0.7342995 0.6781609 0.7112299 0.6960784 0.7222222 0.7577093 0.7068063

> average\_effectiveness <- mean(output1)

> cat("Average Effectiveness:", average\_effectiveness, "\n")

Average Effectiveness: 0.7081198

To get the effectiveness and to perform the cross-validation of M2\_model on PDE.csv we implement the following ‘R’ code which refers to the above implemented functions.

> data2 <- read.csv("PDE.csv")

> output2 <- required\_M2\_cross\_validation(data2)

> print(output2)

[1] 0.6770186 0.6363636 0.6702128 0.6571429 0.6994536 0.5117647 0.5000000 0.4800000 0.6827957 0.5062500 0.4782609

[12] 0.5341615 0.4662577 0.5379310 0.4812500 0.5838926 0.6363636 0.5337838 0.6923077 0.6514286 0.6884422 0.6432749

[23] 0.6187500 0.5060241 0.5031847 0.5657895 0.6395939 0.4965517 0.5179856 0.6822917

> average\_effectiveness <- mean(output2)

> cat("Average Effectiveness:", average\_effectiveness, "\n")

Average Effectiveness: 0.5826176

b. Perform cross-validation of T2 on JDT and PDE (the setting is similar in Q1). Compare with M2 in task a).

The ‘R’ code to perform cross-validation of T2 on JDT.csv and PDE.csv is as follows:

> library(caret)

> required\_actual\_total\_bugs <- function(predicted\_data, actual\_values) {

+ n <- length(predicted\_data)

+ cutoff\_top\_20 <- round(0.2 \* n)

+ sorted\_indices <- order(predicted\_data, decreasing = TRUE)

+ indices\_top\_20 <- sorted\_indices[1:cutoff\_top\_20]

+ actual\_total\_bugs\_in\_top\_20 <- sum(actual\_values[indices\_top\_20])

+ return(actual\_total\_bugs\_in\_top\_20)

+ }

> top\_20\_required\_model <- function(predicted\_data, actual\_values) {

+ actual\_bugs <- sum(actual\_values)

+ actual\_total\_bugs\_in\_top\_20\_percent <- required\_actual\_total\_bugs(predicted\_data, actual\_values)

+ required\_effectiveness <- actual\_total\_bugs\_in\_top\_20\_percent / actual\_bugs

+ return(required\_effectiveness)

+ }

> required\_T2\_cross\_validation <- function(data) {

+ set.seed(450)

+ output\_data <- numeric(30)

+ for (i in 1:30) {

+ train\_indices <- createDataPartition(data$bug, p = 0.5, list = FALSE)

+ train\_data <- data[train\_indices, ]

+ test\_data <- data[-train\_indices, ]

+ T2\_model <- train(bug ~ loc + bf, data = train\_data, method = "rpart")

+ predicted\_data <- predict(T2\_model, newdata = test\_data)

+ required\_effectiveness <- top\_20\_required\_model(predicted\_data, test\_data$bug)

+ output\_data[i] <- required\_effectiveness

+ }

+

+ return(output\_data)

+ }

In the above code at first, we load the library named caret because it is a part of machine learning analysis where we are going to perform cross-validation with T2 on the JDT.csv and PDE.csv datasets. Later we create a function named required\_actual\_total\_bugs. This function calculates the total actual bugs found in the top 20% of predicted data. In this function, at first, we define the length of the predicted\_data and later we then calculate the 20% cutoff based on the length. Now we sort the predicted\_data in descending order and extract the top 20% of sorted indices. Now we compute the sum of actual values corresponding to the top 20% indices. We return the computed sum from this function. Later, we now define the other function named top\_20\_required\_model. This function calculates the model's effectiveness in predicting the top 20% of actual bugs by considering predicted data. Now we calculate the total actual bugs in the top 20% and effectiveness in terms of the ratio of actual bugs in the top 20% to total actual bugs. The effectiveness values are returned from this function. Now we generate another function named required\_T2\_cross\_validation to perform cross-validation from model T2. At first, we set the seed to 450 for reproducibility by referencing the last three digits of my R number. Later we initialize the empty numeric vector named output\_data to store the results. Now we iterate through a for-loop for 30 iterations and train\_indices were generated using createDataPartition to split the data into train\_data and test\_data. We generate a model named T2\_model which fits the linear regression model for the training data. Later we calculate the predicted values for test\_data and call the top\_20\_required\_model function to compute the effectiveness of the generated T2\_model. For each iteration, the generated results are stored in the output\_data vector. Now we exit from a for-loop.

To get the effectiveness and to perform the cross-validation of T2\_model on JDT.csv we implement the following ‘R’ code which refers to the above implemented functions.

> data1 <- read.csv("JDT.csv")

> output1 <- required\_T2\_cross\_validation(data1)

> print(output1)

[1] 0.4367816 0.4756757 0.4074074 0.5959596 0.7117647 0.6187845 0.4918033 0.6559140 0.6903553 0.5064103 0.3720930

[12] 0.6730769 0.5135135 0.3550296 0.5097087 0.5229885 0.3918129 0.3809524 0.6000000 0.5506329 0.5957447 0.6060606

[23] 0.5100000 0.5365854 0.6774194 0.6035503 0.6540541 0.5222222 0.4171123 0.6513761

> average\_effectiveness <- mean(output1)

> cat("Average Effectiveness on JDT:", average\_effectiveness, "\n")

Average Effectiveness on JDT: 0.5411597

To get the effectiveness and to perform the cross-validation of T2\_model on PDE.csv we implement the following ‘R’ code which refers to the above implemented functions.

> data2 <- read.csv("PDE.csv")

> output2 <- required\_T2\_cross\_validation(data2)

> print(output2)

[1] 0.3913043 0.3741007 0.4393939 0.2197802 0.2294118 0.2567568 0.3621622 0.2638889 0.3184713 0.3736842 0.5773196

[12] 0.2413793 0.5297297 0.3279570 0.2864865 0.3492823 0.3899371 0.3409091 0.5824742 0.2808219 0.3417722 0.1818182

[23] 0.2708333 0.4040404 0.3883495 0.3333333 0.3121019 0.5714286 0.3006993 0.2913907

> average\_effectiveness <- mean(output2)

> cat("Average Effectiveness on PDE:", average\_effectiveness, "\n")

Average Effectiveness on PDE: 0.351034

By comparing the output of cross-validation of T2\_model on JDT with the output of cross-validation of M2\_model on JDT, the average effectiveness is 0.7081198 in the case of cross-validation of M2\_model on JDT and the average effectiveness is 0.5411597 in the case of cross-validation of T2\_moel on JDT, which is lower than M2\_model cross-validation.

By comparing the output of cross-validation of T2\_model on PDE with the output of cross-validation of M2\_model on PDE, the average effectiveness is 0.5826176 in the case of cross-validation of M2\_model on PDE and the average effectiveness is 0.351034 in the case of cross-validation of T2\_moel on PDE, which is lower than M2\_model cross-validation.

Note:

* <https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/sample>

I have referred to this website to understand how to split the data randomly from a given data set.

* <https://dplyr.tidyverse.org/articles/dplyr.html>

I have referred to this website to get to know how to sort the data in required conditions such as sorting the dataset, arranging datasets, and selecting the top 20% of data.

* <https://stat.ethz.ch/R-manual/R-devel/library/stats/html/lm.html>

I have referred to this website to understand how to implement the linear regression model.

* <https://stat.ethz.ch/R-manual/R-devel/library/stats/html/glm.html>

I have referred to this website to understand how to implement the logistic regression model.

* <https://stat.ethz.ch/R-manual/R-devel/library/rpart/html/rpart.html>

I have referred to this website to understand how to implement the decision tree.