**Software Analytics - Assignment 8**

Due date: 11:59 pm Nov 5, 2023.

**General information**

You can use all available resources: notes, AI apps, Google search... But you need to do all the work on your own. You cannot discuss or share your work with your friends or classmates.

The instructor and TA will not provide any clarification or suggestions. You should work with the best of your knowledge and understanding.

You need to submit a document containing your answers and the R code you use to produce the analysis and answers. You should copy the analysis results and figures produced to your answer document. Please ensure that you submit both the Word/PDF solution file and the "Rhistory" file, which is necessary for a complete evaluation of your work.

Note: If you don't submit the Rhistory file, you will be deducted 50% of the total points.

For each question, before writing the code, you should explain your ideas on how to solve the problem: Why did you choose a particular method, function, or solution? What was your thought process in approaching the problem? Your descriptions will account for half of the points for each question. Providing clear and thoughtful explanations shows that you understand the topic and will be crucial for achieving full marks.

**Question**

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 colume **defect** to each dataset. If a file has bugs (bug > 0) then **defect** is 1. Otherwise it is 0.

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.

Approach: First, we need to load the dataset into their respective variables for successful usage. Once that step is completed, we will add an extra column named defect to each data set following the instructions indicated above.

After those steps are completed, we will create the logistic regression Model and decision tree using loc and bf as our variables.

Method: we load the datasets using read.csv(“dataset”), add the defect column to each by using dataset$defect =ifelse (dataset$bug > 0, 1, 0). Then create the models using gml() for the logistics model and the library rpart alongside rpart() for the decision tree.

a. Train M1 on JDT and also test it on JDT. Calculte 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.

First, we create the M1 model based on the JDT data and subsequently test it on JDT too, a similar case for PDE. Then, we will create a function that calculates the recall, precision, accuracy, and f-score of multiple thresholds and stores them in a data frame for easy analysis and multiple use along the assignment. Then after running the codes we can analyze and choose the threshold with highest fscore

Method: the function will go through each threshold from 0.1 to 0.9 until it reaches the max, calculating the asked variables and storing them in a dataframe. These are calculated as follows: creating a table that stores the prediction vs the actual values of defects, precision is obtained by dividing the true positive predictions by the total amount of positive predictions made; recall is obtained by dividing the true positive predictions by the total amount of **actual** positive predictions made; accuracy is obtained by dividing all the correct positive predictions by the total amount of predictions attempts, and the f score is the harmonic mean of precision and recall, providing a balance between these two metrics.

Lastly,

> JDT = read.csv("JDT.csv")

> PDE = read.csv("PDE.csv")

> JDT$defect = ifelse(JDT$bug > 0, 1,0)

> PDE$defect = ifelse(PDE$bug > 0, 1,0)

> M1 = glm(defect ~ loc + bf, data = JDT, family = "binomial")

> calculateMetrics = function(predictions, actuals){

+ thresholds = seq(0.1, 0.9, by = 0.1)

+ results = data.frame(threshold = numeric(), recall = numeric(), precision = numeric(), accuracy = numeric(), fscore = numeric())

+

+ for (threshold in thresholds){

+ predDefect = ifelse(predictions > threshold, 1, 0)

+ confMatrix = table(predDefect, actuals)

+ precision = confMatrix[2, 2] / sum(confMatrix[, 2])

+ recall = confMatrix[2, 2] / sum(confMatrix[2, ])

+ accuracy = sum(diag(confMatrix)) / sum(confMatrix)

+ fscore = 2 \* (precision \* recall) / (precision + recall)

+

+ results[nrow(results) + 1, ] = c(threshold, recall, precision, accuracy, fscore)

+ }

+ colnames(results) = c("threshold", "recall", "precision", "accuracy", "fscore")

+ return(results)

+ }

predictM1 = predict(M1, newdata = JDT, type = "response")

metricsJDT = calculateMetrics(predictM1,JDT$defect)

> M2 = glm(defect ~ loc + bf, data = PDE, family = "binomial")

> predictM2 = predict(M2, newdata = PDE, type = "response")

> metricsPDE = calculateMetrics(predictM2,PDE$defect)

A screenshot of a computer

Description automatically generated

For M1 (JDT on JDT): the threshold with highest fscore is 0.3 with a score of 0.5745257

For M2 (PDE on PDE): the threshold with highest fscore is 0.2 with a score of 0.36939314

b. Train M1 on JDT and also test it on PDE. Calculte 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.

Same approach and method will be used from the previous question, main difference is that JDT is test on PDE and viceversa.

Asked ChatGPT to modify my code so it could accept M1 when it had different length to JDT$Defect

Function calculateMetrics did not required modifications, instead, it suggested to remodify the predictM1 variable, so it only had the same number of entries as dataset$Defect

predictM1 = predict(M1, newdata = PDE, type = "response")

> predictM1 = predictM1[1:length(JDT$defect)]

> metricsJDTPDE = calculateMetrics(predictM1, JDT$defect)

> metricsJDTPDE

> predictM2 = predict(M2, newdata = JDT, type = "response")

> predictM2 = predictM2[1:length(PDE$defect)]

> metricsPDEJDT = calculateMetrics(predictM2, PDE$defect)

> metricsPDEJDT

A screenshot of a computer

Description automatically generated

For M1 (JDT on PDE) the highest fscore is 0.3038 with a threshold of 0.1

For M2 (PDE on JDT) the highest fscore is 0.2422 with a threshold of 0.1

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

We would import library(rpart) and create a decision tree using the command rpart(defect ~ loc + bf, data = dataset, method = "class"), then use the a modified version of the function created before that handles errors everytime the confmatrix goes out of bounds (USED CHATGPT TO SOLVE THIS ISSUE)

library(rpart)

> T1 = rpart(defect ~ loc + bf, data = JDT, method = "class")

> T2 = rpart(defect ~ loc + bf, data = PDE, method = "class")

calculateMetrics = function(predictions, actuals) {

+ thresholds = seq(0.1, 0.9, by = 0.1)

+ results = data.frame(threshold = numeric(), recall = numeric(), precision = numeric(), accuracy = numeric(), fscore = numeric())

+

+ for (threshold in thresholds) {

+ predDefect = ifelse(predictions > threshold, 1, 0)

+ actualDefect = ifelse(actuals > 0, 1, 0)

+

+ confMatrix = table(predDefect, actualDefect)

+

+ if (nrow(confMatrix) < 2 || ncol(confMatrix) < 2) {

+ results[nrow(results) + 1, ] = c(threshold, 0, 0, 0, 0)

+ } else {

+ precision = confMatrix[2, 2] / sum(confMatrix[, 2])

+ recall = confMatrix[2, 2] / sum(confMatrix[2, ])

+ accuracy = sum(diag(confMatrix)) / sum(confMatrix)

+ fscore = 2 \* (precision \* recall) / (precision + recall)

+

+ results[nrow(results) + 1, ] = c(threshold, recall, precision, accuracy, fscore)

+ }

+ }

+ colnames(results) = c("threshold", "recall", "precision", "accuracy", "fscore")

+ return(results)

+ }

predictT1 = predict(T1, newdata = JDT, type = "prob")[,2]

> predictT1 = predictT1[1:length(JDT$defect)]

> metricsT1 = calculateMetrics(predictT1, JDT$defect)

> predictT2 = predict(T2, newdata = PDE, type = "prob")[,2]

> predictT2 = predictT2[1:length(PDE$defect)]

> metricsT2 = calculateMetrics(predictT2,PDE$defect)

A screenshot of a computer code

Description automatically generated

T1 highest fscore is 0.5690141 between threshold 0.2 and 0.3

T2 highest fscore is 0.3869347 between threshold 0.2 and 0.3

A screenshot of a computer

Description automatically generated

T1(JDT ON PDE) highest fscore is 0.02703 with threshold 0.4 to 0.7

T2 (PDE ON JDT) highest fscore is 0.2066277 with threshold 0.2 and 0.3

We can see that when we are training and testing on the same dataset, M1 has a higher fscore for JDT and slightly smaller fscore for PDE compared to T1; Whenever M1 was trained in one dataset but tested in the other dataset, M1 has a higher fscore in both situations compared to T1. Is important to point out that the fscores of T1 repeat themselves in multiple thresholds.

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.

For this question, we will use the same calculate metrics function from before and after we create a repetition function that will validate the function 30 times using the specified threshold, M1 will be test it on itself, 50% in training and 50% in testing. For this reason, we will be using sample() to obtain the first and second halves of the data set

A new modified version of the calculate metrics will be used to just obtain the values of a specific threshold. The code goes as such:

calculateMetrics = function(predictions, actuals, optimalThreshold) {

+ predDefect = ifelse(predictions > optimalThreshold, 1, 0)

+ actualDefect = ifelse(actuals > 0, 1, 0)

+

+ confMatrix = table(predDefect, actualDefect)

+

+ if (nrow(confMatrix) < 2 || ncol(confMatrix) < 2) {

+ metrics = c(0, 0, 0, 0)

+ } else {

+ precision = ifelse(sum(confMatrix[, 2]) == 0, 0, confMatrix[2, 2] / sum(confMatrix[, 2]))

+ recall = ifelse(sum(confMatrix[2, ]) == 0, 0, confMatrix[2, 2] / sum(confMatrix[2, ]))

+ accuracy = sum(diag(confMatrix)) / sum(confMatrix)

+ fscore = ifelse((precision + recall) == 0, 0, 2 \* (precision \* recall) / (precision + recall))

+

+ metrics = c(precision, recall, accuracy, fscore)

+ }

+

+ names(metrics) = c("precision", "recall", "accuracy", "fscore")

+ return(metrics)

+ }

> crossvalid = function(dataset, optimalThres){

+ result = list()

+

+ for(i in 1:30) {

+ shufflesamp = dataset[sample(nrow(dataset)),]

+

+ splitindex = nrow(shufflesamp) %/% 2

+ train = shufflesamp[1:splitindex,]

+ test = shufflesamp[(splitindex + 1):nrow(shufflesamp),]

+

+ M1 = glm(defect ~ loc + bf, data = train, family = "binomial")

+

+ predictM1 = predict(M1, newdata = test, type = "response")

+ predictM1 = predictM1[1:length(test$defect)]

+ metrics = calculateMetrics(predictM1, test$defect, optimalThres)

+ result[[i]] = metrics

+ }

+ return(result)

+ }

> m1JDT = crossvalid(JDT,0.3)

> m1PDE = crossvalid(PDE,0.2)

JDT

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer code

Description automatically generated A screenshot of a computer

Description automatically generated

PDE

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

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

The same functions will be used but the cross validation will receive the following modification:

M1 = glm(defect ~ loc + bf, data = train, family = "binomial")

+

+ predictM1 = predict(M1, newdata = test, type = "response")

+ predictM1 = predictM1[1:length(test$defect)]

+ metrics = calculateMetrics(predictM1, test$defect, optimalThres)

Will be replaced by (marked in green)

crossvalid = function(dataset, optimalThres){

+ result = list()

+

+ for(i in 1:30) {

+ shufflesamp = dataset[sample(nrow(dataset)),]

+

+ splitindex = nrow(shufflesamp) %/% 2

+ train = shufflesamp[1:splitindex,]

+ test = shufflesamp[(splitindex + 1):nrow(shufflesamp),]

+

+ T1 = rpart(defect ~ loc + bf, data = train, method = "class")

+

+ predictT1 = predict(T1, newdata = test, type = "prob")[,2]

+ predictT1 = predictT1[1:length(test$defect)]

+ metrics = calculateMetrics(predictT1, test$defect, optimalThres)

+ result[[i]] = metrics

+ }

+ return(result)

+ }

> T1JDT = crossvalid(JDT,0.2)

> T1PDE = crossvalid(PDE,0.2)

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated A screenshot of a computer code

Description automatically generated

PDE

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

One of the most notable differences is the deviation from the original fscore calculated in the previous tests, M1 has a deviation of +6 and -6in average. While T2 has a deviation of almost -10 and 10.

Q2 (4 pt). 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.

For this question, we will be using a modified version of the cross validation function that will load the 50% of the dataset for training purposes and 50% for testing purposes. Then we will load these values in a lm() command and sort the top 20 percent of the model by sorting it manually, then store the sum() of bugs in results and retrieve it for further analysis.

1. Perform cross-validation of M2 on JDT and PDE (the setting is similar in Q1).
   1. ChatGPT was used, as I was encountering an error I was not able to fix by myself here is the corrected version of the code and the specific modifications chatgpt did

> crossvalid\_M2 = function(dataset) {

+ result = list()

+

+ for (i in 1:30) {

+ shufflesamp = dataset[sample(nrow(dataset)),]

+

+ splitindex = nrow(shufflesamp) %/% 2

+ train = shufflesamp[1:splitindex,]

+ test = shufflesamp[(splitindex + 1):nrow(shufflesamp),]

+

+ M2 = lm(bug ~ loc + bf, data = train)

+

+ predictM2 = predict(M2, newdata = test)

+

+ # Find the 20% threshold manually

+ threshold\_index = round(0.2 \* length(predictM2))

+ sorted\_predictions = sort(predictM2, decreasing = TRUE)

+ top20th = sorted\_predictions[threshold\_index]

+

+ top20class = test[test$bug >= top20th, ]

+

+ totalbug = sum(top20class$bug)

+ result[[i]] = totalbug

+ }

+

+ return(result)

+ }

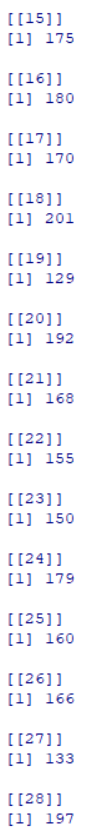
A white background with black text

Description automatically generatedA close up of numbers

Description automatically generatedA close up of numbers

Description automatically generated

A white background with black text

Description automatically generated  A close up of numbers

Description automatically generated

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

Following a similar approach and method to the part a, we will modify the cross validation function by creating a tree method to execute, replacing the lm with rpart() and applying the same analysis as yesterday

> crossvalid = function(dataset) {

+ result = list()

+

+ for (i in 1:30) {

+ shufflesamp = dataset[sample(nrow(dataset)),]

+

+ splitindex = nrow(shufflesamp) %/% 2

+ train = shufflesamp[1:splitindex,]

+ test = shufflesamp[(splitindex + 1):nrow(shufflesamp),]

+

+ T2 = rpart(bug ~ loc + bf, data = train)

+

+ predictT2 = predict(T2, newdata = test)

+

+ threshold\_index = round(0.2 \* length(predictT2))

+ sorted\_predictions = sort(predictT2, decreasing = TRUE)

+ top20th = sorted\_predictions[threshold\_index]

+

+ top20class = test[predictT2 >= top20th, ]

+

+ totalbug = sum(top20class$bug)

+ result[[i]] = totalbug

+ }

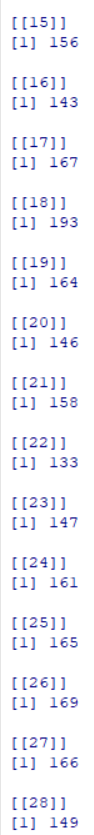
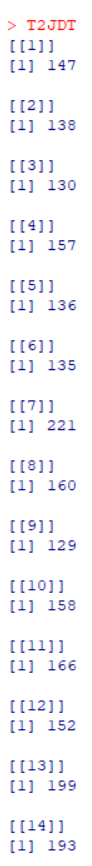
+

+ return(result)

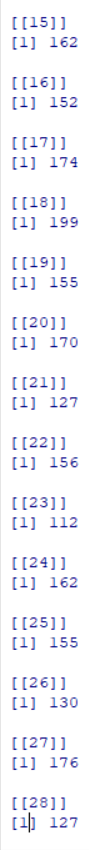
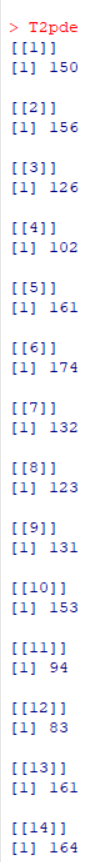
+ }

> T2JDT = crossvalid(JDT)

> T2pde = crossvalid(pde)

A screenshot of a computer

Description automatically generated

A white background with black numbers

Description automatically generated

The most notable difference is the deviation range of decision tree and linear model bug values, the decision tree having a higher deviation value in both JDT and PDE dataset.