

Project4_Part3

Prabhat Ale

2024-05-15

Loading mtcars dataset

```
data <- mtcars  
head(data)
```

```
##           mpg cyl  disp  hp  drat    wt  qsec vs am gear carb  
## Mazda RX4      21.0   6  160 110 3.90 2.620 16.46  0  1    4    4  
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0  1    4    4  
## Datsun 710      22.8   4  108  93 3.85 2.320 18.61  1  1    4    1  
## Hornet 4 Drive  21.4   6  258 110 3.08 3.215 19.44  1  0    3    1  
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2  
## Valiant        18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
data$am <- as.factor(data$am)  
data$am
```

Converting am target variable as categorical variable using factor

```
## [1] 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1  
## Levels: 0 1
```

```
set.seed(22)
```

Setting seed = 22 (my roll number for doing train test splits)

```
index <- sample(2, size = nrow(data), replace = TRUE, prob = c(0.8, 0.2))  
train_full <- data[index==1, ]  
test_full <- data[index==2, ]
```

Dividing the data into train test split of 0.8:0.2

```
data_full <- data
```

Creating a copy of dataset for using all features while training other ML algorithms except logistic regression

```
library(ggplot2)  
library(gridExtra)
```

Importing ggplot2 and gridExtra library for visualizing the relationship between independent and dependent feature

Define a function to create a single scatter plot

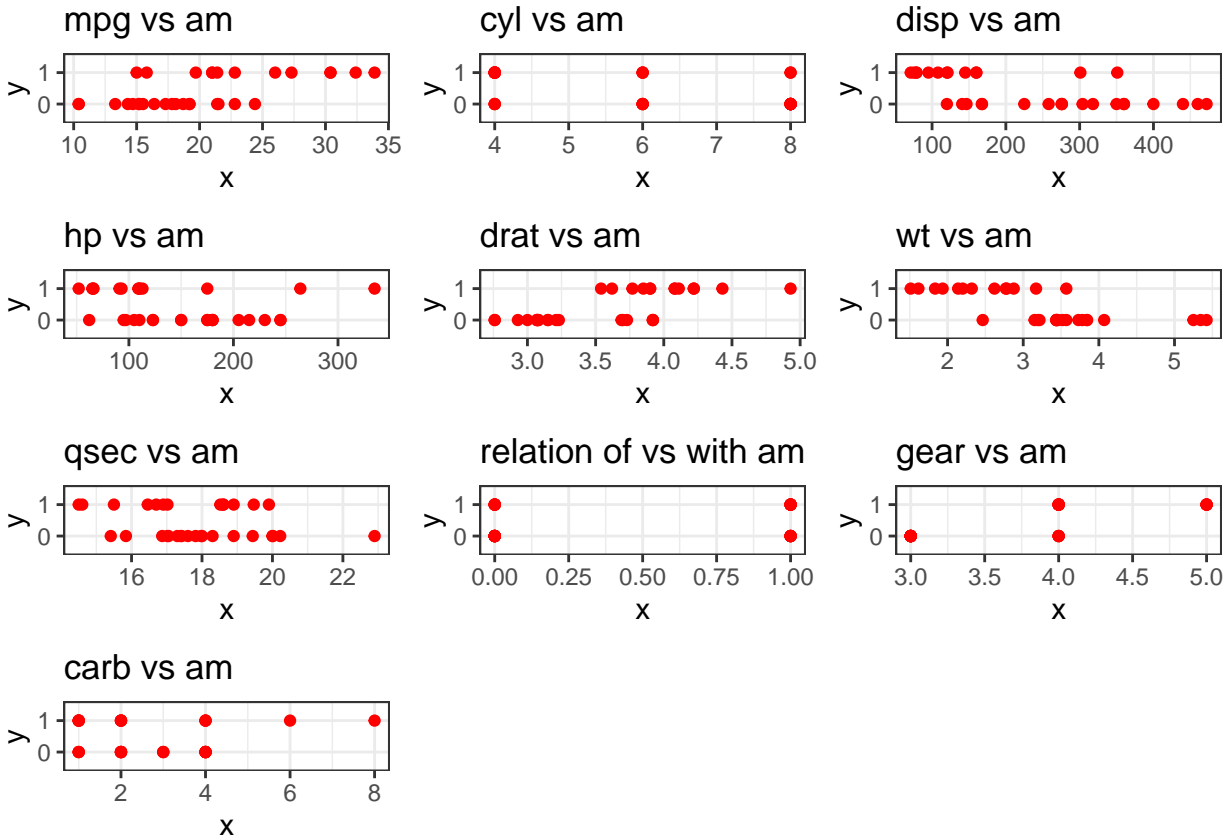
```
create_plot <- function(x, y, title) {  
  ggplot(data, aes(x = !!x, y = !!y)) + # !! for variable evaluation within function  
  geom_point(color = 'red') +  
  labs(title = title) +  
  theme_bw()  
}
```

Create individual plots (replace titles with your own)

```
plot1 <- create_plot(data$mpg, data$am, "mpg vs am")  
plot2 <- create_plot(data$cyl, data$am, "cyl vs am")  
plot3 <- create_plot(data$disp, data$am, "disp vs am")  
plot4 <- create_plot(data$hp, data$am, "hp vs am")  
plot5 <- create_plot(data$drat, data$am, "drat vs am")  
plot6 <- create_plot(data$wt, data$am, "wt vs am")  
plot7 <- create_plot(data$qsec, data$am, "qsec vs am")  
plot8 <- create_plot(data$vs, data$am, "relation of vs with am")  
plot9 <- create_plot(data$gear, data$am, "gear vs am")  
plot10 <- create_plot(data$carb, data$am, "carb vs am")
```

Arrange plots in a 2x3 grid (adjust rows and columns as needed)

```
grid.arrange(plot1, plot2, plot3, plot4, plot5,  
             plot6, plot7, plot8, plot9, plot10,  
             nrow = 4, ncol = 3)
```



Applying Multivariate Logistic regression on the test data

```
model.lr <- glm(am ~ ., data = data, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
library(car)
```

Using car library for calculating vif for logistic regression model to check multicollinearity in independent features

```
## Loading required package: carData
```

```
vif(model.lr)
```

```
##      mpg      cyl      disp      hp      drat      wt      qsec      vs
## 18.150446 38.112972 45.336025 21.288933  3.950868 28.384837  9.214178 11.102033
##      gear      carb
## 16.289577 21.096232
```

Here the vif values for all these features are greater than 2 so remove those feature whose vif value is highest at first disp has a high vif value so removing this feature

```
data <- data[, -which(names(data) == 'disp')]
```

Checking vif again after fitting logistic regression model on remaining features

```
model.lr <- glm(am ~ ., data = data, family = "binomial")

## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
vif(model.lr)

##      mpg      cyl      hp      drat      wt      qsec      vs      gear
## 7.685844 44.338479 16.517186 3.150966 6.315245 13.362245 23.046754 8.067772
##      carb
## 17.430216
```

After dropping disp feature, the vif for remaining feature is calculated. We can still see the features having vif values greater than 2 in all cases.

So removing cyl feature as it has the highest vif value among all others.

```
data <- data[, -which(names(data)=='cyl')]
```

Checking vif again after dropping cyl feature

```
model.lr <- glm(am ~ ., data = data, family = "binomial")

## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
vif(model.lr)

##      mpg      hp      drat      wt      qsec      vs      gear      carb
## 6.455138 10.635368 2.693499 5.441216 11.376122 12.362879 7.992196 17.129364
```

After dropping cyl feature, the vif for remaining feature is calculated. We can still see the features having vif values greater than 2 in all cases.

So removing carb feature as it has the highest vif value among all others.

```
data <- data[, -which(names(data)=='carb')]
ncol(data)
```

```
## [1] 8
```

Checking vif again after dropping cyl feature

```
model.lr <- glm(am ~ ., data = data, family = "binomial")

## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
vif(model.lr)

##      mpg      hp      drat      wt      qsec      vs      gear
## 29.884797 59.304559 6.661025 15.884779 6.288662 32.176847 15.855234
```

After dropping carb feature, the vif for remaining feature is calculated. We can still see the features having vif values greater than 2 in all cases.

So removing hp feature as it has the highest vif value among all others.

```
data <- data[, -which(names(data)=='hp')]
ncol(data)
```

```
## [1] 7
```

Checking vif again after dropping cyl feature

```
model.lr <- glm(am ~ ., data = data, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
vif(model.lr)
```

```
##      mpg      drat      wt      qsec      vs      gear
## 26.817260 51.247955 47.121136  5.483656 14.725850 82.262503
```

After dropping hp feature, the vif for remaining feature is calculated. We can still see the features having vif values greater than 2 in all cases.

So removing gear feature as it has the highest vif value among all others.

```
data <- data[, -which(names(data)=='gear')]
ncol(data)
```

```
## [1] 6
```

Checking vif again after dropping cyl feature

```
model.lr <- glm(am ~ ., data = data, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
vif(model.lr)
```

```
##      mpg      drat      wt      qsec      vs
## 146.146928  3.036565 52.351310 44.468752 142.612528
```

After dropping gear feature, the vif for remaining feature is calculated. We can still see the features having vif values greater than 2 in all cases.

So removing mpg feature as it has the highest vif value among all others.

```
data <- data[, -which(names(data)=='mpg')]
ncol(data)
```

```
## [1] 5
```

Checking vif again after dropping cyl feature

```
model.lr <- glm(am ~ ., data = data, family = "binomial")

## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
vif(model.lr)

##      drat      wt      qsec      vs
## 1.323271  8.216562 16.703997  6.873107
```

After dropping mpg feature, the vif for remaining feature is calculated. We can still see the features having vif values greater than 2 in all cases.

So removing qsec feature as it has the highest vif value among all others.

```
data <- data[, -which(names(data)=='qsec')]
ncol(data)

## [1] 4
```

Checking vif again after dropping cyl feature

```
model.lr <- glm(am ~ ., data = data, family = "binomial")

## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
vif(model.lr)

##      drat      wt      vs
## 23.22034 171.57421 252.95178
```

After dropping qsec feature, the vif for remaining feature is calculated. We can still see the features having vif values greater than 2 in all cases.

So removing vs feature as it has the highest vif value among all others.

```
data <- data[, -which(names(data)=='vs')]
ncol(data)

## [1] 3
```

Checking vif again after dropping cyl feature

```
model.lr <- glm(am ~ ., data = data, family = "binomial")
vif(model.lr)

##      drat      wt
## 1.045007  1.045007
```

After dropping vs feature, the vif for remaining feature is calculated.

The vif for the remaining two features is less than 2 so we can select those feature for creating logistic regression model.

```
train_lr <- data[index==1, ]
test_lr <- data[index==2, ]
```

Splitting training and testing data for logistic regression

```
model_lr <- glm(am ~ ., data = train_lr, family = "binomial")
```

Fitting logistic regression model using training dataset

```
predict_lr <- predict(model_lr, test_lr, type = 'response')
```

Making predictions on the test data using logistic regression

```
predict_lr.test <- factor(ifelse(predict_lr > 0.5, 1, 0))
```

Converting the probability values into categorical features

The logistic regression model's probability output represents the likelihood of an observation belonging to a specific category based on predictor variables, with coefficients indicating the direction and magnitude of their influence.

```
library(caret)
```

Loading a library caret for creating a confusion matrix

```
## Loading required package: lattice
```

```
confusionMatrix(predict_lr.test, factor(test_lr$am))
```

```
## Confusion Matrix and Statistics
```

```
##
```

```
##           Reference
```

```
## Prediction 0 1
```

```
##           0 3 0
```

```
##           1 0 3
```

```
##
```

```
##           Accuracy : 1
```

```
##           95% CI : (0.5407, 1)
```

```
##           No Information Rate : 0.5
```

```
##           P-Value [Acc > NIR] : 0.01563
```

```
##
```

```
##           Kappa : 1
```

```
##
```

```
##           McNemar's Test P-Value : NA
```

```
##
```

```
##           Sensitivity : 1.0
```

```
##           Specificity : 1.0
```

```
##           Pos Pred Value : 1.0
```

```
##           Neg Pred Value : 1.0
```

```
##           Prevalence : 0.5
```

```
##           Detection Rate : 0.5
```

```
## Detection Prevalence : 0.5
## Balanced Accuracy : 1.0
##
## 'Positive' Class : 0
##
```

We get all the prediction correct for the test set. 6 samples were taken in the test set.

We get an accuracy of 1.0, sensitivity of 1.0 and a specificity of 1.0.

```
library(e1071)
```

Loading a library named e1071 for Naive Bayes Algorithm

Naive Bayes Algorithm

Assumptions

- The Naive Bayes algorithm is called “Naive” because it makes the assumption that the occurrence of a certain feature is independent of the occurrence of other features. (satisfied)

Since Naive Bayes is not impacted by multicollinear features so we do not need any vif test for training the model using this algorithm.

```
model.nb <- naiveBayes(am ~ ., data = train_full)
```

Fitting naiveBayes model on the full training data

```
predict.nb <- predict(model.nb, test_full)
```

Making predictions on the test data using the naive bayes model

```
confusionMatrix(predict.nb, factor(test_full$am))
```

Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##           0 2 0
##           1 1 3
##
##           Accuracy : 0.8333
##           95% CI : (0.3588, 0.9958)
##           No Information Rate : 0.5
##           P-Value [Acc > NIR] : 0.1094
##
##           Kappa : 0.6667
##
##           Mcnemar's Test P-Value : 1.0000
##
```



```
##           Sensitivity : 0.6667
##           Specificity : 1.0000
##           Pos Pred Value : 1.0000
##           Neg Pred Value : 0.7500
##           Prevalence : 0.5000
##           Detection Rate : 0.3333
##           Detection Prevalence : 0.3333
##           Balanced Accuracy : 0.8333
##
##           'Positive' Class : 0
##
```

We got an accuracy of 0.8333, sensitivity of 0.6667 and a specificity of 1.000 using Naive Bayes model.

SVM (Support Vector Machine Algorithm)

Identify numeric features that need scaling

Upon examining all features distribution, we found that vs and am are binary features and donot require scaling.

But cyl although being categorical features have value 4, 6 or 8 so I think we need to scale these features as svm is a distance

based algorithm and distance can impact the performance of the model.

```
binary_features <- c('am', 'vs')
scalable_features <- setdiff(colnames(data_full), binary_features)
```

```
selected_data_full <- data_full[, scalable_features]
```

List of features to exclude because we need to scale other feature except categorical features

Apply Min-Max scaling to selected features

```
scaled_data_full <- apply(selected_data_full, 2, function(x) (x - min(x)) / (max(x) - min(x)))
```

Combine scaled selected features with remaining features

```
scaled_data_full <- cbind(data_full[, !names(data_full) %in% scalable_features], scaled_data_full)
```

```
train_scaled_full <- scaled_data_full[index ==1, ]
test_scaled_full <- scaled_data_full[index ==2, ]
```

Creating training and testing data after scaling the features for using svm algorithm

```
model.svm <- svm(am ~ ., data = train_scaled_full,
                 type = 'C-classification',
                 kernel = 'linear')
```

Fitting Liner SVM algorithm on the training datasets

```
predict_svm <- predict(model.svm, test_scaled_full)
```

Making predictions on the test datasets

```
confusionMatrix(predict_svm, factor(test_scaled_full$am))
```

Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##           0 3 0
##           1 0 3
##
##           Accuracy : 1
##           95% CI : (0.5407, 1)
##    No Information Rate : 0.5
##    P-Value [Acc > NIR] : 0.01563
##
##           Kappa : 1
##
## Mcnemar's Test P-Value : NA
##
##           Sensitivity : 1.0
##           Specificity : 1.0
##    Pos Pred Value : 1.0
##    Neg Pred Value : 1.0
##           Prevalence : 0.5
##    Detection Rate : 0.5
##    Detection Prevalence : 0.5
##    Balanced Accuracy : 1.0
##
##    'Positive' Class : 0
##
```

Upon applying Linear SVM Algorithm, we got an accuracy of 1.000, sensitivity of 1.000 and a specificity of 1.000.

Decision Trees

```
library(party)
```

For fitting decision trees, we need to install party package

```
## Loading required package: grid
## Loading required package: mvtnorm
## Loading required package: modeltools
## Loading required package: stats4
##
## Attaching package: 'modeltools'
## The following object is masked from 'package:car':
##
##      Predict
## Loading required package: strucchange
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##      as.Date, as.Date.numeric
## Loading required package: sandwich
```

```
model.dt <- ctree(am~., data = train_full)
```

Fit the decision trees in the training datasets

```
model.dt
```

check the model

```
##
## Conditional inference tree with 2 terminal nodes
##
## Response: am
## Inputs: mpg, cyl, disp, hp, drat, wt, qsec, vs, gear, carb
## Number of observations: 26
##
## 1) gear <= 3; criterion = 0.999, statistic = 16.29
## 2)* weights = 13
## 1) gear > 3
## 3)* weights = 13
```

```
predict_dt <- predict(model.dt, test_full)
```

Making predictions on the test dataset

```
confusionMatrix(predict_dt, factor(test_full$am))
```

Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##           0 2 0
##           1 1 3
##
##           Accuracy : 0.8333
##           95% CI : (0.3588, 0.9958)
##           No Information Rate : 0.5
##           P-Value [Acc > NIR] : 0.1094
##
##           Kappa : 0.6667
##
## Mcnemar's Test P-Value : 1.0000
##
##           Sensitivity : 0.6667
##           Specificity : 1.0000
##           Pos Pred Value : 1.0000
##           Neg Pred Value : 0.7500
##           Prevalence : 0.5000
##           Detection Rate : 0.3333
##           Detection Prevalence : 0.3333
##           Balanced Accuracy : 0.8333
##
##           'Positive' Class : 0
##
```

Upon applying Decision Tree Algorithm, We got an accuracy of 0.833, sensitivity of 0.6667 and a specificity of 1.000.

```
library(ipred)
```

Bagging: Loading a library named ipred for bagging

```
model.bagging <- bagging(am ~ ., data = train_full)
```

```
print(model.bagging)
```

Fitting the training data in bagging model

```
##
## Bagging classification trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = am ~ ., data = train_full)
```

```
predict_bagging <- predict(model.bagging, test_full)
```

Making predictions on the testing dataset using bagging model

```
confusionMatrix(predict_bagging, factor(test_full$am))
```

Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##           0 3 0
##           1 0 3
##
##           Accuracy : 1
##           95% CI : (0.5407, 1)
##       No Information Rate : 0.5
##       P-Value [Acc > NIR] : 0.01563
##
##           Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##       Sensitivity : 1.0
##       Specificity : 1.0
##       Pos Pred Value : 1.0
##       Neg Pred Value : 1.0
##       Prevalence : 0.5
##       Detection Rate : 0.5
##       Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
##
##       'Positive' Class : 0
##
```

Upon applying Bagging Algorithm, We got an accuracy of 1.00, sensitivity of 1.00 and a specificity of 1.000.

```
library(randomForest)
```

Loading a library named randomForest for training random forest algorithm

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:gridExtra':
##
##      combine
## The following object is masked from 'package:ggplot2':
##
##      margin
```

```
model.rf <- randomForest(am ~ ., data = train_full)
```

Fitting randomForest model on the training dataset

```
model.rf
```

Checking the model

```
##
## Call:
## randomForest(formula = am ~ ., data = train_full)
##           Type of random forest: classification
##           Number of trees: 500
## No. of variables tried at each split: 3
##
##           OOB estimate of  error rate: 11.54%
## Confusion matrix:
##      0 1 class.error
## 0 14 2      0.125
## 1  1 9      0.100
```

```
predict_rf <- predict(model.rf, test_full)
```

Making predictions on the test dataset

```
confusionMatrix(predict_rf, test_full$am)
```

Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##           0 3 0
##           1 0 3
##
##           Accuracy : 1
##           95% CI : (0.5407, 1)
##           No Information Rate : 0.5
##           P-Value [Acc > NIR] : 0.01563
##
##           Kappa : 1
##
## Mcnemar's Test P-Value : NA
##
##           Sensitivity : 1.0
##           Specificity : 1.0
##           Pos Pred Value : 1.0
##           Neg Pred Value : 1.0
##           Prevalence : 0.5
##           Detection Rate : 0.5
```

```
## Detection Prevalence : 0.5
## Balanced Accuracy : 1.0
##
## 'Positive' Class : 0
##
```

Upon applying Random Forest Algorithm, We got an accuracy of 1.00, sensitivity of 1.00 and a specificity of 1.000.

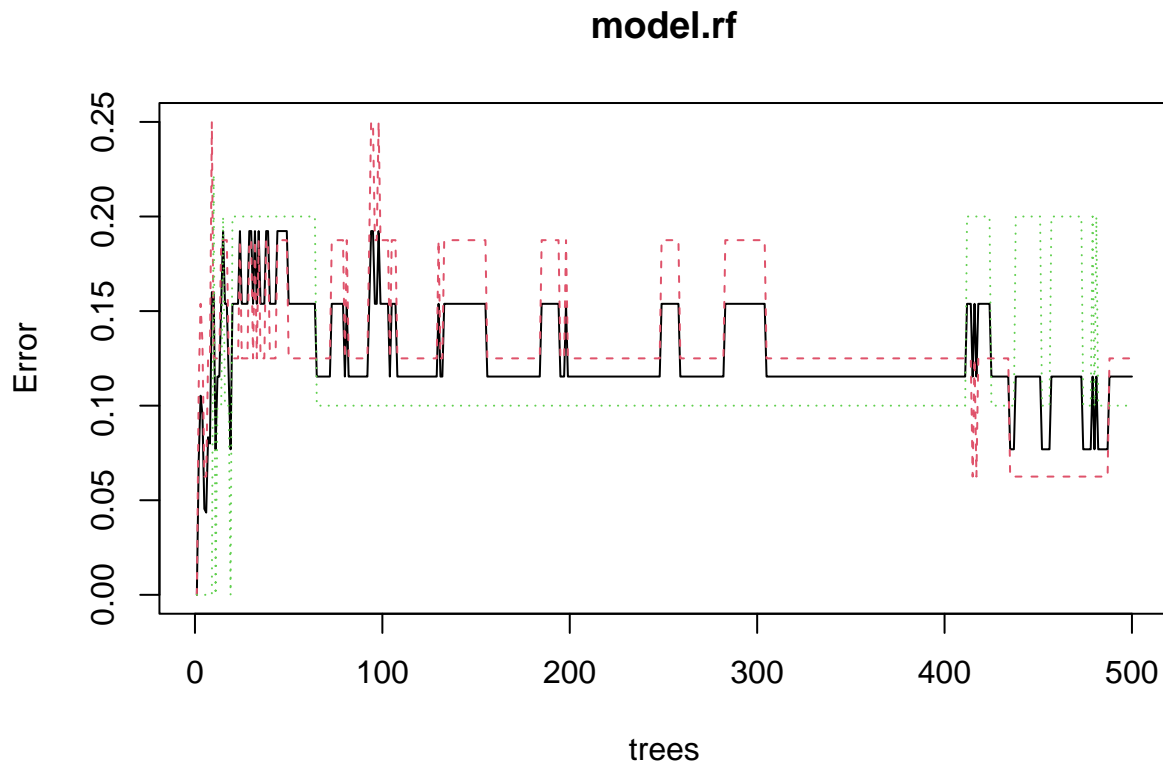
Tuned Random Forest

For tuning the graph, we need to plot the graph to see at what number of trees the random forest has minimum error.

From this graph, we can see that the graph look kind of stationary throughout but we can see the lowest possible error

```
plot(model.rf)
```

in between 200 and 300 so taking number of trees = 250 for further tuning the model.



```
#### install.packages("xgboost")
library(xgboost)
```

Gradient Boosting: Loading a new library for training xgboost algorithm (installing and loading)

```
model.xgb <- train(am ~ ., data = train_full, method = "xgbTree", verbose = FALSE)
```

Fitting the model on the training datasets

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Making predictions on the test dataset

Confusion Matrix

```
##          Neg Pred Value : 1.0
##          Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##          Balanced Accuracy : 1.0
##
##          'Positive' Class : 0
##
```

Upon applying XGBoost Algorithm, We got an accuracy of 1.00, sensitivity of 1.00 and a specificity of 1.000.

Conclusion, being a simple classification task, XGBoost, RandomForest, SVM and Logistic Regression all got 100% accuracy on the test set.