64037_group_proj

wliu16

2023-04-21

```
#Remove large object
rm(list=ls())
gc() # Garbage collection
```

```
##           used  (Mb) gc trigger  (Mb) limit (Mb) max used  (Mb)
## Ncells 2730074 145.9    4601752 245.8         NA  4601752 245.8
## Vcells 4414695  33.7   10146329  77.5     102400  7932994  60.6
```

Data Cleaning - Remove NAs and outliers

Data Cleaning- preprocess and remove near zero variance

```
# Normalization, remove "near zero variance" (loss is removed as nzv)
data_processing_1 <- preProcess(df, method = c("center", "scale", "nzv"))
df_scaled_b4rank <- predict(data_processing_1, df) # 22 removed for near zero variance

# loss is not removed as zv
data_processing_2 <- preProcess(df, method = c("center", "scale", "zv"))
df_scaled_b4rank_loss <- predict(data_processing_2, df)
```

Remove IVs highly correlated with each other and low correlated to target

```
#PLS model for variable selection

set.seed(123)
X <- training_0[, !(names(training_0) %in% "default")]
Y <- training_0$default
Y_numeric <- ifelse(Y == "Class1", 1, 0) # Replace "Class1" with the name of one of the binary classes
train_control <- trainControl(method = "repeatedcv", number = 10, repeats = 3,
                              classProbs = TRUE,
                              summaryFunction = twoClassSummary)

# Convert the factor levels in Y to valid R variable names
Y_clean <- factor(Y, labels = make.names(levels(Y), unique = TRUE))

pls_model <- train(X, Y_clean, method = "pls",
                         tuneLength = 20,
                         trControl = train_control)
```

```
## Warning in train.default(X, Y_clean, method = "pls", tuneLength = 20, trControl
## = train_control): The metric "Accuracy" was not in the result set. ROC will be
## used instead.
```

```
pls_model
```

```
## Partial Least Squares
```

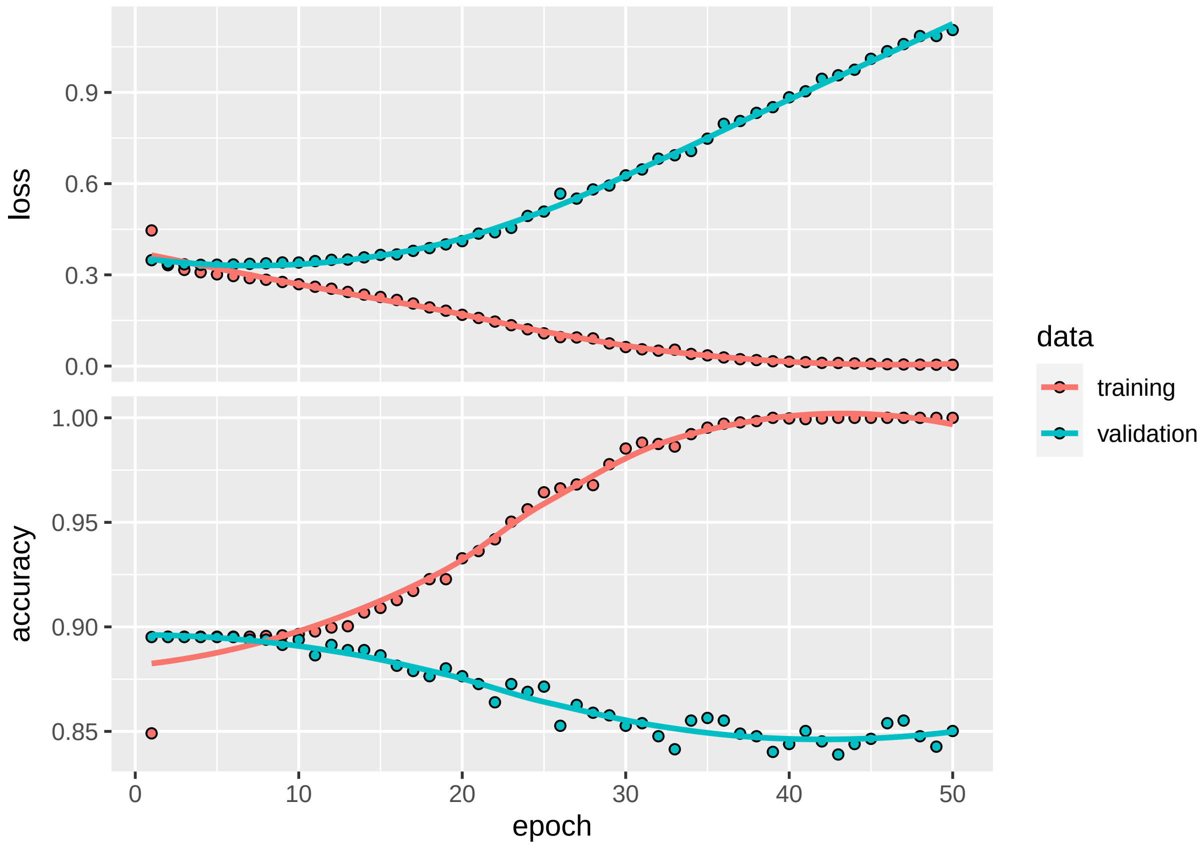
```
##
## 4001 samples
##   57 predictor
##    2 classes: 'X1', 'X2'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3601, 3601, 3601, 3601, 3601, 3601, ...
## Resampling results across tuning parameters:
##
##   ncomp  ROC        Sens  Spec
##    1     0.6293858  1     0
##    2     0.6415022  1     0
##    3     0.6417994  1     0
##    4     0.6438274  1     0
##    5     0.6407832  1     0
##    6     0.6407058  1     0
##    7     0.6396143  1     0
##    8     0.6410412  1     0
##    9     0.6400789  1     0
##   10     0.6415316  1     0
##   11     0.6418584  1     0
##   12     0.6415774  1     0
##   13     0.6415432  1     0
##   14     0.6415148  1     0
##   15     0.6412459  1     0
##   16     0.6410266  1     0
##   17     0.6406976  1     0
##   18     0.6405114  1     0
##   19     0.6406809  1     0
##   20     0.6407044  1     0
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 4.
```

```
# Generate predictions on the test set
predictions_pls <- predict(pls_model, testing_0, type = "prob")
predicted_labels_pls <- factor(ifelse(predictions_pls$'X1' > 0.5, 1, 0), levels = c(1, 0))
levels(predicted_labels_pls) <- levels(testing_0$default)
confusion_mat_pls <- confusionMatrix(predicted_labels_pls, testing_0$default)
confusion_mat_pls
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 895 104
##          2   0   0
##
##                Accuracy : 0.8959
##                  95% CI : (0.8753, 0.9141)
##     No Information Rate : 0.8959
##     P-Value [Acc > NIR] : 0.5261
##
##                   Kappa : 0
```

```
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.8959
##          Neg Pred Value :    NaN
##              Prevalence : 0.8959
##          Detection Rate : 0.8959
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
```

```r
# NNet

set.seed(123)

df_train <- training_0
df_test <- testing_0

# Prepare the data for the neural network model
X_train_nn <- as.matrix(df_train[, -ncol(df_train)])
y_train_nn <- as.numeric(df_train[, "default"]) - 1
X_test_nn <- as.matrix(df_test[, -ncol(df_test)])
y_test_nn <- as.numeric(df_test[, "default"]) - 1

# Build the neural network model
model_nn <- keras_model_sequential() %>%
  layer_dense(units = 60, activation = "relu", input_shape = ncol(X_train_nn)) %>%
  layer_dense(units = 60, activation = "relu") %>%
  layer_dense(units = 30, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

# Compile the model
model_nn %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

# Train the model
history <- model_nn %>% fit(
  X_train_nn, y_train_nn,
  epochs = 50,
  batch_size = 100,
  validation_split = 0.2
)
plot(history)
```




```r
# Generate predictions on the test set
predictions_nn <- predict(model_nn, X_test_nn)
predicted_labels_nn <- ifelse(predictions_nn > 0.5, 1, 0) # Convert the predictions to class labels
confusion_mat_nn <- confusionMatrix(as.factor(predicted_labels_nn), as.factor(y_test_nn))
confusion_mat_nn
```

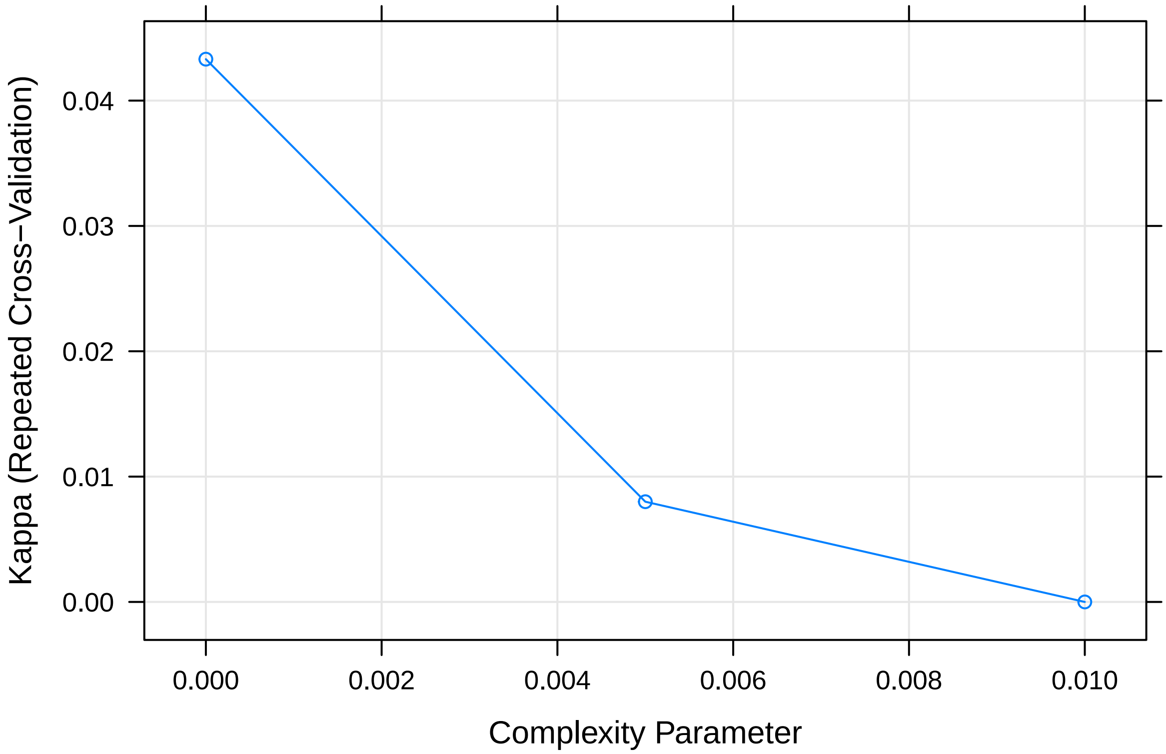
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 838  97
##          1  57   7
##
##                Accuracy : 0.8458
##                  95% CI : (0.8219, 0.8677)
##     No Information Rate : 0.8959
##     P-Value [Acc > NIR] : 1.000000
##
##                   Kappa : 0.0044
##
##  Mcnemar's Test P-Value : 0.001674
##
##             Sensitivity : 0.93631
##             Specificity : 0.06731
##          Pos Pred Value : 0.89626
##          Neg Pred Value : 0.10938
##              Prevalence : 0.89590
```

```
##          Detection Rate : 0.83884
##    Detection Prevalence : 0.93594
##       Balanced Accuracy : 0.50181
##
##        'Positive' Class : 0
##
```

```
# Decision Tree
control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

set.seed(123)
tuning_grid <- expand.grid(
  cp = seq(0, 0.01, by = 0.005),
  minsplit = c(2, 5, 10, 20),
  minbucket = c(1, 5, 10),
  maxdepth = c(5, 7, 9, 11, 13, 15, 50)
)
# Train the decision tree model
mod_tree <- train(default ~ ., data = training_0,
                  method = "rpart",
                  metric = "Kappa",
                  trControl = control,
                  tuneGrid = data.frame(cp = tuning_grid$cp))

mod_tree
```

```
## CART
##
## 4001 samples
##   57 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3601, 3601, 3601, 3601, 3601, 3601, ...
## Resampling results across tuning parameters:
##
##   cp     Accuracy   Kappa
##   0.000  0.8509536  0.043306540
##   0.005  0.8919436  0.007996438
##   0.010  0.8952763  0.000000000
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
```

```
plot(mod_tree)
```




```r
# Generate predictions on the test set
predictions_tree <- predict(mod_tree, testing_0)
confusion_mat_tree <- confusionMatrix(predictions_tree, testing_0$default)
confusion_mat_tree
```

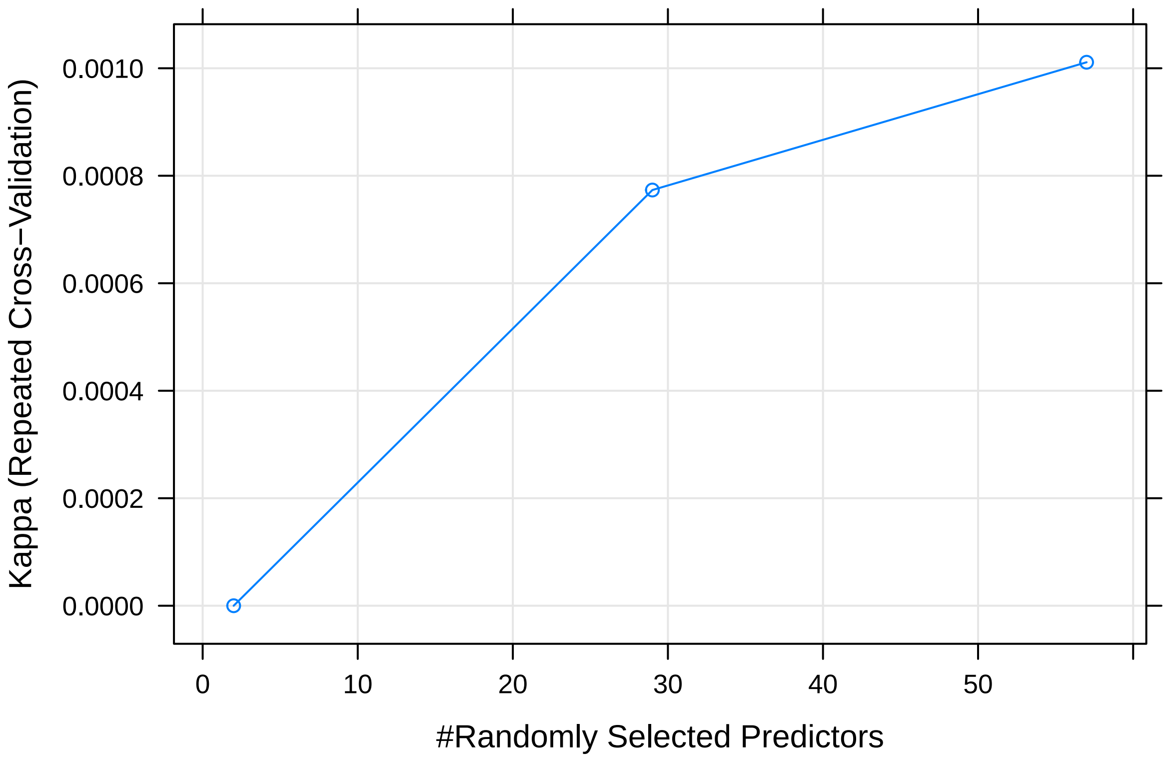
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 834  94
##          2  61  10
##
##                Accuracy : 0.8448
##                  95% CI : (0.8209, 0.8668)
##     No Information Rate : 0.8959
##     P-Value [Acc > NIR] : 1.00000
##
##                   Kappa : 0.0326
##
##  Mcnemar's Test P-Value : 0.01016
##
##             Sensitivity : 0.93184
##             Specificity : 0.09615
##          Pos Pred Value : 0.89871
##          Neg Pred Value : 0.14085
##              Prevalence : 0.89590
##          Detection Rate : 0.83483
##    Detection Prevalence : 0.92893
##       Balanced Accuracy : 0.51400
##
```

```
##        'Positive' Class : 1
##
```

```r
#RF
control <- trainControl(method = "repeatedcv", number = 5, repeats = 3)

set.seed(123)
#tuning_grid <- expand.grid(mtry = seq(1, ncol(training_0) - 1, by = 1))

# Train the Random Forest model with an extensive hyperparameter search
mod_rf_tuned <- train(default ~ ., data = training_0,
                      method = "rf",
                      metric = "Kappa",
                      trControl = control)
                      #tuneGrid = tuning_grid)

mod_rf_tuned
```

```
## Random Forest
##
## 4001 samples
##   57 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 3201, 3202, 3200, 3201, 3200, 3202, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8952765  0.0000000000
##   29    0.8944434  0.0007734536
##   57    0.8939437  0.0010111796
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 57.
```

```r
plot(mod_rf_tuned)
```




```r
# Generate predictions on the test set
predictions_rf_tuned <- predict(mod_rf_tuned, testing_0)
confusion_mat_rf_tuned <- confusionMatrix(predictions_rf_tuned, testing_0$default)
confusion_mat_rf_tuned
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 894 104
##          2   1   0
##
##                Accuracy : 0.8949
##                  95% CI : (0.8742, 0.9132)
##     No Information Rate : 0.8959
##     P-Value [Acc > NIR] : 0.567
##
##                   Kappa : -0.002
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9989
##             Specificity : 0.0000
##          Pos Pred Value : 0.8958
##          Neg Pred Value : 0.0000
##              Prevalence : 0.8959
##          Detection Rate : 0.8949
##    Detection Prevalence : 0.9990
##       Balanced Accuracy : 0.4994
##
```

```
##        'Positive' Class : 1
##
```

```r
# SVM Linear
control <- trainControl("repeatedcv", number = 5, repeats = 3)

set.seed(123)
grid <- expand.grid(C = c(0.001, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2,5, 20))
mod_svm_ln <- train(default~., data = training_0,
                    method = "svmLinear",
                    trControl = control,
                    metric = "Kappa",
                    tuneGrid = grid,
                    tuneLength = 20)
mod_svm_ln
```

```
## Support Vector Machines with Linear Kernel
##
## 4001 samples
##   57 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 3201, 3202, 3200, 3201, 3200, 3202, ...
## Resampling results across tuning parameters:
##
##   C        Accuracy   Kappa
##    0.001   0.8952765  0
##    0.050   0.8952765  0
##    0.100   0.8952765  0
##    0.250   0.8952765  0
##    0.500   0.8952765  0
##    0.750   0.8952765  0
##    1.000   0.8952765  0
##    1.250   0.8952765  0
##    1.500   0.8952765  0
##    2.000   0.8952765  0
##    5.000   0.8952765  0
##   20.000   0.8952765  0
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.001.
```

```r
# Generate predictions on the test set
predictions_svm_ln <- predict(mod_svm_ln, testing_0)
confusion_mat_svm_ln <- confusionMatrix(predictions_svm_ln, testing_0$default)
confusion_mat_svm_ln
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 895 104
##          2   0   0
##
```

```
##                Accuracy : 0.8959
##                  95% CI : (0.8753, 0.9141)
##     No Information Rate : 0.8959
##     P-Value [Acc > NIR] : 0.5261
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.8959
##          Neg Pred Value :    NaN
##              Prevalence : 0.8959
##          Detection Rate : 0.8959
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
```

```
# SVM Radial
control <- trainControl("cv", number = 5)

set.seed(123)
grid <- expand.grid(C = c(0.1, 0.5, 1, 2, 5, 10),
                    sigma = c(0.01, 0.02, 0.05, 0.07, 0.1, 1, 2, 5))

mod_svm_rbf <- train(default ~ ., data = training_0,
                     method = "svmRadial",
                     metric = "Kappa",
                     trControl = control,
                     tuneGrid = grid)
mod_svm_rbf
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 4001 samples
##   57 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3201, 3202, 3200, 3201, 3200
## Resampling results across tuning parameters:
##
##   C     sigma  Accuracy   Kappa
##    0.1  0.01   0.8952765   0.0000000000
##    0.1  0.02   0.8952765   0.0000000000
##    0.1  0.05   0.8952765   0.0000000000
##    0.1  0.07   0.8952765   0.0000000000
##    0.1  0.10   0.8952765   0.0000000000
##    0.1  1.00   0.8952765   0.0000000000
##    0.1  2.00   0.8952765   0.0000000000
##    0.1  5.00   0.8952765   0.0000000000
```

```
##    0.5  0.01   0.8952765   0.0000000000
##    0.5  0.02   0.8952765   0.0000000000
##    0.5  0.05   0.8952765   0.0000000000
##    0.5  0.07   0.8952765   0.0000000000
##    0.5  0.10   0.8952765   0.0000000000
##    0.5  1.00   0.8952765   0.0000000000
##    0.5  2.00   0.8952765   0.0000000000
##    0.5  5.00   0.8952765   0.0000000000
##    1.0  0.01   0.8952765   0.0000000000
##    1.0  0.02   0.8952765   0.0000000000
##    1.0  0.05   0.8952765   0.0000000000
##    1.0  0.07   0.8952765   0.0000000000
##    1.0  0.10   0.8952765   0.0000000000
##    1.0  1.00   0.8952765   0.0000000000
##    1.0  2.00   0.8952765   0.0000000000
##    1.0  5.00   0.8952765   0.0000000000
##    2.0  0.01   0.8952765   0.0000000000
##    2.0  0.02   0.8952765   0.0000000000
##    2.0  0.05   0.8940262   0.0049571927
##    2.0  0.07   0.8940262   0.0012864802
##    2.0  0.10   0.8950268  -0.0004947215
##    2.0  1.00   0.8952765   0.0000000000
##    2.0  2.00   0.8952765   0.0000000000
##    2.0  5.00   0.8952765   0.0000000000
##    5.0  0.01   0.8952762   0.0037274599
##    5.0  0.02   0.8882777   0.0146655896
##    5.0  0.05   0.8882783   0.0215558108
##    5.0  0.07   0.8935271   0.0219585339
##    5.0  0.10   0.8955265   0.0079158503
##    5.0  1.00   0.8952765   0.0000000000
##    5.0  2.00   0.8952765   0.0000000000
##    5.0  5.00   0.8952765   0.0000000000
##   10.0  0.01   0.8932777   0.0322121034
##   10.0  0.02   0.8777833   0.0480159295
##   10.0  0.05   0.8882780   0.0248733770
##   10.0  0.07   0.8935271   0.0219585339
##   10.0  0.10   0.8955265   0.0079158503
##   10.0  1.00   0.8952765   0.0000000000
##   10.0  2.00   0.8952765   0.0000000000
##   10.0  5.00   0.8952765   0.0000000000
##
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.02 and C = 10.
```

```r
# Generate predictions on the test set
predictions_svm_rbf <- predict(mod_svm_rbf, testing_0)
confusion_mat_svm_rbf <- confusionMatrix(predictions_svm_rbf, testing_0$default)
confusion_mat_svm_rbf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 868 101
##          2  27   3
```

```
##
##                Accuracy : 0.8719
##                  95% CI : (0.8495, 0.892)
##     No Information Rate : 0.8959
##     P-Value [Acc > NIR] : 0.9933
##
##                   Kappa : -0.0019
##
##  Mcnemar's Test P-Value : 1.101e-10
##
##             Sensitivity : 0.96983
##             Specificity : 0.02885
##          Pos Pred Value : 0.89577
##          Neg Pred Value : 0.10000
##              Prevalence : 0.89590
##          Detection Rate : 0.86887
##    Detection Prevalence : 0.96997
##       Balanced Accuracy : 0.49934
##
##        'Positive' Class : 1
##
```

```r
set.seed(123)

control <- trainControl("cv", number = 5)
defaultGrid <- expand.grid(interaction.depth = seq(1, 5, by=1),
                           n.trees = c(50, 100, 150),
                           shrinkage = c(0.01, 0.1),
                           n.minobsinnode = c(10, 20))
mod_gbm <- train(default~., data = training_0,
                       method = "gbm", ##gradient boosting machine
                       trControl = control,
                       verbose = FALSE,
                       tuneGrid = defaultGrid,
                       metric = "Kappa")

mod_gbm
```

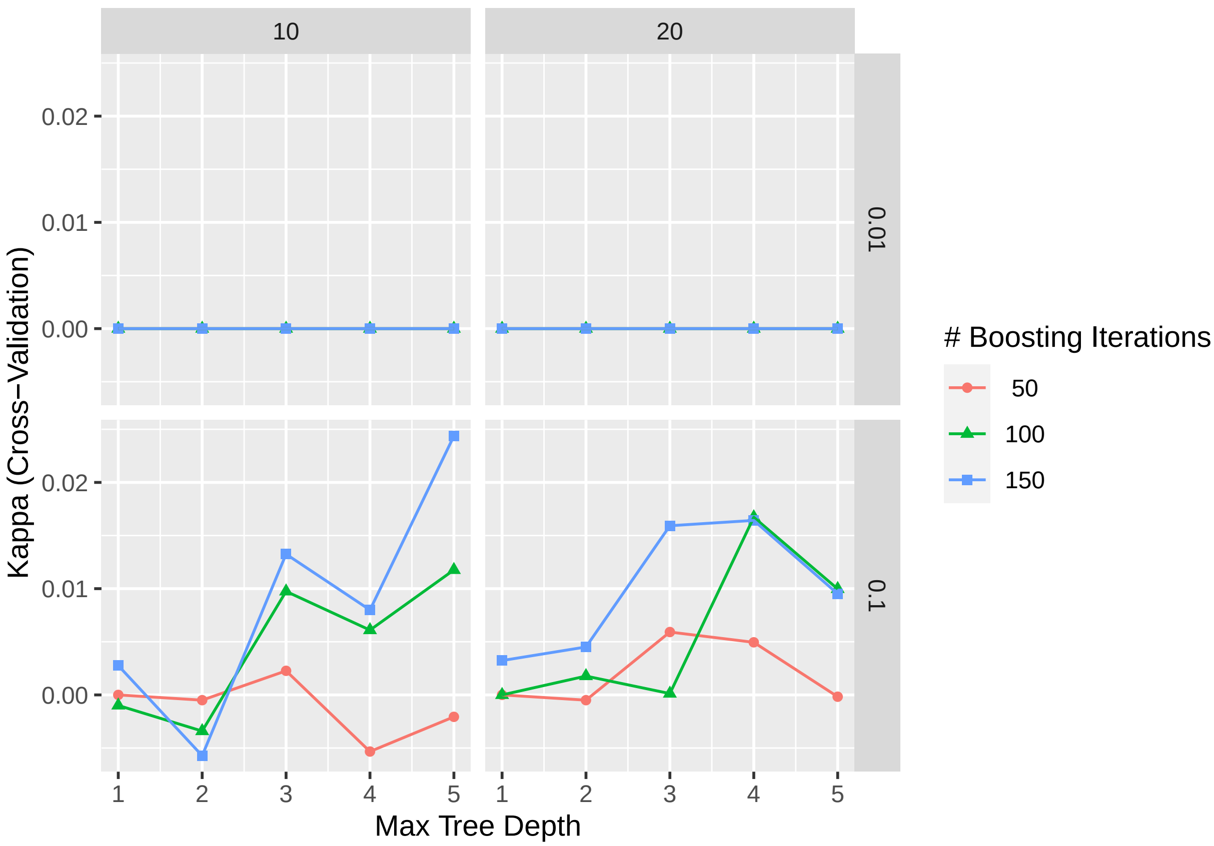
```
## Stochastic Gradient Boosting
##
## 4001 samples
##   57 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3201, 3202, 3200, 3201, 3200
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  n.trees  Accuracy
##   0.01       1                  10               50      0.8952765
##   0.01       1                  10              100      0.8952765
##   0.01       1                  10              150      0.8952765
##   0.01       1                  20               50      0.8952765
##   0.01       1                  20              100      0.8952765
```

```
##   0.01      1             20           150      0.8952765
##   0.01      2             10            50      0.8952765
##   0.01      2             10           100      0.8952765
##   0.01      2             10           150      0.8952765
##   0.01      2             20            50      0.8952765
##   0.01      2             20           100      0.8952765
##   0.01      2             20           150      0.8952765
##   0.01      3             10            50      0.8952765
##   0.01      3             10           100      0.8952765
##   0.01      3             10           150      0.8952765
##   0.01      3             20            50      0.8952765
##   0.01      3             20           100      0.8952765
##   0.01      3             20           150      0.8952765
##   0.01      4             10            50      0.8952765
##   0.01      4             10           100      0.8952765
##   0.01      4             10           150      0.8952765
##   0.01      4             20            50      0.8952765
##   0.01      4             20           100      0.8952765
##   0.01      4             20           150      0.8952765
##   0.01      5             10            50      0.8952765
##   0.01      5             10           100      0.8952765
##   0.01      5             10           150      0.8952765
##   0.01      5             20            50      0.8952765
##   0.01      5             20           100      0.8952765
##   0.01      5             20           150      0.8952765
##   0.10      1             10            50      0.8952765
##   0.10      1             10           100      0.8947762
##   0.10      1             10           150      0.8947752
##   0.10      1             20            50      0.8952765
##   0.10      1             20           100      0.8952765
##   0.10      1             20           150      0.8950268
##   0.10      2             10            50      0.8950262
##   0.10      2             10           100      0.8935255
##   0.10      2             10           150      0.8922749
##   0.10      2             20            50      0.8950265
##   0.10      2             20           100      0.8942758
##   0.10      2             20           150      0.8937761
##   0.10      3             10            50      0.8945252
##   0.10      3             10           100      0.8927761
##   0.10      3             10           150      0.8927762
##   0.10      3             20            50      0.8945258
##   0.10      3             20           100      0.8915252
##   0.10      3             20           150      0.8922768
##   0.10      4             10            50      0.8925261
##   0.10      4             10           100      0.8927768
##   0.10      4             10           150      0.8917755
##   0.10      4             20            50      0.8940255
##   0.10      4             20           100      0.8927749
##   0.10      4             20           150      0.8907743
##   0.10      5             10            50      0.8922765
##   0.10      5             10           100      0.8920265
##   0.10      5             10           150      0.8915268
##   0.10      5             20            50      0.8932768
##   0.10      5             20           100      0.8912783
```

```
##   0.10       5                   20                150       0.8892783
##   Kappa
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##    0.0000000000
##   -0.0009912339
##    0.0027627333
##    0.0000000000
##    0.0000000000
##    0.0032321331
##   -0.0004958925
##   -0.0034051180
##   -0.0057234886
##   -0.0004953414
##    0.0017720651
##    0.0045107609
##    0.0022673919
##    0.0097452320
##    0.0132492507
##    0.0059179821
##    0.0001242025
##    0.0159241586
##   -0.0053313763
##    0.0060992365
##    0.0079936589
##    0.0049493120
```

```
##    0.0167244703
##    0.0164260897
##   -0.0020632597
##    0.0117639013
##    0.0243709866
##   -0.0001704740
##    0.0099730247
##    0.0095170393
##
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  5, shrinkage = 0.1 and n.minobsinnode = 10.
```

```r
ggplot(mod_gbm)
```




```r
# Generate predictions on the test set
predictions_gbm <- predict(mod_gbm, testing_0, type = "prob")
predicted_labels_gbm <- factor(ifelse(predictions_gbm$'1' > 0.5, 1, 0), levels = c(1, 0))
levels(predicted_labels_gbm) <- levels(testing_0$default)
confusion_mat_gbm <- confusionMatrix(predicted_labels_gbm, testing_0$default)
confusion_mat_gbm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 893 103
##          2   2   1
```

```
##
##                Accuracy : 0.8949
##                  95% CI : (0.8742, 0.9132)
##     No Information Rate : 0.8959
##     P-Value [Acc > NIR] : 0.567
##
##                   Kappa : 0.0129
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.997765
##             Specificity : 0.009615
##          Pos Pred Value : 0.896586
##          Neg Pred Value : 0.333333
##              Prevalence : 0.895896
##          Detection Rate : 0.893894
##    Detection Prevalence : 0.996997
##       Balanced Accuracy : 0.503690
##
##        'Positive' Class : 1
##
```

```
class_proportions <- table(training_0$default) / length(training_0$default)
class_weights <- 1 / class_proportions
mod_gbm_weighted <- train(default ~ ., data = training_0,
                          method = "gbm",
                          trControl = control,
                          verbose = FALSE,
                          tuneGrid = defaultGrid,
                          metric = "Kappa",
                          weights = class_weights[training_0$default])
mod_gbm_weighted
```

```
## Stochastic Gradient Boosting
##
## 4001 samples
##   57 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3201, 3200, 3202, 3201, 3200
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  n.trees  Accuracy   Kappa
##   0.01       1                  10               50      0.5003708  0.05979764
##   0.01       1                  10              100      0.5278680  0.06390298
##   0.01       1                  10              150      0.5451068  0.06768218
##   0.01       1                  20               50      0.5016227  0.06015030
##   0.01       1                  20              100      0.5321099  0.06908832
##   0.01       1                  20              150      0.5476074  0.07512802
##   0.01       2                  10               50      0.5598677  0.07669969
##   0.01       2                  10              100      0.5771043  0.07903061
##   0.01       2                  10              150      0.5921037  0.08125207
##   0.01       2                  20               50      0.5896062  0.08313246
```

```
##    0.01       2                  20                 100      0.5918580  0.08847338
##    0.01       2                  20                 150      0.5973496  0.08702954
##    0.01       3                  10                  50      0.6110968  0.08457371
##    0.01       3                  10                 100      0.6155996  0.08615008
##    0.01       3                  10                 150      0.6243484  0.09023284
##    0.01       3                  20                  50      0.5983287  0.08155099
##    0.01       3                  20                 100      0.6165934  0.08451527
##    0.01       3                  20                 150      0.6245931  0.08972090
##    0.01       4                  10                  50      0.6083477  0.08247182
##    0.01       4                  10                 100      0.6275862  0.09085498
##    0.01       4                  10                 150      0.6373372  0.08941350
##    0.01       4                  20                  50      0.6230862  0.09387322
##    0.01       4                  20                 100      0.6280984  0.09202235
##    0.01       4                  20                 150      0.6405903  0.09815456
##    0.01       5                  10                  50      0.6478362  0.10066177
##    0.01       5                  10                 100      0.6535909  0.09390984
##    0.01       5                  10                 150      0.6608378  0.09449768
##    0.01       5                  20                  50      0.6363409  0.08696184
##    0.01       5                  20                 100      0.6428418  0.08693469
##    0.01       5                  20                 150      0.6573353  0.09199238
##    0.10       1                  10                  50      0.5966034  0.08234620
##    0.10       1                  10                 100      0.6135965  0.07933627
##    0.10       1                  10                 150      0.6240997  0.06667278
##    0.10       1                  20                  50      0.5988374  0.09155784
##    0.10       1                  20                 100      0.6038390  0.08629859
##    0.10       1                  20                 150      0.6195837  0.08121065
##    0.10       2                  10                  50      0.6223350  0.08207058
##    0.10       2                  10                 100      0.6535837  0.07328833
##    0.10       2                  10                 150      0.6705806  0.07890336
##    0.10       2                  20                  50      0.6303337  0.08753484
##    0.10       2                  20                 100      0.6535797  0.08140261
##    0.10       2                  20                 150      0.6803185  0.08962512
##    0.10       3                  10                  50      0.6590859  0.09530112
##    0.10       3                  10                 100      0.6910835  0.08619060
##    0.10       3                  10                 150      0.7093260  0.07510947
##    0.10       3                  20                  50      0.6573372  0.09389639
##    0.10       3                  20                 100      0.6825810  0.07226377
##    0.10       3                  20                 150      0.7075735  0.06747761
##    0.10       4                  10                  50      0.6783263  0.08573046
##    0.10       4                  10                 100      0.7148169  0.07996425
##    0.10       4                  10                 150      0.7405601  0.07393359
##    0.10       4                  20                  50      0.6628322  0.07058175
##    0.10       4                  20                 100      0.7045757  0.04335469
##    0.10       4                  20                 150      0.7315641  0.04391892
##    0.10       5                  10                  50      0.6888300  0.07438633
##    0.10       5                  10                 100      0.7308144  0.08307093
##    0.10       5                  10                 150      0.7598038  0.07804076
##    0.10       5                  20                  50      0.6893269  0.06420332
##    0.10       5                  20                 100      0.7355694  0.06536927
##    0.10       5                  20                 150      0.7565670  0.06457037
##
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  5, shrinkage = 0.01 and n.minobsinnode = 10.
```

```r
# Generate predictions on the test set
predictions_gbm_w <- predict(mod_gbm_weighted, testing_0, type = "prob")
predicted_labels_gbm_w <- factor(ifelse(predictions_gbm_w$'1' > 0.5, 1, 0), levels = c(1, 0))
levels(predicted_labels_gbm_w) <- levels(testing_0$default)
confusion_mat_gbm_w <- confusionMatrix(predicted_labels_gbm_w, testing_0$default)
confusion_mat_gbm_w
```

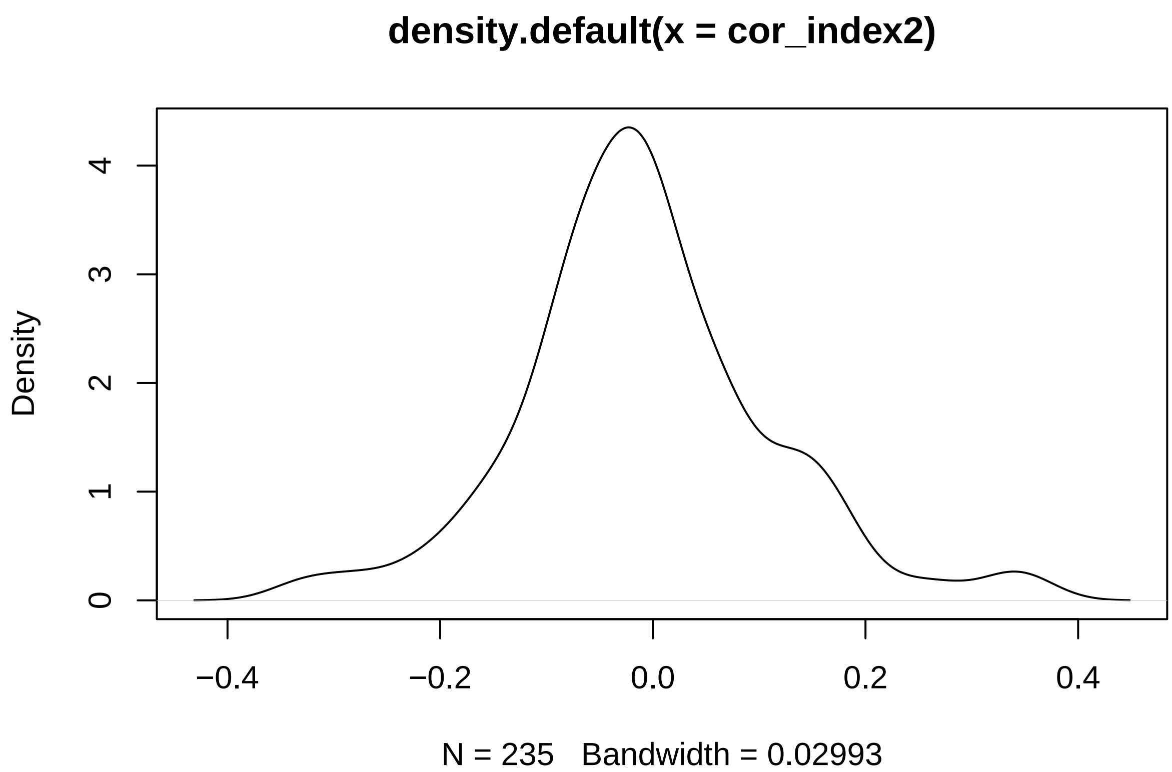
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 565  46
##          2 330  58
##
##                Accuracy : 0.6236
##                  95% CI : (0.5928, 0.6538)
##     No Information Rate : 0.8959
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0856
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6313
##             Specificity : 0.5577
##          Pos Pred Value : 0.9247
##          Neg Pred Value : 0.1495
##              Prevalence : 0.8959
##          Detection Rate : 0.5656
##    Detection Prevalence : 0.6116
##       Balanced Accuracy : 0.5945
##
##        'Positive' Class : 1
##
```

Classification Model Prediction

```r
# Apply model on the testing data

# Prepare the data for the neural network model
X_nn_test <- as.matrix(df2_rmdv_test)
predicted_probs_nn <- predict(model_nn, X_nn_test)

# Convert the predicted probabilities to binary class labels
predicted_labels_nn <- ifelse(predicted_probs_nn > 0.5, 1, 0)

# Convert the predicted_labels_nn to a factor
default_predicted <- factor(predicted_labels_nn)
levels(default_predicted) <- levels(testing_0$default) # levels 1 2
df2_rmdv_test <- cbind(df2_rmdv_test, default_predicted) # Add the predicted default column

# df2_rmdv_test testing dataset
```

Bi-variate analysis: remove variables based on target variable loss

```
# Reorder columns so that they are ranked by number of unique values from highest to lowest.
rm(num_of_unique)
num_of_unique <- c()

for (x in 1:(length(run_data)-1)) {
  num_of_unique = append(num_of_unique,length(unique(run_data[, x])))
}
rkk <- rank(-num_of_unique, ties.method= "first")
rkk<- append(rkk, length(run_data))

rm(reorder_index)
reorder_index <- c()
for (x in 1:(length(run_data))) {
  reorder_index = append(reorder_index,which(rkk == x))
}
run_data <- run_data[, reorder_index]

# In each highly correlated pair, remove the second element
cor_matrix2 <- cor(run_data[, -c(ncol(run_data), ncol(run_data)-1)]) # IV correlation matrix
cor_upper2 <- cor_matrix2 * upper.tri(cor_matrix2, diag = FALSE) # Matrix of 0 and correlation

index2 <- apply(cor_upper2, 1, function(x) paste(colnames(cor_upper2)[which(abs(x) > 0.9)], collapse =
# Store IV names if correlation absolute value is larger than certain threshold
elements2 <- unique(unlist(strsplit(index2, split = ", ")))
# Split names in one string to chr vector and remove duplicated IV names
#elements2

cols_to_keep2 <- setdiff(names(run_data), elements2)
run_data <- run_data[, cols_to_keep2] #  232 IVs remained at 0.9 threshold


# Remove variables that has low correlation with default
cor_index2 <- as.vector(cor(run_data[, -ncol(run_data)], run_data[, ncol(run_data)])) # Calculate corre
plot(density(cor_index2))
```




```r
threshold2 <- 0.2 # Set correlation threshold
selected_indices2 <- which(abs(cor_index2) > threshold2)
selected_variables2 <- names(run_data)[selected_indices2] # 20 variables remained

run_data <- cbind(run_data[, c(selected_variables2)], run_data[, ncol(run_data)])
colnames(run_data)[ncol(run_data)] <- "loss_predicted"
summary(run_data$loss_predicted)
```

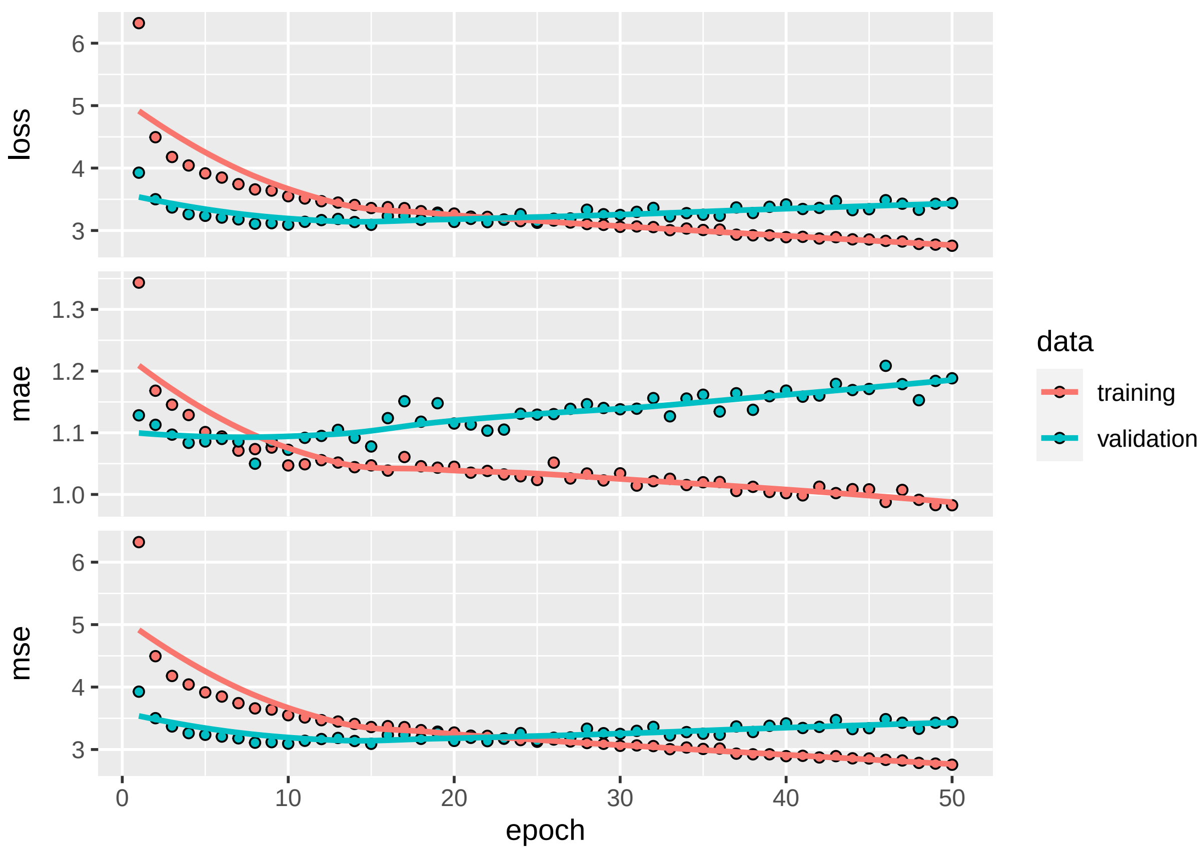
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.03593  0.50730  0.97866  1.75373  2.15707 23.36847
```

```r
#NNet

set.seed(246)

train_control <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 3)

# Define the model architecture
model_nn_loss <- keras_model_sequential() %>%
  layer_dense(units = 150, activation = "relu", input_shape = ncol(training_1) - 1) %>%
  #layer_dense(units = 60, activation = "relu") %>%
  #layer_dense(units = 30, activation = "relu") %>%
  layer_dense(units = 1, activation = "linear")

# Compile the model
model_nn_loss %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(),
```

```r
  metrics = c("mae", "mse")
)

# Train the model
history_nn_loss <- model_nn_loss %>% fit(
  x = as.matrix(training_1[, 1:20]),
  y = as.matrix(training_1[, "loss_predicted"]),
  validation_split = 0.2,
  epochs = 50,
  batch_size = 32,
  trControl = train_control
)

# Evaluate the model
model_nn_loss %>% evaluate(
  x = as.matrix(testing_1[, 1:20]),
  y = as.matrix(testing_1[, "loss_predicted"])
)
```

```
##     loss      mae      mse
## 6.992609 1.362621 6.992609
```

```r
plot(history_nn_loss)
```




```r
# predict on testing data using the trained model
predicted_loss <- predict(model_nn_loss, as.matrix(testing_1[, 1:20]))

# calculate performance metrics
```

```r
MAE <- mean(abs(as.matrix(testing_1[, "loss_predicted"]) - predicted_loss))
MSE <- mean((as.matrix(testing_1[, "loss_predicted"]) - predicted_loss)^2)
RMSE <- sqrt(MSE)
R_squared <- cor(as.matrix(testing_1[, "loss_predicted"]), predicted_loss)^2

MAE
```

```
## [1] 1.362621
```

```r
MSE
```

```
## [1] 6.99261
```

```r
RMSE
```

```
## [1] 2.644354
```

```r
R_squared
```

```
##           [,1]
## [1,] 0.3094586
```

Run the loss prediction

```
```r
# Extract the input variables from the testing data
X_test_loss <- as.matrix(df2_rmdv_test[, 1:20])

# Use the trained model to predict the loss for the NA values in the testing data
y_pred_loss <- predict(model_nn_loss, X_test_loss)

# Replace the NA values in the loss_predicted column with the predicted values
df2_rmdv_test$loss_predicted[is.na(df2_rmdv_test$loss_predicted)] <- y_pred_loss
```

```
## Warning in df2_rmdv_test$loss_predicted[is.na(df2_rmdv_test$loss_predicted)] <-
## y_pred_loss: number of items to replace is not a multiple of replacement length
```

```r
# output the "column_name" column to a csv file
write.csv(df2_rmdv_test$loss_predicted, file = "Results.csv", row.names = FALSE)
```
```