Evaluating Model Accuracy and Interpreting Variable Selection in Predicting Malignant Breast Tumors

Stephanie Daniella Hernandez Prado

2025-12-11

```r
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.4.3
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.3

## Cargando paquete requerido: ggplot2

## Warning: package 'ggplot2' was built under R version 4.4.3

## Cargando paquete requerido: lattice
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.4.3

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Adjuntando el paquete: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(ggplot2)
library(ggcorrplot)
```

```
## Warning: package 'ggcorrplot' was built under R version 4.4.3
```

```r
library(Metrics)
```

```
## Warning: package 'Metrics' was built under R version 4.4.3

##
## Adjuntando el paquete: 'Metrics'

## The following objects are masked from 'package:caret':
##
##     precision, recall
```

Data Set

```r
# Reading the File
Breast_Cancer <- read_excel("~/Stephanie/Maestria/Fall 2025/STST 578/Breas_Cancer/Breast_Cancer.xlsx")

head(Breast_Cancer)
```

```
## # A tibble: 6 x 31
##   radius1 texture1 perimeter1 area1 smoothness1 compactness1 concavity1
##     <dbl>    <dbl>      <dbl> <dbl>       <dbl>        <dbl>      <dbl>
## 1    18.0     10.4      123.  1001       0.118        0.278      0.300
## 2    20.6     17.8      133.  1326       0.0847       0.0786     0.0869
## 3    19.7     21.2      130   1203       0.110        0.160      0.197
## 4    11.4     20.4       77.6  386.      0.142        0.284      0.241
## 5    20.3     14.3      135.  1297       0.100        0.133      0.198
## 6    12.4     15.7       82.6  477.      0.128        0.17       0.158
## # i 24 more variables: concave_points1 <dbl>, symmetry1 <dbl>,
## #   fractal_dimension1 <dbl>, radius2 <dbl>, texture2 <dbl>, perimeter2 <dbl>,
## #   area2 <dbl>, smoothness2 <dbl>, compactness2 <dbl>, concavity2 <dbl>,
## #   concave_points2 <dbl>, symmetry2 <dbl>, fractal_dimension2 <dbl>,
## #   radius3 <dbl>, texture3 <dbl>, perimeter3 <dbl>, area3 <dbl>,
## #   smoothness3 <dbl>, compactness3 <dbl>, concavity3 <dbl>,
## #   concave_points3 <dbl>, symmetry3 <dbl>, fractal_dimension3 <dbl>, ...
```

```r
data <- Breast_Cancer
data$Diagnosis <- as.factor(data$Diagnosis) #Changeing response variable to a factor varaible
```

K-folds CV set up

```r
set.seed(996) #Save the seed for replicavility

folds <- createFolds(data$Diagnosis, k = 5, returnTrain = TRUE) #Set the 5 folds that will
#be used for k-fold CV

Method <- trainControl(method = "cv", index = folds) #We set the K-folds CV method to fit
#the models
```

Base Model

```
suppressWarnings({ #Suopress Warning Messages

  Base_Time <- system.time({  # Count the execution time
    Base_model <- train(
      Diagnosis ~ ., data = data, #model and data used
      method = "glm", #Logistic model setting
      family = binomial, #Binomial response
      trControl = Method #CV setting
    ) #Logistic Model Fitting

  })["elapsed"]
})

summary(Base_model$finalModel)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -2.881e+06  2.816e+05 -10.233  < 2e-16 ***
## radius1              2.427e+06  2.693e+05   9.014  < 2e-16 ***
## texture1             1.958e+05  1.471e+04  13.313  < 2e-16 ***
## perimeter1           1.473e+06  2.464e+04  59.791  < 2e-16 ***
## area1               -1.301e+05  3.907e+03 -33.301  < 2e-16 ***
## smoothness1         -1.525e+08  8.361e+06 -18.234  < 2e-16 ***
## compactness1        -6.428e+06  3.213e+06  -2.001  0.04539 *
## concavity1           1.042e+06  1.408e+06   0.740  0.45959
## concave_points1     -1.716e+07  5.382e+06  -3.188  0.00143 **
## symmetry1            4.049e+07  7.772e+05  52.093  < 2e-16 ***
## fractal_dimension1 -4.233e+07  2.169e+06 -19.519  < 2e-16 ***
## radius2              3.328e+07  1.169e+06  28.478  < 2e-16 ***
## texture2             6.368e+06  2.005e+05  31.763  < 2e-16 ***
## perimeter2           1.701e+06  4.720e+04  36.032  < 2e-16 ***
## area2               -6.393e+05  1.835e+04 -34.840  < 2e-16 ***
## smoothness2          7.492e+08  1.224e+07  61.213  < 2e-16 ***
## compactness2        -1.773e+08  5.732e+06 -30.931  < 2e-16 ***
## concavity2           1.529e+08  5.340e+06  28.624  < 2e-16 ***
## concave_points2     -1.260e+09  4.012e+07 -31.398  < 2e-16 ***
## symmetry2            2.890e+08  4.126e+06  70.055  < 2e-16 ***
## fractal_dimension2  1.512e+09  6.597e+07  22.921  < 2e-16 ***
## radius3             -6.130e+06  2.143e+05 -28.606  < 2e-16 ***
## texture3            -5.832e+05  2.437e+04 -23.935  < 2e-16 ***
## perimeter3          -3.538e+05  1.219e+04 -29.023  < 2e-16 ***
## area3                8.950e+04  2.741e+03  32.658  < 2e-16 ***
## smoothness3         -2.161e+07  3.298e+06  -6.553 5.65e-11 ***
## compactness3         8.986e+06  3.999e+05  22.470  < 2e-16 ***
## concavity3          -3.028e+07  1.523e+06 -19.876  < 2e-16 ***
## concave_points3      1.431e+08  5.471e+06  26.162  < 2e-16 ***
## symmetry3          -2.474e+07  3.392e+05 -72.923  < 2e-16 ***
## fractal_dimension3 -3.698e+07  5.340e+06  -6.926 4.32e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance:   751.44  on 568  degrees of freedom
## Residual deviance: 32006.76  on 538  degrees of freedom
## AIC: 32069
##
## Number of Fisher Scoring iterations: 25
```

We can see that all variables are being used, even if some are not significant, as they have a very low p-value.

Stepwise

```
suppressWarnings({
    Stepwise_Time <- system.time({
      Stepwise_Model <- train(
        Diagnosis ~ ., data = data,
        method = "glmStepAIC",
        trControl = Method,
        direction = "both", #Set stepwise method
        trace = FALSE # Suppress training trace
      )
    })["elapsed"]
})
summary(Stepwise_Model$finalModel)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -5.914e+03  2.619e+05  -0.023    0.982
## radius1            -6.630e+03  1.150e+05  -0.058    0.954
## texture1            1.913e+02  1.345e+03   0.142    0.887
## area1               6.077e+01  1.079e+03   0.056    0.955
## smoothness1         3.914e+04  2.517e+05   0.155    0.876
## compactness1       -8.621e+04  9.326e+05  -0.092    0.926
## concavity1          2.852e+04  2.402e+05   0.119    0.905
## concave_points1     5.886e+04  1.544e+06   0.038    0.970
## symmetry1          -1.964e+04  1.347e+05  -0.146    0.884
## fractal_dimension1  1.626e+05  1.120e+06   0.145    0.885
## perimeter2         -1.253e+03  1.822e+04  -0.069    0.945
## area2               1.562e+02  2.259e+03   0.069    0.945
## smoothness2        -9.793e+04  1.472e+06  -0.067    0.947
## compactness2        9.217e+04  7.142e+05   0.129    0.897
## concavity2         -8.131e+04  1.097e+06  -0.074    0.941
## concave_points2     4.398e+05  6.736e+06   0.065    0.948
## symmetry2          -1.038e+05  2.160e+06  -0.048    0.962
## fractal_dimension2 -1.092e+06  1.065e+07  -0.103    0.918
## radius3             2.226e+03  2.134e+04   0.104    0.917
## texture3            7.269e+01  3.150e+03   0.023    0.982
## perimeter3          1.267e+02  1.355e+03   0.093    0.926
## area3              -1.626e+01  1.165e+02  -0.140    0.889
## concavity3          6.737e+03  1.051e+05   0.064    0.949
## symmetry3           2.201e+04  3.283e+05   0.067    0.947
## fractal_dimension3  5.899e+04  1.032e+06   0.057    0.954
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7.5144e+02  on 568  degrees of freedom
## Residual deviance: 1.6713e-04  on 544  degrees of freedom
## AIC: 50
##
## Number of Fisher Scoring iterations: 25
```

With a quick inspection, we can see that the p-values of this model are high for all the variables used, so they are highly significant.

Backward

```r
suppressWarnings({
    Backward_Time <- system.time({
      Backward_Model <- train(
        Diagnosis ~ ., data = data,
        method = "glmStepAIC",
        trControl = Method,
        direction = "backward", # Set backward method
        trace = FALSE
      )
    })["elapsed"]
})
summary(Backward_Model$finalModel)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -5.914e+03  2.619e+05  -0.023    0.982
## radius1            -6.630e+03  1.150e+05  -0.058    0.954
## texture1            1.913e+02  1.345e+03   0.142    0.887
## area1               6.077e+01  1.079e+03   0.056    0.955
## smoothness1         3.914e+04  2.517e+05   0.155    0.876
## compactness1       -8.621e+04  9.326e+05  -0.092    0.926
## concavity1          2.852e+04  2.402e+05   0.119    0.905
## concave_points1     5.886e+04  1.544e+06   0.038    0.970
## symmetry1          -1.964e+04  1.347e+05  -0.146    0.884
## fractal_dimension1  1.626e+05  1.120e+06   0.145    0.885
## perimeter2         -1.253e+03  1.822e+04  -0.069    0.945
## area2               1.562e+02  2.259e+03   0.069    0.945
## smoothness2        -9.793e+04  1.472e+06  -0.067    0.947
## compactness2        9.217e+04  7.142e+05   0.129    0.897
## concavity2         -8.131e+04  1.097e+06  -0.074    0.941
## concave_points2     4.398e+05  6.736e+06   0.065    0.948
## symmetry2          -1.038e+05  2.160e+06  -0.048    0.962
## fractal_dimension2 -1.092e+06  1.065e+07  -0.103    0.918
## radius3             2.226e+03  2.134e+04   0.104    0.917
## texture3            7.269e+01  3.150e+03   0.023    0.982
## perimeter3          1.267e+02  1.355e+03   0.093    0.926
## area3              -1.626e+01  1.165e+02  -0.140    0.889
## concavity3          6.737e+03  1.051e+05   0.064    0.949
## symmetry3           2.201e+04  3.283e+05   0.067    0.947
## fractal_dimension3  5.899e+04  1.032e+06   0.057    0.954
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7.5144e+02  on 568  degrees of freedom
## Residual deviance: 1.6713e-04  on 544  degrees of freedom
## AIC: 50
##
## Number of Fisher Scoring iterations: 25
```

With a quick inspection, we can see that the p-values of this model are high for all the variables used, so they are highly significant.

Forward

```r
suppressWarnings({
    Forward_Time <- system.time({
      Forward_Model <- train(
        Diagnosis ~ ., data = data,
        method = "glmStepAIC",
        trControl = Method,
        direction = "forward", # Set forward method
        trace = FALSE
      )
    })["elapsed"]
})
summary(Forward_Model$finalModel)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -44.28703   11.47044  -3.861 0.000113 ***
## perimeter3         0.08387    0.07434   1.128 0.259268
## smoothness3       56.53029   30.08184   1.879 0.060215 .
## texture3           0.53014    0.13654   3.883 0.000103 ***
## radius2           -9.94253   16.81165  -0.591 0.554248
## symmetry3         16.97161    8.01255   2.118 0.034164 *
## compactness2    -130.42302   40.78433  -3.198 0.001384 **
## concavity1        39.87363   16.30611   2.445 0.014472 *
## texture2          -2.68316    1.51555  -1.770 0.076657 .
## area2              0.35067    0.20024   1.751 0.079892 .
## concave_points3   36.51134   19.42158   1.880 0.060117 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 751.440  on 568  degrees of freedom
## Residual deviance:  54.713  on 558  degrees of freedom
## AIC: 76.713
##
## Number of Fisher Scoring iterations: 11
```

With a quick inspection, we can see that the p-values of this model are high for all the variables used, so they are highly significant.

Lasso

```r
suppressWarnings({
    Lasso_Time <- system.time({
      Lasso_Model <- train(
        Diagnosis ~ ., data = data,
        method = "glmnet",
        trControl = Method,
        tuneGrid = expand.grid(alpha = 1,
                              lambda = seq(0.0001, 1, length = 50)) # alpha=1 means lasso
        #model and we fit for different lambdas
      )
    })["elapsed"]
})

best_lambda_lasso <- Lasso_Model$bestTune$lambda #Get the lambda of the best model

coef(Lasso_Model$finalModel, s = best_lambda_lasso) #Coef of the model with the best lambda
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##                            s1
## (Intercept)      -14.96636168
## radius1             .
## texture1            .
## perimeter1          .
## area1               .
## smoothness1         .
## compactness1         .
## concavity1           .
## concave_points1     13.57467042
## symmetry1            .
## fractal_dimension1   .
## radius2              0.78991905
## texture2             .
## perimeter2           .
## area2                .
## smoothness2          .
## compactness2         .
## concavity2           .
## concave_points2      .
## symmetry2            .
## fractal_dimension2   .
## radius3              0.43324388
## texture3             0.11076710
## perimeter3           .
## area3                .
## smoothness3          6.03826173
## compactness3         .
## concavity3           0.00699303
## concave_points3     16.78847079
## symmetry3            2.25946856
## fractal_dimension3   .
```

With the Lasso penalty, using the best lambda, we can see how too many variables were discarded; now we need to compare its performance and see if this much simpler model is at least as good as the others.

Ridge

```r
suppressWarnings({
    Ridge_Time <- system.time({
      Ridge_Model <- train(
        Diagnosis ~ ., data = data,
        method = "glmnet",
        trControl = Method,
        tuneGrid = expand.grid(alpha = 0, lambda = seq(0.0001, 1, length = 50)) # alpha=0
        #means Ridge regression and we fit for different lambdas
      )
    })["elapsed"]
})

best_lambda_ridge <- Ridge_Model$bestTune$lambda #Best Ridge lambda

coef(Ridge_Model$finalModel, s = best_lambda_ridge) #Coef of the model with the best lambda
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##                               s1
## (Intercept)        -1.389314e+01
## radius1             8.081935e-02
## texture1             6.184018e-02
## perimeter1           1.152165e-02
## area1                7.557894e-04
## smoothness1          8.107265e+00
## compactness1         1.603673e+00
## concavity1           2.801163e+00
## concave_points1      7.519044e+00
## symmetry1            2.671773e+00
## fractal_dimension1 -2.010059e+01
## radius2              8.894382e-01
## texture2            -2.790517e-02
## perimeter2           9.921854e-02
## area2                4.526886e-03
## smoothness2          3.463722e-01
## compactness2        -4.442906e+00
## concavity2          -1.176142e+00
## concave_points2      1.340849e+01
## symmetry2           -8.259528e+00
## fractal_dimension2 -4.889345e+01
## radius3              6.990693e-02
## texture3             5.460557e-02
## perimeter3           9.566350e-03
## area3                5.233578e-04
## smoothness3          1.085056e+01
## compactness3         9.892990e-01
## concavity3           1.161959e+00
## concave_points3      5.067763e+00
## symmetry3            3.876396e+00
## fractal_dimension3  5.047323e+00
```

Regression Rigge uses many variables, resulting in a more complex model; however, many of the coefficients show very small values. We will have to see if this complexity and the minimal contribution of some variables help its performance.

Random Forest

```
suppressWarnings({
    Random_Forest_Time <- system.time({
      RF_model <- train(
        Diagnosis ~ ., data = data,
        method = "rf", # Train a random forest model
        trControl = Method
      )
    })["elapsed"]
})

varImp(RF_model) # Importance of the 20 most important variables
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 30)
```

```
##
##                  Overall
## area3            100.000
## radius3           95.167
## concave_points3   94.321
## perimeter3        80.770
## concavity1        70.969
## concave_points1   70.938
## radius1           68.643
## area1             53.316
## concavity3        51.989
## perimeter1        49.280
## area2             43.858
## perimeter2        27.614
## compactness3      27.437
## compactness1      25.748
## texture3          20.682
## radius2           20.054
## texture1          14.380
## smoothness3       11.988
## symmetry3          9.938
## concavity2         9.432
```

Table of coefficient comparison

```
#Get the names of all the variables
vars <- names(coef(Base_model$finalModel))

# Data frame with the first column the
#name of all the variables
df_coefs <- data.frame(
  variable = vars,
  stringsAsFactors = FALSE
)

#Function to extract the falue of the coef,
#mo te sure that all the columns have the
# same variables in the same order
extract_coefs <- function(m, vars) {

  co <- m

  # Check if it is a sparse matrix
  #(Lasso and Ridge)
  if ("dgCMatrix" %in% class(co)) {
    #If so, extract the first column
    #that contains the names.
    values <- as.numeric(co[,1])
    names(values) <- rownames(co)
  } else {
    #If not, extract the names of the
    #coefficients.
    values <- as.numeric(co)
    names(values) <- names(co)
  }

  return(values[vars])
}

#Extract the coefficient values for each model
df_coefs$Full_Model    <- extract_coefs(coef(Base_model$finalModel), vars)

df_coefs$Stepwise <- extract_coefs(coef(Stepwise_Model$finalModel), vars)

df_coefs$Backward <- extract_coefs(coef(Backward_Model$finalModel), vars)

df_coefs$Forward  <- extract_coefs(coef(Forward_Model$finalModel), vars)

df_coefs$Lasso <- extract_coefs(
  coef(Lasso_Model$finalModel, s = Lasso_Model$bestTune$lambda),
  vars
)

df_coefs$Ridge <- extract_coefs(
  coef(Ridge_Model$finalModel, s = Ridge_Model$bestTune$lambda),
  vars
)

#Round coefficients to 2 decimal places
df_coefs[ , -1] <- round(df_coefs[ , -1], 2)

#Replace NA and 0 with -
df_coefs[is.na(df_coefs)|df_coefs==0] <- "-"

df_coefs
```

```
##              variable    Full_Model    Stepwise    Backward Forward  Lasso
## 1         (Intercept)    -2881304.4    -5914.46    -5914.46  -44.29 -14.97
## 2             radius1     2427013.0    -6629.52    -6629.52       -      -
## 3            texture1      195783.2      191.26      191.26       -      -
## 4          perimeter1     1473188.4           -           -       -      -
## 5               area1     -130118.2       60.77       60.77       -      -
## 6         smoothness1  -152452270.8     39142.9     39142.9       -      -
## 7        compactness1    -6428388.9   -86211.47   -86211.47       -      -
## 8           concavity1     1041553.7    28520.95    28520.95   39.87      -
## 9     concave_points1   -17156866.7    58858.72    58858.72       -  13.57
## 10          symmetry1    40485942.1   -19644.82   -19644.82       -      -
## 11 fractal_dimension1   -42329195.5   162556.55   162556.55       -      -
## 12            radius2    33284830.2           -           -   -9.94   0.79
## 13           texture2     6368395.0           -           -   -2.68      -
## 14         perimeter2     1700716.8    -1253.11    -1253.11       -      -
## 15              area2     -639346.7      156.21      156.21    0.35      -
## 16        smoothness2   749172456.3   -97931.96   -97931.96       -      -
## 17       compactness2  -177307723.7    92173.51    92173.51 -130.42      -
## 18          concavity2   152864835.6   -81310.72   -81310.72       -      -
## 19     concave_points2 -1259854447.6   439795.13   439795.13       -      -
## 20           symmetry2   289010997.2  -103758.08  -103758.08       -      -
## 21 fractal_dimension2  1512104102.9 -1092014.08 -1092014.08       -      -
## 22             radius3    -6130234.0     2226.39     2226.39       -   0.43
## 23            texture3     -583246.0       72.69       72.69    0.53   0.11
## 24          perimeter3     -353820.5      126.66      126.66    0.08      -
## 25               area3       89504.5      -16.26      -16.26       -      -
## 26        smoothness3   -21611286.5           -           -   56.53   6.04
## 27       compactness3     8986340.5           -           -       -      -
## 28          concavity3   -30279288.9     6736.58     6736.58       -   0.01
## 29     concave_points3   143130487.1           -           -   36.51  16.79
## 30           symmetry3   -24735907.8    22008.73    22008.73   16.97   2.26
## 31 fractal_dimension3   -36983257.4    58988.94    58988.94       -      -
##     Ridge
## 1  -13.89
## 2    0.08
## 3    0.06
## 4    0.01
## 5       -
## 6    8.11
## 7     1.6
## 8     2.8
## 9    7.52
## 10   2.67
## 11  -20.1
## 12   0.89
## 13  -0.03
## 14    0.1
## 15      -
## 16   0.35
## 17  -4.44
## 18  -1.18
## 19  13.41
## 20  -8.26
## 21 -48.89
## 22   0.07
## 23   0.05
## 24   0.01
## 25      -
## 26  10.85
## 27   0.99
## 28   1.16
## 29   5.07
## 30   3.88
## 31   5.05
```

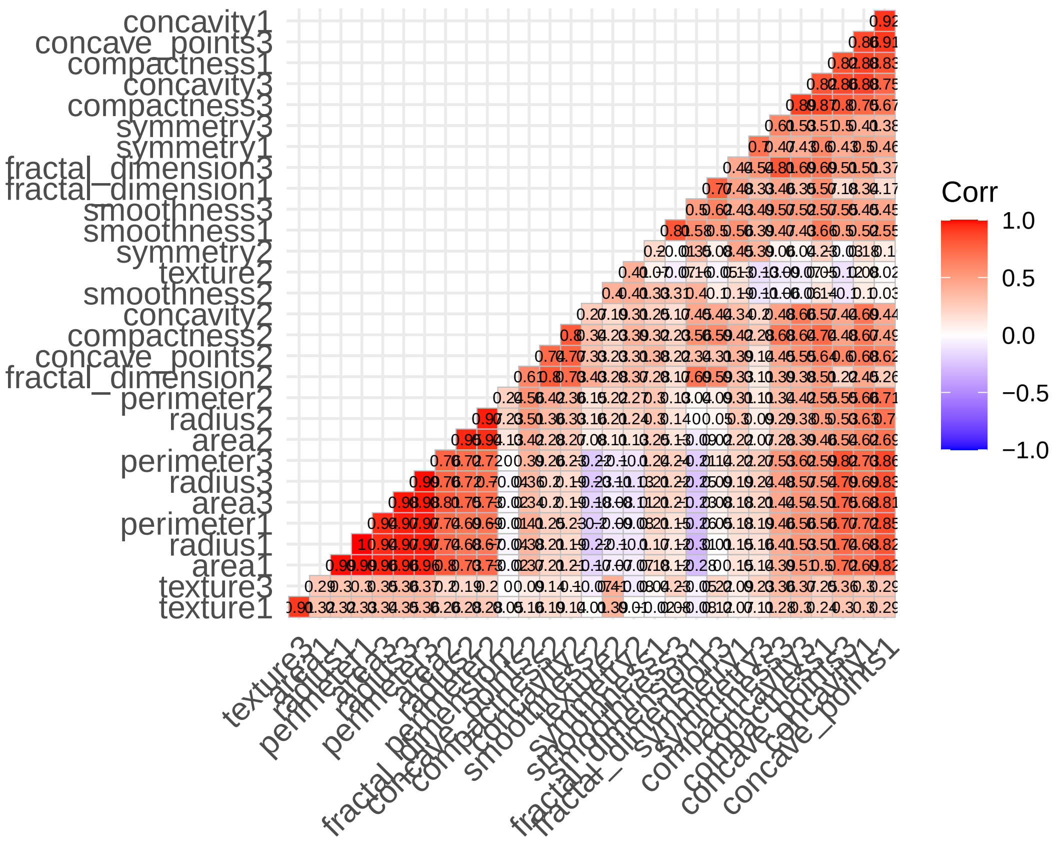
We can see that the Lasso and Forward models are the simplest, with much smaller coefficient values. The Ridge model is slightly more complex, but its coefficient values are also very small. The Backward, Stepwise, and especially the Full model are more complex, and their coefficients are much larger.

Correlation analysis of variables

A heat map showing the correlation of the explanatory variables will be displayed below.

```r
M <- cor(data[sapply(data, is.numeric)])

ggcorrplot(M, hc.order = TRUE, type = "lower",
           lab = TRUE, lab_size = 2)
```

```
## Warning: ‘aes_string()‘ was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with ‘aes()‘.
## i See also ‘vignette("ggplot2-in-packages")‘ for more information.
## i The deprecated feature was likely used in the ggcorrplot package.
##   Please report the issue at <https://github.com/kassambara/ggcorrplot/issues>.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



There are too many variables, but we can easily notice that most of the squares are painted red, which indicates a strong correlation between the different variables in our data set.

This highly correlated data creates a multicollinearity problem in the logistic model that uses all the variables. This makes the model unstable and requires large coefficients to balance the repeated effects explained by other variables.

It’s surprising that the base model has such high accuracy given its clear collinearity issues. But how does Lasso achieve slightly better precision without using so many variables and with much smaller coefficients?

First, let's make a list of the variables with the absolute value lowest average correlation.

```r
numeric_data <- data[, sapply(data, is.numeric)]

#Correlation matrix
M <- cor(numeric_data, use = "pairwise.complete.obs")

diag(M) <- NA

#We take the average of the correlations
avg_corr <- apply(abs(M), 1, mean, na.rm = TRUE)

# We ordered from highest to lowest correlation
avg_corr_sorted <- sort(avg_corr, decreasing = TRUE)

most_correlated <- names(avg_corr_sorted)[tail(order(avg_corr_sorted), 10)]

most_correlated
```

```
##  [1] "area3"           "compactness3"    "perimeter1"      "radius3"
##  [5] "perimeter3"      "concavity3"      "concave_points3" "compactness1"
##  [9] "concave_points1" "concavity1"
```

We can see that 4 of the 10 most correlated variables are used by Lasso: concave_points1, radius3, concavity3 and concave_points3. Let's see if we can extract those with the lowest absolute value average correlation.

```r
most_correlated <- names(avg_corr_sorted)[20:30]

most_correlated
```

```
##  [1] "symmetry1"          "fractal_dimension1" "fractal_dimension3"
##  [4] "smoothness3"        "symmetry3"          "fractal_dimension2"
##  [7] "texture3"           "texture1"           "smoothness2"
## [10] "symmetry2"          "texture2"
```

We can see that Lasso uses 3 of the variables with the least average correlation in absolute value: texture3, smoothness3 and symmetry3.

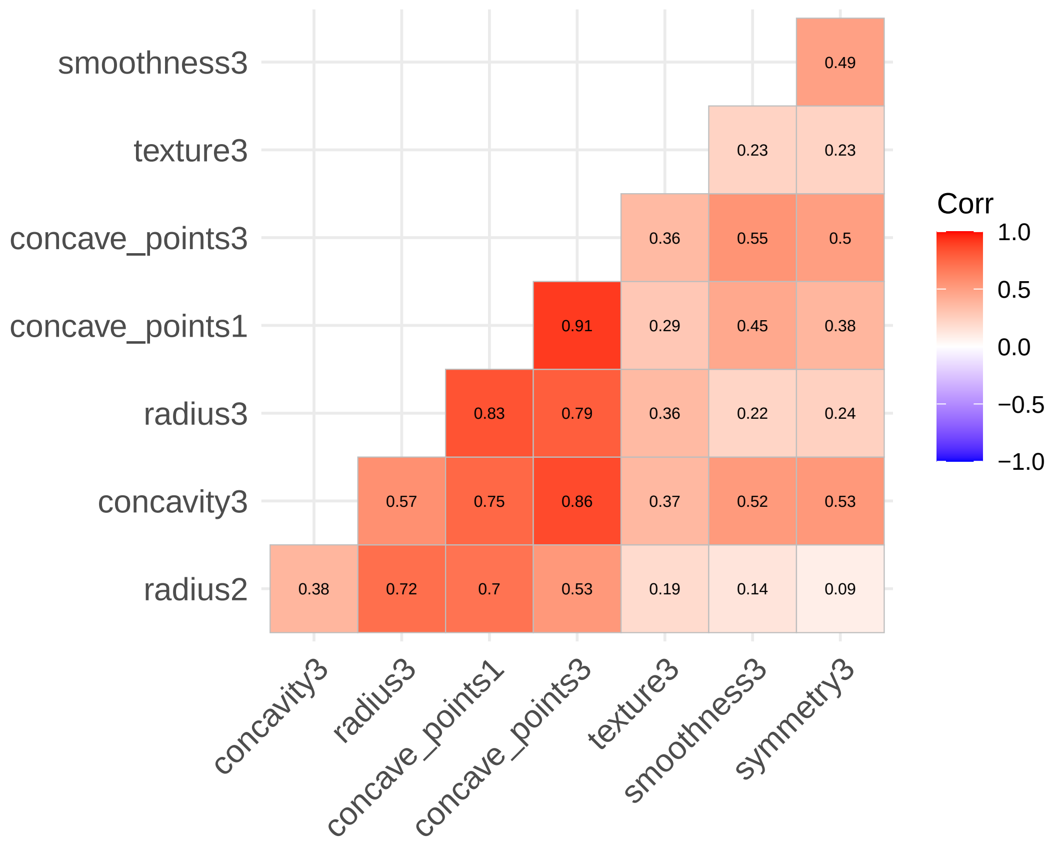
And finally there is the variable radius 2, which remains in the variables with intermediate average absolute correlation, specifically it is the 14th ordered from highest to lowest.

We can see that Lasso uses a combination of highly correlated variables, with poorly correlated variables and an intermediate variable.

The heat map of the variables used by Lasso is shown below.

```r
vars <- c(
  "concave_points1", "radius2", "radius3", "texture3",
  "smoothness3", "concavity3", "concave_points3", "symmetry3"
)

data_sub <- data[, vars]

M <- cor(data_sub[sapply(data_sub, is.numeric)])
```

```r
ggcorrplot(M, hc.order = TRUE, type = "lower",
           lab = TRUE, lab_size = 2)
```



We can see that variables with low correlation are in more orange or white tones, while variables with higher correlation tend to be more red.

This choice of variables is not accidental; in fact, it's quite logical. The explanation I can offer is that Lasso, being penalized, uses highly correlated variables to explain the greater amount of information contained in many variables, synthesizing it into a few. Meanwhile, he uses some less correlated variables to explain more specific information not found in other variables.

Accuracy comparison

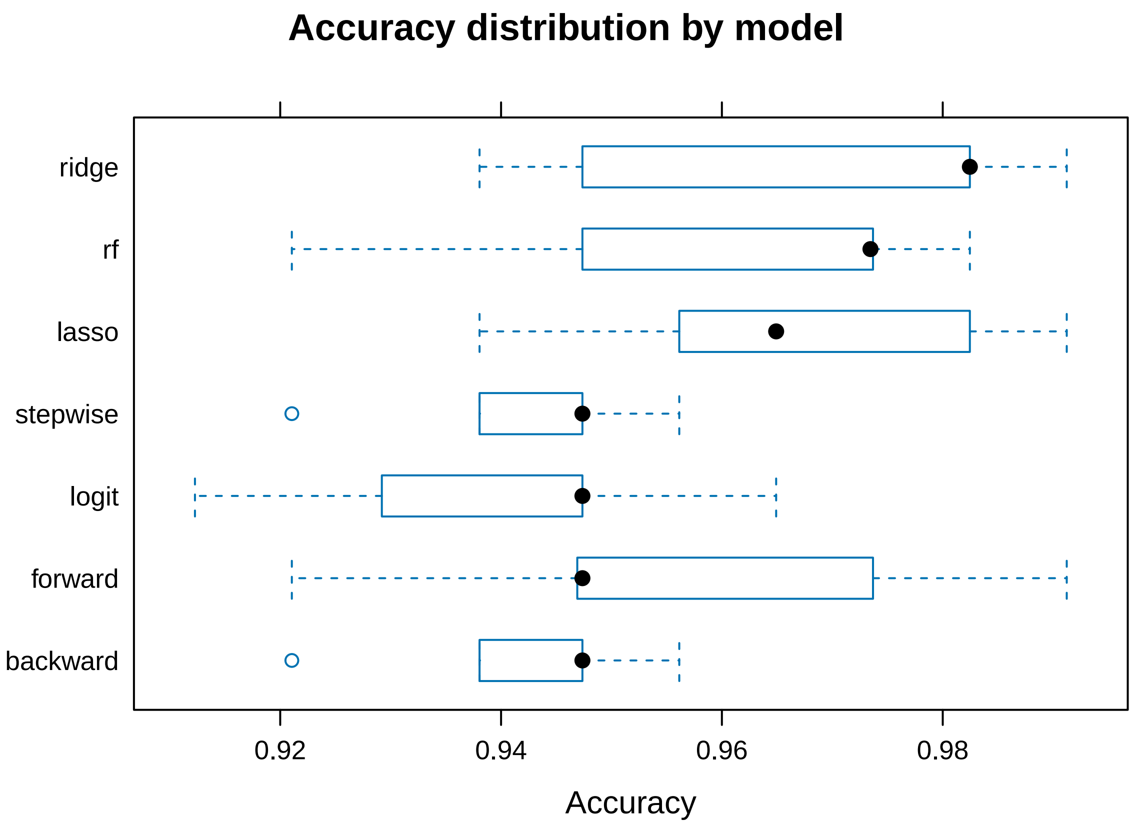
```r
#Statistical summary of the 5-fold CV
#for each model

results <- resamples(list(
  logit = Base_model,
  stepwise = Stepwise_Model,
  backward = Backward_Model,
  forward = Forward_Model,
  lasso = Lasso_Model,
  ridge = Ridge_Model,
  rf = RF_model
))

summary(results)$statistics$Accuracy
```

```
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## logit    0.9122807 0.9292035 0.9473684 0.9402267 0.9473684 0.9649123    0
## stepwise 0.9210526 0.9380531 0.9473684 0.9419966 0.9473684 0.9561404    0
## backward 0.9210526 0.9380531 0.9473684 0.9419966 0.9473684 0.9561404    0
## forward  0.9210526 0.9469027 0.9473684 0.9560472 0.9736842 0.9912281    0
## lasso    0.9380531 0.9561404 0.9649123 0.9665580 0.9824561 0.9912281    0
## ridge    0.9380531 0.9473684 0.9824561 0.9683124 0.9824561 0.9912281    0
## rf       0.9210526 0.9473684 0.9734513 0.9596025 0.9736842 0.9824561    0
```

Accuracy is the proportion of correct predictions out of the total number of observations.

$$Accurracy = \frac{\text{Correct predictions}}{\text{Total predictions}}$$

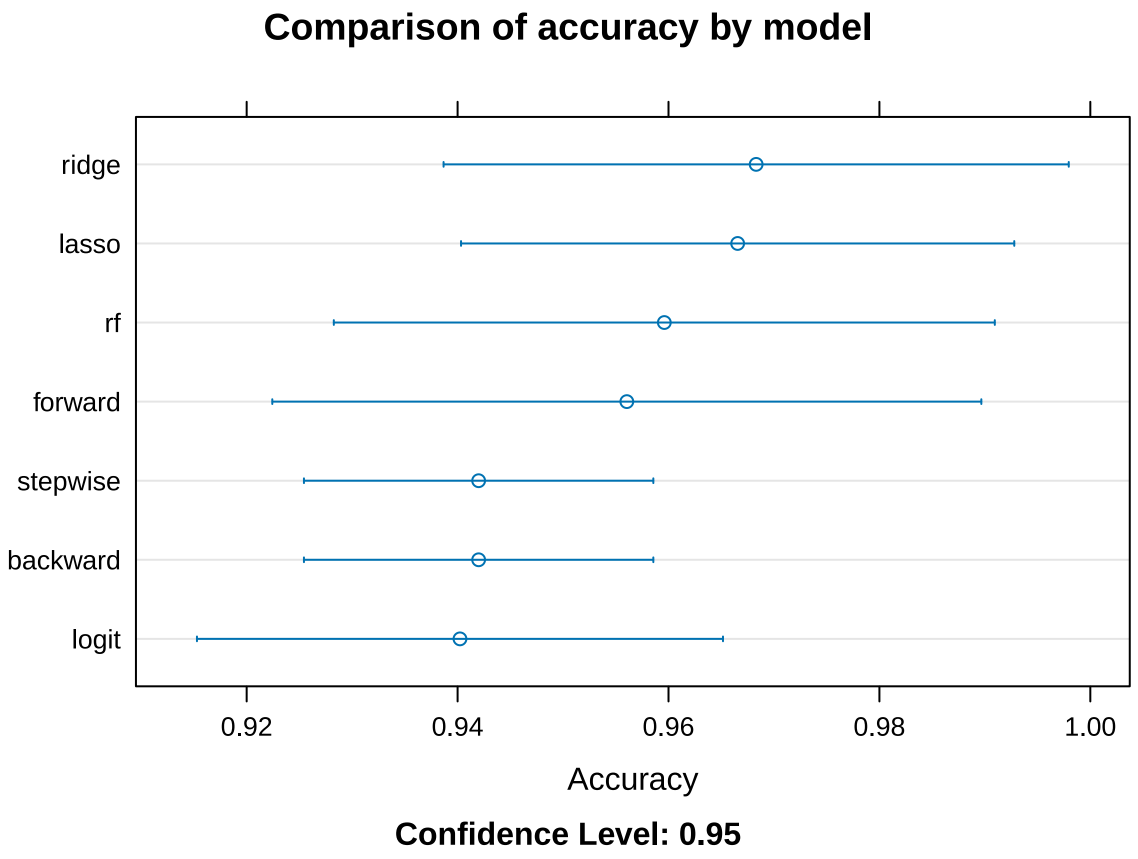
```
bwplot(results, metric = "Accuracy", main = "Accuracy distribution by model")
```



This chart shows a statistical summary of the accuracy of the 5 folds used.

The box shows the first and third quartiles; the black dot is the median. Points outside the whiskers are outliers.

```r
dotplot(results, metric = "Accuracy", main = "Comparison of accuracy by model")
```



In this graph, the dot shows the average accuracy across the 5 folds for each model. The line around it shows the dispersion in accuracy. Shorter lines indicate a more consistent metric, while longer lines indicate less consistency and greater variability.

Fitting Time

```r
#Combine the training times into one data frame
Times_DF <- data.frame(
  #Models column
  Model = c("Logistic", "Stepwise", "Backward", "Forward", "Lasso", "Ridge", "RandomForest"),
  #Column of Times
  Fitting_Time = c(Base_Time, Stepwise_Time, Backward_Time, Forward_Time,
                   Lasso_Time, Ridge_Time, Random_Forest_Time)
)

#Order from smallest to largest
Times_DF <- Times_DF[order(Times_DF$Fitting_Time), ]

Times_DF
```

```
##            Model Fitting_Time
## 6        Ridge         0.34
## 1     Logistic         0.45
## 5        Lasso         0.80
## 7 RandomForest         3.23
## 4      Forward         4.89
## 3     Backward        21.50
## 2     Stepwise        29.98
```

This table shows the time it takes to train each model.

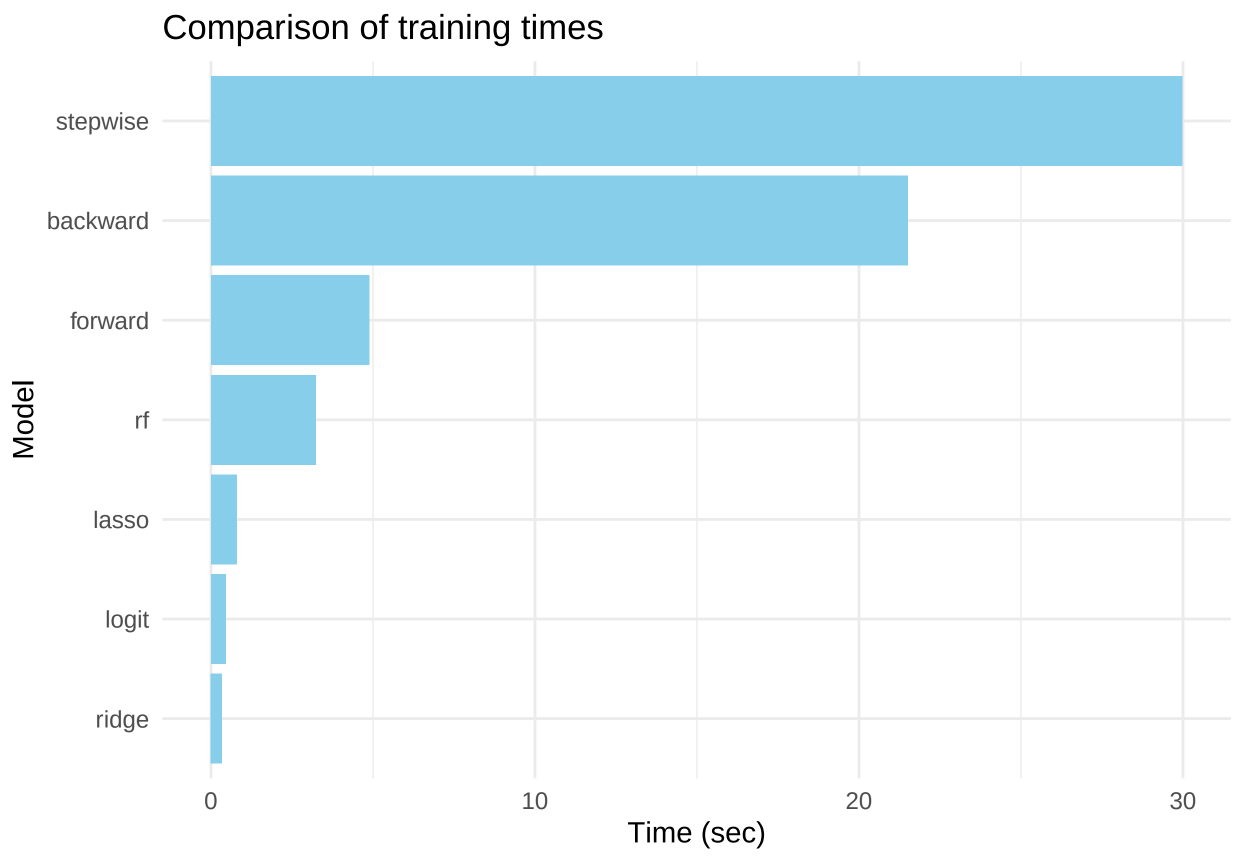
Comparison

```
#We extract the average of Accuracy
Metrics<- summary(results)
metrics_summary <- Metrics$statistics

accuracy_mean <- metrics_summary$Accuracy[,'Mean']

#We create a vector with the times
tiempos <- c(Base_Time, Stepwise_Time, Backward_Time, Forward_Time, Lasso_Time, Ridge_Time,
             Random_Forest_Time)

#Date Frame with the values to compare the models
Comparison <- data.frame(
  Model = names(accuracy_mean),
  Accuracy_Mean = unname(accuracy_mean),
  Time = tiempos
)

Comparison
```

```
##      Model Accuracy_Mean  Time
## 1    logit     0.9402267  0.45
## 2 stepwise     0.9419966 29.98
## 3 backward     0.9419966 21.50
## 4  forward     0.9560472  4.89
## 5    lasso     0.9665580  0.80
## 6    ridge     0.9683124  0.34
## 7       rf     0.9596025  3.23
```

Time Graph

```
ggplot(Comparison,
       #Order the bars from shortest to longest time
       aes(x = reorder(Model, Time), y = Time)) +

  #draw bars according to the time values
  geom_bar(stat = "identity", fill = "skyblue") +
  coord_flip() + #makes the bars horizontal
    #Labels
  labs(title = "Comparison of training times",
       x = "Model",
       y = "Time (sec)") +
  theme_minimal()
```



Note that the most accurate models are Ridge, followed by Lasso, and then Random Forest. Ridge's training time is slightly faster than Lasso, but Random Forest takes almost six times longer to be adjusted. Recall that in terms of complexity, the Lasso model was the simplest, but Ridge, despite being more complex, wasn't as complex as the other models.

We can conclude that the Lasso model is the best option, as it offers simplicity, good accuracy, and a short adjustment time. If we are willing to sacrifice simplicity for a shorter training time and greater accuracy, Ridge is the best choice. Random Forest, being a machine learning model and therefore much less interpretable, has similar accuracy to the other models, which, while not bad, are inferior to Lasso and Ridge. Even so, it is faster and more accurate than the other models. The other models are not worthwhile in this situation; they have a significantly longer training time, their accuracy is worse, and they are much more complex.

Let's repeat this analysis 50 times

Due to the time it takes for them to adjust, we will not be using the Backward and Stepwise models. These models did not demonstrate outstanding performance, so I decided to forgot them at this stage.

To obtain more conclusive data, we will repeat this same experiment 50 times, with different folds, to obtain a more precise sample of accuracy and not attribute it to chance.

Creation of the folds for the different iterations

```r
n_repeats <- 50
k <- 5

set.seed(996)

#Vector that stores the folds for each iteration.
folds_list <- vector("list", n_repeats)

for (i in 1:n_repeats) {
  folds_list[[i]] <- createFolds(y = data$Diagnosis, k = k, returnTrain = TRUE)
}
```

Training and storage of results

```r
results_list <- list()

for (i in 1:n_repeats) {
  # I establish the folds that correspond
  #to each iteration.
  Method <- trainControl(
    method = "cv",
    number = k,
    index = folds_list[[i]],
    summaryFunction = twoClassSummary,
    classProbs = TRUE,
    savePredictions = "final"
  )

  suppressWarnings({

  Base_Time <- system.time({
    Base_model <- train(
      Diagnosis ~ ., data = data,
      method = "glm",
      family = binomial,
      trControl = Method
    )
  })["elapsed"]


  Forward_Time <- system.time({
      Forward_Model <- train(
        Diagnosis ~ ., data = data,
        method = "glmStepAIC",
        trControl = Method,
        direction = "forward",
        trace = FALSE
      )
    })["elapsed"]


  Lasso_Time <- system.time({
      Lasso_Model <- train(
        Diagnosis ~ ., data = data,
        method = "glmnet",
        trControl = Method,
        tuneGrid = expand.grid(alpha = 1,
                               lambda = seq(0.0001, 1, length = 50))
      )
    })["elapsed"]

  Lasso_Model$pred <- Lasso_Model$pred[
    Lasso_Model$pred$lambda == Lasso_Model$bestTune$lambda, ]


  Ridge_Time <- system.time({
      Ridge_Model <- train(
        Diagnosis ~ ., data = data,
        method = "glmnet",
        trControl = Method,
        tuneGrid = expand.grid(alpha = 0, lambda = seq(0.0001, 1, length = 50))
      )
    })["elapsed"]

  Ridge_Model$pred <- Ridge_Model$pred[
    Ridge_Model$pred$lambda == Ridge_Model$bestTune$lambda, ]


  Random_Forest_Time <- system.time({
      RF_model <- train(
        Diagnosis ~ ., data = data,
        method = "rf",
        trControl = Method
      )
    })["elapsed"]

})

  results_list[[i]] <- list(Logit = Base_model, Forward=Forward_Model,
                            LASSO=Lasso_Model, Ridge=Ridge_Model, RF = RF_model)

}
```

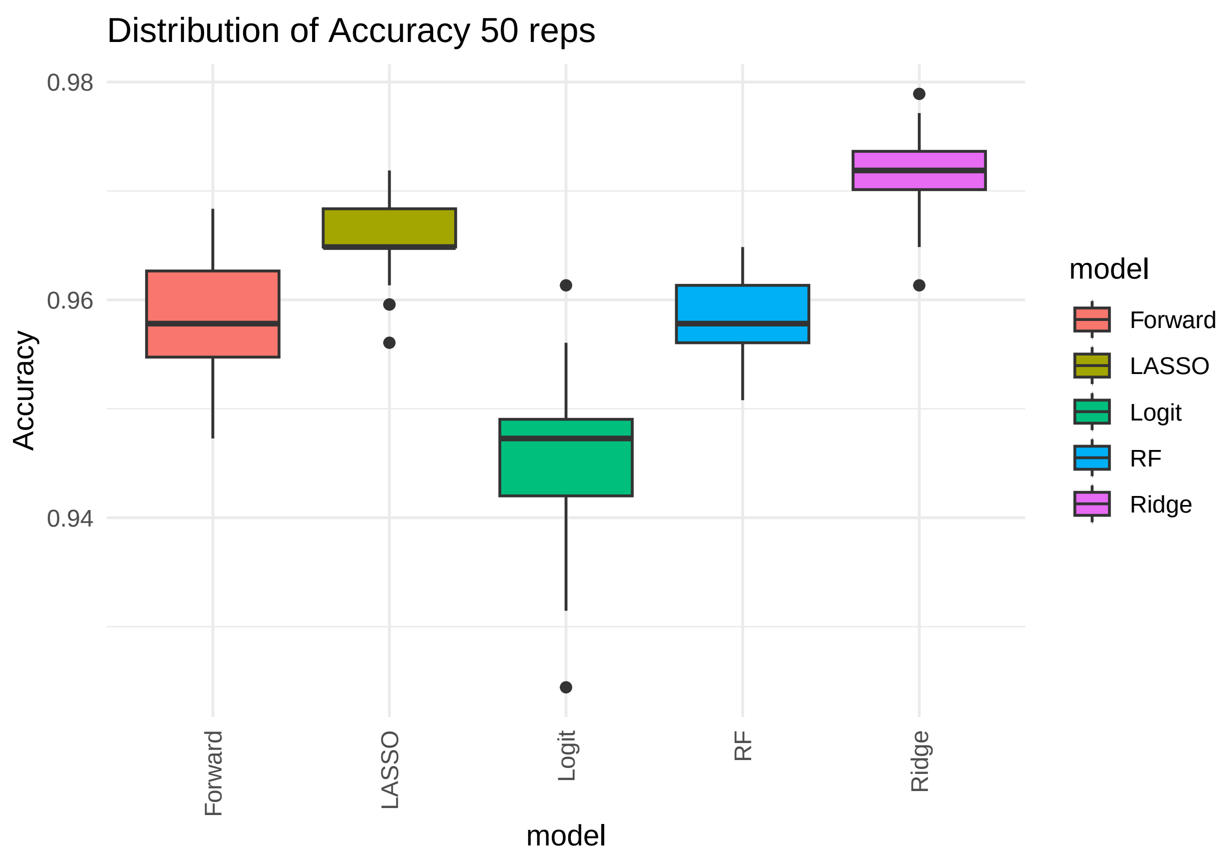
Summary the results

```
all_results <- data.frame()
```

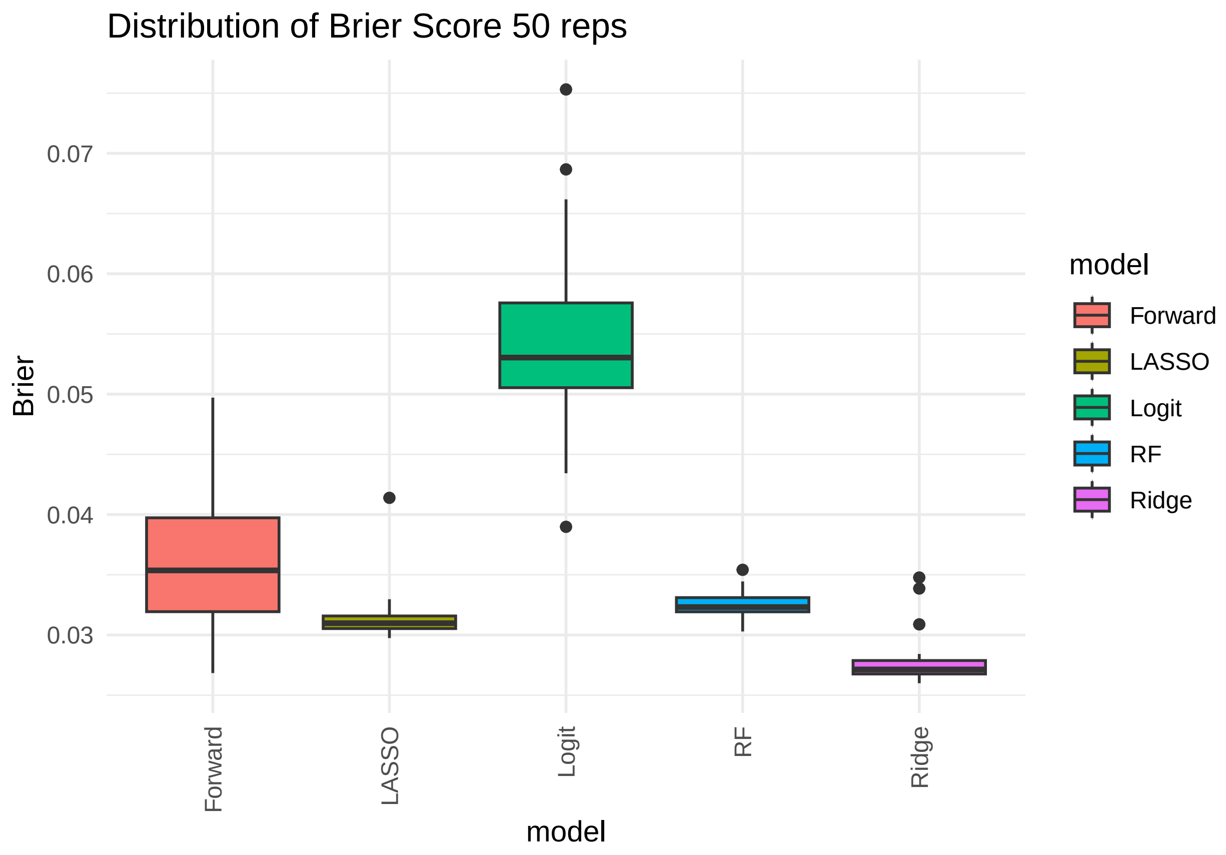
```r
for (i in 1:n_repeats) {
  for (model_name in names(results_list[[i]])) {

    pred <- results_list[[i]][[model_name]]$pred

    # Accuracy average
    acc <- mean(pred$pred == pred$obs)

    # Brier Score
    probs <- pred$M
    obs <- ifelse(pred$obs == "M", 1, 0)
    brier <- mean((probs - obs)^2)

    all_results <- rbind(all_results,
                         data.frame(
                           iteration = i,
                           model = model_name,
                           Accuracy = acc,
                           Brier = brier
                         ))
  }
}
```

Graphs

```r
ggplot(all_results, aes(x = model, y = Accuracy, fill = model)) +
  geom_boxplot() +
  theme_minimal() +
  ggtitle("Distribution of Accuracy 50 reps")+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```



```r
# Boxplot Brier
ggplot(all_results, aes(x = model, y = Brier, fill = model)) +
  geom_boxplot() +
  theme_minimal() +
  ggtitle("Distribution of Brier Score 50 reps")+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```



Accuracy is the proportion of correct predictions out of the total number of observations.

$$Accurracy = \frac{\text{Correct predictions}}{\text{Total predictions}}$$

Brier Score is the mean squared error of the forecast. It tells us how confidently the model predicts the response variable in each observation.

$$\text{Brier Score} = \frac{1}{N}\sum_{i=1}^{N}(p_i = y_i)^2$$

Where:

p_i: Predicted probability of the positive class.

y_i: Actual value encoded as 0 or 1.

Final Reflection

Looking at the box plot for the 50 repetitions, we can conclude that Ridge remains the best-performing model, followed by LASSO. Random Forest is still a good candidate but is on par with Forward, falling behind these two.

Using the Brier Score, we can conclude that the models predict with high accuracy, particulary Ridge and Lasso, in the same order of performance. Ridge and Lasso have smaller boxes, suggesting that their prediction accuracy is more consistent than the others.

Our final conclusion is the same as with the single-repetition model. The most accurate and reliable predictor is Ridge, but if we don't mind sacrificing a little precision for a much simpler model, LASSO is undoubtedly the clear winner. Although RF is also a good candidate, it still lags behind the two mentioned above and completely lacks interpretability.

In my opinion, the best model is LASSO; it is simple, accurate, reliable, and interpretable, not to mention that its penalty is much easier to understand.