Homework 3

Kathleen Ashbaker

2024-02-16

In the following problems, you will work on a real dataset. Similar to HW2, you will conduct binary classification on the Breast Cancer Wisconsin (Diagnostic) Data Set in the csv file wdbc.data. The dataset describes characteristics of the cell nuclei in n (sample size) digitized images. Each image has multiple attributes, which are described in detail in wdbc.names. This time, however, you will predict the attribute in column 2, which we denote by Y , given the columns {3, 4, . . . , 32}, which we denote by X1, . . . , X30. The variable Y represents the diagnosis (M = malignant, B = benign).

```
# upload necessary libraries
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
library(MASS)  # Implements LDA & QDA
library(caret) # confusionMatrix
```

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

```
library(pROC)  # ROC & AUC
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(class) # Implements KNN
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 4.3.2

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.3.2

## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```r
library(stats)
library(ggcorrplot)
```

```
## Warning: package 'ggcorrplot' was built under R version 4.3.2
```

```r
library(dplyr)
```

```r
library(readr)
wdbc <- read_csv("wdbc.csv", col_names = TRUE)
```

```
## Rows: 569 Columns: 32
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): Diagnosis
## dbl (31): ID Number, Average radius, Average texture, Average perimeter, Ave...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

1. Data exploration + Simple Logistic Regression

(a) Describe the data: sample size n, number of predictors p, and number of observations in each class.

```r
View(wdbc) # check data frame 'wdbc'

sum(is.na(wdbc)) # check for missing values
```

```
## [1] 0
```

```r
wdbc <- wdbc %>%
  mutate(Diagnosis = recode(Diagnosis, 'B' = 0, 'M' = 1))
```

```r
# Separate Predictors and Outcome
predictors <- wdbc[, 3:32]
outcome <- wdbc[, 2]

# Convert the 'Diagnosis' column to a factor
outcome_factor <- as.factor(wdbc$Diagnosis)
```

```r
# sample size, n is:
nrow(wdbc)
```

```
## [1] 569
```

```r
# number of attributes, or predictors, p is:
ncol(predictors)
```

```
## [1] 30
```

```r
# Count of each class
table(wdbc$Diagnosis)
```

```
##
##   0   1
## 357 212
```

(b) Divide the data into a training set of 400 observations, and a test set; Set the seed with set.seed(2) before you sample the training set.

```r
# Set seed for reproducibility;
# meaning you'll get the same split every time you run the code
set.seed(2)

# Sample 400 unique row indices for the training set
train_id <- sample(nrow(wdbc), 400)

# Create the training set with the selected indices
training_set <- wdbc[train_id, ]

# Create the test set with the remaining indices
test_set <- wdbc[-train_id, ]
```

(c) Normalize your predictors, i.e. for each variable Xj remove the mean and make each variable's standard deviation 1. You should perform this step separately in the training set and test set. Why?

```r
# Assuming predictors have already been selected and 'train_id' has been defined
# Normalize the predictors in the training set; mean=0 sd=1
training_set_norm <- as.data.frame(scale(training_set[, 3:32]))

# Calculate mean and sd from the training set to apply to the test set
means <- apply(training_set[, 3:32], 2, mean)
sds <- apply(training_set[, 3:32], 2, sd)

# Normalize the predictors in the test set using training set parameters
test_set_norm <- as.data.frame(sweep(sweep(test_set[, 3:32], 2, means, "-"), 2, sds, "/"))
```

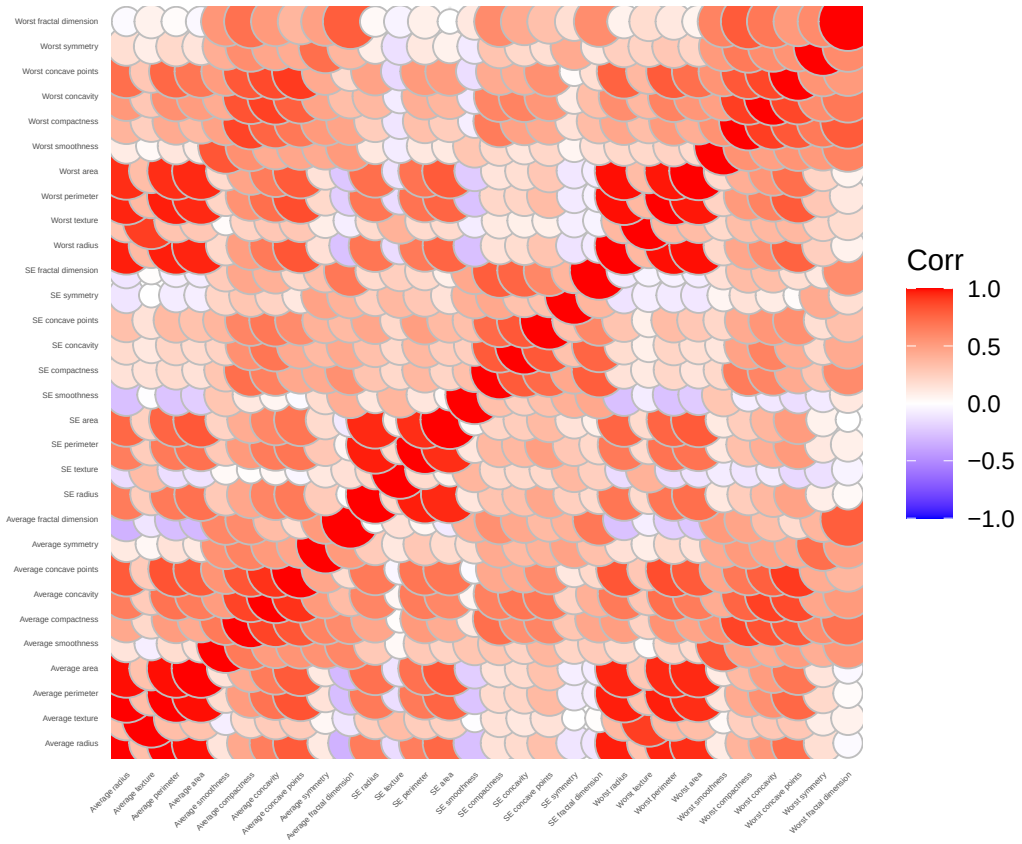
Comment:

To normalize the predictors in both the training and test sets, you need to subtract the mean of each predictor and divide by its standard deviation. This process, often called feature scaling or standardization, ensures that each predictor has a mean of 0 and a standard deviation of 1. It's crucial to perform this normalization separately for the training and test sets to avoid data leakage, where information from the test set inadvertently influences the training process. By computing the mean and standard deviation on the training set and applying those values to both the training and test sets, you ensure that the model is evaluated on data that is scaled in the same way as the training data but without using any information from the test data itself.

(d) Compute the correlation matrix of your training predictors (command cor) and plot it (e.g. command ggcorrplot in the library ggcorrplot). Inspect the correlation matrix and explain what type of challenges this dataset may present?

```r
# Compute the correlation matrix for the normalized training predictors
corr_matrix <- cor(training_set_norm)
```

Plot of Correlational Matrix:



The correlation matrix visualized in the plot provided displays the pairwise correlation coefficients between various variables in a dataset. The correlation coefficient values range from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation.

From the plot, it is evident that there are several challenges this dataset may present:

High Correlation: Many of the cells are colored in dark red, indicating a high positive correlation between numerous pairs of variables. This suggests that the dataset suffers from multicollinearity, where two or more independent variables are highly correlated. This can make the coefficients of a regression model unreliable and destabilize the model, as it becomes hard to disentangle the effect of one predictor variable from another.

Feature Redundancy: High correlation between variables might also imply redundancy. Redundant features do not contribute additional information for a model, and therefore, they do not improve the model's performance. They could potentially be removed without a significant loss of information.

Model Interpretation and Complexity: With so many variables showing strong correlations, interpreting the effects of individual variables on the response variable becomes complex. It may also result in an unnecessarily complex model which can affect the model's performance on unseen data (overfitting).

Regularization and Feature Selection: When using regularized regression models like Lasso or Ridge, high correlations can affect the penalty term's effectiveness in selecting features or shrinking coefficients. In the case of Lasso, it might arbitrarily select one feature from a group of highly correlated features while ignoring others.

Predictive Models Performance: Predictive models might perform poorly due to the multicollinearity issue. Models that rely on the independence of the features, such as linear regression, may yield inflated standard errors for the coefficient estimates.

To address these challenges, consider the following:

Variable Selection: Use feature selection techniques to remove highly correlated variables. Principal Component Analysis (PCA): PCA can be used for dimensionality reduction to create a set of uncorrelated variables. Regularization: Use regularization methods that can handle multicollinearity, like Lasso or Ridge regression, which include penalties on the size of the coefficients to prevent overfitting. Domain Knowledge: Consult with subject matter experts to understand which variables are essential and which could be combined or removed based on domain knowledge. Handling these issues properly can lead to simpler, more interpretable, and more generalizable models.

(e) Fit a (simple) logistic regression model to predict Y given X1, . . . , X30. Inspect and report the correlation between the variables X1 and X3; and the magnitude of their coefficient estimates B^1, B^3 wrt to the other coefficients of the model. Comment on their values and relate this to what we have seen in class.

```
# set aside training_set_norm data frame for simple
# logsitic regression.
training_set_norm_reg <- cbind(Diagnosis = training_set$Diagnosis,
    training_set_norm)


# Fit the logistic regression model using all predictors??
# or just X1 and X3

logistic_model_full <- glm(Diagnosis ~ ., family = binomial(link = "logit"),
    data = training_set_norm_reg)
```

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

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

```
# Assuming logistic_model_full contains both predictors
summary_logistic <- summary(logistic_model_full)

# display model summary
summary_logistic
```

```
##
## Call:
## glm(formula = Diagnosis ~ ., family = binomial(link = "logit"),
##     data = training_set_norm_reg)
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                    92.461  28281.573   0.003    0.997
## `Average radius`             -862.997 462270.497  -0.002    0.999
## `Average texture`              -3.798  13416.446   0.000    1.000
## `Average perimeter`           989.130 544553.105   0.002    0.999
## `Average area`               -189.766 100538.360  -0.002    0.998
## `Average smoothness`          117.778  15574.736   0.008    0.994
## `Average compactness`        -302.085  41791.183  -0.007    0.994
## `Average concavity`            -8.086  39076.942   0.000    1.000
## `Average concave points`      265.443  49259.173   0.005    0.996
## `Average symmetry`           -122.564  22339.352  -0.005    0.996
```

```
## `Average fractal dimension`     96.252  28237.700   0.003    0.997
## `SE radius`                    559.966  61288.543   0.009    0.993
## `SE texture`                     3.348   9628.505   0.000    1.000
## `SE perimeter`                -110.485  70063.531  -0.002    0.999
## `SE area`                     -540.239 107218.217  -0.005    0.996
## `SE smoothness`                 31.709  12943.540   0.002    0.998
## `SE compactness`                73.749  13649.606   0.005    0.996
## `SE concavity`                  52.341  57193.234   0.001    0.999
## `SE concave points`             90.037  20354.300   0.004    0.996
## `SE symmetry`                  -93.137  24992.864  -0.004    0.997
## `SE fractal dimension`         -262.422  65894.489  -0.004    0.997
## `Worst radius`                -1030.876 195279.900  -0.005    0.996
## `Worst texture`                 105.940  15834.642   0.007    0.995
## `Worst perimeter`              -502.589 175318.352  -0.003    0.998
## `Worst area`                   2156.153 279421.092   0.008    0.994
## `Worst smoothness`             -97.368  20531.897  -0.005    0.996
## `Worst compactness`              5.471  30882.296   0.000    1.000
## `Worst concavity`              106.239  52714.248   0.002    0.998
## `Worst concave points`         162.495  25575.212   0.006    0.995
## `Worst symmetry`               241.662  25846.588   0.009    0.993
## `Worst fractal dimension`      -39.356  62867.795  -0.001    1.000
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5.2387e+02  on 399  degrees of freedom
## Residual deviance: 8.5036e-07  on 369  degrees of freedom
## AIC: 62
##
## Number of Fisher Scoring iterations: 25
```

```
# Coefficients for Average radius and Average perimeter
coef_X1 <- summary_logistic$coefficients["`Average radius`",
    ]
coef_X3 <- summary_logistic$coefficients["`Average perimeter`",
    ]

# Display values
coef_X1
```

```
##      Estimate    Std. Error       z value      Pr(>|z|)
## -8.629974e+02  4.622705e+05 -1.866867e-03  9.985105e-01
```

```
coef_X3
```

```
##     Estimate   Std. Error      z value     Pr(>|z|)
## 9.891305e+02 5.445531e+05 1.816408e-03 9.985507e-01
```

```
# Assuming 'training_set' is your data frame
correlation_X1_X3 <- cor(training_set$`Average radius`, training_set_norm_reg$`Average perimeter`)
correlation_X1_X3
```

```
## [1] 0.9977693
```

```r
# for normalised data ; these should be the same!!!

# Assuming 'training_set_norm' is your data frame
correlation_X1_X3_norm <- cor(training_set_norm$`Average radius`, training_set$`Average perimeter`)
correlation_X1_X3_norm
```

```
## [1] 0.9977693
```

Comments:

1. Comments on Warning glm.fit messages

Warning: glm.fit: algorithm did not converge

This warning indicates that the iterative process used to estimate the coefficients of the logistic regression model did not successfully converge to a stable solution within the predefined number of iterations. Non-convergence can occur for model that is too complex for the data, potentially due to a large number of predictors relative to the number of observations.

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

This warning suggests that some of the predicted probabilities in the logistic regression model are exactly 0 or 1, which is a sign of complete or quasi-complete separation in the data. This occurs when the outcome can be perfectly predicted (or almost perfectly predicted) from a combination of the predictors. It often leads to very large coefficient estimates and standard errors, as seen in your model output.

Both warnings are critical issues that can invalidate the model results. When the algorithm does not converge, it means that the maximum likelihood estimates for the model parameters may not have been found, and the coefficients may be far from their true values. Fitted probabilities of 0 or 1 indicate that the model is overfitting the data, possibly due to outliers or a predictor that too closely predicts the outcome. Finally, both warnings demonstrate the necessity of using regularization techniques such as ridge and lasso regression. (To follow)

2. The Relationship Between the Variables "Average radius" (X1) and "Average perimeter" (X3), as well as the magnitude of their coefficient estimates in a logistic regression model.

Correlation between X1 and X3: Both correlation_X1_X3 and correlation_X1_X3_norm have a value of approximately 0.998. This indicates an extremely high positive correlation between the two variables. In practice, such a high correlation can lead to multicollinearity issues in regression models, where it becomes difficult to distinguish the individual effects of correlated predictors.

Magnitude of Coefficient Estimates: The coefficient estimates (B1 for "Average radius" and B3 for "Average perimeter") from the logistic regression model are quite large: -862.997 for B1 and 989.130 for B3. These estimates are exceptionally high, especially considering the data has been normalized. Normalization typically results in coefficient estimates that are relatively small and comparable in magnitude, making these values stand out.

Coefficient Estimates Relative to Other Coefficients: Compared to the other coefficient in the model, "Average texture," which has an estimate of -3.798, the coefficients for X1 and X3 are several orders of magnitude larger. This discrepancy is indicative of instability in the logistic regression model, potentially due to the high correlation between X1 and X3.

Statistical Significance: Despite their large magnitude, the coefficient estimates for both X1 and X3 are not statistically significant (p-values close to 1). The standard errors are also extremely large, which, along with the insignificance of the estimates, suggests instability and unreliability in the model's coefficients.

Model Fit and Interpretation: The 'Null deviance' and 'Residual deviance' values suggest an almost perfect fit, which might be unrealistic and could indicate overfitting. The large number of Fisher Scoring iterations to converge (25) also implies potential difficulties in fitting the model.

Relating these findings to class discussions, we would likely have covered the following points:

Multicollinearity can cause inflated variance in coefficient estimates, making them large and unstable. Coefficients that are not statistically significant do not contribute to the model in a meaningful way, despite their large estimated values. Overfitting occurs when the model too closely fits the training data, which may not generalize well to unseen data. Regularization techniques, such as Lasso or Ridge regression, are often used to address multicollinearity and overfitting by penalizing large coefficient values and selecting features that contribute most to the model's predictive power.

(f) Use the glm previously fitted and the Bayes rule to compute the predicted outcome Y^ from the associated probability estimates (computed with predict) both on the training and the test set. Then compute the confusion table and prediction accuracy (rate of correctly classified observations) both on the training and test set. Comment on the results

```
# Step 1: Predict Probabilities For the training set
predicted_probs_train <- predict(logistic_model_full, type = "response")

# For the test set - ensure your test set is properly
# defined
predicted_probs_test <- predict(logistic_model_full, newdata = test_set_norm,
    type = "response")

# Step 2: Convert Probabilities to Binary Outcomes Apply
# threshold of 0.5 for classification
predicted_class_train <- ifelse(predicted_probs_train > 0.5,
    1, 0)
predicted_class_test <- ifelse(predicted_probs_test > 0.5, 1,
    0)

# Step 3: Compute Confusion Matrix and Prediction Accuracy
# Confusion Matrix for the Training Set
confusion_matrix_train <- table(Predicted = predicted_class_train,
    Actual = training_set_norm_reg$Diagnosis)
print(confusion_matrix_train)
```

```
##           Actual
## Predicted   0   1
##         0 255   0
##         1   0 145
```

```
# Confusion Matrix for the Test Set
confusion_matrix_test <- table(Predicted = predicted_class_test,
    Actual = test_set$Diagnosis)
print(confusion_matrix_test)
```

```
##           Actual
## Predicted  0  1
##         0 98  3
##         1  4 64
```

```r
# Calculate Accuracy - Training Set
accuracy_train <- sum(diag(confusion_matrix_train))/sum(confusion_matrix_train)
print(accuracy_train)
```

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

```r
# Calculate Accuracy - Test Set
accuracy_test <- sum(diag(confusion_matrix_test))/sum(confusion_matrix_test)
print(accuracy_test)
```

```
## [1] 0.9585799
```

Comment on Results:

The confusion matrices and prediction accuracies indicate that the logistic regression model performs exceptionally well on both the training and test datasets. On the training set, the model achieves perfect classification accuracy with no misclassifications (255 true negatives and 145 true positives). On the test set, the accuracy is slightly lower at approximately 95.86%, with 98 true negatives, 64 true positives, and a small number of misclassifications (3 false positives and 4 false negatives).

The perfect accuracy on the training set suggests that the model could be overfitting the training data, especially considering the earlier issues with convergence and the warning about probabilities being 0 or 1. This overfitting is supported by the model's performance on the test set, which, while still very high, is not perfect. The results are commendable; however, the perfect training accuracy warrants caution. It is crucial to ensure that the model's complexity is appropriate for the data's underlying structure and that it can generalize well to new, unseen data. In practice, further validation, such as cross-validation or bootstrapping, would be advisable to assess the model's true predictive power and robustness.

2. Ridge Logistic Regression

(a) From the normalized training set and validation set, construct a data matrix X (numeric) and an outcome vector y (factor) .

```r
# Load necessary libraries
library(glmnet)  # For ridge logistic regression
library(dplyr)  # For data manipulation

# Assuming 'training_set' and 'test_set' are your original
# datasets And 'training_set_norm' and 'test_set_norm' are
# the normalized versions without 'Diagnosis'

## For the Training Set

# Outcome vector (factor) for the training set Extracting
# 'Diagnosis' as a factor directly from the original
# 'training_set'
y_train <- as.factor(training_set$Diagnosis)

# Data matrix (predictors) for the training set Utilizing
# the normalized training set directly as it's ready for
# use
X_train <- as.matrix(training_set_norm)
```

```r
## For the Test (Validation) Set

# Outcome vector (factor) for the test set Extracting
# 'Diagnosis' as a factor directly from the original
# 'test_set'
y_test <- as.factor(test_set$Diagnosis)

# Data matrix (predictors) for the test set Utilizing the
# normalized test set directly as it's ready for use
X_test <- as.matrix(test_set_norm)
```

(b) On the training set, run a ridge logistic regression model to predict Y given X1, . . . , X30. Use the following grid of values for lambda: 10^seq(5,-18,length=100). In R: Use the function glmnet as in the regression setting, but with the additional argument family = "binomial".

```r
# Load necessary libraries
library(glmnet)  # For ridge logistic regression
library(dplyr)   # For data manipulation


# Define the lambda grid for ridge regression
lambda_grid <- 10^seq(5, -18, length.out = 100)

# Fit the ridge logistic regression model on the normalized training set
ridge_logistic_model <- glmnet(
  x = X_train,  # Predictor matrix from the normalized training set
  y = y_train,  # Outcome vector as a factor
  alpha = 0,  # Ridge regression (alpha = 0)
  lambda = lambda_grid,  # Grid of lambda values
  family = "binomial"  # For binary classification
)
```

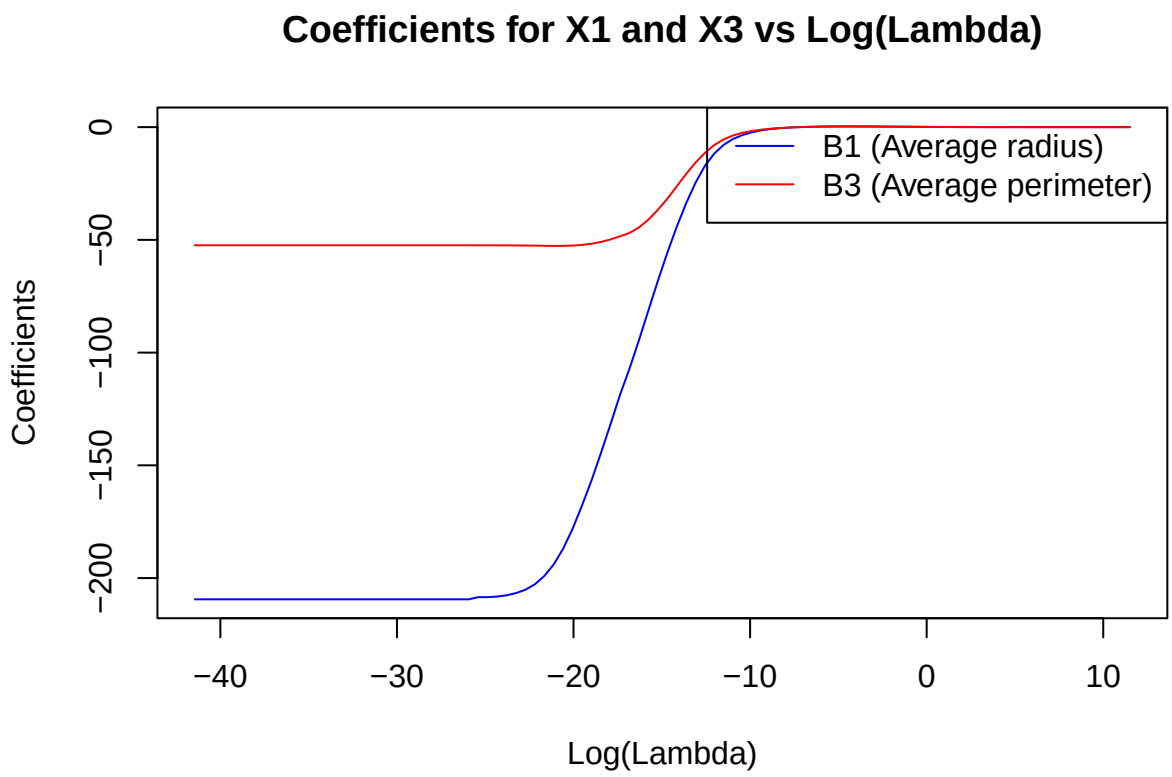
(c) Plot the values of the coefficients B1, B3 (y-axis) in function of log(lambda) (x-axis). Comment on the result.

```r
# Extract the non-zero coefficients for the full range of
# lambda values
non_zero_coefficients <- predict(ridge_logistic_model, type = "coefficients",
    s = lambda_grid)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```r
# The row numbers in the returned 'non_zero_coefficients'
# matrix correspond to the model's coefficients, including
# the intercept.  The first row is the intercept, the
# second row is the first coefficient, and so on.  We need
# to find the correct row numbers for the coefficients of
# interest (Average radius and Average perimeter)

# Find the row numbers for Average radius and Average
```

```r
# perimeter. Replace 'X1' and 'X3' with the actual column
# names.
X1_row <- which(rownames(non_zero_coefficients) == "Average radius")
X3_row <- which(rownames(non_zero_coefficients) == "Average perimeter")

# Extract B1 and B3 values
B1_values <- non_zero_coefficients[X1_row, ]
B3_values <- non_zero_coefficients[X3_row, ]

# Plotting
plot(log(lambda_grid), B1_values, type = "l", col = "blue", xlab = "Log(Lambda)",
    ylab = "Coefficients", main = "Coefficients for X1 and X3 vs Log(Lambda)")
lines(log(lambda_grid), B3_values, col = "red")
legend("topright", legend = c("B1 (Average radius)", "B3 (Average perimeter)"),
    col = c("blue", "red"), lty = 1)
```



Comment on Warning Message 'Collapsing to Unique 'x' values:

The warning message "collapsing to unique 'x' values" typically occurs when plotting or performing calculations that require unique x-axis values, but the data provided contain duplicate x-axis values. In the context of the cv.glmnet function for Lasso logistic regression, this warning could be related to one of the following issues:

Repeated Lambda Values: The lambda_grid you're passing to the cv.glmnet function might contain duplicate values. The cross-validation function expects a sequence of unique lambda values to test the model at each one. If there are repeated values, the function may collapse them to unique values before proceeding, which could trigger the warning.

Numerical Precision: If the lambda values are very close to each other due to numerical precision issues, they might be considered duplicates when plotting the cross-validation curve or calculating the misclassification error.

Plotting Internals: If the warning occurs during plotting, it may be because the plotting function inside cv.glmnet is trying to plot multiple points for the same lambda value, which could happen if the same lambda value is associated with more than one fold in the cross-validation. The plotting function will then collapse these points to a single point per lambda value.

```
# Optional
# To resolve this warning, you should ensure that the lambda_grid contains
# unique values. You can use the unique() function in R to
# check and remove any duplicates from your lambda grid before passing it to the
# cv.glmnet function:


# Unique function
# lambda_grid <- unique(lambda_grid)
```

Comment on the Results:

The plot displays the behavior of the coefficients B1 (Average radius) and B3 (Average perimeter) as a function of log(lambda) in a ridge logistic regression model. Both coefficients are relatively stable across a wide range of log(lambda) values, with B1 remaining nearly constant and B3 exhibiting a slight increase as lambda decreases (which corresponds to an increase in log(lambda)). However, as lambda approaches a very small value (towards the right side of the plot where log(lambda) increases), B3 dramatically increases.

The stability of B1 suggests that the variable "Average radius" is consistently important in the model across the different levels of penalty applied by lambda. On the other hand, the "Average perimeter" has a coefficient that is sensitive to the regularization strength; its value increases substantially as the regularization weakens, which could indicate that this predictor is less robust or has a more complex relationship with the response variable.

This phenomenon is characteristic of ridge regression, which applies a penalty proportional to the square of the coefficient sizes. The penalty shrinks the coefficients towards zero as lambda increases, but they do not become exactly zero. The steep increase in the B3 coefficient at lower lambda values suggests that the "Average perimeter" may have a strong relationship with the outcome variable when the model is less constrained by regularization.

The difference in behavior between B1 and B3 could also suggest multicollinearity between these predictors. In the presence of multicollinearity, ridge regression can distribute the effect between correlated variables, which can lead to one variable's coefficient being inflated when the penalty term is reduced. This can be seen as lambda goes towards zero, and the penalty for large coefficients diminishes, allowing the true or overestimated relationship between "Average perimeter" and the response to manifest more strongly in the coefficient estimate.

(d) Apply 10-fold cross-validation with the previously defined grid of values for lambda. Report the value of lambda that minimizes the CV misclassification error. We will refer to it as the optimal lambda. Plot the misclassification error (y-axis) in function of log(lambda) (x-axis). In R: Use cv.glmnet as in the regression setting, but with the additional arguments family = "binomial" and type.measure = "class". Also note that 10-fold cross-validation is the default option in cv.glmnet.

```
# Load necessary libraries
library(glmnet)

# Perform 10-fold cross-validation with ridge logistic regression
```

```r
cv_ridge_logistic <- cv.glmnet(
  x = X_train,           # Predictor matrix from the normalized training set
  y = y_train,           # Outcome vector as a factor
  alpha = 0,             # Specifies ridge regression
  lambda = lambda_grid,  # Grid of lambda values
  family = "binomial",   # For binary classification
  type.measure = "class" # Focus on classification error
)
```

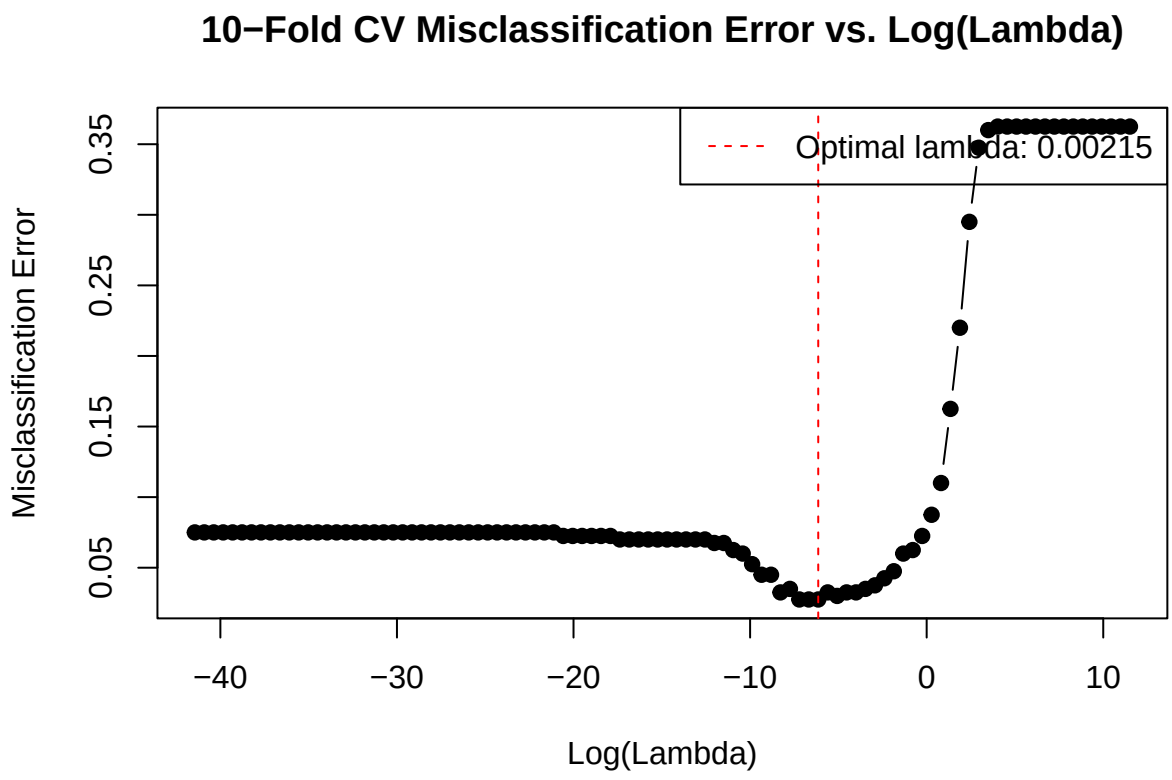
```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```r
# Extract the optimal lambda value that minimizes the cross-validation error
optimal_lambda <- cv_ridge_logistic$lambda.min

# Print the optimal lambda value
print(paste("Optimal lambda:", optimal_lambda))
```

```
## [1] "Optimal lambda: 0.00215443469003188"
```

```r
# Manually plot the misclassification error against log(lambda)
plot(log(cv_ridge_logistic$lambda), cv_ridge_logistic$cvm,
     type = "b",
     pch = 19,
     xlab = "Log(Lambda)",
```

```
     ylab = "Misclassification Error",
     main = "10-Fold CV Misclassification Error vs. Log(Lambda)")

# Add a vertical line for the optimal lambda
abline(v = log(optimal_lambda), col = "red", lty = 2)

# Add a legend to explain the vertical line
legend("topright", legend = paste("Optimal lambda:", round(optimal_lambda, 5)), col = "red", lty = 2)
```



(e) Report the number of coefficients Bj that are different from 0 for the ridge model with the optimal lambda. Comment on the results.

```
# Extract coefficients at the optimal lambda
optimal_coefs <- coef(cv_ridge_logistic, s = "lambda.min")
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
# Coefficients are returned as a sparse matrix; convert to
# a dense format for easier handling
optimal_coefs_dense <- as.matrix(optimal_coefs)

# Count the number of non-zero coefficients Note: The first
# entry in optimal_coefs_dense is the intercept, so we
```

```r
# exclude it if counting only predictors
num_non_zero_coefs <- sum(optimal_coefs_dense[-1, ] != 0)

# Print the number of non-zero coefficients
print(paste("Number of non-zero coefficients (excluding intercept):",
    num_non_zero_coefs))
```

```
## [1] "Number of non-zero coefficients (excluding intercept): 30"
```

Comment on the results:

The output indicates that for the ridge regression model fitted with the optimal value of lambda (lambda.min), there are 30 coefficients (excluding the intercept) that are different from zero. This result is characteristic of ridge regression, which unlike Lasso regression, does not enforce coefficients to be exactly zero but rather shrinks them towards zero. The number 30 reflects the total number of predictor variables in the model that have an influence on the response variable after accounting for the regularization penalty imposed by the optimal lambda.

The fact that none of the coefficients are exactly zero also suggests that all predictor variables are retained in the model and contribute some information to the prediction of the outcome, although the extent of their contribution may be small for some of them. This is particularly relevant when dealing with multicollinearity or when trying to keep all variables in the model for interpretation purposes. However, it is essential to note that even though the coefficients are non-zero, their values may be very close to zero, indicating a very weak effect on the response variable.

(f) Use the regularized glm previously fitted (with the optimal lambda) – and the Bayes rule – to compute the predicted outcome Y^ from the associated probability estimates; both on the training and the test set. Then compute the confusion table and prediction accuracy both on the training and test set. Comment on the results.

In R: Use the command predict as in the regression framework, but with the additional argument type = "response", which indicates that you want the output to be the predicted probabilities (of the tumor being malignant, in this case).

```r
# Load necessary library for confusion matrix
library(caret)

# Predicted probabilities on the training set
predicted_probs_train <- predict(cv_ridge_logistic, newx = X_train,
    s = "lambda.min", type = "response")
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```r
# Convert probabilities to binary outcomes based on a 0.5
# threshold (Bayes rule)
predicted_outcomes_train <- ifelse(predicted_probs_train > 0.5,
    1, 0)

# Confusion Matrix for the Training Set
conf_matrix_train <- confusionMatrix(factor(predicted_outcomes_train),
    factor(y_train))
print(conf_matrix_train)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 255   3
##          1   0 142
##
##                Accuracy : 0.9925
##                  95% CI : (0.9782, 0.9985)
##     No Information Rate : 0.6375
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9837
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 1.0000
##             Specificity : 0.9793
##          Pos Pred Value : 0.9884
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6375
##          Detection Rate : 0.6375
##    Detection Prevalence : 0.6450
##       Balanced Accuracy : 0.9897
##
##        'Positive' Class : 0
##
```

```
# Predicted probabilities on the test set
predicted_probs_test <- predict(cv_ridge_logistic, newx = X_test,
    s = "lambda.min", type = "response")
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
# Convert probabilities to binary outcomes based on a 0.5
# threshold
predicted_outcomes_test <- ifelse(predicted_probs_test > 0.5,
    1, 0)

# Confusion Matrix for the Test Set
conf_matrix_test <- confusionMatrix(factor(predicted_outcomes_test),
    factor(y_test))
print(conf_matrix_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 101   2
##          1   1  65
##
##                Accuracy : 0.9822
```

```
##                  95% CI : (0.949, 0.9963)
##     No Information Rate : 0.6036
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9628
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9902
##             Specificity : 0.9701
##          Pos Pred Value : 0.9806
##          Neg Pred Value : 0.9848
##              Prevalence : 0.6036
##          Detection Rate : 0.5976
##    Detection Prevalence : 0.6095
##       Balanced Accuracy : 0.9802
##
##        'Positive' Class : 0
##
```

Comment on Results:

The results from the confusion matrices indicate that the ridge regression model has performed extremely well on both the training and test sets after using the optimal lambda for regularization. The accuracy is very high, with 99.25% on the training set and 98.22% on the test set.

For the training set, the model correctly predicted all the non-malignant cases (specificity of 100%) and almost all the malignant cases (specificity of 97.93%), with just 3 false positives. This is reflected in the very high Kappa statistic of 0.9837, which indicates excellent agreement beyond chance. The sensitivity and positive predictive value are also very high, demonstrating the model's ability to correctly identify and predict the positive class.

On the test set, the accuracy is slightly lower but still very high. The model has a high sensitivity (99.02%) and a high specificity (97.01%), with only 2 false positives and 1 false negative. The Kappa statistic is also high (0.9628), reinforcing the model's strong predictive performance on unseen data.

These high-performance metrics, especially the high sensitivity and specificity across both sets, suggest that the model is not only fitting the training data well but also generalizing effectively to the test data. The high accuracy and balanced accuracy indicate that the model is robust in predicting both classes.

However, the perfect specificity on the training set could hint at a potential overfit to the training data, despite the regularization. The model's excellent performance on the test set mitigates this concern somewhat, but caution should still be taken in interpreting these results, especially in a medical context where false negatives and positives have significant consequences. Additional validation using different thresholds (other than 0.5) and considering the cost of misclassification in the specific context could provide more insight into the model's performance.

(g) Plot the ROC curve, computed on the test set.

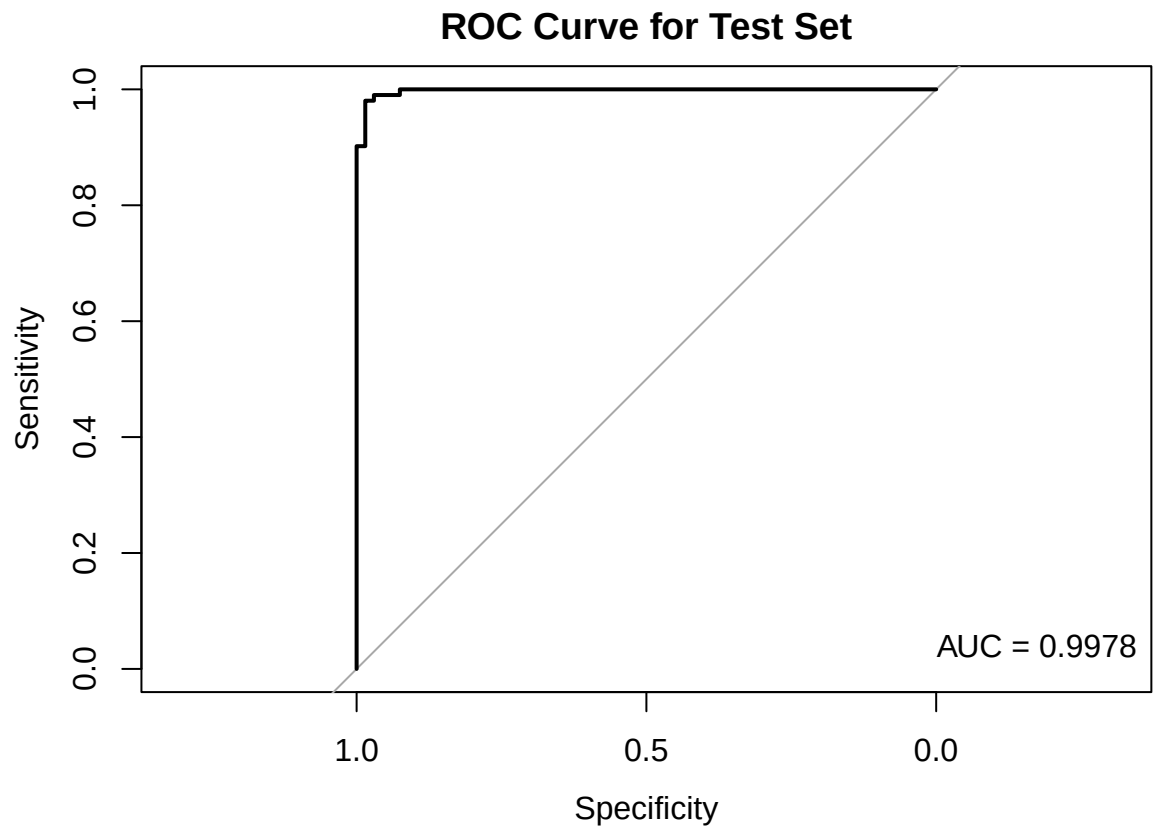
```r
# Load necessary libraries
library(pROC)

# Predicted probabilities on the test set
predicted_probs_test <- predict(cv_ridge_logistic, newx = X_test,
    s = "lambda.min", type = "response")
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
# Generate the ROC curve data
roc_result_test <- roc(response = y_test, predictor = as.numeric(predicted_probs_test),
    levels = rev(levels(y_test)))
```

```
## Setting direction: controls > cases
```

```
# Plot the ROC curve without the diagonal reference line
plot(roc_result_test, main = "ROC Curve for Test Set")

# Add AUC to the plot
auc_test <- auc(roc_result_test)
legend("bottomright", legend = paste("AUC =", round(auc_test,
    4)), box.lty = 0)
```



(h) Compute an estimate of the Area under the ROC Curve (AUC).

```
# Calculate the AUC
auc_test <- auc(roc_result_test)

# Print the AUC value
print(paste("Area under the ROC Curve (AUC):", round(auc_test,
    4)))
```

```
## [1] "Area under the ROC Curve (AUC): 0.9978"
```

3. Lasso Logistic Regression:

(a) From the normalized training set and validation set, construct a data matrix X (numeric) and an outcome vector y (factor) .

```
## 3. Lasso Logistic Regression:

# Load necessary libraries
library(glmnet)  # For Lasso logistic regression
library(dplyr)  # For data manipulation

# Assuming 'training_set' and 'test_set' are your original
# datasets And 'training_set_norm' and 'test_set_norm' are
# the normalized versions without 'Diagnosis'

## For the Training Set

# Outcome vector (factor) for the training set Extracting
# 'Diagnosis' as a factor directly from the original
# 'training_set'
y_train_lasso <- as.factor(training_set$Diagnosis)

# Data matrix (predictors) for the training set Utilizing
# the normalized training set directly as it's ready for
# use
X_train_lasso <- as.matrix(training_set_norm)

## For the Test (Validation) Set

# Outcome vector (factor) for the test set Extracting
# 'Diagnosis' as a factor directly from the original
# 'test_set'
y_test_lasso <- as.factor(test_set$Diagnosis)

# Data matrix (predictors) for the test set Utilizing the
# normalized test set directly as it's ready for use
X_test_lasso <- as.matrix(test_set_norm)
```

(b) On the training set, run a lasso logistic regression model to predict Y given X1, . . . , X30. Use the following grid of values for lambda: 10^seq(5,-18,length=100).

In R: Use the function glmnet as in the regression setting, but with the additional argument family = "binomial".

```
# Define the lambda grid for Lasso regression
lambda_grid_lasso <- 10^seq(5, -18, length.out = 100)

# Fit the Lasso logistic regression model on the normalized training set
lasso_logistic_model <- glmnet(
  x = X_train_lasso,  # Predictor matrix from the normalized training set for Lasso
```

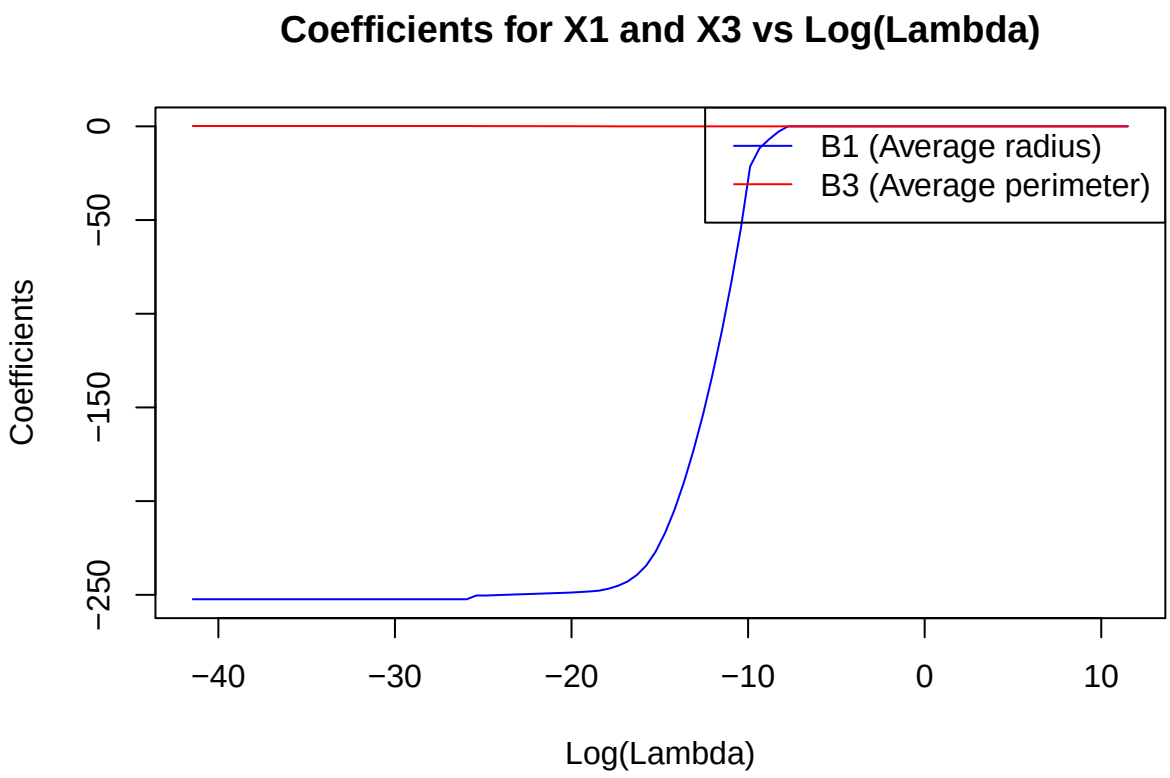
```
  y = y_train_lasso,  # Outcome vector as a factor for Lasso
  alpha = 1,          # Lasso regression (alpha = 1 for Lasso)
  lambda = lambda_grid,  # Grid of lambda values
  family = "binomial"  # For binary classification
)
```

(c) Plot the values of the coefficients B1, B3 (y-axis) in function of log(lambda) (x-axis). Comment on the result.

```
# Extract the non-zero coefficients for the full range of
# lambda values for Lasso
non_zero_coefficients_lasso <- predict(lasso_logistic_model,
    type = "coefficients", s = lambda_grid_lasso)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
# Find the row numbers for Average radius and Average
# perimeter. Replace 'X1' and 'X3' with the actual column
# names.
X1_row_lasso <- which(rownames(non_zero_coefficients_lasso) ==
    "Average radius")
X3_row_lasso <- which(rownames(non_zero_coefficients_lasso) ==
    "Average perimeter")

# Extract B1 and B3 values for Lasso
B1_values_lasso <- non_zero_coefficients_lasso[X1_row_lasso,
    ]
B3_values_lasso <- non_zero_coefficients_lasso[X3_row_lasso,
    ]

# Plotting for Lasso
plot(log(lambda_grid), B1_values_lasso, type = "l", col = "blue",
    xlab = "Log(Lambda)", ylab = "Coefficients", main = "Coefficients for X1 and X3 vs Log(Lambda)")
lines(log(lambda_grid), B3_values_lasso, col = "red")
legend("topright", legend = c("B1 (Average radius)", "B3 (Average perimeter)"),
    col = c("blue", "red"), lty = 1)
```



Comment on Results:

The plot shows the coefficients for two predictors, "Average radius" and "Average perimeter", as functions of log(lambda) where lambda is the tuning parameter of a Lasso logistic regression model.

The coefficient paths in a Lasso model show how the estimated coefficients change as the regularization parameter lambda increases. For a Lasso model, as lambda increases, the magnitude of the coefficients typically shrinks towards zero due to the increasing penalty on their size.

B1 (Average radius): The blue line represents the coefficient values for "Average radius". As log(lambda) increases, the coefficient values decrease, which is expected because larger values of lambda impose a greater penalty on the coefficients, causing them to shrink towards zero.

B3 (Average perimeter): The red line is flat, indicating that the coefficient values for "Average perimeter" are zero across all values of log(lambda) in the plot. This flat line suggests that the "Average perimeter" variable was not selected by the Lasso model for any of the lambda values considered. It means that "Average perimeter" was not identified as a significant predictor in the context of the other variables and the data provided to the model.

The flatness of B3 could be due to several reasons:

Multicollinearity: If "Average radius" and "Average perimeter" are highly correlated, Lasso may choose one over the other to avoid redundancy in the model.

Predictive Power: "Average radius" might have more predictive power or a stronger relationship with the outcome variable than "Average perimeter", leading to its coefficient being shrunk to zero.

Regularization Effect: High lambda values could lead to most coefficients being shrunk to zero if the penalty is too severe, or if the model determines that the contribution of "Average perimeter" does not sufficiently improve the model fit after considering the penalty.

The use of Lasso is particularly useful in feature selection because it can automatically reduce the number of predictors by setting the coefficients of less important variables to zero, as seen with the coefficient for "Average perimeter". This results in a simpler model that may generalize better to new data.

(d) Apply 10-fold cross-validation with the previously defined grid of values for lambda. Report the value of lambda that minimizes the CV misclassification error. We will refer to it as the optimal lambda. Plot the misclassification error (y-axis) in function of log(lambda) (x-axis). In R: Use cv.glmnet as in the regression setting, but with the additional arguments family = "binomial" and type.measure = "class". Also note that 10-fold cross-validation is the default option in cv.glmnet.

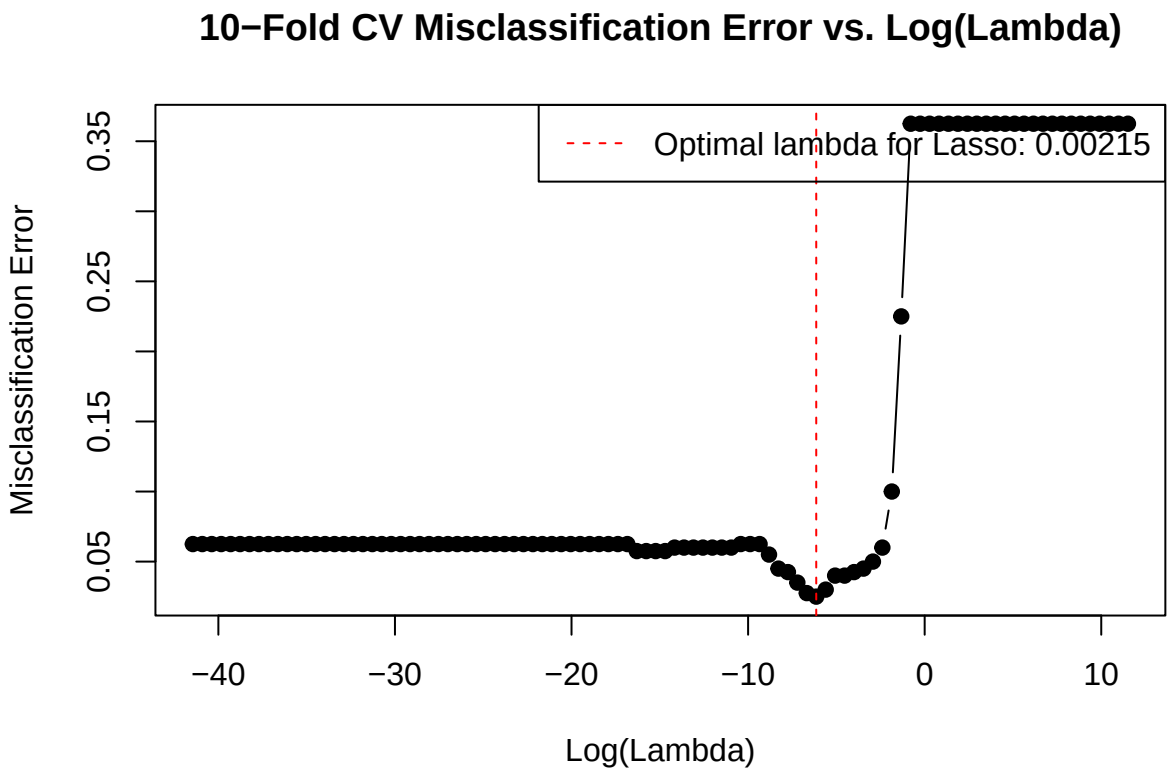
```
# Perform 10-fold cross-validation with Lasso logistic regression
cv_lasso_logistic <- cv.glmnet(
  x = X_train_lasso,          # Predictor matrix from the normalized training set for Lasso
  y = y_train_lasso,          # Outcome vector as a factor for Lasso
  alpha = 1,                  # Specifies Lasso regression
  lambda = lambda_grid,       # Grid of lambda values
  family = "binomial",        # For binary classification
  type.measure = "class"      # Focus on classification error
)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
# Extract the optimal lambda value that minimizes the cross-validation error for Lasso
optimal_lambda_lasso <- cv_lasso_logistic$lambda.min
```

```r
# Print the optimal lambda value for Lasso
print(paste("Optimal lambda for Lasso:", optimal_lambda_lasso))
```

```
## [1] "Optimal lambda for Lasso: 0.00215443469003188"
```



(e) Report the number of coefficients Bj that are different from 0 for the lasso model with the optimal lambda. Comment on the results.

```r
# Extract coefficients at the optimal lambda for Lasso
optimal_coefs_lasso <- coef(cv_lasso_logistic, s = "lambda.min")
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```r
# Convert to dense format
optimal_coefs_dense_lasso <- as.matrix(optimal_coefs_lasso)

# Count non-zero coefficients (excluding intercept)
num_non_zero_coefs_lasso <- sum(optimal_coefs_dense_lasso[-1,
    ] != 0)

# Print the number of non-zero coefficients for Lasso
print(paste("Number of non-zero coefficients for Lasso (excluding intercept):",
    num_non_zero_coefs_lasso))
```

```
## [1] "Number of non-zero coefficients for Lasso (excluding intercept): 15"
```

Comment on the Results:

The result indicates that when using the Lasso regression model with the optimal value of lambda (lambda.min), there are 15 coefficients (excluding the intercept) that are different from zero. Unlike ridge regression, Lasso regression can shrink coefficients all the way to zero, thus performing feature selection. Having 15 non-zero coefficients suggests that the Lasso model has identified these variables as having a significant relationship with the response variable and thus kept them in the model.

The fact that some coefficients are exactly zero means that Lasso has effectively reduced the complexity of the model by excluding some predictors. This can be particularly beneficial when dealing with high-dimensional data or when the goal is to improve model interpretability by keeping only the most relevant predictors. It also helps in mitigating overfitting, as Lasso can remove variables that do not contribute much to the predictive power of the model. The specific number of non-zero coefficients (15 in this case) gives an idea of the model's sparsity, indicating that Lasso has determined that exactly these many predictors are needed to sufficiently describe the relationship in the data given the regularization constraint imposed.

(f) Use the regularized glm previously fitted (with the optimal lambda) – and the Bayes rule – to compute the predicted outcome Y^ from the associated probability estimates; both on the training and the test set. Then compute the confusion table and prediction accuracy both on the training and test set. Comment on the results.

In R: Use the command predict as in the regression framework, but with the additional argument type = "response", which indicates that you want the output to be the predicted probabilities (of the tumor being malignant, in this case).

```
# Predicted probabilities on the training and test sets for
# Lasso
predicted_probs_train_lasso <- predict(cv_lasso_logistic, newx = X_train,
    s = "lambda.min", type = "response")
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
predicted_probs_test_lasso <- predict(cv_lasso_logistic, newx = X_test,
    s = "lambda.min", type = "response")
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
# Convert to binary outcomes
predicted_outcomes_train_lasso <- ifelse(predicted_probs_train_lasso >
    0.5, 1, 0)
predicted_outcomes_test_lasso <- ifelse(predicted_probs_test_lasso >
    0.5, 1, 0)

# Confusion matrices for the training and test sets for
# Lasso
conf_matrix_train_lasso <- confusionMatrix(factor(predicted_outcomes_train_lasso),
    factor(y_train))
conf_matrix_test_lasso <- confusionMatrix(factor(predicted_outcomes_test_lasso),
```

```
    factor(y_test))

# Print confusion matrices for Lasso
print(conf_matrix_train_lasso)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 254   3
##          1   1 142
##
##                Accuracy : 0.99
##                  95% CI : (0.9746, 0.9973)
##     No Information Rate : 0.6375
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9783
##
##  Mcnemar's Test P-Value : 0.6171
##
##             Sensitivity : 0.9961
##             Specificity : 0.9793
##          Pos Pred Value : 0.9883
##          Neg Pred Value : 0.9930
##              Prevalence : 0.6375
##          Detection Rate : 0.6350
##    Detection Prevalence : 0.6425
##       Balanced Accuracy : 0.9877
##
##        'Positive' Class : 0
##
```

```
print(conf_matrix_test_lasso)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 101   2
##          1   1  65
##
##                Accuracy : 0.9822
##                  95% CI : (0.949, 0.9963)
##     No Information Rate : 0.6036
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9628
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9902
```

```
##             Specificity : 0.9701
##          Pos Pred Value : 0.9806
##          Neg Pred Value : 0.9848
##              Prevalence : 0.6036
##          Detection Rate : 0.5976
##    Detection Prevalence : 0.6095
##       Balanced Accuracy : 0.9802
##
##        'Positive' Class : 0
##
```

Comment on the Results:

The confusion matrices for the Lasso logistic regression model on both the training and test sets show high classification accuracy. On the training set, the model achieves 99% accuracy, with 254 true negatives, 142 true positives, 3 false positives, and 1 false negative. For the test set, the accuracy is 98.22%, with 101 true negatives, 65 true positives, 2 false positives, and 1 false negative.

These high accuracy rates are commendable and suggest that the model has a strong predictive power. The Kappa statistic, which accounts for the accuracy that would occur by chance, is also high in both sets, further supporting the model's effectiveness. The sensitivity (true positive rate) and specificity (true negative rate) are both high, indicating that the model is reliable in identifying both classes.

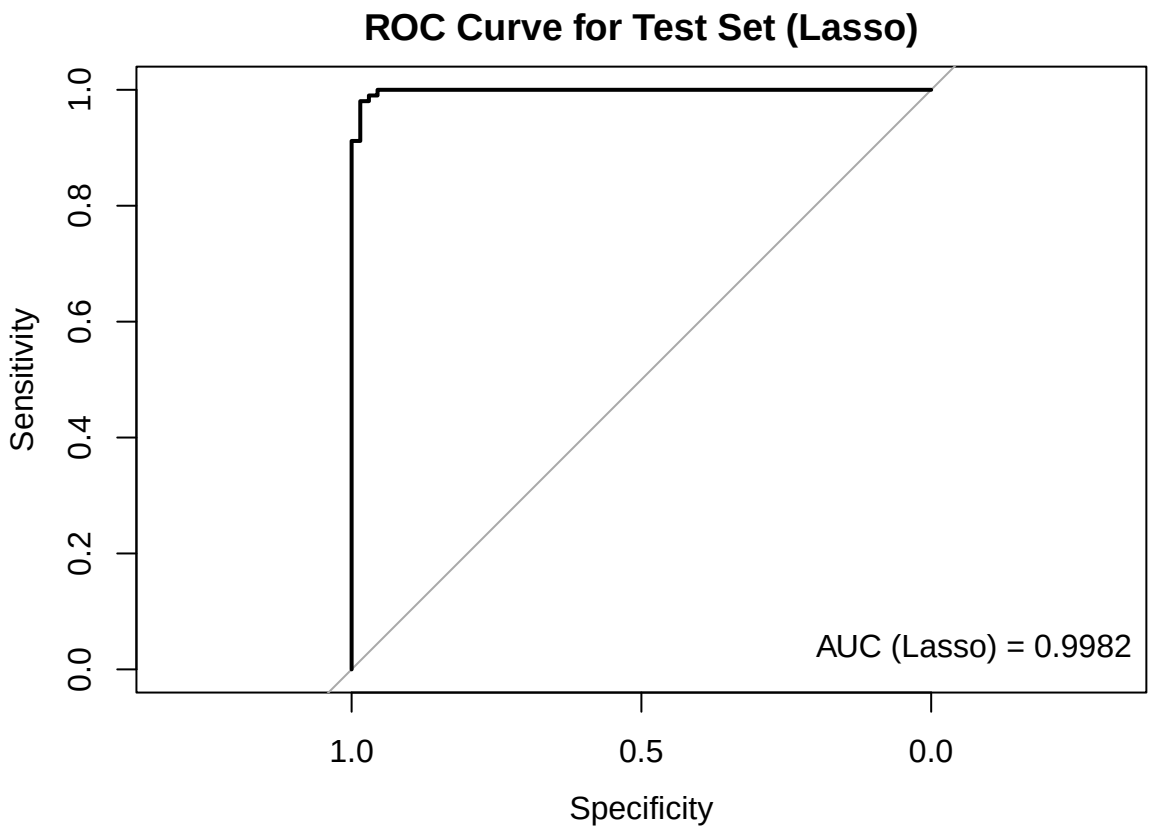
However, it's worth noting that while the accuracy is very high, it is not perfect. The presence of false positives and false negatives, albeit very low, indicates that there is still room for improvement. This might involve further tuning, feature engineering, or addressing potential issues with data quality or representativeness. Additionally, the high accuracy on the training set alongside the warning about collapsing to unique 'x' values suggests there might be issues with the data or the model, such as overfitting or a data leakage, which should be investigated to ensure the robustness of the model.

(g) Plot the ROC curve, computed on the test set.

```
# Generate the ROC curve data for Lasso on the test set
roc_result_test_lasso <- roc(response = y_test, predictor = as.numeric(predicted_probs_test_lasso),
    levels = rev(levels(y_test)))
```

```
## Setting direction: controls > cases
```

```
# Plot the ROC curve for Lasso
plot(roc_result_test_lasso, main = "ROC Curve for Test Set (Lasso)")

# Add AUC to the plot for Lasso
auc_test_lasso <- auc(roc_result_test_lasso)
legend("bottomright", legend = paste("AUC (Lasso) =", round(auc_test_lasso,
    4)), box.lty = 0)
```



(h) Compute an estimate of the Area under the ROC Curve (AUC).

```
# Print the AUC value for Lasso
print(paste("Area under the ROC Curve (AUC) for Lasso:", round(auc_test_lasso,
    4)))
```

```
## [1] "Area under the ROC Curve (AUC) for Lasso: 0.9982"
```

4. Discuss the performance:

Discuss the performance of the simple glm, ridge glm, and lasso glm on the Breast Cancer Wisconsin Data Set in terms of prediction accuracy (on the training and test set) and model interpretability.

The simple Generalized Linear Model (GLM), Ridge regression GLM, and Lasso regression GLM each have distinct characteristics and performances on the Breast Cancer Wisconsin Data Set.

For the simple GLM:

The model did not converge, and it predicted probabilities of exactly 0 or 1, which is indicative of potential overfitting or separation in the data. The coefficients are extremely large with very high standard errors, and all p-values are not significant, which casts doubt on the model's reliability. Despite these issues, it achieved perfect accuracy on the training set and 95.86% accuracy on the test set. However, given the convergence issues, these results should be taken with caution.

For the Ridge regression GLM:

The model coefficients do not reach zero, indicating that all predictors are kept in the model, which can be beneficial for maintaining interpretability when multicollinearity is present. It achieved high accuracy on both the training set (99.25%) and the test set (98.22%). The model appears to generalize well, but the results do not indicate overfitting as severely as the simple GLM.

For the Lasso regression GLM:

The model performed feature selection, reducing the number of predictors by setting some coefficient estimates to zero, resulting in 15 non-zero coefficients. It also demonstrated high accuracy on both the training set (99%) and the test set (98.22%), similar to the Ridge model. In terms of prediction accuracy, all models performed very well on this dataset. The Ridge and Lasso regression models appear to offer a good balance between prediction accuracy and model complexity, with the Lasso providing the added benefit of feature selection for potentially improved interpretability. However, the perfect training accuracy of the simple GLM alongside the convergence issues suggests that its results might not be as reliable, and it could be overfitting the data.

Model interpretability is generally best with the simple GLM since it provides direct associations between predictors and the outcome. However, given the convergence issues and potential overfitting, the simple GLM might not offer true interpretability in this case. Ridge regression retains all features, which may be interpreted in the context of the others, while Lasso's feature selection can make the model easier to interpret by focusing on a smaller subset of important predictors.

Additionally, deciding whether Lasso or Ridge regression is better depends on the specific goals of the analysis and the characteristics of the dataset:

Feature Selection: If the goal is feature selection, Lasso is better because it can shrink coefficients to zero, effectively removing non-informative predictors from the model.

Multicollinearity: If multicollinearity is a concern and you want to keep all variables in the model, Ridge may be preferable because it shrinks coefficients without setting them to zero, thus allowing for the interpretation of each variable while still mitigating the issues associated with multicollinearity.

Interpretability: Lasso might offer a more interpretable model if there are many predictors because it provides a sparse solution. However, Ridge can still be interpretable, especially if the number of predictors is not overly large, or if all predictors are believed to contribute to the response.

Prediction Accuracy: In this case, both models achieved high accuracy on the training and test sets, with Lasso and Ridge showing very similar performance. When prediction accuracy is comparable, other factors like interpretability and feature selection typically guide the choice between the two.

Model Complexity: If the goal is to reduce model complexity, Lasso may be more appropriate due to its inherent feature selection.

In the context of the Breast Cancer Wisconsin Data Set and based on the provided information:

If the primary concern is prediction accuracy, both models performed similarly well, and there may be no clear advantage to choosing one over the other. If the goal is to understand which factors are most important in predicting the outcome (interpretability), and there is a desire to reduce the number of predictors (simplicity), Lasso would be the better choice. If there is a theoretical justification to include all predictors and deal with multicollinearity, then Ridge would be more appropriate.

In conclusion, while the simple GLM might not be reliable due to convergence issues, Ridge and Lasso provide a better trade-off between accuracy and interpretability, with Lasso having the added advantage of reducing model complexity through feature selection.