Midterm Project: Bayesian Linear Model, Ridge, and Lasso Regression for MPG Prediction

In this Jupyter Notebook, we will explore and apply Bayesian Linear Model, Ridge, and Lasso regression techniques to predict the miles per gallon (MPG) of cars based on various features. The objective is to employ these regression models with variable selection techniques using 5-fold cross-validation to optimize the Root Mean Squared Error (RMSE).

Overview

The dataset used for this project contains information about different cars, including features such as cylinders, displacement, horsepower, weight, acceleration, model year, origin, and possibly others. The goal is to predict the MPG of cars using these features.

Goals

1. Implement Bayesian Linear Model, Ridge, and Lasso regression models.
2. Utilize 5-fold cross-validation for model evaluation and hyperparameter tuning.
3. Achieve a smaller RMSE with Bayesian Ridge and Lasso compared to Bayesian Linear Model.
4. Explore techniques for reducing collinearity, such as removing redundant categorical features, to potentially improve prediction accuracy of Bayesian Ridge and Lasso models.

Methodology

- **Data Loading and Preparation**: Load the MPG dataset and preprocess as necessary (handling missing values, encoding categorical variables).
- **Model Implementation**:
  - **Bayesian Linear Model**: Establish a baseline using Bayesian regression.
  - **Ridge Regression**: Apply Ridge regression with hyperparameter tuning using cross-validation.
  - **Lasso Regression**: Implement Lasso regression with cross-validation for feature selection.
- **Evaluation**: Compare the performance of Bayesian Ridge and Lasso models with the Bayesian Linear Model in terms of RMSE using 5-fold cross-validation.
- **Feature Selection**: Explore methods to reduce collinearity and improve model performance, particularly for Bayesian Ridge and Lasso models.

By the end of this notebook, we aim to have a clear understanding of how Bayesian Ridge and Lasso regression models perform in predicting MPG, compared to a Bayesian Linear Model. Additionally, we will explore the impact of feature selection techniques on model accuracy and provide insights into further improving predictions.

Let's dive into the implementation!

```
library(MASS)


gibbs_sampler_blr <- function(X, y, n_iter = 1000, burn_in = 100) {
  n <- nrow(X)
  p <- ncol(X)

  # Prior parameters
  beta_prior_mean <- rep(0, p)
  beta_prior_cov <- diag(p)
  sigma2_prior_shape <- 0.01
  sigma2_prior_scale <- 0.01

  # Storage for samples
  beta_samples <- matrix(0, nrow = n_iter, ncol = p)
  sigma2_samples <- numeric(n_iter)

  # Initial values
  beta <- rep(0, p)
  sigma2 <- 1

  for (iter in 1:n_iter) {
    # Sample beta | sigma2, y
    beta_cov <- solve(t(X) %*% X / sigma2 + solve(beta_prior_cov))
    beta_mean <- beta_cov %*% (t(X) %*% y / sigma2)
    beta <- MASS::mvrnorm(1, beta_mean, beta_cov)

    # Sample sigma2 | beta, y
    resid <- y - X %*% beta
    sigma2_shape <- sigma2_prior_shape + n / 2
    sigma2_scale <- sigma2_prior_scale + sum(resid^2) / 2
    sigma2 <- 1 / stats::rgamma(1, shape = sigma2_shape, rate = sigma2_scale)

    # Store samples
    if (iter > burn_in) {
      beta_samples[iter, ] <- beta
      sigma2_samples[iter] <- sigma2
    }
  }

  list(beta_samples = beta_samples[(burn_in + 1):n_iter, ],
       sigma2_samples = sigma2_samples[(burn_in + 1):n_iter])
}
```

Data Preprocessing

in this first step, we will be using the nessecary libraries in order to load and preform accurate bayesian models in order get a better read of the data and see any simple observations

```
library(ggplot2)
data(mpg)
```

```
# Use all available features for prediction
data_mpg <- na.omit(mpg)
data_mpg$drv <- as.factor(data_mpg$drv)
data_mpg$fl <- as.factor(data_mpg$fl)
data_mpg$class <- as.factor(data_mpg$class)

X <- model.matrix(hwy ~ ., data = data_mpg)[, -1]
y <- data_mpg$hwy
n <- length(y)
```

Defining Bayesian Models using nessecary Functions

```
gibbs_sampler_brr <- function(X, y, n_iter = 1000, burn_in = 100, lambda = 0.1) {
  n <- nrow(X)
  p <- ncol(X)

  # Prior parameters
  beta_prior_mean <- rep(0, p)
  beta_prior_cov <- diag(p) / lambda
  sigma2_prior_shape <- 0.01
  sigma2_prior_scale <- 0.01

  # Storage for samples
  beta_samples <- matrix(0, nrow = n_iter, ncol = p)
  sigma2_samples <- numeric(n_iter)

  # Initial values
  beta <- rep(0, p)
  sigma2 <- 1

  for (iter in 1:n_iter) {
    # Sample beta | sigma2, y
    beta_cov <- solve(t(X) %*% X / sigma2 + solve(beta_prior_cov))
    beta_mean <- beta_cov %*% (t(X) %*% y / sigma2)
    beta <- MASS::mvrnorm(1, beta_mean, beta_cov)

    # Sample sigma2 | beta, y
    resid <- y - X %*% beta
    sigma2_shape <- sigma2_prior_shape + n / 2
    sigma2_scale <- sigma2_prior_scale + sum(resid^2) / 2
    sigma2 <- 1 / stats::rgamma(1, shape = sigma2_shape, rate = sigma2_scale)

    # Store samples
    beta_samples[iter, ] <- beta
    sigma2_samples[iter] <- sigma2
  }

  list(beta_samples = beta_samples[(burn_in + 1):n_iter, ],
       sigma2_samples = sigma2_samples[(burn_in + 1):n_iter])
}

# Example usage:
 brr_samples <- gibbs_sampler_brr(X, y)
 beta_samples_brr <- brr_samples$beta_samples
 sigma2_samples_brr <- brr_samples$sigma2_samples
```

```
gibbs_sampler_lasso <- function(X, y, n_iter = 1000, burn_in = 100, lambda = 1) {
  n <- nrow(X)
  p <- ncol(X)

  # Storage for samples
  beta_samples <- matrix(0, nrow = n_iter, ncol = p)
  sigma2_samples <- numeric(n_iter)
  lambda_samples <- matrix(0, nrow = n_iter, ncol = p)

  # Initial values
  beta <- rep(0, p)
  sigma2 <- 1
  lambda2 <- rep(lambda^2, p)

  for (iter in 1:n_iter) {
    # Sample beta | sigma2, lambda, y
    beta_cov <- solve(t(X) %*% X / sigma2 + diag(1 / lambda2))
    beta_mean <- beta_cov %*% (t(X) %*% y / sigma2)
    beta <- MASS::mvrnorm(1, beta_mean, beta_cov)

    # Sample sigma2 | beta, y
    resid <- y - X %*% beta
    sigma2_shape <- 0.01 + n / 2
    sigma2_scale <- 0.01 + sum(resid^2) / 2
    sigma2 <- 1 / stats::rgamma(1, shape = sigma2_shape, rate = sigma2_scale)

    # Sample lambda2 | beta
    lambda2 <- 1 / stats::rgamma(p, shape = 1, rate = beta^2 / (2 * sigma2) + 1 / la

    # Store samples
    beta_samples[iter, ] <- beta
    sigma2_samples[iter] <- sigma2
    lambda_samples[iter, ] <- sqrt(lambda2)
  }

  list(beta_samples = beta_samples[(burn_in + 1):n_iter, ],
       sigma2_samples = sigma2_samples[(burn_in + 1):n_iter],
       lambda_samples = lambda_samples[(burn_in + 1):n_iter, ])
}

# Function to tune lambda
tune_lambda <- function(X, y, lambdas, top_n = 4, n_iter = 1000, burn_in = 100) {
  best_lambda <- NULL
  min_diff <- Inf

  for (lambda in lambdas) {
    lasso_samples <- gibbs_sampler_lasso(X, y, n_iter, burn_in, lambda)
    beta_means <- colMeans(lasso_samples$beta_samples)
    significant <- abs(apply(lasso_samples$beta_samples, 2, quantile, probs = 0.025)
                   abs(apply(lasso_samples$beta_samples, 2, quantile, probs = 0.975)
    nonzero_count <- sum(significant)
```

```
    diff <- abs(nonzero_count - top_n)

    if (diff < min_diff) {
      min_diff <- diff
      best_lambda <- lambda
    }
  }

  best_lambda
}

# Example usage:
 lambdas <- seq(0.1, 1, by = 0.1)
 best_lambda <- tune_lambda(X, y, lambdas)
lasso_samples <- gibbs_sampler_lasso(X, y, lambda = best_lambda)
beta_samples_lasso <- lasso_samples$beta_samples
sigma2_samples_lasso <- lasso_samples$sigma2_samples
```

```
install.packages("caret")
library(caret)

# Function to calculate RMSE
calculate_rmse <- function(y_true, y_pred) {
  sqrt(mean((y_true - y_pred)^2))
}

# Function to perform cross-validation and compute RMSE for each model
cross_validate_models <- function(X, y, lambdas = seq(0.1, 1, by = 0.1)) {
  set.seed(123)
  folds <- createFolds(y, k = 5, list = TRUE, returnTrain = TRUE)

  blr_rmse <- numeric(5)
  brr_rmse <- numeric(5)
  lasso_rmse <- numeric(5)

  for (i in 1:5) {
    train_indices <- folds[[i]]
    test_indices <- setdiff(1:nrow(X), train_indices)

    X_train <- X[train_indices, ]
    y_train <- y[train_indices]
    X_test <- X[test_indices, ]
    y_test <- y[test_indices]

    # Bayesian Linear Regression
    blr_samples <- gibbs_sampler_blr(X_train, y_train)
    beta_blr <- colMeans(blr_samples$beta_samples)
    lower_blr <- apply(blr_samples$beta_samples, 2, quantile, probs = 0.025)
    upper_blr <- apply(blr_samples$beta_samples, 2, quantile, probs = 0.975)
    significant_blr <- which(sign(lower_blr) == sign(upper_blr))
    y_pred_blr <- X_test[, significant_blr] %*% beta_blr[significant_blr]
    blr_rmse[i] <- calculate_rmse(y_test, y_pred_blr)

    # Bayesian Ridge Regression
    brr_samples <- gibbs_sampler_brr(X_train, y_train)
    beta_brr <- colMeans(brr_samples$beta_samples)
    lower_brr <- apply(brr_samples$beta_samples, 2, quantile, probs = 0.025)
    upper_brr <- apply(brr_samples$beta_samples, 2, quantile, probs = 0.975)
    significant_brr <- which(sign(lower_brr) == sign(upper_brr))
    y_pred_brr <- X_test[, significant_brr] %*% beta_brr[significant_brr]
    brr_rmse[i] <- calculate_rmse(y_test, y_pred_brr)

    # Bayesian Lasso
    best_lambda <- 1  # Change to tune_lambda(X_train, y_train, lambdas) for automat
    lasso_samples <- gibbs_sampler_lasso(X_train, y_train, lambda = best_lambda)
    beta_lasso <- colMeans(lasso_samples$beta_samples)
    lower_lasso <- apply(lasso_samples$beta_samples, 2, quantile, probs = 0.025)
    upper_lasso <- apply(lasso_samples$beta_samples, 2, quantile, probs = 0.975)
    significant_lasso <- which(sign(lower_lasso) == sign(upper_lasso))
```

```
    y_pred_lasso <- X_test[, significant_lasso] %*% beta_lasso[significant_lasso]
    lasso_rmse[i] <- calculate_rmse(y_test, y_pred_lasso)
  }

  blr_rmse_mean <- mean(blr_rmse)
  brr_rmse_mean <- mean(brr_rmse)
  lasso_rmse_mean <- mean(lasso_rmse)

  results <- data.frame(
    Model = c("Bayesian Linear Regression", "Bayesian Ridge Regression", "Bayesian L
    RMSE = c(blr_rmse_mean, brr_rmse_mean, lasso_rmse_mean)
  )

  return(results)
}

# Example usage:
# Replace X and y with your actual data
lambdas <- seq(0.1, 2, by = 0.1)
 results <- cross_validate_models(X, y, lambdas)
print(results)
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

                       Model     RMSE
1 Bayesian Linear Regression 1.619512
2  Bayesian Ridge Regression 2.571378
3             Bayesian Lasso 2.341974
```

```
library(ggplot2)
library(MASS)  # For mvrnorm function (multivariate normal sampling)

# Load the mpg dataset
data(mpg)
# Assuming you want to predict highway mileage (hwy) using other variables

# Prepare the predictor matrix X and response vector y
X <- model.matrix(hwy ~ . - 1, data = mpg)  # Exclude intercept (column of 1s)
y <- mpg$hwy

# Function to simulate Bayesian samples for Linear Regression
gibbs_sampler_blr <- function(X, y, n_iter = 1000, burn_in = 100) {
  n <- nrow(X)
  p <- ncol(X)

  # Prior parameters
  beta_prior_mean <- rep(0, p)
  beta_prior_cov <- diag(1, p)
  sigma2_prior_shape <- 0.01
  sigma2_prior_scale <- 0.01

  # Storage for samples
  beta_samples <- matrix(0, nrow = n_iter, ncol = p)
  sigma2_samples <- numeric(n_iter)

  # Initial values
  beta <- rep(0, p)
  sigma2 <- 1

  for (iter in 1:n_iter) {
    # Sample beta | sigma2, y
    beta_cov <- solve(t(X) %*% X / sigma2 + solve(beta_prior_cov))
    beta_mean <- beta_cov %*% (t(X) %*% y / sigma2)
    beta <- mvrnorm(1, beta_mean, beta_cov)

    # Sample sigma2 | beta, y
    resid <- y - X %*% beta
    sigma2_shape <- sigma2_prior_shape + n / 2
    sigma2_scale <- sigma2_prior_scale + sum(resid^2) / 2
    sigma2 <- 1 / rgamma(1, shape = sigma2_shape, rate = sigma2_scale)

    # Store samples
    beta_samples[iter, ] <- beta
    sigma2_samples[iter] <- sigma2
  }

  list(beta_samples = beta_samples[(burn_in + 1):n_iter, ],
       sigma2_samples = sigma2_samples[(burn_in + 1):n_iter])
}
```

```
# Simulate Bayesian samples
blr_samples <- gibbs_sampler_blr(X, y)

# Compute summary statistics
blr_summary <- data.frame(
  Mean = colMeans(blr_samples$beta_samples),
  Median = apply(blr_samples$beta_samples, 2, median),
  Lower = apply(blr_samples$beta_samples, 2, quantile, probs = 0.025),
  Upper = apply(blr_samples$beta_samples, 2, quantile, probs = 0.975)
)

# Determine significance based on quantiles
blr_summary$Significant <- with(blr_summary, sign(Lower) == sign(Upper))

# Assign row names from original predictor matrix X
rownames(blr_summary) <- colnames(X)

# Print the summary
print(blr_summary)
```

```
modelrange rover          1.973228815      FALSE
modelsonata               1.352408614      FALSE
modeltiburon              1.193767430      FALSE
modeltoyota tacoma 4wd    0.874970596      FALSE
displ                     0.789024136      FALSE
year                      0.004681464       TRUE
cyl                       0.034125967      FALSE
transauto(l3)             0.167836474      FALSE
transauto(l4)             0.534645888      FALSE
transauto(l5)             1.472540430       TRUE
transauto(l6)             1.765908352      FALSE
transauto(s4)             0.791882383      FALSE
transauto(s5)             2.393793056       TRUE
transauto(s6)             1.099270079      FALSE
transmanual(m5)           0.706465442      FALSE
transmanual(m6)           1.139886769      FALSE
drvf                      2.453555816       TRUE
drvr                      2.380160583       TRUE
cty                       1.135830911       TRUE
fld                       1.727558942      FALSE
fle                      -0.326955072       TRUE
flp                      -0.043604713       TRUE
flr                       0.375161205      FALSE
classcompact              1.617217624      FALSE
classmidsize              1.944115516      FALSE
classminivan              1.442555431      FALSE
classpickup              -0.115276607       TRUE
classsubcompact           1.332211101      FALSE
classsuv                 -0.540152446       TRUE
```

```
# Assuming you have already defined gibbs_sampler_brr function and loaded the data

# Example: Bayesian Ridge Regression
brr_samples <- gibbs_sampler_brr(X, y)

# Compute summary statistics
brr_summary <- data.frame(
  Mean = colMeans(brr_samples$beta_samples),
  Median = apply(brr_samples$beta_samples, 2, median),
  Lower = apply(brr_samples$beta_samples, 2, quantile, probs = 0.025),
  Upper = apply(brr_samples$beta_samples, 2, quantile, probs = 0.975)
)

# Determine significance based on quantiles
brr_summary$Significant <- with(brr_summary, sign(Lower) == sign(Upper))

# Assign row names from original predictor matrix X
rownames(brr_summary) <- colnames(X)

# Print the summary
print(brr_summary)

# Example: Using significant variables for further analysis
significant_vars <- colnames(X)[brr_summary$Significant == TRUE]

# Example: Fit a linear model using significant variables
lm_model <- lm(y ~ X[, significant_vars] - 1)

# Example: Print summary of the linear model
print(summary(lm_model))
```

```
drvf                          FALSE
drvr                          FALSE
cty                            TRUE
fld                           FALSE
fle                            TRUE
flp                            TRUE
flr                            TRUE
classcompact                  FALSE
classmidsize                  FALSE
classminivan                  FALSE
classpickup                   FALSE
classsubcompact               FALSE
classsuv                      FALSE

Call:
lm(formula = y ~ X[, significant_vars] - 1)

Residuals:
    Min      1Q  Median      3Q     Max 
-5.1740 -1.1248  0.0026  1.2076  4.2784 

Coefficients:
                                     Estimate Std. Error t value Pr(>|t|)    
X[, significant_vars]year          -0.0007036  0.0009521  -0.739   0.4607    
X[, significant_vars]cyl            0.1347461  0.1204589   1.119   0.2645    
X[, significant_vars]transauto(s5)  1.0583256  0.9614264   1.101   0.2722    
X[, significant_vars]cty            1.3715577  0.0495210  27.696   <2e-16 ***
X[, significant_vars]fle            0.2458309  1.0448431   0.235   0.8142    
X[, significant_vars]flp            2.0413210  0.7889931   2.587   0.0103 *  
X[, significant_vars]flr            0.6378365  0.7684117   0.830   0.4074    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.652 on 227 degrees of freedom
Multiple R-squared:  0.9955,	Adjusted R-squared:  0.9953 
F-statistic:  7129 on 7 and 227 DF,  p-value: < 2.2e-16
```

Start coding or generate with AI.

```
# Define a sequence of lambda values to tune
lambdas <- seq(0.1, 5, by = 0.1)

# Tune lambda using the provided function
best_lambda <- tune_lambda(X, y, lambdas)

# Run the Gibbs sampler for Lasso with the best lambda
lasso_samples <- gibbs_sampler_lasso(X, y, lambda = best_lambda)

# Compute summary statistics
lasso_summary <- data.frame(
  Mean = colMeans(lasso_samples$beta_samples),
  Median = apply(lasso_samples$beta_samples, 2, median),
  Lower = apply(lasso_samples$beta_samples, 2, quantile, probs = 0.025),
  Upper = apply(lasso_samples$beta_samples, 2, quantile, probs = 0.975)
)

# Determine significance based on quantiles
lasso_summary$Significant <- with(lasso_summary, sign(Lower) == sign(Upper))

# Assign row names from original predictor matrix X
rownames(lasso_summary) <- colnames(X)

# Print the summary
print(lasso_summary)
```

```
                              Mean       Median         Lower       Upper
manufactureraudi        -0.25540091 -0.447475552 -1.519226e+01 14.83297220
manufacturerchevrolet   -0.58203925 -0.455841481 -1.439768e+01 13.64268818
manufacturerdodge       -1.35579803 -1.375067214 -1.528053e+01 12.71971686
manufacturerford        -1.04646340 -1.472294621 -1.391126e+01 12.97177195
manufacturerhonda       -0.77846678 -0.464640357 -2.875827e+01 25.50546047
```