**Bayesian Linear Regression** 2024-11-15 Bayesian Linear Regression: Comprehensive Analysis of Principles, Hyperparameter Tuning, and Model Performance In statistical modeling, linear regression is a fundamental method for predicting continuous outcomes based on one or more predictors. While classical linear regression provides point estimates of parameters, it lacks the capacity to capture uncertainty in these estimates, which is ofter critical for robust predictions and informed decision-making. Bayesian linear regression addresses this limitation by incorporating prior beliefs and updating them with observed data via Bayes' theorem. Unlike traditional approaches, it yields posterior distributions for parameters, enabling uncertainty quantification and probabilistic interpretation This report is aim to explores Bayesian linear regression with a focus on its application to the Boston Housing Dataset. Key objectives include analyzing feature impacts on housing prices, making uncertainty-aware predictions, and investigating the effects of hyperparameters and training Load Packages The purpose of this step is to load the necessary R packages for conducting Bayesian Linear Regression. If any of these packages are not installed, the code will automatically install them. # Install packages if not already installed
if (!requireNamespace("mutnorm", quietly - TRUE)) install.packages("mutnorm
if (!requireNamespace("dplyr", quietly - TRUE)) install.packages("dplyr")
if (!requireNamespace("gplot2", quietly - TRUE)) install.packages("gplot2
if (!requireNamespace("reshape2", quietly - TRUE)) install.packages("reshape2") Section 1: Bayesian Linear Regression Section 1: Bayesian Linear Regression and The purpose of this section is to implement Bayesian Linear Regression using Glibbs Sampling, a Markov Chain Monte Carlo method for drawing samples from the posterior distributions of model parameters. It begins by validating the input dimensions to ensure consistency with the Bayesian model. The process involves precomputing key matrices for efficiency, initializing storage for posterior samples, and setting initial values for the regression coefficients ( $\beta$ ) and error variance ( $\sigma^2$ ). Glibbs sampling alternates between sampling  $\beta$  from a multivariate normal posterior and  $\sigma^2$  from an inverse-gamma posterior, capturing the uncertainty in parameter estimates. After discarding the burn-in samples, the function computes summary statistics, including the posterior means and covariance of  $\beta$  and  $\sigma^2$ , providing a comprehensive representation of the posterior distributions. The output is a structured list containing the sampled values and these summary metrics, facilitating further analysis of the Bayesian linear regression model. # Load the necessary package "mwtnorm " # for multivariate normal distribution sampling library(mwtnorm) # Bayesian Linear Regression using Gibbs Sampling
bayesian\_linear\_regression <- function(X, y, beta0, V0, a0, b0, n\_iter, burn\_in) {</pre> # to prevent dimension mismatches.
if (ncol(X) != length(beta0)) {
 stop("Dimensions\_of\_X\_and\_beta0\_do\_not\_match.") if (nrow(V0) != ncol(V0) || ncol(V0) != ncol(X)) {
 stop("Dimensions\_of\_V0\_do\_not\_match\_with\_X\_or\_are\_not\_square.") #### Step 3: Precompute Matrices ####

# In this step, we precompute matrices used repeatedly in the Gibbs sampling

# process to save computation time.

XXX <- crossprod(X) # Compute the predictor covariance matrix

XXY <- crossprod(X, y) # Compute the predictor sand response

VO\_inv <- solve(VO) # Inverse of the prior covariance matrix for beta

betaO\_VO\_inv <- VO\_inv %+% betaO # Combines prior mean and covariance for efficient calculations #### Step 4: Initialize Storage for Posterior Samples ####
# In this step we create matrices to store the sampled beta and
# sigma^2 values for later analysis.
beta\_ample <- matrix(NA, nrow = n\_iter, ncol = p) # To store sampled beta values
sigma2\_samples <- numeric(n\_iter) # To store sampled sigma^2 values</pre> #### Step 5: Set Initial Values for Beta and Sigma^2 ####
# In this step, we start the Gibbs sampling process with
# reasonable initial values.
beta\_current <- beta # Initialize beta as the prior mean
sigma2\_current <- 1 # Initialize sigma^2 with a default value</pre> #### Step 6: Gibbs Sampling Iterations ####
# In this step, we alternate between sampling beta and sigma^2 from their
f conditional posterior distributions.
for (iter in lin\_iter) { # Compute the posterior precision matrix (inverse of posterior covariance).
V.N.inv <- V0\_inv + xtx / sigma2\_current
# Compute the posterior covariance matrix for beta.
V.N.<- solve(V.N.inv)
# Compute the posterior mean for beta.
beta\_N <- V\_N % % (beta\_V0\_inv) + xty / sigma2\_current)
# Sample beta from the posterior distribution (multivariate normal).
beta\_current <- as.numeric(rmvnorm(1, mean - beta\_N, sigma - V\_N))</pre> ## Store the sampled values for beta and sigma^2 ##
# Save the current samples of beta and sigma^2 for later analysis.
beta\_samples[iter] <- beta\_current
sigma2\_samples[iter] <- sigma2\_current</pre> # means and covariance.
beta\_mean <- colMeans(beta\_samples) # Posterior mean of beta samples
sigma2\_mean <- mean(sigma2\_samples) # Posterior mean of sigma'2 samples
beta\_cov <- cov(beta\_samples) # Posterior Covariance matrix of beta samples</pre> return(list(
beta\_samples - beta\_samples, # All sampled beta values
sigma2\_samples - sigma2\_samples, # All sampled sigma^2 values
beta\_mean - beta\_mean, # Mean of the posterior beta samples
sigma2\_mean - sigma2\_mean, # Mean of the posterior sigma^2 samples
beta\_cov - beta\_cov # Covariance matrix of the posterior beta samples The purpose of this section is to implement a prediction function for Bayesian Linear Regression, leveraging posterior samples of regression coefficients ( $\beta$ ) and residual variance ( $\sigma^{-}$ ) to generate predictions for new data points. It begins by determining the dimensions of the posterior samples and the new design matrix ( $X_{\text{inve}}$ ), ensuring proper alignment. Predictions are generated iteratively for each posterior sample by combining sampled coefficients and variance with the design matrix, capturing the uncertainty in predictions. The function then calculates summary statistics, including the mean and 95% credible intervals, to provide both point estimates and uncertainty quantificant for each prediction. The output is a data frame containing the predictive mean and credible intervals, offering a clear representation of the posterior predictive distribution. #### Step 1: Get Dimensions of Inputs ####
# In this step, we determine the number of posterior samples and the
# number of new data points.
n\_mamples <- nrow(beta\_mamples) # Number of posterior samples
n\_new <- nrow(X\_new) # Number of new data points to predict</pre> #### Step 2: Initialize Storage for Predictions ####

# In this step, we create a matrix to store the predicted values for
# all posterior samples.
# Each row corresponds to predictions for one posterior sample,
# and each column corresponds to a new data point.
y\_pred\_samples <- matrix(NA, nrow - n\_samples, ncol - n\_new) #### Step 3: Generate Predictive Values ####

# For each posterior sample of beta and sigma^2, we generate predict
for (i in 1:n\_samples) {
# Extract the 1-th posterior sample of beta coefficients.
beta\_i <- beta\_samples[i, ]
# Extract the i-th posterior sample of sigma^2 (residual variance)
sigma2\_i <- sigma2\_samples[i] # Predictions are calculated as the linear model plus random noise: y\_pred\_samples[i, ] <- X\_new %\*% beta\_i + rnorm(n\_new, mean - 0, sd - sqrt(sigma2\_i))</pre> # Compute the posterior predictive mean for each new data point.
pred\_mean <- colMeans(y\_pred\_samples) # Mean across all posterior samples</pre> # Compute the 97.5th percentile (upper bound of 95% credible interval)
# for each prediction.
pred\_upper <- apply(y\_pred\_samples, 2, quantile, probs - 0.975)</pre> ### Step 5: Return Predictions ###
# In this step, we return a data frame containing the predictive mean and
# 938 credible intervals for each new data point.
return(data.frame(
 pred\_mean - pred\_mean,
 pred\_lower - pred\_lower,
 pred\_upper Section 2: Data Loading and Preprocessing The purpose of this section is to prepare the Boston Housing dataset for Bayesian Linear Regression. The dataset is loaded from a CSV file and previewed to inspect its structure, data types, and content, ensuring it is correctly imported. Missing values are identified and addressed using mean imputation, where each missing value is replaced with the mean of its respective column, ensuring a complete and clean dataset. After cleaning, the data is split into training, validation, and test sets with proportions of 80%, 10%, and 10%, respectively, to enable robust model training and evaluation. The Boston Housing dataset is selected because it provides a well-known benchmark for regression tasks, with features that capture various aspects of housing data, such as socioeconomic and environmental factors. This dataset's size and feature diversity make it ideal for demonstrating Bayesian Linear Regression, as it allows exploration of uncertainty in parameter estimates and predictions, which are central to Bayesian approaches. # Load the necessary package "dplyr" # for data manipulation functions. library(dplyr) ##
## Attaching package: 'dplyr' ## The following objects are masked from 'package:stats': ## The following objects are masked from 'package:base': ## ## intersect, setdiff, setequal, union ### Step 1: Load the Dataset ####
# In this step, we load the Boston Housing dataset from a CSV file.
Boston <- read.csv("BostonHousing\_data.csv")</pre> #### Step 2: Check for Missing Values #### # In this step, we calculate the total numb missing\_count <- sum(is.na(Boston)) if (missing\_count -- 0) {
 cat("The dataset has no missing values.\n") cat("Ine usease ""
} else {
 cat("The dataset has", missing\_count, "missing values.\n") ## The dataset has 5 missing values. # For each column, replace NA values with the mean of that column.
Boston <- Boston %>% mutate\_all(~ ifelse(is.na(.), mean(., na.rm - TRUE), .)) #### Step 5: Inspect the Data #### # In this step, we print the first few rows of the dataset to examine its structure # and ensure it is ready for further analysis. head(Boston) #### Step 1: Recalculate Missing Values After Imputation ####
# In this step, we confirm that all missing values have been handled.
# After performing mean imputation, it's important to verify that
# there are no remaining missing values in the dataset.
issing\_count <- sum(is.na(Boston))
cat("After cleaning, the dataset has", missing\_count, "missing values.\n")</pre> ## After cleaning, the dataset has 5 missing values. #### Step 2: Determine the Total Number of Observations ####
# In this step, we obtain the total number of data points for splitting.
# Get the total number of observations (rows) in the dataset.
n <- nrow(Roston)</pre> #### Step 3: Define Proportions for Data Splitting ### # In this step, we set the suitable proportions for each dataset train\_prop <- 0.8 # 80% for training validation.prop <- 0.1 # 10% for validation test\_prop <- 0.1 # 10% for testing Ensure the proportions sum to 1.

tal\_prop <- train\_prop + validation\_prop + test\_prop

( total\_prop != 1) {

stop(\*The sum of the split proportions must equal 1.\*) set.seed(123) # Calculate the number of observations for the training set.  $train_{n}size \leftarrow floor(train_{n}prop * n) \\ \neq Calculate the number of observations for the validation set validation_size \leftarrow floor(validation_prop * n)$ # Assign the remaining observations to the test set test\_size <- n - train\_size - validation\_size # preventing any order bias.
shuffled\_indices <- sample(n) # Create a vector of shuffled row indices</pre> # Assign indices for the training set.
train\_indices <- shuffled\_indices[1:train\_size]</pre> validation\_indices <- shuffled\_indices[(train\_size + 1):(train\_size + validation\_size)]</pre> test\_indices <- shuffled\_indices[(train\_size + validation\_size + 1):n] # Create the training set.
train\_data <- Boston[train\_indices, ]</pre> # Create the validation set.
validation\_data <- Boston[validation\_indices, ]</pre> # Create the test set.
test\_data <- Boston[test\_indices, ] # Print the sizes of each set
cat("Training set size:", nrow(train\_data), "\n") ## Training set size: 404 cat("Validation set size:", nrow(validation\_data), "\n") ## Validation set size: 50 cat("Test set size:", nrow(test\_data), "\n") ## Test set size: 52 The purpose of this section is to perform Bayesian Linear Regression to estimate the impact of each feature on the response variable. The training, validation, and test datasets are constructed by separating predictors and the response variable while adding an intercept term. Prior distributions are defined with specified mean, covariance, and hyperparameters for the coefficients and residual variance. Gibbs Sampling is then executed to draw posterior samples, and the posterior means and 95% credible intervals for each regression coefficient are computed to quantify their impact. These results are summarized in a table for clear interpretation of the feature effects. # Select all columns except 'medv' as predictors and convert to matrix format.
X\_train <- train\_data %>% dplyr::select(-medv) # exclude 'medv' since it's the target variable.
X\_train <- as.matrix(X\_train)
# Set the response variable as the 'medv' column.
Y\_train <- train\_data%medv
# Add an intercept term to the predictors allows the model to learn a bias term.
X\_train <- cbind(Intercept - 1, X\_train)</pre> X\_validation <= as.matrix(x\_validation)
# Add an intercept term.
X\_validation <= cbind(Intercept = 1, X\_validation)</pre> # Set the response variable.
y\_validation <- validation\_data\$medv</pre> # Select predictors and convert to matrix fo.
X\_test <- test\_data %>% dplyr::select(-medv)
X\_test <- as.matrix(X\_test)</pre> # Add an intercept term.
X\_test <- cbind(Intercept = 1, X\_test)</pre> #### Step 2: Set Prior Parameters for Bayesian Linear Regression ####
# In this step, we define the prior distributions for the model parame
# Number of predictors (including the intercept) # Number of precurence
p <- ncol(X\_train)
# Prior mean vector for regression coefficients (initialized to zeros)</pre> beta0 <- rep(0, p)

Defor variance determined from previous analysis. # Prior variance determined from previous analysis.
tau2 <- 10
# Prior covariance matrix (scaled identity matrix).
VO <- diag(tau2, p)
# Prior shape parameter for the inverse-gamma distribution
a0 <- 0.01 # Prior scale parameter for the inverse-gamma distribution b0 <- 0.01 #### Step 3: Specify Gibbs Sampling Parameters ####
# In this step, we define the sampling parameters for the Gibbs sampler.
# Total number of iterations for Gibbs sampling.
n\_iter <- 5000
# Number of initial samples to discard (burn-in period).
burn\_in <- 1000</pre> Coefficient Mean Lower Upper Intercept 8.384326796 2.914380257 14.0464547667 crim -0.090719917 -0.161440459 -0.0224730401 ro. 0.052041992 0.02132925 0.0843211767 indus -0.015550637 -0.155130792 0.1288457513 chas 2.98442344 1.28855963 4.7994189881 nox -2.381771602 -7.400037089 2.3641946108 rm 5.100699313 4.380822042 5.3442944333 age 0.003622849 -0.025829234 0.0330945615 dis -0.999619143 -1.415026574 -0.566950194 0.61090413 0.3489958958 1 tax -0.009110066 -0.018727149 -0.0099417213 2 ptratio -0.50247175 -0.762722306 -0.247885175 -0.762723206 -0.247885175 -0.76272306 -0.24785175 -0.76272306 -0.247885175 -0.76272306 -0.247885175 -0.76272306 -0.247885175 -0.76272306 -0.247885175 -0.76272306 -0.247885175 -0.76272306 -0.247885175 -0.76272306 -0.247885175 -0.76272306 -ggtitle("Posterior Means and 95% Credible Intervals of Regression Coefficients") + Posterior Means and 95% Credible Intervals of Regression Coefficients tax indus crim -Istat -The purpose of this section is to apply the Bayesian Linear Regression prediction function to generate predictions for the test set, combining them Ine purpose or this section is to apply the sayesian Linear Regiression prediction function to generate predictions for the test set, combining them with actual values for evaluation. The predicted mean values and 95% credible intervals are computed for each test data point to reflect uncertainty in the predictions. Residuals are calculated to measure prediction errors, providing insights into the model's performance and areas for potential refinement. These results are consolidated into a data frame alongside the actual test values, enabling a comparison of predictions to ground truth. Additionally, #Set a Random Seed for Reproducibility ####
set.seed(123) predict\_bayesian\_lr <- function(beta\_samples, sigma2\_samples, X\_new) {</pre> # Determine the number of posterior samples and new data points.
n\_samples <- nrow(beta\_samples) # Number of posterior samples
n\_new <- nrow(X\_new) # Number of new data points to predict</pre> # Loop through each posterior sample to generate predictions for (i in 1:n\_samples) { # Extract the i-th posterior sample of beta coefficients. beta\_i <- beta\_samples[i, ] # Extract the i-th posterior sample of sigma^2 (residual variance).sigma2\_i <- sigma2\_samples[i] # Generate predictive values for the new data points.
y\_pred\_samples[i, ] <- X\_new %\*% beta\_i + rnorm(n\_new, mean = 0, sd = sqrt(sigma2\_i))</pre> # Calculate the mean and 95% credible intervals for the predictions.
pred\_mean <- colMeans(y\_pred\_mamples) # Mean predicted value across posterior samples
pred\_lower <- apply(y\_pred\_samples, 2, quantile, probs = 0.025) # Lower bound (2.5th percentile)
pred\_upper <- apply(y\_pred\_samples, 2, quantile, probs = 0.975) # Upper bound (97.5th percentile)</pre> predictions <- predict\_bayesian\_lr(beta\_samples, sigma2\_samples, X\_test) # Residuals provide insignt into the prediction errors
prediction\_results\$Residuals <- prediction\_results\$Predicted library(dplyr) cat("\nSignificant predictors at 95% credible interval:\n") ##
## Significant predictors at 95% credible interval: print(significant\_predictors\$Coefficient) ## [1] "Intercept" "crim" ## [7] "rad" "tax" "zn" "chas" "rm"
"ptratio" "b" "lstat" # Plot actual vs predicted values with credible intervals
plot\_actual\_vs\_predicted <- gaplot(prediction\_results, aes(x = Actual, y = Predicted)) +
geom\_point(color = "blue") +</pre> geom\_point(color = "blue") + geom\_point(color = "gray") + geom\_arbnia(sa(ymin = Lower, ymax = Upper), width = 0.2, color = "gray") + geom\_abline(slope = 1, intercept = 0, linetype = "dashed", color = "red") + geom\_abline(slope = 1, intercept = 0, linetype = theme\_bw() + ggtifle("Predicted vs Actual Housing Prices") + xlab("Actual MEDV") + ylab("Predicted MEDV")+ theme( plot.title = element\_text(hjust = 0.4) ) Predicted vs Actual Housing Prices # Load the packa
library(ggplot2) # Flot residuals vs predicted values with adjusted y-axis
plot\_residuals\_vs\_predicted <- gqplot(prediction\_results, aes(x = Predicted, y = Residuals)) +
geom\_point(color = "darkgreen") +
geom\_pline(yintercept = 0, linetype = "dashed") +
theme\_bw() +
gqttle("Residuals vs Predicted Values") +
xlab("Predicted MEDV") +
ylab("Residuals") +
ylab("Residuals") +
theme(</pre> plot.title - element\_text(hjust - 0.4) Residuals vs Predicted Values Predicted MEDV # Load the package 'ggplot2'
library(ggplot2) # Histogram of residuals with adjusted x-axis
plot\_residuals\_histogram <- gaplot(prediction\_results, aes(x - Residuals)) +
geom\_histogram(binwidth - 1, fill - "orange", color - "black", alpha - 0.7) +
theme\_bw() +
ggtitle("Distribution of Residuals") +
xlab("Residuals") +
ylab("Frequency") +
theme(</pre> plot.title = element\_text(hjust = 0.4)
) print (plot\_residuals\_histogram) Distribution of Residuals Section 3: Performance Evaluation of the Bayesian Linear Regression Model The purpose of this section is to evaluate the performance of a Bayesian Linear Regression model on the Boston Housing dataset by calculating the RMSE for training, validation, and test sets across varying training set proportions. Firstly, we define the feature matrix and target vector, adding an intercept term for the regression model. Training set proportions are specified, and the dataset is randomly shuffled and split into training, validation, and test subsets. For each training set proportion, Bayesian Linear Regression is performed with specified prior parameters, and the posterior mean of coefficients is used to make predictions on all subsets. The RMSE is computed for each subset to assess model performance. Finally, we combine the RMSE results a data frame. # Set the response variable as the "medu" column.
y <= BoatonSmedu # Target variable (median house values)
# Add an intercept term to the feature matrix.
X <= cbind(intercept = 1, X) # Adds a column of ones for the inter</pre> #### Step 2: Define Training Set Proportions #### # In this step, we specify different proportions of data to be used for trainintrain\_proportions <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8) # Training set sizes rmse\_train <- numeric(length(train\_proportions)) # Stores RMSE for training data
rmse\_val <- numeric(length(train\_proportions)) # Stores RMSE for validation data
rmse\_test <- numeric(length(train\_proportions)) # Stores RMSE for test data # Randomly shuffle indices to randomize data splitting.
n\_total <- nrow(X) # Total number of observations in the dataset
indices <- sample(1:n\_total) # Shuffled indices.</pre> #### Step 4: Loop Over Different Training Set Sizes #### # In this step, we evaluate model performance for each tr for (i in seq\_along(train\_proportions)) { # Assign indices to each set based on calculated sizes.
train\_indices <- indices[in\_train] # Indices for training set
val\_indices <- indices[in\_train + 1].(in\_train + n\_val)] # Indices for validation set
test\_indices <- indices[(n\_train + n\_val + 1):n\_total] # Indices for test set</pre> # Training set.
X\_train <- X[train\_indices, ] # Predictor matrix for training
y\_train <- y[train\_indices] # Target vector for training se</pre> # validation set.
X\_val <- X[val\_indices, ] # Predictor matrix for validation set
y\_val <- y[val\_indices] # Target vector for validation set</pre> # Test set.
X\_test <- X[test\_indices, ] # Predictor matrix for test set
y\_test <- y[test\_indices] # Target vector for test set</pre> # Set parameters for the Gibbs sampling algorithm.
n\_iter <- 5000 # Total number of iterations for Gibbs sampling
burn\_in <- 1000 # Number of initial samples to discard (burn-in period)</pre> # Estimate regression coefficients using posterior means.
beta\_posterior\_mean <- colMeans(results\$beta\_samples) # Posterior mean estimates | # Posterior mean est \* Predictions on training data.
y\_pred\_train <- X\_train %\*% beta\_posterior\_mean # Predicted values for training set
rmse\_train[i] <- sqrt(mean((y\_train - y\_pred\_train)^2)) # RMSE for training data # Predictions on validation data.
y\_pred\_val <- X\_val %\*% beta\_posterior\_mean # Predicted values for validation
rmse\_val[i] <- sqrt(mean((y\_val - y\_pred\_val)^2)) # RMSE for validation data</pre> # Predictions on test data.
y\_pred\_test <- X\_test %\*% beta\_posterior\_mean # Predicted values for test set
rmse\_test[1] <- sqrt(mean(y\_test - y\_pred\_test)^2]) # RMSE for test data</pre> #### Step 5: Combine RMSE Results into a Data Frame ####
# In this step, we organize RMSE results for further analysis and plotting.

Tmme\_results <- data.frame(
TrainingProportion - train\_proportions \* 100, # Convert proportions to per

RMSE\_train = Tmme\_train, # RMSE values for training data

RMSE\_valuation = Tmme\_val, # RMSE values for validation data

RMSE\_train = Tmme\_test # RMSE values for test data print(rmse\_results) # Display the RMSE results for different training proportions ## TrainingProportion RMSE\_Train RMSE\_Validation RMSE\_Test
## 1 20 4.365591 4.880260 5.494768
## 2 30 4.225963 5.49260 4.924971
## 4 50 4.275963 5.49260 4.924971
## 4 50 4.427996 5.488652 4.951604
## 5 6 70 4.699030 4.577155 5.027338
## 7 8 80 4.655953 4.442790 5.245174 # Find the optimal training set size
optimal\_index <- which.min(mme\_val) # Index of minimum RMSE in validation set
optimal\_min\_size <- train\_proportions[optimal\_index] \* 100 # Convert to percentage
optimal\_mmse <- rmse\_val[optimal\_index] # Corresponding validation RMSE</pre> # Output the results
cat("Optimal Training Size (%):", optimal\_train\_size, "\n") ## Optimal Training Size (%): 80 cat("Validation RMSE at Optimal Training Size:", optimal\_rmse, "\n") ## Validation RMSE at Optimal Training Size: 4.44279# Create RMSE data for plotting with more descriptive data types
rmse\_data <- data.frame(
 Training@etSize - rep(train\_proportions \* 100, 2),
 RMSE - c(rmse\_train, rmse\_val),
 Type - rep(c("Train RMSE (Train Size)", "Validation RMSE (Train Size)"), each - length(train\_proportions))</pre> \* Flot RMSE for Training and Validation

rmse\_plot <- ggplot(rmse\_data, aes(x - TrainingSetSize, y - RMSE, color - Type)) +

geom\_point(size - 3) +

labs( ubs(
title - "Impact of Training Set Size on RMSE for Bayesian Linear Regression",
x - "Training Set Size (%)",
y - "RMSE",
color - "Data Type" ) + theme\_bw() Impact of Training Set Size on RMSE for Bayesian Linear Regression Data Type Train RMSE (Train Size) Training Set Size (%) Section 4: Exploring Model Performance by Varying a Hyperparameter The purpose of this section is to evaluate the impact of different prior variances ( $\tau^2$ ) on the performance of the Bayesian Linear Regression model by computing RMSE for both training and validation datasets. The feature matrix and target vector are first defined, with an intercept term added to the predictors. Using the previously determined optimal training set size of 80%, the data is split into training, validation, and test sets. For each value of  $\tau^2$ , Bayesian Linear Regression is performed with corresponding prior parameters, and the posterior mean of the coefficients is used to make predictions. The RMSE is calculated for both the training and validation sets to assess the model's fitting ability and generalization performance. Finally, the results are compiled into a data frame for comparison, providing insights into how the choice of prior variance affects the model's fitting ability and generalization. model's performance. library(dplyr) # Set the response variable as the "medo" column.
y <- BostonSmedot # Target variable (median house values)
# Add an intercept term to the feature matrix.
X <- cbind(Intercept - 1, X) # Adds a column of ones for the intercept term.</pre> # test in the model.
tau\_squared\_values <- c(0.01, 0.1, 1, 10, 100, 1000) # Different values of tau\_squared\_values</pre> #### Step 3: Initialize Vectors to Store RMSE Values ####
# In this step, we prepare storage for RMSE values computed for each prior variance
rmse\_train\_hyper <- numeric(length(tau\_squared\_values)) # Store RMSE for training data
rmse\_val\_hyper <- numeric(length(tau\_squared\_values)) # Store RMSE for validation da</pre> # Total number of samples.
n\_total <- nrow(X)</pre> # Randomly shuffle indices for splitting data.
indices <- sample(1:n\_total)</pre> # Split data into training, validation, and test sets.
train\_indices <- indices[:n\_train]
val\_indices <- indices[(n\_train + 1):(n\_train + n\_val)]
test\_indices <- indices[(n\_train + n\_val + i):n\_total]</pre> #### Step 5: Loop Over Different Prior Variances (tau\_squared) ####
# In this step, we evaluate model performance for each tau\_squared value
for (i in seq\_along(tau\_squared\_values)) {
 tau\_squared <- tau\_squared\_values[i] # Current tau\_squared value</pre> # Define prior distributions for model parameters.

beta0 <- rep(0, ncol(X)) # Prior mean vector for regression coefficients
V0 <- diag(tau\_squared, ncol(X)) # Prior covariance matrix scaled by tau\_squared
a0 <- 0.01 # Shape parameter for inverse-gamma prior on variance
b0 <- 0.01 # Scale parameter for inverse-gamma prior on variance # Estimate regression coefficients using posterior means.
beta\_posterior\_mean <- colMeans(results\$beta\_samples) # Posterior mean estimates \*\*Predictions on training data
y\_pred\_train <- X\_train %\*\* beta\_posterior\_mean # Predicted values for training data.
rmse\_train\_hyper[i] <- sqrt(mean((y\_train - y\_pred\_train)^2)) # RMSE for training data # Predictions on validation data
y\_pred\_val <- X\_val %\*% beta\_posterior\_mean # Predicted values for validation data
rmse\_val\_hyper[i] <- sqrt(mean((y\_val - y\_pred\_val)\*2)) # Validation RMSE</pre> ## 5 1e+02 4.679821 ## 6 1e+03 4.655981 4.444334 # Identify the optimal TauSquared value based on validation RMSE
optimal\_index <- which.min(rmse\_val\_hyper) # Index of the minimum validation RMSE
optimal\_quayquared <- tau\_qquared\_values[optimal\_index] # Optimal TauSquared value
optimal\_rmse <- rmse\_val\_hyper[optimal\_index] # Minimum validation RMSE</pre> # Output the optimal TauSquared value and corresponding validation RMSE cat("Optimal TauSquared Value:", optimal\_tau\_squared, "\n") ## Optimal TauSquared Value: 10 cat("Validation RMSE at Optimal TauSquared Value:", optimal\_rmse, "\n") ## Validation RMSE at Optimal TauSquared Value: 4.307508 # Load necessary
library(reshape2)
library(dplyr)
library(ggplot2) \* Flot RMSE vs. TauSquared for Training and Validation using ggplot2 tau\_squared\_rmse\_plot <- ggplot( rmse hover -ma'>-- 1 rmse\_hyper\_melted,
aes(x - log10(TauSquared), y - RMSE, color - Dataset) title - "Impact of Prior Variance (TauSquared) on RMSE for Bayesian Linear Regression", # Flot title x - "Log10(TauSquared)", y = "RMSE", color = "Data Type" ) + theme\_bw() Impact of Prior Variance (TauSquared) on RMSE for Bayesian Linear Regression Data Type Train RMSE (TauSquared) Log10(TauSquared) section 5: Further Exploring Model Performance The purpose of this section is to evaluate the performance of a Bayesian Linear Regression model on a test set using an optimal prior variance ( Ine purpose or this section is to evaluate the performance of a Bayesian Linear Regression model on a test set using an optimal prior variance (  $\tau^2=10$ ) determined from prior analysis. The process begins by merging the training and validation sets to ensure the model is trained on all available data except the test set, enhancing robustness. The prior parameters, including a zero-initialized mean vector for  $\beta$  and a diagonal covariance matrix scaled by the optimal  $\tau^2$ , are defined alongside inverse-gamma hyperparameters for  $\sigma^2$ . Gibbs sampling is then performed to estimate the posterior distributions of  $\beta$  and  $\sigma^2$ , with a burn-in period to discard initial samples and ensure convergence. The posterior mean of  $\beta$  is extracted to make predictions on the test set, and the model's accuracy is quantified using the Root Mean Square Error (RMSE), which reflects the average deviation of predicted values from the true test values. The test RMSE is finally printed to summarize the model's predictive performance. # Combine rows of training and validat
X\_train\_full <- rbind(X\_train, X\_val)</pre> # Combine responses from training and validation sets y\_train\_full <- c(y\_train, y\_val) # Prior mean vector for regression coefficients (initialized to zeros). beta0 <- rep(0, ncol(X)) # Prior covariance matrix for beta with optimal tau\_squared = 10 VO <- diag(10, ncol(X)) #### Step 4: Retrain the Model on the Combined Training and Validation Sets ####
# In this step, we fit the Bayesian Linear Regression model using
# the combined dataset.
results\_final <- bayesian\_linear\_regression(
 X\_train\_full, y\_train\_full, beta0, VO, a0, b0, # Input data and prior parameters
 n\_iter - n\_iter, burn\_in = burn\_in # Sampling parameters
}</pre> # We compute the posterior mean of the regression coefficients.
beta\_posterior\_mean\_final <- colMeans(results\_final\$beta\_samples) # on the test set.
y\_pred\_test <- X\_test %\*% beta\_posterior\_mean\_final</pre> #### Step 7: Calculate RMSE on the Test Set ####
# In this setp, we compute the Root Mean Squared Error (
# evaluate model performance.
rmse\_test\_final <- sqrt(mean((v\_test - y\_pred\_test)^2))</pre> ## Test RMSE with optimal tau\_squared (10): 5.456036 The purpose of this section is to evaluate the impact of different prior variances  $(\tau^2)$  on the performance of a Bayesian Linear Regression model using the Boston dataset. The feature matrix (X) and target vector (y) are defined, with an intercept term added to X. Various  $\tau^2$  values are specified, and training and validation datasets are merged to maximize the data available for model training. For each  $\tau^2$ , the model is trained using Gibbs sampling to estimate the posterior distributions of  $\beta$  and  $\sigma^2$ . The posterior mean of  $\beta$  is used to make predictions on the test set, and the Root Mean Square Error (RMSE) is calculated to measure prediction accuracy. Finally, create a data frame to summarize the test RMSE values for each  $\tau^2$ , providing insights into how the choice of prior variance affects model performance. # The first step is define the feature matrix (X) and target vector (y) from the Boston data
X <- as.matrix(Boston %>% dplyr::select(-medv)) # Select predictors and convert to matrix
y <- BostonSendev # Target variable (median home value)</pre> # The second step is define prior variances to test
# Create a vector of different tau\_squared values (prior variances for beta) to e
tau\_squared\_values <- c(0.01, 0.1, 1, 10, 100, 1000)</pre> # Initialize vectors to store RMSE values for test data
# Emplanation: Prepare a numeric vector to store RMSE results for each tau\_squared value
rmse\_test\_byper <- numeric (length(tau, Squared, values)) # The third step is merge training and validation datasets into a single dataset for model re X\_train\_full <- rbind(X\_train, X\_val) # Combine predictors from training and validation sets y\_train\_full <- c(y\_train, y\_val) # Combine target values from training and validation sets # Evaluate model performance for each tau\_squared value.
for (i in seq\_alonq(tau\_squared\_values)
 tau\_squared <- tau\_squared\_values[i] # Select the current tau\_squared value</pre> # Set prior parameters # Define the prior parameters for Bayesian linear regression: beta0 < rep(0,  $\operatorname{ncl}(X)$ ) # Prior mean vector for beta V0 < diag(tau\_squared,  $\operatorname{ncl}(X)$ ) # Prior covariance matrix for beta a 0 < 0.01 # Shape parameter for inverse-gamma prior on sigma\*2 b0 < 0.01 # Scale parameter for inverse-gamma prior on sigma\*2 # Set random seed for reproducibility
set.seed(123 + i) # Set different seed for each iteratio. # Retrain the model on the combined training and validation sets
results\_final <- bayesian\_linear\_regression(
 X\_train\_full, y\_train\_full, betaO, VO, aO, bO,
 n\_iter = n\_iter, burn\_in = burn\_in</pre> # Obtain Compute the posterior mean of beta from the Gibbs samplin
# This mean is used for making predictions on the test data.
beta\_posterior\_mean\_final <- colMeans(results\_final\$beta\_samples)</pre> # Predict the target variable on the test data using the posterior mean of beta y\_pred\_test <- X\_test %\*% beta\_posterior\_mean\_final</pre> # Compute the Root Mean Square Error (RMSE) to evaluate the model's prediction accuracy on the test set
rmse\_test\_hyper[i] <- sqrt(mean((y\_test - y\_pred\_test)^2)) # RMSE formula</pre> print (rmse\_test\_results) ## TauSquared RMSE\_Test ## 1 1e-02 7.219654 ## 2 1e-01 5.893419 ## 3 1e+00 5.410184 ## 4 1e+01 5.454303 ## 5 1e+02 5.366311 ## 6 1e+03 5.365310 # Load the package 'ggplot2' library(ggplot2) tabs(
 title - "Impact of Prior Variance (TauSquared) on Test RMSE",
 x - "Log10(TauSquared)",
 y - "Test RMSE" Impact of Prior Variance (TauSquared) on Test RMSE 5.5 Log10(TauSquared) The purpose of this section is to evaluate the performance of different prior variances  $(\tau^2)$  for Bayesian Linear Regression using k-fold cross-validation. The first setp is define the number of folds, and observations are randomly assigned to one of the k folds to ensure reproducibility. For each  $\tau^2$  value, the code iterates over the folds, splitting the data into training and validation sets, where one fold is used for validation and the remaining folds for training. The Bayesian model is trained on the training set using Gibbs sampling, and the posterior mean of  $\beta$  is used to redict the target variable on the validation set. The Root Mean Square Error (RMSE) is calculated for each fold to measure prediction accuracy. Finally, the mean RMSE across all folds is computed for each  $\tau^2$ , providing a comprehensive evaluation of how prior variance affects model performance. #### Step 1: Set Up Cross-Validation Parameters #### # In this step, we define the number of folds for k-fold cross-validation k\_folds <- 5 # Number of folds for cross-validation #### Step 2: Create Fold Assignments ####
# In this step, we randomly assign each observation to one of the k folds.
n\_total < -nov(X) # Total number of observations (ensure X is defined)
folds <- sample(rep(1:k\_folds, length.out - n\_total)) # Assign observations to folds</pre> rmse\_cv <- matrix(0, nrow = length(tau\_squared\_values), ncol = k\_folds) # Perform k-fold cross-validation for the current tau\_squared value.
for (k in 1:k\_folds) { # Use fold 'k' as the validation set and the remaining folds # as the training set. test\_indices\_cv <- which(folds -- k) # Indices for the validation set train\_indices\_cv <- which(folds !- k) # Indices for the training set X\_val\_cv <- X[test\_indices\_cv, ] # Validation predictors
y\_val\_cv <- y[test\_indices\_cv] # Validation response variable</pre> # Define prior parameters for Bayesian Linear Regression.

beta0 <- rep(0, ncol(X)) # Prior mean vector for beta

V0 <- diag(tau\_squared, ncol(X)) # Prior covariance matrix for beta
a0 <- 0.01 # Shape parameter for inverse-gamma prior on sigma^2
b0 <- 0.01 # Scale parameter for inverse-gamma prior on sigma^2

RMSE

RMSE

# Fit the model using the training set for the current fold.
results\_cv <- bayesian\_linear\_regression(
 X\_train\_cv, y\_train\_cv, beta0, V0, a0, b0,
 n\_iter = n\_iter, burn\_in = burn\_in</pre>

# Obtain the posterior mean of the regression coefficients. beta\_posterior\_mean\_cv <- colMeans(results\_cv\$beta\_samples)

# Purpose: Use the posterior mean of beta to predict on the validation set y\_pred\_cv <- x\_val\_cv %\*% beta\_posterior\_mean\_cv</pre>

#### Step 5: Compute Mean RMSE Across Folds ####
# In this step, we aggregate the RMSE across all folds for each tau\_squared value.
rmse\_cv\_mean <- rowdeena(rmse\_cv) # Average RMSE across folds for each tau\_square</pre>

Impact of Prior Variance (TauSquared) on Cross-Validation RMSE

Log10(TauSquared)

# Select all columns except 'medv' as predictors and convert to matrix format  $X \leftarrow as.matrix(Boston \ \$ \ \$ \ select(-medv)) # 'medv' is the target variable.$ 

# Add an intercept term to the feature matrix.

X <- cbind(Intercept - 1, X) # Adds a column of ones for the intercept term

# Calculate the number of training and validation samples.
n\_train <- floor(optimal\_train\_prop \* n\_total) # Number of training samples
n\_val <- n\_total - n\_train # Number of validation sample</pre>

# Split indices for training and validation sets.
train\_indices <- indices[1:n\_train] # Indices for training set
val\_indices <- indices[(n\_train + 1):n\_total] # Indices for validation s</pre>

#### Step 5: Set Number of Simulations #### # In this step, we fefine the number of simulations to perform for each tau\_squared value  $n_simulations \leftarrow 50$ 

#### Step 6: Evaluate Bias, Variance, and MSE for Each Tau Squared Value ####
# Loop over tau\_squared values to compute bias, variance, and MSE.
for (i in ac\_along(tau\_squared\_values)) {
 tau\_squared <- tau\_squared\_values[i] # Current tau\_squared value.</pre>

# Create a matrix to store predictions from each simulation.
predictions\_matrix <- matrix(0, nrow - nrow(X\_val), ncol - n\_simulations)</pre> # Estimate the model multiple times to compute bias and variance
for (sim in 1:n\_simulations) {
# Randomly symmetry

# Obtain point estimates of the regression coefficient beta\_posterior\_mean <- colMeans(results\$beta\_samples)

# Use the estimated coefficients to predict on the validation set y\_pred\_val <- X\_val %\*% beta\_posterior\_mean predictions\_matrix[, sim] <- y\_pred\_val</pre>

# Mean prediction for each validation sample across all simulations.
y\_pred\_mean <- rowMeans(predictions\_matrix)</pre>

# Calculate the squared bias.
bias\_squared <- mean((y\_pred\_mean - y\_val)^2)
bias\_list[i] <- bias\_squared # Store bias for the current tau\_squ</pre>

mse <- bias\_squared + variance
mse\_list[i] <- mse # Store MSE for the current tau\_squared.</pre>

theme\_Dw() theme\_Dw()

) +
scale\_linetype\_manual(
values = c("BiasSquared" - "longdash",
 "Variance" - "dashed",
 "MEE" - "solid"),
break = - c("MEE", "BiasSquared", "Variance"),
labels = c("Bias^2", "Variance", "MSE")

) +
scale\_shape\_manual(
values - c("maissquared" - 16,
 "variance" - 17,
 "msE" - 15),
labels - c("maiss2", "variance", "MSE"),
breaks - c("msE", "maissquared", "variance"),

Bias-Variance Trade-off in Bayesian Linear Regression

Log10(TauSquared)

shape - "Performance Metrics"

print(bias\_variance\_plot)

# Purpose: Set prior distributions for Bayesian Linear Regression.

beta0 <- rep(0, ncol(X)) # Prior mean vector for beta.

V0 <- diag(tau\_squared, ncol(X)) # Prior covariance matrix for beta.

a0 <- 0.01 # Shape parameter for inverse-gamma prior on sigma^2.

b0 <- 0.01 # Scale parameter for inverse-gamma prior on sigma^2.

# with 'TauSquared' as the identifier, and 'Component' and 'Value' as columns.
bias\_variance\_melted <- melt(bias\_variance\_df, id.vars - 'TauSquared', variable.name - 'Component', value.name -</pre>

# Flot the bias\_variance\_plot
blas\_variance\_plot <= gplot(bias\_variance\_melted, aes(x - log10(TauSquared), y - Value, color - Component, linet
ype - Component, shape - Component)) +
geom\_line() +
geom\_plott(size - 2, show.legend - FALSE) +</pre>

Bias^2
Variance --- MSE

The purpose of this section is to evaluate the impact of prior variance  $(\tau^2)$  on the bias-variance tradeoff and prediction accuracy in Bayesian Linear Regression. The dataset is prepared by defining the feature matrix and target vector, splitting it into training and validation sets, and specifying a range of  $\tau^2$  values to assess. For each  $\tau^2$ , multiple simulations are performed using bootstrapped training samples, where a Bayesian model is fitted and predictions on the validation set are generated. The bias, variance, and mean squared error (MSE) are computed for each  $\tau^2$ .

quantifying the tradeoff between underfitting and overfitting. Results are consolidated into a data frame, enabling systematic comparison of model performance across different prior variances.

#### Step 6: Compile Results into Data Frame ####
# In this step, Organize the results for further analysis
data\_cv <- data.frame(
tau\_squared\_values,
rmse - rmse\_cv\_mean)</pre>

library(ggplot2)

theme\_bw() +

EWSE 6.5

Cross

library(dplyr)

# Total number of observations.
n\_total <- nrow(X)</pre>

# Randomly shuffle indices to randomize data splitting indices <- sample(1:n\_total)

# Initialize vectors to store bias, variance, and MSE
blas\_list <- numeric(length(tau\_squared\_values))
variance\_list <- numeric(length(tau\_squared\_values))
mse\_list <- numeric(length(tau\_squared\_values))</pre>

print(tau\_squared\_cv\_plot)

plot.title = element\_text(hjust = 0.5)
)

Section 6: Exploring the Bias-Variance Trade-off

# Evaluate the model's prediction accuracy on the validation set.
rmse\_cv[i, k] <- sqrt(mean((y\_val\_cv - y\_pred\_cv)^2))</pre>