FinalExam

Mansi Sharma

2023-12-14

###Problem 1

```
require(car)
```

```
## Loading required package: car
```

```
## Loading required package: carData
```

```
dat<-read.csv("C:/Users/mansi/OneDrive/Desktop/StatModel/stackloss.csv")
dat
```

```
##    Air.Flow Water.Temp Acid.Conc Stack.Loss
## 1        80         27      58.9        4.2
## 2        80         27      58.8        3.7
## 3        75         25      59.0        3.7
## 4        62         24      58.7        2.8
## 5        62         22      58.7        1.8
## 6        62         23      58.7        1.8
## 7        62         24      59.3        1.9
## 8        62         24      59.3        2.0
## 9        58         23      58.7        1.5
## 10       58         18      58.0        1.4
## 11       58         18      58.9        1.4
## 12       58         17      58.8        1.3
## 13       58         18      58.2        1.1
## 14       58         19      59.3        1.2
## 15       50         18      58.9        0.8
## 16       50         18      58.6        0.7
## 17       50         19      57.2        0.8
## 18       50         19      57.9        0.8
## 19       50         20      58.0        0.9
## 20       56         20      58.2        1.5
## 21       70         20      59.1        1.5
```

##1(a)

```
ols <-lm(Stack.Loss~Acid.Conc, data=dat)
summary(ols)
```

```
##
## Call:
## lm(formula = Stack.Loss ~ Acid.Conc, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1584 -0.5584 -0.3066  0.1247  2.2416
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -42.7441    23.4027  -1.826   0.0835 .
## Acid.Conc     0.7590     0.3992   1.901   0.0725 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9565 on 19 degrees of freedom
## Multiple R-squared:  0.1599, Adjusted R-squared:  0.1156
## F-statistic: 3.615 on 1 and 19 DF,  p-value: 0.07252
```

Slope of this model is 0.7590 (positive) which means that as acid.conc increases, stack.loss increases. However, the p-value of acid.conc is 0.0725 which is more than the significance level (0.05) so we can say that is not statistically significant.

##1(b)

```
#Computing centered variable
dat$Acid.Conc.c <- dat$Acid.Conc - mean(dat$Acid.Conc)
#Refitting model
ols1 <- lm(Stack.Loss~Acid.Conc.c, data = dat)
summary(ols1)
```

```
##
## Call:
## lm(formula = Stack.Loss ~ Acid.Conc.c, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1584 -0.5584 -0.3066  0.1247  2.2416
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.7524     0.2087   8.395 8.13e-08 ***
## Acid.Conc.c   0.7590     0.3992   1.901   0.0725 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9565 on 19 degrees of freedom
## Multiple R-squared:  0.1599, Adjusted R-squared:  0.1156
## F-statistic: 3.615 on 1 and 19 DF,  p-value: 0.07252
```

Estimated slope, standard error, t-value and p-value of this model remain same as in part (a). Everything else is similar to ols except the intercept value that changes to 1.7524.

##1(c)

```r
# Calculate leverage values
leverage_values <- hatvalues(ols1)

# Find the indices of the top three points with the highest leverage
top_leverage_indices <- order(leverage_values, decreasing = TRUE)[1:3]

# Display top leverage points
top_leverage_indices
```

```
## [1] 17 18  7
```

```r
for (i in 1:3) {
  # Exclude the top i leverage points
  excluded_data <- dat[-top_leverage_indices[1:i], ]

  # Fit the model
  ols_excluded <- lm(Stack.Loss ~ Acid.Conc.c, data = excluded_data)

  # Get the summary
  summary_ols_excluded <- summary(ols_excluded)

  # Display changes in slope parameter and standard error
  cat("Excluding top", i, "leverage point(s):\n")
  cat("Slope:", coef(ols_excluded)[2], "\n")
  cat("Standard Error:", summary_ols_excluded$coefficients[2, "Std. Error"], "\n\n")
}
```

```
## Excluding top 1 leverage point(s):
## Slope: 0.8138889
## Standard Error: 0.5175218
##
## Excluding top 2 leverage point(s):
## Slope: 0.7134892
## Standard Error: 0.5879628
##
## Excluding top 3 leverage point(s):
## Slope: 0.7929855
## Standard Error: 0.6405008
```

In all three scenarios, the standard error increases as more high leverage points are excluded (from 0.5175 for the top 1, to 0.5880 for the top 2, and finally to 0.6405 for the top 3). This increment in standard error can be attributed to the decreasing sample size and the removal of influential points, which tends to increase the uncertainty in the slope estimate.

Removing high leverage points can improve the regression model if these points are outliers or unduly influencing the model. However, the increased standard error upon their exclusion indicates a trade-off between reducing undue influence and increasing uncertainty in the model. Careful consideration is needed to decide whether excluding these points genuinely improves the model's predictive accuracy and interpretability.

##1(d)

```r
set.seed(123)
bootslope_99 <- numeric(1000)
```

```r
for (b in 1:1000) {
  ind <- sample(nrow(dat), replace = TRUE)
  bootfit <- lm(Stack.Loss ~ Acid.Conc.c, data = dat[ind, ])
  bootslope_99[b] <- coef(bootfit)[2]
}
```

```r
# Bootstrap standard error
se_boot_99 <- sd(bootslope_99)
se_boot_99
```

```
## [1] 0.2836936
```

```r
# 99% bootstrap confidence interval
ci_boot_99 <- quantile(bootslope_99, c(0.005, 0.995))
ci_boot_99
```

```
##      0.5%     99.5%
## 0.2246571 1.8029769
```

```r
# Standard error of the OLS model
se_ols1 <- sqrt(diag(vcov(ols1)))[2]
se_ols1
```

```
## Acid.Conc.c
##   0.3991529
```

```r
# 99% confidence interval of the OLS model
ci_ols1_99 <- confint(ols1, level = 0.99)[2, ]
ci_ols1_99
```

```
##      0.5 %     99.5 %
## -0.3829952  1.9009056
```

The bootstrap method shows a lower standard error (0.2837) than OLS (0.3992), suggesting more precise slope estimates. However, the wider bootstrap confidence interval (0.2247 to 1.8030) compared to OLS (-0.3830 to 1.9009) indicates greater variability in these estimates.

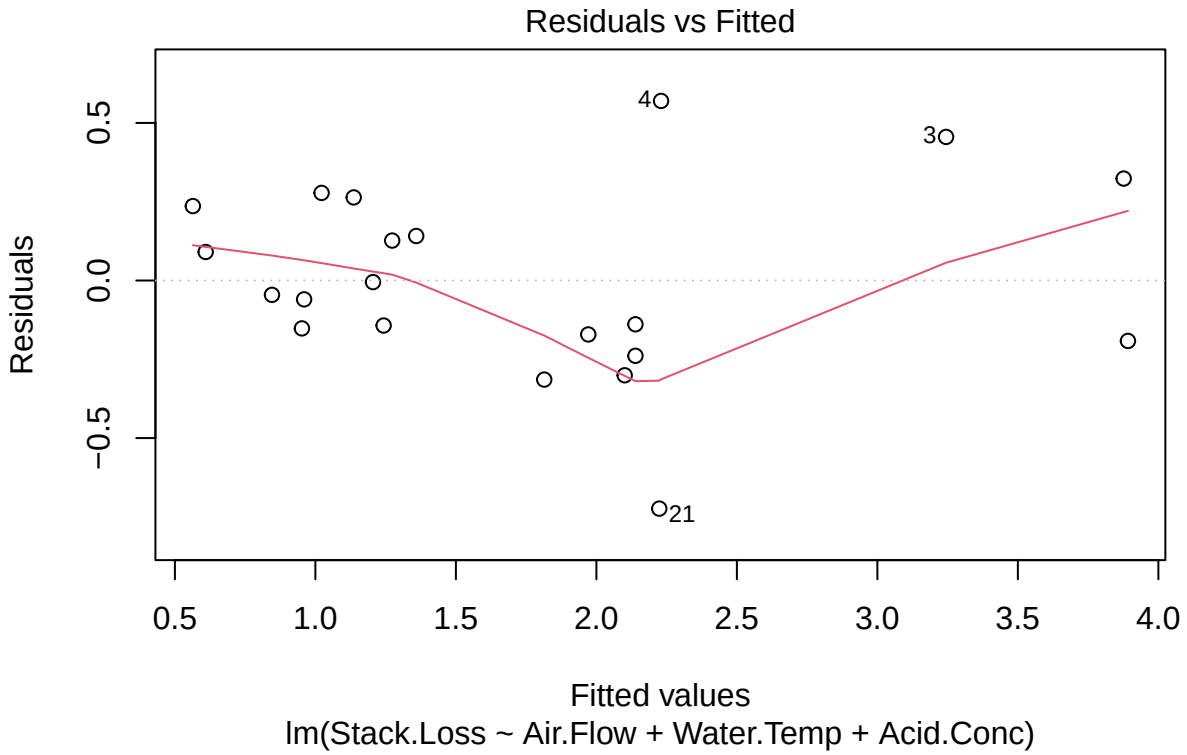
###Problem 2 ##2(a)

```r
fit2 <- lm(Stack.Loss~Air.Flow + Water.Temp + Acid.Conc, data = dat)
summary(fit2)
```

```
##
## Call:
## lm(formula = Stack.Loss ~ Air.Flow + Water.Temp + Acid.Conc,
##     data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72377 -0.17117 -0.04551  0.23614  0.56978
```

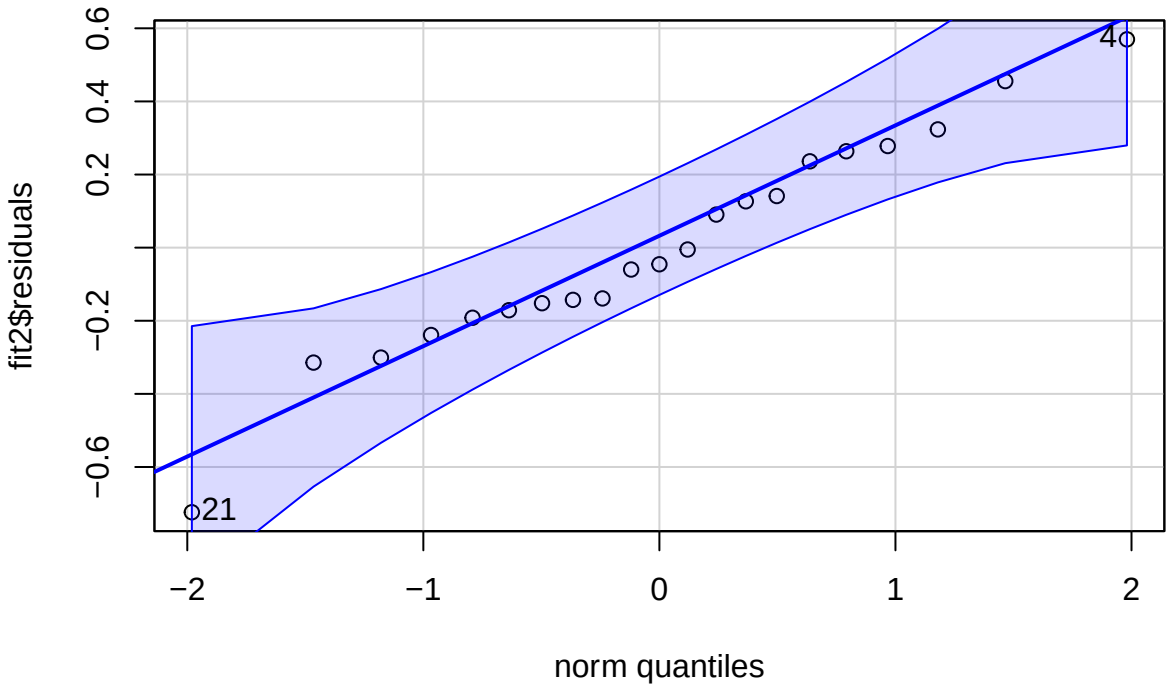
```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.61416    8.90213   0.406  0.68982
## Air.Flow     0.07156    0.01349   5.307  5.8e-05 ***
## Water.Temp   0.12953    0.03680   3.520  0.00263 **
## Acid.Conc   -0.15212    0.15629  -0.973  0.34405
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3243 on 17 degrees of freedom
## Multiple R-squared:  0.9136, Adjusted R-squared:  0.8983
## F-statistic:  59.9 on 3 and 17 DF,  p-value: 3.016e-09
```

```
#Mean 0 and Homoscedasticity
plot(fit2, which = 1)
```



There is clear curvature in the residual plot implying that zero mean assumption is violated. For different fitted values the residual variance keeps varying, hence it does not comply with the homoscedasticity assumption.

```
#Normality
qqPlot(fit2$residuals)
```




```
## [1] 21  4
```

The points in the Q-Q plot do not deviate significantly from the diagonal line so the normality assumption holds. Reason for this might be that the new model fits better.

##2(b)

```
#Variance inflation factors
vif(fit2)
```

```
##   Air.Flow Water.Temp  Acid.Conc
##   2.906484   2.572632   1.333587
```

For Air.Flow and Water.Temp, VIF is between 1 to 5 which means there is some level (moderate) of correlation between those individual variables with the remaining. The larger the value of VIF of a predictor variable the more collinear it is. Acid.Conc has the least correlation with the other predictor variables.

```
fit2b <- lm(Stack.Loss~Water.Temp + Acid.Conc, data=dat)
summary(fit2b)
```

```
##
## Call:
## lm(formula = Stack.Loss ~ Water.Temp + Acid.Conc, data = dat)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.78197 -0.28600 -0.06656  0.31235  0.83295
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -11.57221   13.35206  -0.867    0.398
## Water.Temp    0.27320    0.03949   6.919 1.82e-06 ***
## Acid.Conc     0.12897    0.23291   0.554    0.587
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5137 on 18 degrees of freedom
## Multiple R-squared:  0.7704, Adjusted R-squared:  0.7449
## F-statistic:  30.2 on 2 and 18 DF,  p-value: 1.772e-06
```

Effect of multicollinearity on the model fit: After removing the predictor with highest VIF i.e., Air.Flow, the standard error of the model increases from 0.32 to 0.51.

##2(c)

```
#AIC
library(leaps)
```

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

```
library(MASS)  # for AIC calculation

# Fit all possible models
model.fit <- regsubsets(Stack.Loss ~ Air.Flow + Water.Temp + Acid.Conc, data = dat, nvmax = 3)
model.summary <- summary(model.fit)

# Initialize an empty vector to store AIC values
aic.values <- numeric(length = length(model.summary$rss))

# Calculate AIC for each model
for (i in 1:length(model.summary$rss)) {
    model <- lm(Stack.Loss ~ Air.Flow + Water.Temp + Acid.Conc, data = dat, subset = model.summary$whic
    aic.values[i] <- AIC(model)
}

# Find the model with the lowest AIC
best.model.index <- which.min(aic.values)
best.model.vars <- names(which(model.summary$which[best.model.index, -1]))

best.model.vars
```

```
## [1] "Air.Flow"
```

```
#LASSO
library(glmnet)
```

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

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
library(MASS)  # For AIC calculation

# Prepare data
X <- as.matrix(dat[, c("Air.Flow", "Water.Temp", "Acid.Conc")])
Y <- dat$Stack.Loss

# Fit LASSO models over a range of lambda values
lasso.fit <- glmnet(X, Y, alpha = 1)

# Extract a sequence of lambda values from the fit
lambda_seq <- lasso.fit$lambda

# Initialize a vector to store AIC values
aic_values <- numeric(length(lambda_seq))

# Loop over lambda values, fit model, and calculate AIC
for (i in seq_along(lambda_seq)) {
  # Fit LASSO model at specific lambda
  lasso_model <- glmnet(X, Y, alpha = 1, lambda = lambda_seq[i])

  # Make predictions
  predictions <- predict(lasso_model, newx = X, s = lambda_seq[i])

  # Calculate the residual sum of squares (RSS)
  rss <- sum((Y - predictions)^2)

  # Estimate degrees of freedom: number of non-zero coefficients
  df <- sum(coef(lasso_model)[-1] != 0)

  # Calculate AIC (using an approximation method)
  n <- nrow(X)
  aic_values[i] <- n * log(rss / n) + 2 * df
}

# Find the best model (lowest AIC)
best_lambda_index <- which.min(aic_values)
best_lambda <- lambda_seq[best_lambda_index]

# Fit and display the best LASSO model
best_lasso <- glmnet(X, Y, alpha = 1, lambda = best_lambda)
coef(best_lasso)
```

```
## 4 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -4.71306412
## Air.Flow     0.06449103
## Water.Temp   0.12174994
## Acid.Conc    .
```

The AIC for predictor subset selection identifies "Air.Flow" as the most significant predictor, while the LASSO method retains both "Air.Flow" and "Water.Temp", excluding "Acid.Conc." This difference highlights LASSO's ability to consider multiple predictors' combined influence, leading to a potentially more complex model than AIC-based selection, which favors simplicity.

###Problem 3 ##3(a)

```
RandomSubsetSelection <- function(x, y) {
  # Number of predictors
  p <- ncol(x)

  # Initialize
  best_aic <- Inf
  best_model <- NULL
  improvement <- TRUE
  coeff_path <- list()
  aic_path <- numeric()

  while (improvement) {
    # Randomly sample a subset of predictors
    sampled_vars <- sample(1:p, size = ceiling(p/2))

    # Fit the model using forward selection with the sampled subset
    current_model <- step(lm(y ~ 1, data = x),
                          scope = list(lower = formula(lm(y ~ 1, data = x)),
                                       upper = formula(lm(y ~ ., data = x[, sampled_vars]))),
                          direction = "forward",
                          trace = 0)

    # Evaluate the model using AIC
    current_aic <- AIC(current_model)

    # Check for improvement
    if (current_aic < best_aic) {
      best_aic <- current_aic
      best_model <- current_model
      coeff_path[[length(coeff_path) + 1]] <- coef(current_model)
      aic_path[length(aic_path) + 1] <- current_aic
    } else {
      improvement <- FALSE
    }
  }

  list(coeff_path = coeff_path, aic_path = aic_path)
}
```

##3(b)

```
RandomSubsetSelectionVisualize <- function(coeffPath) {
  # Plot of the estimated coefficients
  par(mfrow = c(2, 1))
  plot(unlist(coeffPath$coeff_path), main = "Coefficients Path", xlab = "Step", ylab = "Coefficient", ty

  # Plot of the corresponding AIC
```

```r
  plot(coeffPath$aic_path, main = "AIC Path", xlab = "Step", ylab = "AIC", type = "b")
}
```

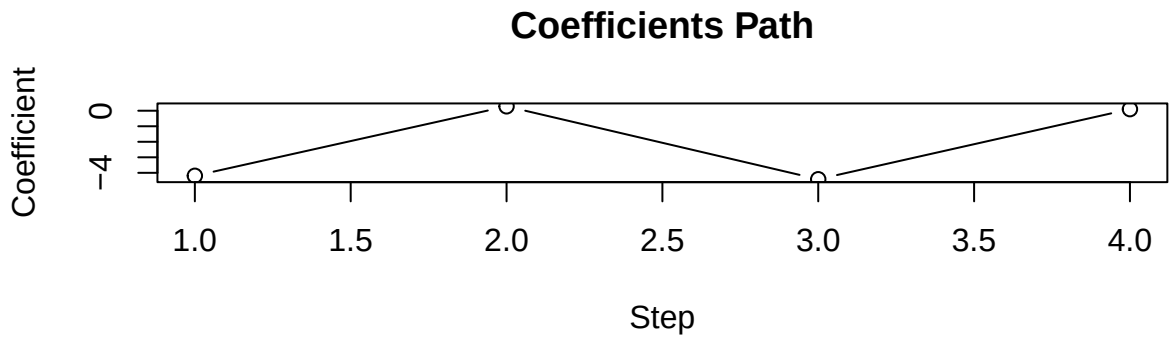
##3(c)

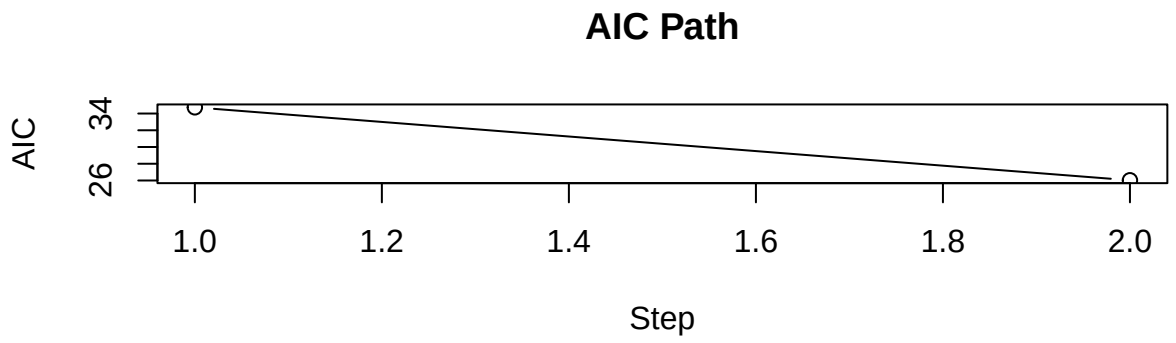
Applying above functions to the stackloss dataset

```r
# Assuming 'dat' is your dataset
coeffPath <- RandomSubsetSelection(dat[, 1:3], dat$Stack.Loss)
coeffPath
```

```
## $coeff_path
## $coeff_path[[1]]
## (Intercept)  Water.Temp
##  -4.1910867   0.2817445
##
## $coeff_path[[2]]
## (Intercept)    Air.Flow
##  -4.4132025   0.1020309
##
##
## $aic_path
## [1] 34.73877 26.02853
```

```r
# Visualize
RandomSubsetSelectionVisualize(coeffPath)
```





From 2(c) we saw that AIC gives best model with Air.Flow. However, a clear minimum AIC corresponds to a model including "Air.Flow" indicating a stable selection, whereas fluctuating AIC values reflect the influence of randomness on model selection. The random element of our subset selection is introducing variability in the selected model.Thus, this model is considered the best according to this random sampling approach.

###Problem 4 ##4(a) Generate random predictors and fit a multiple regression model

```
# Load the stackloss dataset
stackloss <- read.csv("C:/Users/mansi/OneDrive/Desktop/StatModel/stackloss.csv")

# Add 50 random predictors to the stackloss dataset
set.seed(123) # For reproducibility
for (i in 1:50) {
  stackloss[[paste0("x", i)]] <- runif(nrow(stackloss), min = -1, max = 1)
}

# Fit the multiple regression model with all predictors, including the random ones
fit_full <- lm(Stack.Loss ~ ., data = stackloss)
summary(fit_full)
```

```
##
## Call:
## lm(formula = Stack.Loss ~ ., data = stackloss)
##
## Residuals:
## ALL 21 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (33 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 119.813134        NaN     NaN      NaN
## Air.Flow     -0.009309        NaN     NaN      NaN
## Water.Temp    0.217879        NaN     NaN      NaN
## Acid.Conc    -2.091052        NaN     NaN      NaN
## x1           -0.325116        NaN     NaN      NaN
## x2            0.213455        NaN     NaN      NaN
## x3           -1.462047        NaN     NaN      NaN
## x4           -0.006009        NaN     NaN      NaN
## x5           -0.558790        NaN     NaN      NaN
## x6            0.274726        NaN     NaN      NaN
## x7            0.379826        NaN     NaN      NaN
## x8           -1.761561        NaN     NaN      NaN
## x9           -0.510479        NaN     NaN      NaN
## x10          -0.282343        NaN     NaN      NaN
## x11          -1.445377        NaN     NaN      NaN
## x12          -1.676413        NaN     NaN      NaN
## x13           0.384496        NaN     NaN      NaN
## x14           0.110121        NaN     NaN      NaN
## x15          -0.128839        NaN     NaN      NaN
## x16          -0.622871        NaN     NaN      NaN
## x17           0.862459        NaN     NaN      NaN
## x18                 NA         NA      NA       NA
## x19                 NA         NA      NA       NA
## x20                 NA         NA      NA       NA
## x21                 NA         NA      NA       NA
```

```
## x22                 NA         NA      NA       NA
## x23                 NA         NA      NA       NA
## x24                 NA         NA      NA       NA
## x25                 NA         NA      NA       NA
## x26                 NA         NA      NA       NA
## x27                 NA         NA      NA       NA
## x28                 NA         NA      NA       NA
## x29                 NA         NA      NA       NA
## x30                 NA         NA      NA       NA
## x31                 NA         NA      NA       NA
## x32                 NA         NA      NA       NA
## x33                 NA         NA      NA       NA
## x34                 NA         NA      NA       NA
## x35                 NA         NA      NA       NA
## x36                 NA         NA      NA       NA
## x37                 NA         NA      NA       NA
## x38                 NA         NA      NA       NA
## x39                 NA         NA      NA       NA
## x40                 NA         NA      NA       NA
## x41                 NA         NA      NA       NA
## x42                 NA         NA      NA       NA
## x43                 NA         NA      NA       NA
## x44                 NA         NA      NA       NA
## x45                 NA         NA      NA       NA
## x46                 NA         NA      NA       NA
## x47                 NA         NA      NA       NA
## x48                 NA         NA      NA       NA
## x49                 NA         NA      NA       NA
## x50                 NA         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 20 and 0 DF,  p-value: NA
```

##4(b) Use AIC for Best Subset Selection

```r
# Using regsubsets from the leaps package to perform best subset selection
#fit_subset <- regsubsets(Stack.Loss ~ ., data = stackloss, nvmax = 53, method = "exhaustive", really.b
#subset_summary <- summary(fit_subset)
# Using stepwise selection for best subset selection based on AIC
# Starting with the full model and using both directions (forward and backward)
#fit_stepwise <- step(lm(Stack.Loss ~ ., data = stackloss), direction = "both", trace = 0)
#summary(fit_stepwise)

forward_selection <- function(data, response) {
  # Start with a model with only the intercept
  formula <- as.formula(paste(response, "~ 1"))
  current_model <- lm(formula, data = data)
  current_aic <- AIC(current_model)

  # Get predictor names
  predictors <- setdiff(names(data), response)

  # Iteratively add predictors and check AIC
```

```r
  for (pred in predictors) {
    # Try adding each predictor to the model
    new_formula <- as.formula(paste(deparse(formula), "+", pred))
    new_model <- lm(new_formula, data = data)
    new_aic <- AIC(new_model)

    # Update the model if AIC improves
    if (new_aic < current_aic) {
      current_model <- new_model
      current_aic <- new_aic
      formula <- new_formula
    }
  }

  return(current_model)
}

# Apply forward selection to the stackloss dataset
best_model_forward <- forward_selection(stackloss, "Stack.Loss")
```

```
## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.
```

```
## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.
```

```
## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.

## Warning: Using formula(x) is deprecated when x is a character vector of length > 1.
##   Consider formula(paste(x, collapse = " ")) instead.
```

```r
summary(best_model_forward)
```

```
##
## Call:
## lm(formula = new_formula, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35508 -0.10486  0.05328  0.10510  0.28464
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.467884   0.371034 -14.737 1.72e-09 ***
## Air.Flow     0.063284   0.009632   6.570 1.80e-05 ***
## Water.Temp   0.163318   0.025680   6.360 2.50e-05 ***
## x2           0.241288   0.097637   2.471  0.02807 *
## x4          -0.321392   0.103255  -3.113  0.00824 **
## x5          -0.045869   0.099253  -0.462  0.65162
```

```
## x8           0.501250   0.140921   3.557  0.00351 **
## x25         -0.226145   0.089970  -2.514  0.02592 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2037 on 13 degrees of freedom
## Multiple R-squared:  0.9739, Adjusted R-squared:  0.9599
## F-statistic: 69.37 on 7 and 13 DF,  p-value: 2.796e-09
```

Implemented forward selection to identify a more parsimonious model. This will help in comparing how variable selection methods deal with the added noise from random predictors

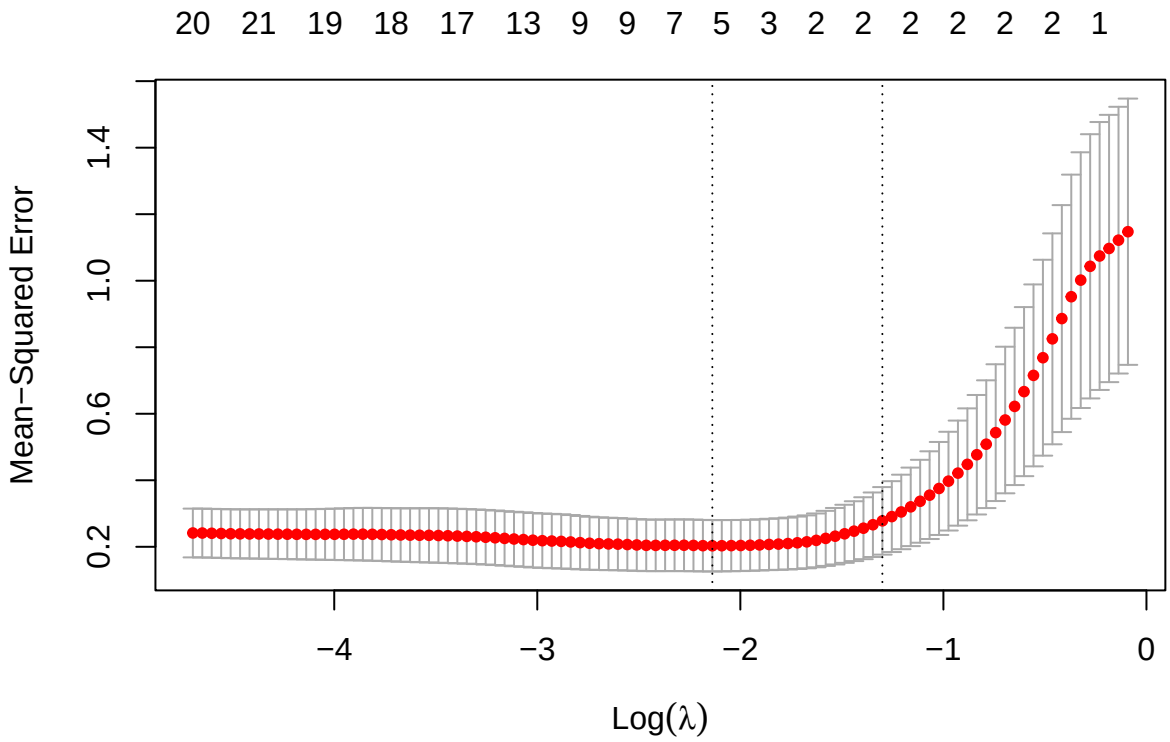
Use LASSO for Best Subset Selection

```
# Prepare matrix for glmnet
x_matrix <- model.matrix(~ . - Stack.Loss, data = stackloss)
y_vector <- stackloss$Stack.Loss

# Fit LASSO model
lasso_mod <- cv.glmnet(x_matrix, y_vector, alpha = 1)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
plot(lasso_mod)
```



Fitted a LASSO model to enforce sparsity in the coefficients. LASSO is particularly useful in dealing with high-dimensional data and should help in identifying the most relevant predictors.

Apply Random Subset Selection functions to the stackloss dataset

```
RandomSubsetSelection <- function(x, y) {
    p <- ncol(x)  # Number of predictors in x

    best_aic <- Inf
    best_model <- NULL
    coeff_path <- list()
    aic_path <- numeric()
    improvement <- TRUE

    data_for_model <- cbind(y = y, x)

    while (improvement) {
        sampled_vars <- sample(1:p, size = max(1, ceiling(p/2)))  # Ensure at least one predictor is se

        current_best_aic <- Inf
        current_best_model <- NULL

        for (var_index in sampled_vars) {
            predictors <- names(data_for_model)[var_index + 1]  # +1 for 'y'
            if (length(predictors) > 0) {
                formula <- reformulate(predictors, response = "y")
                model <- lm(formula, data = data_for_model)

                aic_value <- AIC(model)
                if (aic_value < current_best_aic) {
                    current_best_aic <- aic_value
                    current_best_model <- model
                }
            }
        }

        if (!is.null(current_best_model) && current_best_aic < best_aic) {
            best_aic <- current_best_aic
            best_model <- current_best_model
            coeff_path[[length(coeff_path) + 1]] <- coef(current_best_model)
            aic_path[length(aic_path) + 1] <- current_best_aic
        } else {
            improvement <- FALSE
        }
    }

    list(best_model = best_model, coeff_path = coeff_path, aic_path = aic_path)
}



# Apply the RandomSubsetSelection function
coeff_path <- RandomSubsetSelection(x_matrix, y_vector)


# Define the RandomSubsetSelectionVisualize function
RandomSubsetSelectionVisualize <- function(coeffPath) {
```

```
  # Ensure the data for plotting is well-defined
  if (length(coeffPath$coeff_path) > 0 && length(coeffPath$aic_path) > 0) {
    coeff_values <- unlist(coeffPath$coeff_path)
    aic_values <- coeffPath$aic_path

    # Check for finite values before plotting
    if (all(is.finite(coeff_values)) && all(is.finite(aic_values))) {
      par(mfrow = c(2, 1))
      plot(coeff_values, main = "Coefficients Path", xlab = "Step", ylab = "Coefficient", type = "b")
      plot(aic_values, main = "AIC Path", xlab = "Step", ylab = "AIC", type = "b")
    } else {
      cat("Non-finite values detected in coefficient or AIC paths. Cannot generate plot.\n")
    }
  } else {
    cat("No data available for plotting in coeff_path or aic_path.\n")
  }
}



# Apply the visualization function
RandomSubsetSelectionVisualize(coeff_path)
```

```
## No data available for plotting in coeff_path or aic_path.
```

Developed a Random Subset Selection function to explore various combinations of predictors. This method should provide insights into which variables are consistently deemed important across different random subsets.

The coefficient path plot indicates how coefficients evolve as the model complexity increases. Significant fluctuations might suggest unstable model selection due to the presence of many irrelevant variables. The AIC path plot shows the model's goodness-of-fit as more variables are added. A steep initial decline followed by a plateau or increase could indicate the point of diminishing returns in adding more variables.

##4(c) Simulation of Adding Random Predictors

```
# Repeat the process of adding random predictors and selection 50 times
selection_counts <- replicate(50, {
  # Add random predictors
  for (i in 1:50) {
    stackloss[[paste0("x", i)]] <- runif(nrow(stackloss), min = -1, max = 1)
  }

  # Fit LASSO model
  lasso_mod <- cv.glmnet(x_matrix, y_vector, alpha = 1)

  # Return the number of times each predictor is selected
  as.integer(coef(lasso_mod, s = "lambda.min") != 0)
})
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
```

```
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
```

```
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
```

```
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```r
# Calculate the proportion of selection for each predictor
proportion_selected <- colMeans(selection_counts)
proportion_selected
```

```
##  [1] 0.14545455 0.07272727 0.14545455 0.05454545 0.07272727 0.10909091
##  [7] 0.07272727 0.07272727 0.07272727 0.09090909 0.10909091 0.10909091
## [13] 0.05454545 0.07272727 0.18181818 0.09090909 0.07272727 0.07272727
## [19] 0.07272727 0.09090909 0.07272727 0.05454545 0.07272727 0.07272727
## [25] 0.14545455 0.07272727 0.10909091 0.05454545 0.09090909 0.05454545
## [31] 0.07272727 0.07272727 0.14545455 0.10909091 0.12727273 0.14545455
## [37] 0.07272727 0.14545455 0.07272727 0.07272727 0.14545455 0.09090909
## [43] 0.10909091 0.05454545 0.07272727 0.10909091 0.14545455 0.09090909
## [49] 0.05454545 0.07272727
```

The proportion of times each predictor is selected gives us a sense of which variables are most robust to the inclusion of noise from random predictors. The presence of many random predictors seems to complicate the

model selection process, as evidenced by the variability in selected models and the challenges in fitting a full model. This highlights the importance of careful variable selection and the potential pitfalls of overfitting in regression analysis.

1 Problem 5

We consider a linear regression model with Laplace errors (double-exponential errors):

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i,$$

where Y_i is the observed response, X_i is the predictor variable, ε_i is the error term distributed according to a Laplace distribution with mean 0 and scale parameter b, and β_0, β_1 are the parameters to be estimated.

Likelihood Function

The probability density function for the Laplace distribution is given by:

$$f(\varepsilon_i) = \frac{1}{2b} e^{-\frac{|\varepsilon_i|}{b}}.$$

For the observed data, the likelihood function is the product of the individual probabilities:

$$L(\beta_0, \beta_1) = \prod_{i=1}^{n} \frac{1}{2b} e^{-\frac{|Y_i - (\beta_0 + \beta_1 X_i)|}{b}}.$$

Log-Likelihood Function

The log-likelihood function is the natural logarithm of the likelihood:

$$\ell(\beta_0, \beta_1) = \sum_{i=1}^{n} \log\left(\frac{1}{2b}\right) - \frac{1}{b} \sum_{i=1}^{n} |Y_i - (\beta_0 + \beta_1 X_i)|.$$

Maximum Likelihood Estimation

The MLE seeks to maximize the log-likelihood function. Since the first term is constant with respect to β_0 and β_1, maximization is equivalent to minimizing the second term, which is the sum of absolute deviations:

$$\min_{\beta_0, \beta_1} \sum_{i=1}^{n} |Y_i - (\beta_0 + \beta_1 X_i)|.$$

Proof that MLE Leads to LAR

Minimizing the sum of absolute deviations is precisely the objective of least absolute regression (LAR). Therefore, by maximizing the log-likelihood function for a linear regression model with Laplace errors, we are performing LAR.

Conclusion

We have shown that the MLE for the parameters of a linear regression model with Laplace errors results in the same estimates as those obtained by LAR, which minimizes the sum of absolute deviations between the observed and predicted values.

2 Problem 6

2.1 Logistic Regression Model

In a logistic regression model predicting the probability of a customer making a purchase $P(\text{Purchase})$, the coefficient for the variable "Time Spent on Website" (x_j) is $\beta_j = 0.02$. The interpretation of this coefficient in the context of the log-odds ratio is as follows:

For every one-minute increase in the time spent on the website, the log-odds of making a purchase increases by 0.02, provided that all other variables in the model are held constant.

Effect of Additional Time Spent on the Website

To interpret the effect of a customer spending an additional 10 minutes on the website, we calculate the odds ratio (OR) for a 10-minute increase:

$$\text{OR} = e^{\beta_j \Delta x_j} = e^{0.02 \times 10} = e^{0.2}.$$

This OR represents the factor by which the odds of making a purchase are multiplied when the time spent on the website increases by 10 minutes.

Thus, if a customer spends an additional 10 minutes on the website, the odds of making a purchase are multiplied by $e^{0.2}$, which we can approximate as:

$$e^{0.2} \approx 1.221.$$

This means that the odds of making a purchase are expected to increase by about 22.1% for every additional 10 minutes spent on the website, assuming all other variables remain constant.

2.2 Poisson Regression Model

In a Poisson regression model predicting the number of customer support calls received in a day Y, the coefficient for the variable "Number of Marketing Emails Sent" (x_j) is $\beta_j = 0.05$. The interpretation of this coefficient in the context of the Poisson distribution is as follows:

For every one-unit increase in the number of marketing emails sent, the expected count of customer support calls increases by 0.05, provided that all other variables in the model are held constant.

Effect of an Increase in Marketing Emails Sent

To interpret the change when the number of marketing emails sent increases by 20 (i.e., $x_j = 20 + a$), where a was the number of marketing emails sent before, we can calculate the expected counts for both scenarios:

For $x_j = a$, the expected count of customer support calls is given by the Poisson distribution:

$$E(Y|x_j = a) = \lambda = e^{\beta_j a} = e^{0.05a}.$$

For $x_j = 20 + a$, the expected count is:

$$E(Y|x_j = 20 + a) = \lambda = e^{\beta_j (20+a)} = e^{0.05(20+a)}.$$

The change in the expected count is given by:

$$\Delta E(Y) = E(Y|x_j = 20 + a) - E(Y|x_j = a) = e^{0.05(20+a)} - e^{0.05a}.$$

This means that if the number of marketing emails sent increases by 20, the expected count of customer support calls is expected to change by $\Delta E(Y)$, assuming all other variables remain constant.