MTH4430 FinalProj Work

Mth 4330

Christopher Guzman, Dereck Lin, Haris Nioulikos, Samiha Uddin

2025-11-17

```r
# Data Processing
student_health_data <- read.csv('Data/student_health_data.csv')
#convert response label to an ordered factor
student_health_data$Health_Risk_Level <- factor(student_health_data$Health_Risk_Level,levels = c("Low", "Moderate", "High"),orde

#Encoding ordinal features, and the nominal feature gender
#cat features:  Gender, Physical_Activity, Sleep_Quality, Mood
student_health_data <- student_health_data %>%
  mutate(
    Gender = case_when(
      Gender == "M" ~ 1,
      Gender == "F" ~ 0
      ),
    Physical_Activity = case_when(
      Physical_Activity == "High" ~ 2,
      Physical_Activity == "Moderate" ~ 1,
      Physical_Activity == "Low" ~ 0
      ),
    Sleep_Quality = case_when(
      Sleep_Quality == "Good" ~ 2,
      Sleep_Quality == "Moderate" ~ 1,
      Sleep_Quality == "Poor" ~ 0
      ),
    Mood = case_when(
      Mood == "Happy" ~ 2,
      Mood == "Neutral" ~ 1,
      Mood == "Stressed" ~ 0
      )
    )

# remove identifier(Student_ID)
student_health_data$Student_ID <- NULL

#OVR encode into separate cols
classes <- levels(student_health_data$Health_Risk_Level)
for(class in classes){
  isClass <- paste0("y_", class)
  student_health_data[[isClass]] <- ifelse(student_health_data$Health_Risk_Level == class, 1, 0)
}

#numerical encode for

set.seed(1)#for reproducibility

# split into train&test(8:2)
# src: https://scikit-learn.org/stable/common_pitfalls.html
idx <- createDataPartition(student_health_data$Health_Risk_Level, p = 0.8, list = FALSE)
tr <- student_health_data[idx,]
te <- student_health_data[-idx,]

#cv split assignments
```

```r
v <- sample(1:5, nrow(tr), replace = TRUE)
```

```r
#Baseline Model: Majority Rule by Christopher Guzman
ProjData <- student_health_data # To load the dataset
#print(df) # To view how the dataset would look like
dim(ProjData) # If curious about the dimensions of dataset
```

```
## [1] 1000   16
```

```r
head(ProjData) # Want to see first few rows
```

```
##   Age Gender Heart_Rate Blood_Pressure_Systolic Blood_Pressure_Diastolic
## 1  24      1   50.66322                122.1730                 84.41986
## 2  21      0   57.92604                110.7784                 75.69615
## 3  22      1   59.29422                109.3757                 83.80381
## 4  24      1   76.82623                125.1422                 78.09159
## 5  20      1   68.34277                107.5156                 80.67494
## 6  22      1   61.74415                 90.0000                 84.45086
##   Stress_Level_Biosensor Stress_Level_Self_Report Physical_Activity
## 1               3.137350                 9.028669                 2
## 2               3.699078                 5.819697                 1
## 3               6.785156                 5.892360                 0
## 4               6.408509                 6.884001                 2
## 5               7.264719                 4.483450                 1
## 6               4.262518                 6.825001                 1
##   Sleep_Quality Mood Study_Hours Project_Hours Health_Risk_Level y_Low
## 1             1    2   34.520973      16.80096          Moderate     0
## 2             2    0   16.763846      15.79115          Moderate     0
## 3             1    2   44.203798      25.67844          Moderate     0
## 4             0    2   21.776645      20.80839              High     0
## 5             0    2    8.964999      15.19405          Moderate     0
## 6             2    2   44.948229      15.65120          Moderate     0
##   y_Moderate y_High
## 1          1      0
## 2          1      0
## 3          1      0
## 4          0      1
## 5          1      0
## 6          1      0
```

```r
#print(df$Health_Risk_Level) # To view the variable we are planning to predict
# names(ProjData) # To see the names of the columns of dataset


# str(ProjData) # if you want a summary of dataset
table(ProjData$Health_Risk_Level)   # check the class distribution
```

```
## 
##      Low Moderate     High 
##      190      672      138
```

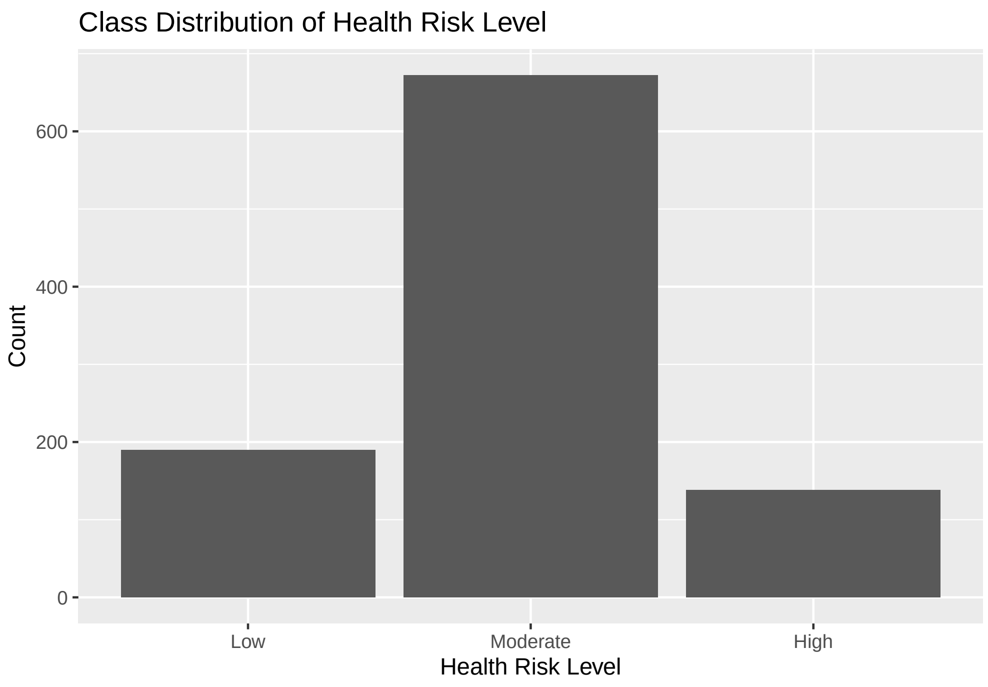
```r
# We can use the table() function to create frequency tables of categorical data of a certain vector.
# It will count the number of times a unique value or combination of values shows up in a vector.
# Can use for combination of vectors, creating a contingency table


ProjData$Health_Risk_Level = factor(ProjData$Health_Risk_Level) # Factor function used to handle categorical variables

# Baseline: Majority Class Predictor
# Find most frequent class
majority_class = names(which.max(table(ProjData$Health_Risk_Level)))
print(majority_class)
```

```
## [1] "Moderate"
```

```
# We will predict this class for every student
baseline_predictions = rep(majority_class, nrow(ProjData))

# Compute Accuracy
baseline_accuracy = mean(baseline_predictions == ProjData$Health_Risk_Level)
print(baseline_accuracy)
```

```
## [1] 0.672
```

```
ggplot(ProjData, aes(x = Health_Risk_Level)) +
  geom_bar() +
  labs(title = "Class Distribution of Health Risk Level",
       x = "Health Risk Level",
       y = "Count")
```



Base ovr Log Reg Model

```
# predictors: "Age + Gender + Heart_Rate + Blood_Pressure_Systolic + Blood_Pressure_Diastolic + Stress_Level_Biosensor + Stress_L
logr_cv_log_loss <- c(0,0,0,0,0)

#itr over cv folds
for(fold in 1:5){
  #split tr data into folds
  cv_tr <- tr[v != fold, ]
  cv_te <- tr[v == fold, ]

  #fit on all, excluding response labels(Health_Risk_Level, y_Low, y_Moderate, y_High)
  logr_High <- glm(y_High ~ Age + Gender + Heart_Rate + Blood_Pressure_Systolic + Blood_Pressure_Diastolic + Stress_Level_Biosen
  logr_Moderate <- glm(y_Moderate ~ Age + Gender + Heart_Rate + Blood_Pressure_Systolic + Blood_Pressure_Diastolic + Stress_Leve
  logr_Low <- glm(y_Low ~ Age + Gender + Heart_Rate + Blood_Pressure_Systolic + Blood_Pressure_Diastolic + Stress_Level_Biosenso

  # #predict
  logr_High_Pred <- predict(logr_High, newdata = cv_te, type = 'response')
  logr_Moderate_Pred <- predict(logr_Moderate, newdata = cv_te, type = 'response')
  logr_Low_Pred <- predict(logr_Low, newdata = cv_te, type = 'response')

  #normalize probabilities
  logr_y_pred <- cbind(logr_Low_Pred,logr_Moderate_Pred,logr_High_Pred)
  logr_y_pred_row_sum <- rowSums(logr_y_pred)
  logr_y_pred_normalized <- logr_y_pred / logr_y_pred_row_sum

  #real label
  logr_y_real <- cv_te[,c("y_Low","y_Moderate","y_High")]

  #Cross entropy loss
  logr_cv_log_loss[fold] <-  -mean(rowSums(logr_y_real * log(logr_y_pred_normalized)))
}

logr_log_loss <- mean(logr_cv_log_loss)
logr_log_loss
```

```
## [1] 0.4337477
```

Refitting base ovr Logr on entirety of train set

```
final_logr_High <- glm(y_High ~ Age + Gender + Heart_Rate + Blood_Pressure_Systolic + Blood_Pressure_Diastolic + Stress_Level_Bi
                       data = tr, family = "binomial")

final_logr_Moderate <- glm(y_Moderate ~ Age + Gender + Heart_Rate + Blood_Pressure_Systolic + Blood_Pressure_Diastolic + Stress_
                           data = tr, family = "binomial")

final_logr_Low <- glm(y_Low ~ Age + Gender + Heart_Rate + Blood_Pressure_Systolic + Blood_Pressure_Diastolic + Stress_Level_Bios
                      data = tr, family = "binomial")
```

Pred Final ovr Log R model on Train Set

```
## [1] 0.4067402
```

Pred Final ovr Log R model on Test Set

```
#Predict
final_logr_High_pred <- predict(final_logr_High, newdata = te, type = 'response')
final_logr_Moderate_pred <- predict(final_logr_Moderate, newdata = te, type = 'response')
final_logr_Low_pred <- predict(final_logr_Low, newdata = te, type = 'response')

# bind into prob vec
final_logr_pred_prob <- cbind(final_logr_Low_pred,final_logr_Moderate_pred,final_logr_High_pred)
final_logr_pred_prob_rsum <- rowSums(final_logr_pred_prob)
final_logr_pred_prob_normalized <- final_logr_pred_prob / final_logr_pred_prob_rsum

final_logr_log_loss <- -mean(rowSums(te[,c("y_Low","y_Moderate","y_High")] * log(final_logr_pred_prob_normalized)))
final_logr_log_loss
```

```
## [1] 0.4616341
```

Sanity check, comparing Majority rule with Logr

```
#Vector of hard classes extracted from test set
y_te_real <- te[,"Health_Risk_Level"]

#collapsing 'logr_y_pred_normalized' into hard classes
logr_y_pred_class <- max.col(final_logr_pred_prob_normalized)

#using accuracy as a baseline metric
desired_levels <- c("Low", "Moderate", "High")
colnames(final_logr_pred_prob_normalized) <- desired_levels
final_predictions <- colnames(final_logr_pred_prob_normalized)[logr_y_pred_class]

final_predictions <- factor(final_predictions, levels = desired_levels)

logr_conf_matrix <- table(Predicted = final_predictions, Actual = te$Health_Risk_Level)

confusionMatrix(logr_conf_matrix, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Actual
## Predicted  Low Moderate High
##   Low       31        1    3
##   Moderate   7      130   14
##   High       0        3   10
##
## Overall Statistics
##
##                Accuracy : 0.8593
##                  95% CI : (0.8031, 0.9044)
##     No Information Rate : 0.6734
##     P-Value [Acc > NIR] : 1.644e-09
##
##                   Kappa : 0.6849
##
##  Mcnemar's Test P-Value : 0.002174
##
## Statistics by Class:
##
##                      Class: Low Class: Moderate Class: High
## Sensitivity              0.8158          0.9701     0.37037
## Specificity              0.9752          0.6769     0.98256
## Pos Pred Value           0.8857          0.8609     0.76923
## Neg Pred Value           0.9573          0.9167     0.90860
## Precision                0.8857          0.8609     0.76923
## Recall                   0.8158          0.9701     0.37037
## F1                       0.8493          0.9123     0.50000
## Prevalence               0.1910          0.6734     0.13568
## Detection Rate           0.1558          0.6533     0.05025
## Detection Prevalence     0.1759          0.7588     0.06533
## Balanced Accuracy        0.8955          0.8235     0.67646
```

ovr LASSO

```
lasso_cv_log_loss <- c(0,0,0,0,0)
for(fold in 1:5){
  #13-16 = health risk lvl, y_...,y_...,y_...
  #split tr data into folds and extract feature matrix and response matrix for lasso using glmnet, also exlcuding hard class labe
  cv_x_tr <- as.matrix(tr[v != fold, - c(13,14,15,16)])
  cv_y_tr <- tr[v != fold, c("y_Low","y_Moderate","y_High")]
  cv_x_te <- as.matrix(tr[v == fold, - c(13,14,15,16)])
  cv_y_te <- tr[v == fold,c("y_Low","y_Moderate","y_High")]

  #train
```

```r
  lasso_high <- cv.glmnet(cv_x_tr, cv_y_tr[,"y_High"], family = "binomial", type.measure = "deviance")
  lasso_moderate <- cv.glmnet(cv_x_tr, cv_y_tr[,"y_Moderate"], family = "binomial", type.measure = "deviance")
  lasso_low <- cv.glmnet(cv_x_tr, cv_y_tr[,"y_Low"], family = "binomial", type.measure = "deviance")

  #pred
  lasso_high_pred <- predict(lasso_high, newx = cv_x_te, s = "lambda.min", type = "response")
  lasso_moderate_pred <- predict(lasso_moderate, newx = cv_x_te, s = "lambda.min", type = "response")
  lasso_low_pred <- predict(lasso_low, newx = cv_x_te, s = "lambda.min", type = "response")

  #prob matrix
  lasso_y_pred <- cbind(lasso_low_pred, lasso_moderate_pred ,lasso_high_pred)
  lasso_y_pred_rowsums <- rowSums(lasso_y_pred)
  lasso_y_pred_normalized <- lasso_y_pred / lasso_y_pred_rowsums

  #Log loss calc
  lasso_cv_log_loss[fold] <- -mean(rowSums(cv_y_te * log(lasso_y_pred_normalized)))
}

lasso_log_loss <- mean(lasso_cv_log_loss)
lasso_log_loss
```

```
## [1] 0.4404227
```

LASSO refit on Train set

```r
X_tr_full_matrix <- as.matrix(tr[, -c(13, 14, 15, 16)])
Y_tr_full_targets <- tr[, c(14, 15, 16)]

best_lasso_high <- cv.glmnet(X_tr_full_matrix, tr[,16 ], family = "binomial", type.measure = "deviance")
best_lasso_moderate <- cv.glmnet(X_tr_full_matrix, tr[,15 ], family = "binomial", type.measure = "deviance")
best_lasso_low <- cv.glmnet(X_tr_full_matrix, tr[,14 ], family = "binomial", type.measure = "deviance")

best_lasso_high_pred <- predict(best_lasso_high, newx = X_tr_full_matrix, s = "lambda.min", type = "response")
best_lasso_moderate_pred <- predict(best_lasso_moderate, newx = X_tr_full_matrix, s = "lambda.min", type = "response")
best_lasso_low_pred <- predict(best_lasso_low, newx = X_tr_full_matrix, s = "lambda.min", type = "response")

best_lasos_y_pred <- cbind(best_lasso_low_pred,best_lasso_moderate_pred,best_lasso_high_pred)
best_lasso_y_pred_rowsum <- rowSums(best_lasos_y_pred)
best_lasso_y_pred_normalized <- best_lasos_y_pred / best_lasso_y_pred_rowsum

log_loss <- -mean(rowSums(Y_tr_full_targets * log(best_lasso_y_pred_normalized)))
log_loss
```

```
## [1] 0.4244026
```

```r
best_lambda <- c(best_lasso_low$lambda.min, best_lasso_moderate$lambda.min, best_lasso_high$lambda.min)
best_lambda
```

```
## [1] 0.004015451 0.009399397 0.006117351
```

Optimized Lasso Model

```r
X_te_full_matrix <- as.matrix(te[, -c(13, 14, 15, 16)])
Y_te_full_targets <- te[, c(14, 15, 16)]

te_best_lasso_high_pred <- predict(best_lasso_high, newx = X_te_full_matrix, s = best_lambda[3], type = "response")
te_best_lasso_moderate_pred <- predict(best_lasso_moderate, X_te_full_matrix, s = best_lambda[2], type = "response")
te_best_lasso_low_pred <- predict(best_lasso_low, newx = X_te_full_matrix, s = best_lambda[1], type = "response")
```

```
te_best_lasso_y_pred <- cbind(te_best_lasso_low_pred, te_best_lasso_moderate_pred, te_best_lasso_high_pred)
te_best_lasso_y_pred_rsum <- rowSums(te_best_lasso_y_pred)
te_best_lasso_y_pred_normalized <- te_best_lasso_y_pred / te_best_lasso_y_pred_rsum

te_best_lasso_log_loss <- -mean(rowSums(Y_te_full_targets * log(te_best_lasso_y_pred_normalized)))
te_best_lasso_log_loss
```

```
## [1] 0.4676428
```

## Optimized Lasso conf mat

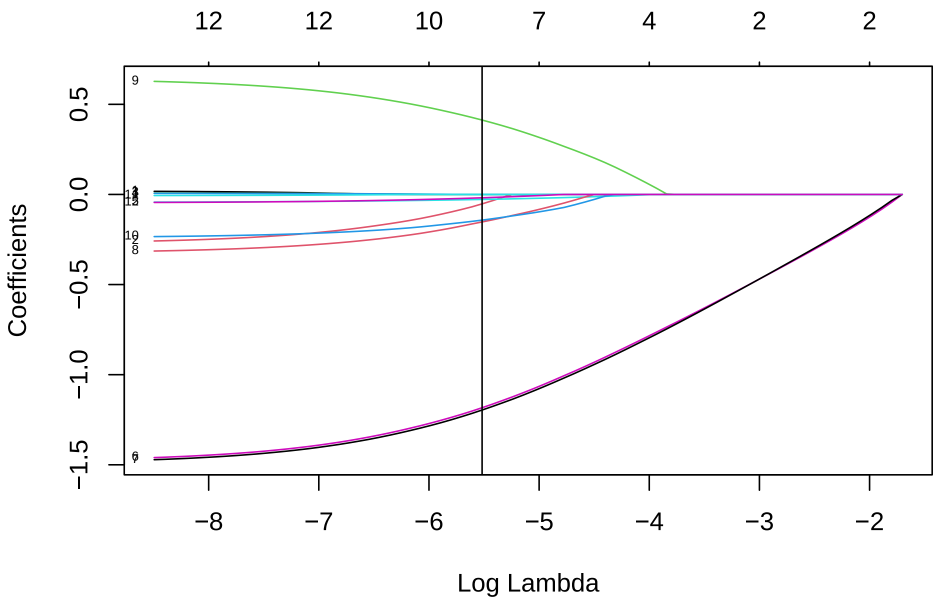
```
#y_te_real = true hard classes
#collapsing 'te_best_lasso_y_pred_normalized' into hard classes
colnames(te_best_lasso_y_pred_normalized) <- c("Low", "Moderate", "High")
best_lasso_y_pred_class <- max.col(te_best_lasso_y_pred_normalized)

final_predictions <- colnames(te_best_lasso_y_pred_normalized)[best_lasso_y_pred_class]

final_predictions <- factor(final_predictions, levels = desired_levels)

lasso_conf_matrix <- table(Predicted = final_predictions, Actual = te$Health_Risk_Level)

confusionMatrix(lasso_conf_matrix, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Actual
## Predicted  Low Moderate High
##   Low       28        1    3
##   Moderate  10      132   18
##   High       0        1    6
##
## Overall Statistics
##
##                Accuracy : 0.8342
##                  95% CI : (0.7751, 0.883)
##     No Information Rate : 0.6734
##     P-Value [Acc > NIR] : 2.382e-07
##
##                   Kappa : 0.6081
##
##  Mcnemar's Test P-Value : 1.171e-05
##
## Statistics by Class:
##
##                      Class: Low Class: Moderate Class: High
## Sensitivity              0.7368          0.9851     0.22222
## Specificity              0.9752          0.5692     0.99419
## Pos Pred Value           0.8750          0.8250     0.85714
## Neg Pred Value           0.9401          0.9487     0.89062
## Precision                0.8750          0.8250     0.85714
## Recall                   0.7368          0.9851     0.22222
## F1                       0.8000          0.8980     0.35294
## Prevalence               0.1910          0.6734     0.13568
## Detection Rate           0.1407          0.6633     0.03015
## Detection Prevalence     0.1608          0.8040     0.03518
## Balanced Accuracy        0.8560          0.7772     0.60820
```

```
confusionMatrix(logr_conf_matrix, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Actual
## Predicted  Low Moderate High
##   Low       31        1    3
##   Moderate   7      130   14
##   High       0        3   10
##
## Overall Statistics
##
##                Accuracy : 0.8593
##                  95% CI : (0.8031, 0.9044)
##     No Information Rate : 0.6734
##     P-Value [Acc > NIR] : 1.644e-09
##
##                   Kappa : 0.6849
##
##  Mcnemar's Test P-Value : 0.002174
##
## Statistics by Class:
##
##                      Class: Low Class: Moderate Class: High
## Sensitivity              0.8158          0.9701     0.37037
## Specificity              0.9752          0.6769     0.98256
## Pos Pred Value           0.8857          0.8609     0.76923
## Neg Pred Value           0.9573          0.9167     0.90860
## Precision                0.8857          0.8609     0.76923
## Recall                   0.8158          0.9701     0.37037
## F1                       0.8493          0.9123     0.50000
## Prevalence               0.1910          0.6734     0.13568
## Detection Rate           0.1558          0.6533     0.05025
## Detection Prevalence     0.1759          0.7588     0.06533
## Balanced Accuracy        0.8955          0.8235     0.67646
```

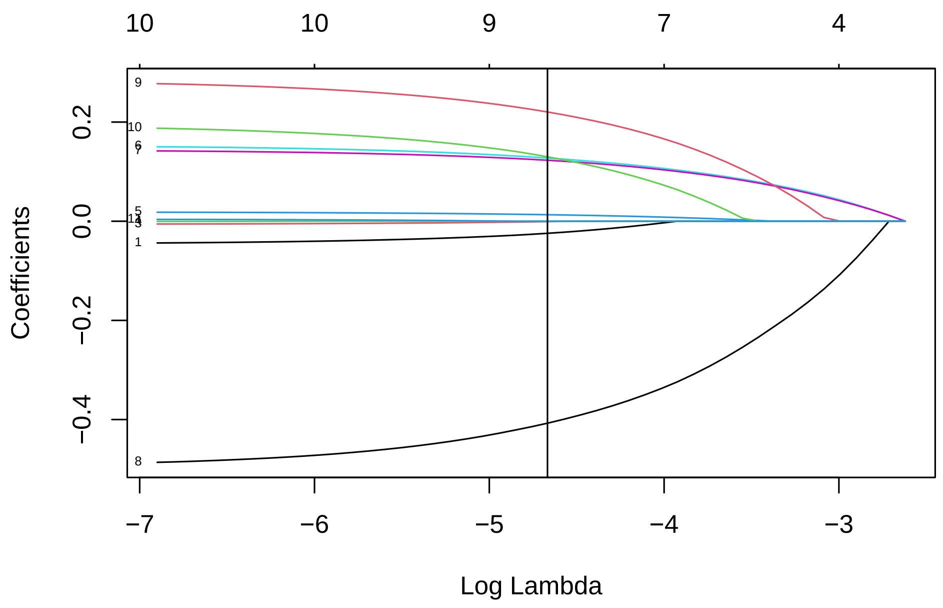
```
plot(best_lasso_low$glmnet.fit,xvar = "lambda", label = TRUE)
abline(v = log(best_lambda)[1])
```



```
coef(best_lasso_low, s = "lambda.min")
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                    s1
## (Intercept)               10.91029653
## Age                         .
## Gender                    -0.05234152
## Heart_Rate                  .
## Blood_Pressure_Systolic     .
## Blood_Pressure_Diastolic -0.02768900
## Stress_Level_Biosensor   -1.18340279
## Stress_Level_Self_Report -1.19615320
## Physical_Activity        -0.15274027
## Sleep_Quality             0.41269211
## Mood                     -0.14247052
## Study_Hours                 .
## Project_Hours            -0.01861177
```

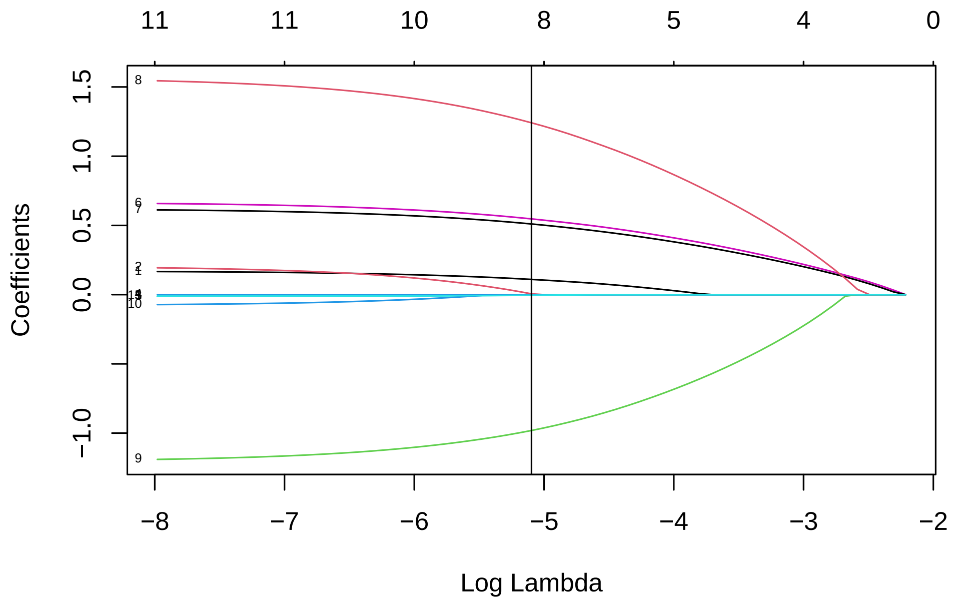
```r
plot(best_lasso_moderate$glmnet.fit,xvar = "lambda", label = TRUE)
abline(v = log(best_lambda)[2])
```



```r
coef(best_lasso_moderate, s = "lambda.min")
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                      s1
## (Intercept)              -1.1382434826
## Age                      -0.0243767507
## Gender                     .
## Heart_Rate               -0.0007822639
## Blood_Pressure_Systolic    .
## Blood_Pressure_Diastolic  0.0131325741
## Stress_Level_Biosensor    0.1273952093
## Stress_Level_Self_Report  0.1229408006
## Physical_Activity        -0.4073551144
## Sleep_Quality             0.2203588768
## Mood                      0.1299367738
## Study_Hours                .
## Project_Hours              .
```

```r
plot(best_lasso_high$glmnet.fit,xvar = "lambda", label = TRUE)
abline(v = log(best_lambda)[3])
```



```r
coef(best_lasso_high, s = "lambda.min")
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                     s1
## (Intercept)              -10.441838638
## Age                        0.109930955
## Gender                     0.005887929
## Heart_Rate                -0.001298608
## Blood_Pressure_Systolic    .
## Blood_Pressure_Diastolic  -0.004741060
## Stress_Level_Biosensor     0.547092707
## Stress_Level_Self_Report   0.510406793
## Physical_Activity          1.241046947
## Sleep_Quality             -0.981167905
## Mood                       .
## Study_Hours               -0.004480365
## Project_Hours              .
```