ELE39123 - Group Exam

1 Exercise One

For the initial exercise in this assignment, I am tasked with developing an algorithm aimed at surpassing a Mean Squared Error (MSE) target of 115.455. My strategy for this task is methodical yet straightforward. I plan to commence with a Multiple Linear Regression (MLR) to gain insights into the significance of various predictors through their t- and p-values. I will then streamline the model by excluding variables that do not significantly contribute to the predictive power of the model. This deliberate choice to begin with a foundational approach allows me to establish a solid baseline from which to iterate and enhance the model's performance.

The evolution of our model begins with a foundational Multiple Linear Regression (MLR). This will later culminate into our modeling strategy for the adoption of a neural network, selected for its advanced capabilities in pattern recognition and non-linear prediction.

1.1 Multiple Linear Regression Version 1

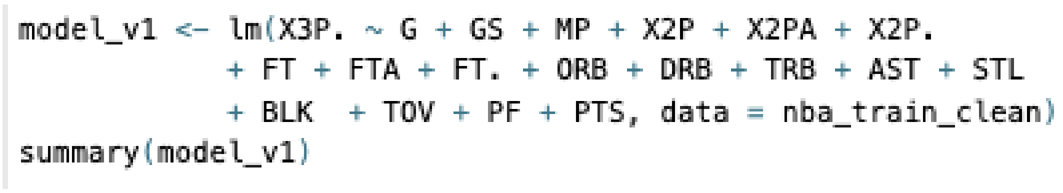
Firstly, I initiated the process by cleaning the datasats "nba_train.txt" and "nba_test.txt", ensuring the removal of all instances of NA values to enhance data quality and integrity. This preliminary step is cruical for establishing a good foundation for the model.

$$\hat{Y} = \beta_0 + \beta_1 \cdot G + \beta_2 \cdot GS + \beta_3 \cdot X2P. + ... + \beta_{18} \cdot PTS + \varepsilon$$

Secondly, I establihsed the Multiple Linear Regression (MLR) model by incorporating all relevant variables across the dataset. The formula encapsulated within this model serves as the mathematical represention of the MLR, effectively capturing the relationship between the predictors and the response variable.

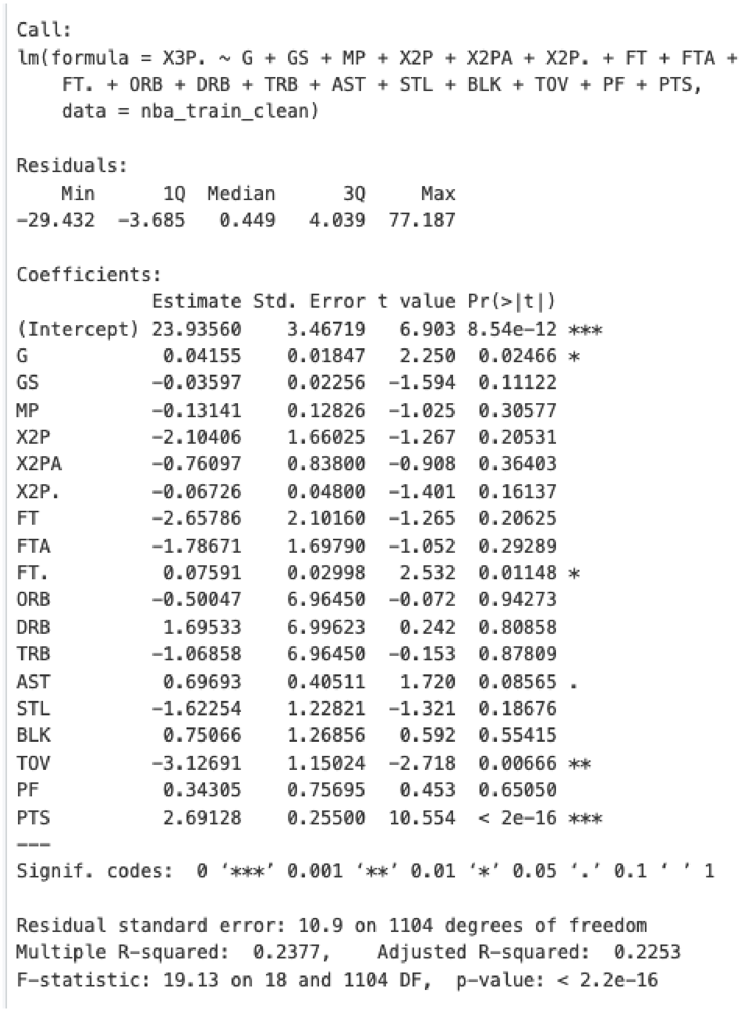
1.1.1 Results and code

In this section, I aim to showcase the code used for the regression analysis along with a summary of its results. This aspect proved to be particularly insightful, as it allowed for a detailed examination of which parameters significantly influence the model and which do not.



```
model_v1 <- lm(X3P. ~ G + GS + MP + X2P + X2PA + X2P.
               + FT + FTA + FT. + ORB + DRB + TRB + AST + STL
               + BLK  + TOV + PF + PTS, data = nba_train_clean)
summary(model_v1)
```

Figure 1: Piece of code used to build the model.



```
Call:
lm(formula = X3P. ~ G + GS + MP + X2P + X2PA + X2P. + FT + FTA +
    FT. + ORB + DRB + TRB + AST + STL + BLK + TOV + PF + PTS,
    data = nba_train_clean)

Residuals:
    Min      1Q  Median      3Q     Max
-29.432  -3.685   0.449   4.039  77.187

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 23.93560    3.46719   6.903 8.54e-12 ***
G            0.04155    0.01847   2.250  0.02466 *
GS          -0.03597    0.02256  -1.594  0.11122
MP          -0.13141    0.12826  -1.025  0.30577
X2P         -2.10406    1.66025  -1.267  0.20531
X2PA        -0.76097    0.83800  -0.908  0.36403
X2P.        -0.06726    0.04800  -1.401  0.16137
FT          -2.65786    2.10160  -1.265  0.20625
FTA         -1.78671    1.69790  -1.052  0.29289
FT.          0.07591    0.02998   2.532  0.01148 *
ORB         -0.50047    6.96450  -0.072  0.94273
DRB          1.69533    6.99623   0.242  0.80858
TRB         -1.06858    6.96450  -0.153  0.87809
AST          0.69693    0.40511   1.720  0.08565 .
STL         -1.62254    1.22821  -1.321  0.18676
BLK          0.75066    1.26856   0.592  0.55415
TOV         -3.12691    1.15024  -2.718  0.00666 **
PF           0.34305    0.75695   0.453  0.65050
PTS          2.69128    0.25500  10.554  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 10.9 on 1104 degrees of freedom
Multiple R-squared:  0.2377,    Adjusted R-squared:  0.2253
F-statistic: 19.13 on 18 and 1104 DF,  p-value: < 2.2e-16
```

Figure 2: Piece of code used to build the model.

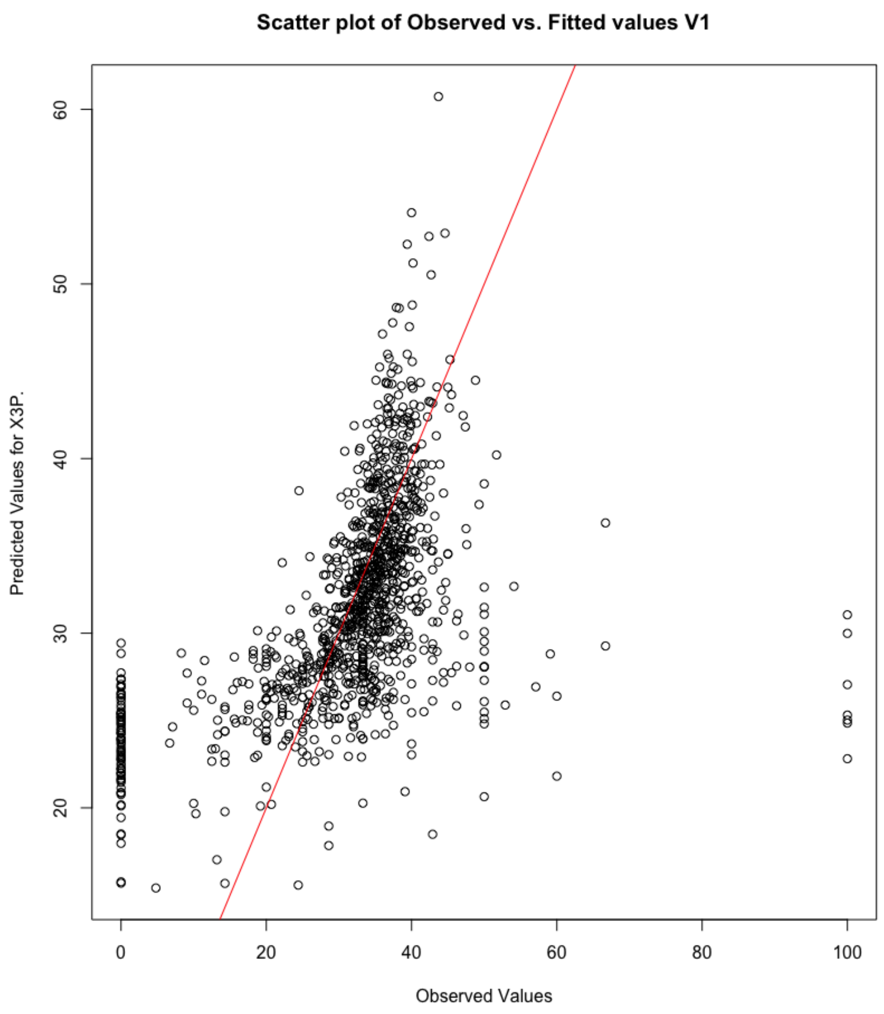


Figure 3: Plotting the MLR version 1

The examination of the plot reveals a discernible linear relationship between the predicted values for X3P. and their observed values. This relationship is established through the utilization of all coefficients in the model, indicating a direct correlation between the predicted and actual performance in this aspect of basketball statistics.

Upon reviewing the model summary, several significant variables have been identified based on their t-values and p-values. These variables stand out in the statistical analysis, demonstrating a notable impact on the predictive power of the model. Their significance is cruical in understanding the factors that influence the prediction of X3P values.

1. PTS: t-val = 10.554 and p-val = $< 2e - 16$
2. TOV: t-val = -2.718 and p-val = 0.0666
3. FT.: t-val = 2.532 and p-val = 0.01148
4. G: t-val = 2.250 and p-val = 0.02466
5. X2P: t-val = $-1,267$ and p-val = 0.20531

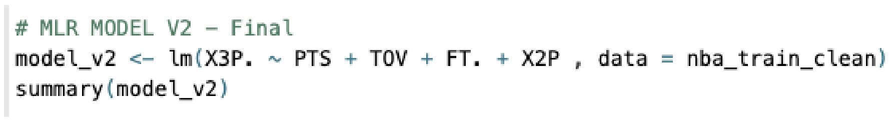
The decision to incorporate the X2P coefficient in the model was primarily guided by intuition rather than a deeply analytical approach. The rationale was that predicting X3P. would logically benefit from considering the X2P coefficient, despite the absence of sufficient theoretical backing for this. This approach was adopted with acknowledgement that I have limited knowledge of basketball in the United States.

With the initial model established, the focus now shifts towards refinement and improvement. To enhance the model's accuracy and reliability, further adjustments and modifications are required. This progression lead to a second version of the MLR (MLR v2), MLR v2 builds on foundations of the current model.

1.2 Multiple Linear Regression Version 2

For the second iteration, MLR v2, all of the insignificant variables have now been removed from the first initial model. This is the mathematical representation for the adjusted and new MLR v2.

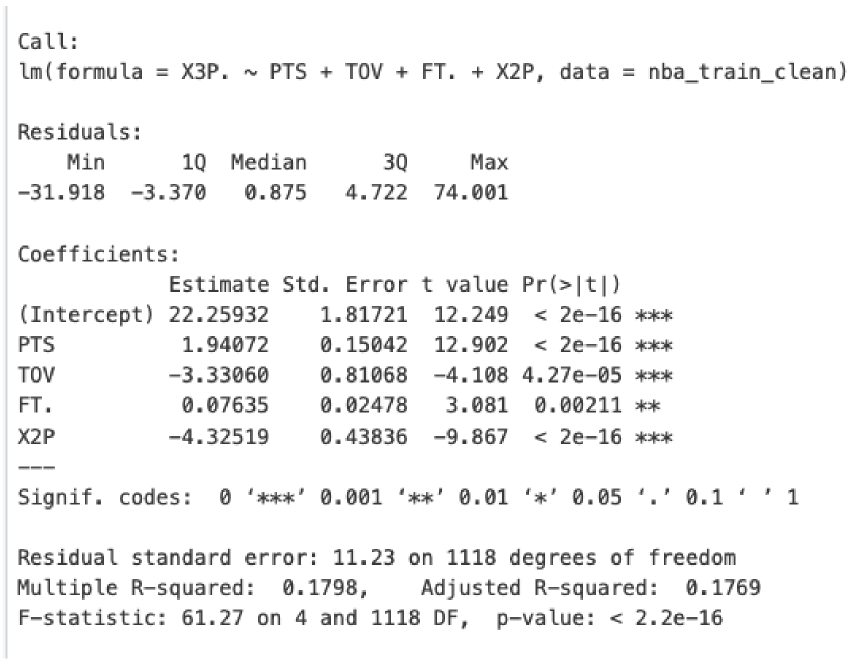
$$\hat{Y} = \beta_0 + \beta_1 \cdot PTS + \beta_2 \cdot TOV. + \beta_3 \cdot FT. + \beta_1 \cdot X2P$$



```
# MLR MODEL V2 - Final
model_v2 <- lm(X3P. ~ PTS + TOV + FT. + X2P , data = nba_train_clean)
summary(model_v2)
```

Figure 4: This is the code used to build MLR v2.

1.2.1 Results



```
Call:
lm(formula = X3P. ~ PTS + TOV + FT. + X2P, data = nba_train_clean)

Residuals:
    Min      1Q  Median      3Q     Max 
-31.918  -3.370   0.875   4.722  74.001 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 22.25932    1.81721  12.249  < 2e-16 ***
PTS          1.94072    0.15042  12.902  < 2e-16 ***
TOV         -3.33060    0.81068  -4.108 4.27e-05 ***
FT.          0.07635    0.02478   3.081  0.00211 ** 
X2P         -4.32519    0.43836  -9.867  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 11.23 on 1118 degrees of freedom
Multiple R-squared:  0.1798,	Adjusted R-squared:  0.1769 
F-statistic: 61.27 on 4 and 1118 DF,  p-value: < 2.2e-16
```

Figure 5: This is the code used to build MLR v2.

When looking at the summary results for this new model, we see heavy improvement. The t-values for this model are great. We have t-values indicating a highly significant effect on the intercept, and high t-values suggesting that they are a significant predictor of X3P.. What is even more interesting is the negative t-values as well, that suggest a significant negative effect on X3P, with the most interesting being X2P with a -9.867 (the coefficient picked intuitively) that suggests it significantly decreases X3P.

Look at the code section for the submitted code to put in your dataset, the submitted set is from the neural network, clearly marked in image.

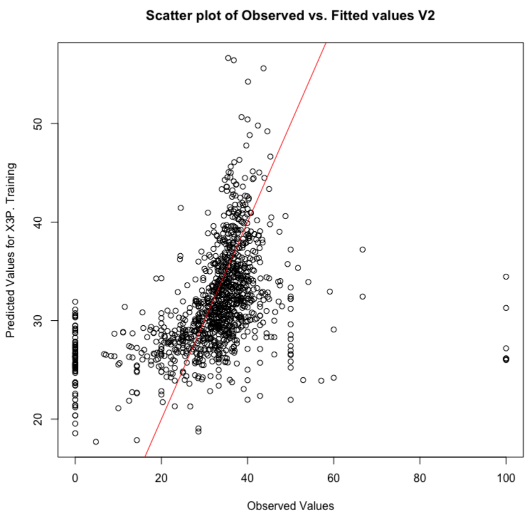


Figure 6: The new plot for MLR v2, suggesting a linear relationship.

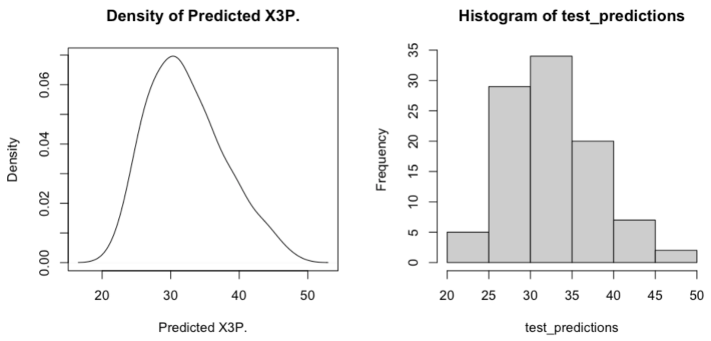


Figure 7: A density plotting a histogram of the predicted X3P.

Let's start by looking at the density plot. The density on the Y-axis represents the probability density function for the predicted X3P. values. The higher the curve at a particular point on the X-axis, the higher the probability density of X3P. values around that point. The plot appears to be skewed to the right, meaning there are a tail of predictions extending towards higher values of X3P.. Suggesting that while most of the predicted values are clustered around the 30s, there is a non-negligible probability of higher values occurring. The peak of the curve seems to be around the mid-30s, indicating that the most common predicted value (the model) for X3P. is in this region.

For the histogram, shows a unimodal distribution, meaning there is one primary peak. This distribution is slightly right-skewed, similar to the density plot, as there are longer tails towards the higher values. The tallest bar is centered around 30-35, indicating that the most predicted X3P. values fall within this range. The width of the bars and the range of the x-axis show that the predictions vary from about 20 to just under 50. There are more frequent predictions in the middle ranges and fewer as the predictions get larger or smaller.

In comparison to the density plot, which gives a smooth estimate of the distribution of predictions, the histogram provides the actual frequency of predictions within each interval. Both plots suggest that most of the predicted X3P values are in the mid-30s, with fewer predictions as the values increase or decrease, and a tendency for the occurrence of higher values, though they are less common.

1.3 Ridge and Lasso Regression

1.3.1 Ridge Regression

Ridge regression is a technique used to analyze multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are so large so they may be far from the true value. Ridge regression adds just enough bias to make these estimates meaningful and improves the predictions made by the model.

1.3.2 Mathematical Foundation of Ridge Regression

In OLS, the regression coefficients are estimated by minimizing the sum of SSR, which is the sum of squares of the differences between the observed and predicted values. Mathematically, this can be presented as:

$$min_\beta SSR = min_\beta \sum_{i=1}^{n}(y_i - \beta 0 - \sum_{j=1}^{p}\beta_j x_{ij})^2$$

Here, y_i is the observed response, β_0 is the intercept, β_j are the regression coefficients, and x_{ij} are the predicted variables.

Ridge regression modifies this objective by adding a penalty term that is equal to the square of the magnitude of the coefficients:

$$min_\beta \Big\{ \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}) + \lambda \sum_{j=1}^{p}\beta_j^2 \Big\}$$

The parameter λ is a complexity parameter that controls the amount of shrinkage: the larger the value of λ, the greater the amount of shrinkage. By increasing λ, the coefficients are shrunk towards zero (and each other), but if $\lambda = 0$, the ridge regression estimates coincide with the OLS estimates.

1.3.3 How Ridge Regression Works

The key concept behind ridge regression is bias-variance trade-off. By introducing bias into the estimates (through the λ penalty), ridge regression reduces the variance of the coefficient estimates. This is particularly beneficial in the presence of multicollinearity, where small changes in the model can lead to large changes in the coefficient estimates.

Ridge regression works by shrinking the coefficients towards zero, which also means that all variables are kept in the model. This difference from lasso regression, which can shrink coefficients completely to zero, thus performing variable selection.

1.3.4 Lasso Regression

Lasso regression, which stands for Least Absolute Shrinkage and Selection Operator, is a type of linear regresion that includes a penalty term to regularize the coefficients of the model. The lasso technique encourages simple, sparse models (i.e. models with fewer parameters).

1.3.5 Mathematical Foundation of Lasso Regression

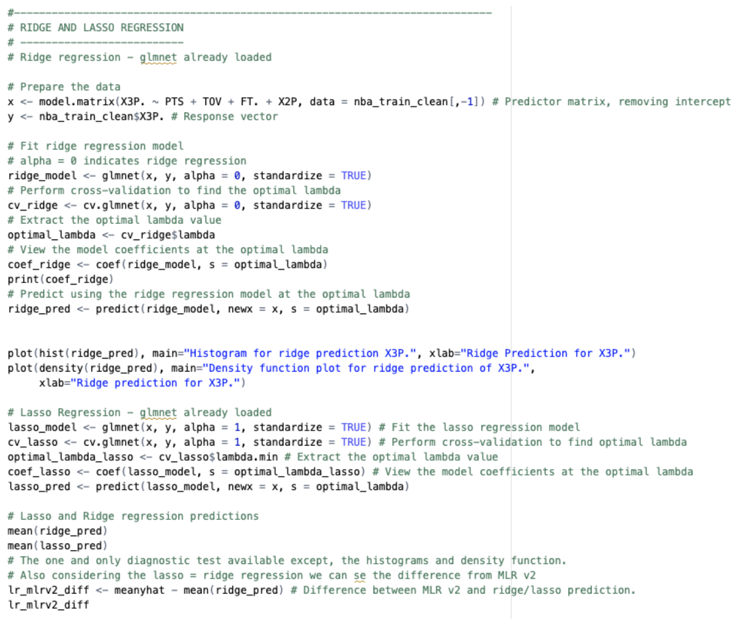
The lasso regression modifies the least squares objective function by adding a penalty term which is the absolute value of the sum of the coefficients, leading to the following optimization problem:

$$min_\beta \Big\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \Big\}$$

Here:

- y_i are the observed values,
- β_0 is the intercept,
- β_j are the coefficients for predictors x_{ij}
- λ is the regularization parameter, a non-negative hyperparameter that controls the strength of the penalty. The higher the value of λ, the more absolute values of the coefficients are penalized, which leads to more coefficients being set to zero.

1.3.6 code and results



```r
#-----------------------------------------------------------------------------
# RIDGE AND LASSO REGRESSION
# -------------------------
# Ridge regression - glmnet already loaded

# Prepare the data
x <- model.matrix(X3P. ~ PTS + TOV + FT. + X2P, data = nba_train_clean[,-1]) # Predictor matrix, removing intercept
y <- nba_train_clean$X3P. # Response vector

# Fit ridge regression model
# alpha = 0 indicates ridge regression
ridge_model <- glmnet(x, y, alpha = 0, standardize = TRUE)
# Perform cross-validation to find the optimal lambda
cv_ridge <- cv.glmnet(x, y, alpha = 0, standardize = TRUE)
# Extract the optimal lambda value
optimal_lambda <- cv_ridge$lambda
# View the model coefficients at the optimal lambda
coef_ridge <- coef(ridge_model, s = optimal_lambda)
print(coef_ridge)
# Predict using the ridge regression model at the optimal lambda
ridge_pred <- predict(ridge_model, newx = x, s = optimal_lambda)


plot(hist(ridge_pred), main="Histogram for ridge prediction X3P.", xlab="Ridge Prediction for X3P.")
plot(density(ridge_pred), main="Density function plot for ridge prediction of X3P.",
     xlab="Ridge prediction for X3P.")

# Lasso Regression - glmnet already loaded
lasso_model <- glmnet(x, y, alpha = 1, standardize = TRUE) # Fit the lasso regression model
cv_lasso <- cv.glmnet(x, y, alpha = 1, standardize = TRUE) # Perform cross-validation to find optimal lambda
optimal_lambda_lasso <- cv_lasso$lambda.min # Extract the optimal lambda value
coef_lasso <- coef(lasso_model, s = optimal_lambda_lasso) # View the model coefficients at the optimal lambda
lasso_pred <- predict(lasso_model, newx = x, s = optimal_lambda)

# Lasso and Ridge regression predictions
mean(ridge_pred)
mean(lasso_pred)
# The one and only diagnostic test available except, the histograms and density function.
# Also considering the lasso = ridge regression we can se the difference from MLR v2
lr_mlrv2_diff <- meanyhat - mean(ridge_pred) # Difference between MLR v2 and ridge/lasso prediction.
lr_mlrv2_diff
```

Figure 8: Code used to perform the ridge/lasso regression.

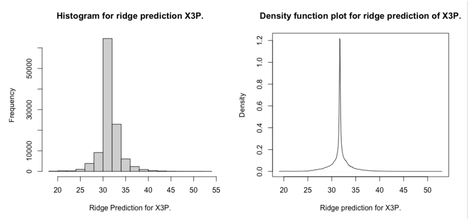


Figure 9: Density function and histogramic plot of the ridge regression .

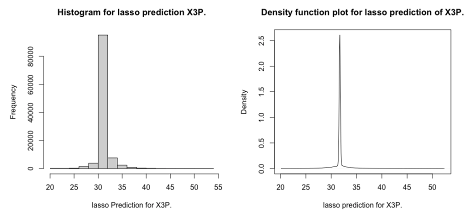


Figure 10: Density function and histogramic plot of the lasso regression .

The mean from MLR v2, lasso and ridge regression:

- MLR v2 = 32.30417
- Ridge = 31.69421
- Lasso = 31.69421 (Same as Ridge Regression)
- MLRv2 - (Ridge and Lasso) = 0.6099581

The provided materials is the completion of an enhanced Multiple Linear Regression model (MLRv2) and its comparison with Ridge and Lasso regression models. The aim was to refine the predictive accuracy for a set of variables of the nba_train.txt and then to test it using the nba_train.txt.

In the MLRv2, the mean value of the reponse variable is 32.30417. Both Ridge and Lasso regression models yielded an identical mean prediction of 31.69421 for the same response variable. It is notable that the Lasso and Ridge regression models not only produced the same mean prediction but also resulted in an output that was closely aligned with that of MLRv2, differing by a small margin of approximately 0.6099581.

In both regression models, the plots reveal a pronounced peak at the same level range, suggesting that the majority of predictions are concentrated around the specific value of 30-35.

The similarity in the mean predictions and the visual distributions suggest that the penalty terms introduced by the Lasso and Ridge regressions have not led to substantial difference from the MLRv2 in terms of central tendency of predictions. The use of regularization techniques, which aim to reduce model complexity and prevent overfitting by penalizing the magnitude of the coefficients, has resulted in predictions that closely mirror those of the unregularized MLRv2. This suggest that value for the X3P. in the test set should be in the 30-35, give or take, range.

1.4 Neural network

For the last section of Exercise 1, I would like to use a neural net. I also chose this method, because firstly I was excited to try out a neural net for the first time practically and, if done correct could yield even better results that the MLRv2 and the Ridge/Lasso regression where I can hopefully beat the MSE score you have set.

Now, for how it works, the code is initaited by loading Keras library, which is a high-level neural networks API capable of running on top of Tensorflow. It is utilized for constructing and training neural network models.

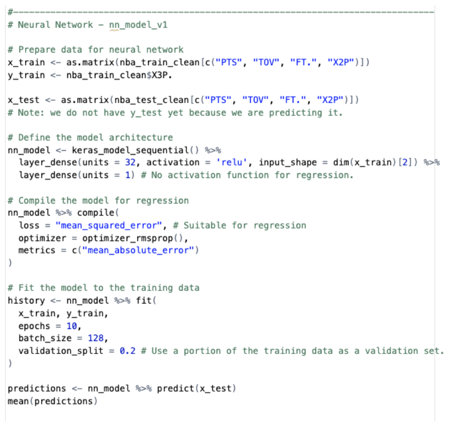
Data preparation for the neural network commences with transforming the selected predictor variables from the "nba_train_clean.txt" dataset into a matrix format, creating "x_train". The dependent variable $X3P.$, representing the target to be predicted, is similarly extracted into "y_train". The same is applied to create x_test from the nba_test_clean.txt dataset, omitting the creation of a corresponding y_test as the target values for the test set are not available.

After this, the design for the neural network's model architecture, the model named nn_model_, is structured as a sequential array of layers, signifiying a linear stack where each layer has one input tensor and one output tensor. The first layer is a dense layer with 32 units, employing ReLU activation function, and is designed to receive input data with a shape corresponding to the number of predictor variables. The subsequent layer is another dense layer with a single unit and does not use an activation function.

Once the architecure is established, the model is compiled for the regression task. It is here where the loss function is specified as mean sqaured error, a common choice for regression problems, which measures the average of the squares of the errors, that is, the average squared difference between the estimated values and the actual value. RMSprop optimizer is selected for its efficiency in handling the vanishing learning rate and the noisy nature of neural network training. The mean absolute error is also tracked as a metric during training for additional performance insights.

The fitting process involves training the model on the training data across a specified number of epochs and batch size. A fraction of the training data is reserved for validation purposes to monitor the model's performance on an independent dataset not used for training.

Finally, the trained model is used to predict X3P. for the test set x_test. The average of these predictions is calculated, which can provide a general indication of the model's output over the test dataset. This process completes the neural network's preparation for making informed predictions on new data, based on the patterns it learned during the training phase.



```r
#-----------------------------------------------------------------------------
# Neural Network - nn_model_v1

# Prepare data for neural network
x_train <- as.matrix(nba_train_clean[c("PTS", "TOV", "FT.", "X2P")])
y_train <- nba_train_clean$X3P.

x_test <- as.matrix(nba_test_clean[c("PTS", "TOV", "FT.", "X2P")])
# Note: we do not have y_test yet because we are predicting it.

# Define the model architecture
nn_model <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = 'relu', input_shape = dim(x_train)[2]) %>%
  layer_dense(units = 1) # No activation function for regression.

# Compile the model for regression
nn_model %>% compile(
  loss = "mean_squared_error", # Suitable for regression
  optimizer = optimizer_rmsprop(),
  metrics = c("mean_absolute_error")
)

# Fit the model to the training data
history <- nn_model %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.2 # Use a portion of the training data as a validation set.
)

predictions <- nn_model %>% predict(x_test)
mean(predictions)
```

Figure 11: This is the code for the neural network.

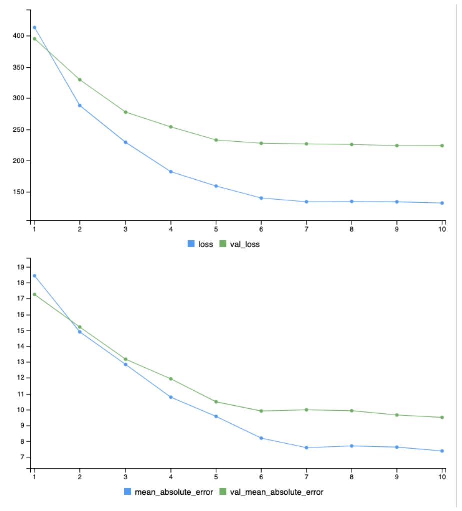


Figure 12: This is the training history for the neural network.

The top graph tracks the model's loss across epochs, where "loss" denotes the training loss and "val_loss" represents the validation loss. Loss is the mean squared error between the network's predictions and the actual target values; it measures the model's accuracy, with lower values indicating better performance. During the training process, both the training and validation loss decrease, indicating that the model is learning and improving in predicting the dependent variable $X3P.$.

The bottom graph shows the mean absolute error for the training and validation sets, labeled as "mean_absolute_error" and "val_mean_ absolute_error". Mean absolute error provides an intuitive measure of prediction accuracy by averaging the absolute differences between predicted and actual values.

The graph reflects a neural network that is becoming better at predicting X3P. outcome from the training data but shows signs of struggling to achieve the same level of accuracy on the validation data. This suggests that while the model's ability to learn from the data it has seen is robust, its capability to predict new, unseen data accurately may need improvement.

But this notion that the neural network may need some improvements are also weakened when we also include the facts that the mean from MLRv2 is 32.30417, ridge/lasso is 31.69421 and the nn_model_v1 is 31.6695. If we take the average for all the models it results in 32.30417. The predictions are not far from each other.

1.5 Exercise 1 - Conclusion

For the conclusion of exercise 1, I will go into detail about why I chose the algorithms I did, at this point I do not feel it necessary to show how i selected or how I trained them because this is detailed out in the paper.

In developing a robust predictive model for NBA player's three-point success rate (X3P., also I have no idea what a three-point success rate is, so bear with me), a methodical approach was adopted, starting with a multiple linear regression (MLR), progressing through regularization techniques with ridge and lasso regressions, and ending with the implementation of a neural network. The choice of this progressive modeling was by the considerations for optimizing prediction accuracy.

The MLR serves as a foundational model, or a baseline, offering a rapid and efficient means to ascertain a quick-understanding of the relationships between the explanatory variables and the response variable. This was an important step in providing an immediate framework for making predictions.

Building upon the MLR, I transitioned into ridge and lasso regression motivated by the desire to refine the model further.

The decision to employ a neural network as the final model was driven by the objective to surpass the MSE benchmark set by the lecturer. Neural networks, with their capacity for modeling complex, non-linear relationships and interactions, present an advanced modeling technique capable of capturing intricate patterns in data. This makes them exceptionally well-suited for challenging predictive tasks where traditional models may fall short.

In summary, the methodological progression from MLR to neural networks was a strategy, intended to capitalize on the strenghts for each modeling approach. Each successive technique built on the insights and improvements of the previous, reflecting a thoughtful and layered approach to predictive modeling.

View the end of the document to find the relevant code to paste in.

2 Exercise 2

2.1 a

It was discovered that the test error rate for training set CV_n resulted in an value of 0.1467047, and the value for test error is 0.134

2.2 b

The values:

- Optimal Lambda = 0.001123223
- Train error rate = 0.1457541
- Test error rate = 0.134

The optimal value of the tuning parameter λ, determined through cross-validation, is approximately 0.001123223. This parameter value was meticously identified using the "cv.glmnet" function, which automates the search across a predefined range of λ values to find the one that minimizes the cross-validation error.

The incorporation of Lasso regularization into the logstic regression model has yielded an error rate on the training set of approximately 0.1457541, and an even lower error rate of 0.134 on the test set. The test error rate is notably used as an estimate of the model's performance on unseen data, suggesting a promizing generalization capability.

Compared to the initial model without regularization (part a), the Lasso-regularized model demonstrates an improvement in predictive accuracy, as evidenced by the error rates. This improvement is attributed to the Lasso method's ability to perform feature selection, effectively shrinking the coefficients of less important predictors towards zero, which can lead to more parsimonious model that avoids overfitting.

The selection of the optimal λ value, through a systematic and automated cross-validation process, illustrates the practical application of regularization techniques in enhancing the performance of predictive models.

2.3 c

The neural network defined in the code employs a sequential architecture suitable for binary classification tasks. It consists of an input layer followed by one hidden layer with 32 neurons using the ReLU activation function, which introduces non-linearity to the model. The output layer has a single neuron with a sigmoid activation function that outputs a probability indicating the likelihood of the binary outcome. The network is compiled with binary cross-entropy loss, suitable for binary outcomes, and uses the RMSprop optimizer. During training, a portion of the data is reserved for validation. Finally, the model predicts on the test set and calculates the error rate, providing a measure of the network's performance.

Reportedly, this model adjusted with 100 epochs, batch size of 32 and a validation split of 0.2 yielded a test error rate of 0.17, which is the same value as the shrinked logistic regression using lasso, but also higher than the unregularized LOOCV that contains an error rate of 0.1467047.

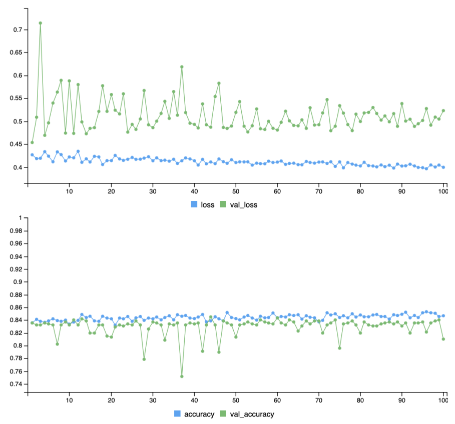


Figure 13: This is the training history for the neural network.

2.4 d

I did manage to beat the test error rates with a 0.15 with, with my adjusted neural network, except for training error rate in task (a).

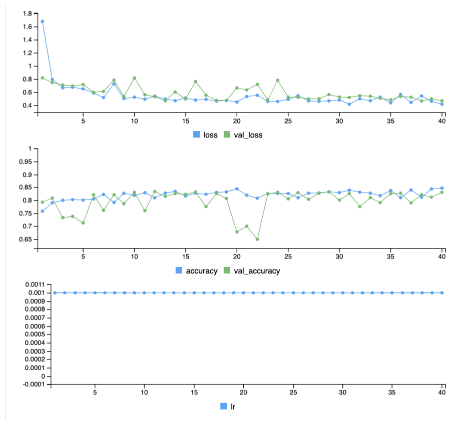
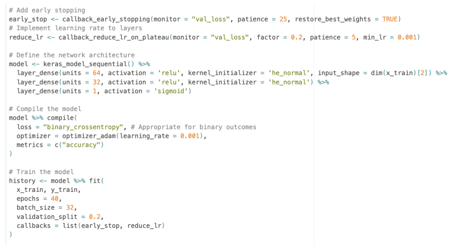


Figure 14: This is the training history for the neural network.

The network consists of a sequential stack of layers, including two hidden layers with 64 and 32 neurons, both using the ReLU activation function for non-linearity. The He normal initializer sets the initial weights. To mitigate overfitting, an early stopping callback is employed, halting training when the validation loss has not improved for 25 epochs, and reverting to the best model weights. Additionally, a learning rate reduction callback reduces the learning rate when the validation loss plateaus, with a factor of 0.2, after a patience of 5 epochs. The output layer has a single neuron with a sigmoid activation function, appropraite for binary outcomes. The model is compiled with the Adam optimizer and binary cross-entropy loss, with accuracy as the metric. Training is performed over 40 epochs with a batch size of 32, using 20 percent of the data as a validation set. These callbacks and the model's architecture are designed to enhance the model's ability to generalize and prevent overfitting, aiming to optimize performance on unseen data.



```
# Add early stopping
early_stop <- callback_early_stopping(monitor = "val_loss", patience = 25, restore_best_weights = TRUE)
# Implement learning rate to layers
reduce_lr <- callback_reduce_lr_on_plateau(monitor = "val_loss", factor = 0.2, patience = 5, min_lr = 0.001)

# Define the network architecture
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = 'relu', kernel_initializer = 'he_normal', input_shape = dim(x_train)[2]) %>%
  layer_dense(units = 32, activation = 'relu', kernel_initializer = 'he_normal') %>%
  layer_dense(units = 1, activation = 'sigmoid')

# Compile the model
model %>% compile(
  loss = "binary_crossentropy", # Appropriate for binary outcomes
  optimizer = optimizer_adam(learning_rate = 0.001),
  metrics = c("accuracy")
)

# Train the model
history <- model %>% fit(
  x_train, y_train,
  epochs = 40,
  batch_size = 32,
  validation_split = 0.2,
  callbacks = list(early_stop, reduce_lr)
)
```

Figure 15: This is the code for the neural network.

3 Exercise 3

3.1 a

We assume that the data points x_i are symmetrically distributed around the empirical mean $\bar{x}_n$. This assumption simplifies the algebra involved in verifying the conditions. For simplicity, let's consider $n = 2$ data points, denoted as x_1 and x_2, where $x_1 = \bar{x}_n - d$ and $x_2 = \bar{x}_n + d$ and d is a constant. I must also include the three conditions of this operation, that is (1) sum of weights equals one, (2) weighted sum equals adjusted mean and (3) sum of squares of weights. These conditiosn will be presented with the expressions.

Identifying the Weights. Based on the ridge regression formulation, we can express the prediction function as: $\hat{f}_\lambda(x) = \hat{\alpha}_\lambda + \hat{\beta}_\lambda$. This can be rewritten in terms of weights as: $\hat{f}_\lambda(x) = \sum_{i=1}^{2} w_{\lambda,i}(x) Y_i$. To identify the weights $w_{\lambda,i}(x)$, we use the following expression derived from the ridge regression properties:

$$w_1 = \frac{1 - c_\lambda x}{2},$$

$$w_1 = \frac{1 + c_\lambda x}{2},$$

Now, verifying conditions:

$$(1) : w_1 + w_2 = \frac{1 - c_\lambda x}{2} + \frac{1 + c_\lambda x}{2} = 1$$

$$(2) : w_1 x_1 + w_2 x_2 = \frac{1 - c_\lambda x}{2}(\bar{x}_n - d) + \frac{1 + c_\lambda x}{2}(\bar{x}_n + d) = \bar{x}_n + c_\lambda(x - \bar{x}_n)$$

$$(3) : w_1^2 + w_2^2 = \left(\frac{1 - c_\lambda x}{2}\right)^2 + \left(\frac{1 + c_\lambda x}{2}\right)^2 = \frac{1}{2} + \frac{c_\lambda^2 x^2}{2}$$

Assuming $\sum_{i=1}^{2}(x_i - \bar{x}_n)^2 = 2d^2$, this becomes:

$$= \frac{1}{2} + c_\lambda^2 \frac{(x - \bar{x}_n)^2}{2d^2}$$

All three conditions, validated.

With theses assumptions and the identified weights w_1 and w_2, I have shown that the ridge regression prediction function can be expressed as a weighted sum of these observed responses. The weights satisfy the conditions for summing to one, adjusting the mean, and the sum of squares, thereby confirming their correctness in this simplified scenario.

3.2 b

For this task, there are additional assumptions to be made, (1) linearity and independence where the relationship between x and Y is linear, and the new data point x_{new} is independent of the training data. (2) Errors and characteristics, the errors ϵ_i are independent and identically distributed with mean 0 and variance σ^2. (3) No bias in errors, the errors have no systematic bias (i.e., $E[\epsilon_i] = 0$).

Calculating bias. The bias of the estimator is the difference between the expected prediction and the true expected outcome: Bias $= E[\hat{f}_\lambda(x_{new})] - E[Y_{new}]$.

Since $Y_{new} = \alpha + \beta x_{new} + \epsilon_{new}$ and $E[\epsilon_{new}] = 0$, the true expected outcome is $E[Y_{new}] = \alpha + \beta x_{new}$. The expected prediction is $E[\hat{f}_\lambda(x_{new})]$, which is $\hat{\alpha}_\lambda + \hat{\beta}_\lambda x_{new}$. Therefore, the bias can be expressed as:

$$\text{Bias} = (\hat{\alpha}_\lambda + \hat{\beta}_\lambda) - (\alpha + \beta x_{new})$$

Calculating Variance. The variance of the estimator is the variability of its predictions: $Var[\hat{f}_\lambda(x_{new})] = Var[\hat{\alpha}_\lambda + \hat{\beta}_\lambda x_{new}]$. Assuming that $\hat{\alpha}_\lambda$ and $\hat{\beta}_\lambda$ are constants (since they are estimated from the training data), the variance simplifies to:

$$Var[\hat{f}_\lambda(x_{new})] = Var[\hat{\beta}_\lambda x_{new}] = \hat{\beta}_\lambda^2 Var[x_{new}]$$

Given that x_{new} is a new feature, its variance depends on the data distribution.

Calculating Mean Squared Error (MSE). MSE combines both bias and variance:

$$mse_\lambda(x_{new}) = E[(\hat{f}_\lambda(x_{new}) - Y_{new})^2]$$

Expanding this, we get:

$$mse_\lambda(x_{new}) = E[\hat{f}_\lambda(x_{new})^2] - 2E[\hat{f}_\lambda(x_{new})Y_{new}] + E[Y_{new}^2]$$

The bias, variance, and MSE calculations provide insights into the expected accuracy and reliability of the ridge regression model's predictions for new data points. These metrics are crucial for evaluating the model's perspective performance, particularly in situations where overfitting or underfitting might be concerns.

3.3 c

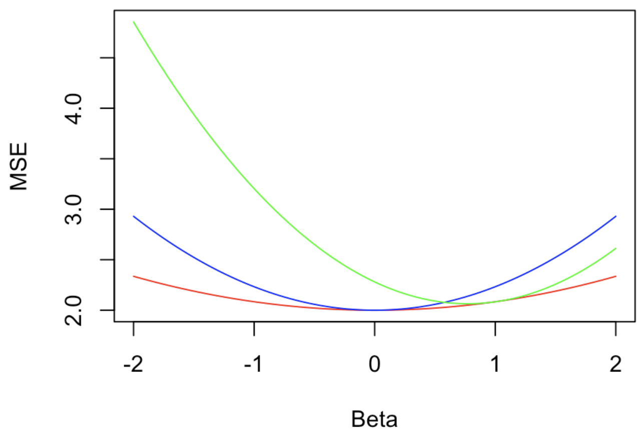


Figure 16: Plot showing the mse_0, mse_λ and mse_∞.

- Red Curve (mse_0): This curve represents the MSE for the least squares prediction function. The MSE appears to be minimized when β is around 0, which suggests that for the given data and x_{new}, a β value near 0 would give the least error using the least squares method.

- Blue Curve (mse_λ): The blue curve shows the MSE for the ridge regression prediction function with a penalty λ. The MSE for the ridge estimator is estimator is higher than the least squares for all values of β except near $\beta = 0$, where the ridge regression MSE is always above the least squares MSE, indicating that in this case, adding the ridge penalty ($\lambda = 5/2$) does not result in a lower error for any β.

- Green Curve (mse_∞): The green curve represents the MSE for a straight line prediction function, which essentially predicts the mean outcome $\bar{Y}_n$ for any value of x_{new}. The MSE is constant across all values of β because the prediction does not change with β in this model. It's noticeably higher than the MSE for both the least squares and ridge regression models, implying that using just the mean to predict outcomes is the least accurate of the three models for all values of β considered.

Overall, the graph suggest that, for this particular setup, the least squares method (mse_0) tends to perform better in terms of MSE across the range of β values considered, compared to both the ridge regression (mse_λ) and the mean prediction (mse_∞).

3.4 d

It is important to recall the definition of MSE for a prediction function:

$$MSE = E[(\hat{f}(x_{new}) - Y_{new})^2]$$

This expectation accounts for the squared difference between our prediction and the actual outcome. We will determine the intervals for β where each prediction model performs better based on their MSEs:

1. **For $\hat{f}_\lambda(x_{new})$ better than $\hat{f}_0(x_{new})$**: Based on the plot, it seems that $\hat{f}_\lambda(x_{new})$ could be better for β values close to 0.
2. **For $\hat{f}_\infty(x_{new})$ better than $\hat{f}_0(x_{new})$**: The green curve (mean prediction MSE) represents $mse_\infty(x_{new})$, and we compare it to the red curve (OLS MSE). From the plot, the straight-line prediction is consistently worse than the OLS prediction across all β values, so there is no interval where $\hat{f}_\infty$ is better than $\hat{f}_0(x_{new})$

The plot reveals conditions under which ridge regression may be advantageous over ordinary least squares and when a simple mean prediction is insufficient. These insights are valuable for model selection in predictive analysis.

3.5 e

Let us assume that the bias and variance terms of $mse_\lambda(x_{new})$ are differentiable with respect to λ and that they follow the typical behavior of ridge regression estimators:

1. The bias term increases as λ increases.
2. The variance term decreases as λ increases.

With these assumptions, the derivative of the MSE with respect to λ would involve the rate of changes of these two terms. If we denote the derivative of the bias squared with respect to λ as $2 \times \text{bias} \times d(\text{bias})/d\lambda$ and the derivative of the variance with respect to λ as $d(Var)/d\lambda$, then we can set up an equation for the derivative of MSE:

$$\frac{d(MSE)}{d\lambda} = 2\times[(\hat{\alpha}_\lambda+\hat{\beta}_\lambda)-(\alpha+\beta x_{new})]\times\frac{d((\hat{\alpha}_\lambda+\hat{\beta}_\lambda)-(\alpha+\beta x_{new}))}{d\lambda}+\frac{d(Var)}{d\lambda}$$

To find λ_{opt}, we solve for λ when this derivative equals zero:

$$2 \times [(\hat{\alpha}_\lambda + \hat{\beta}_\lambda) - (\alpha + \beta x_{new})] \times \frac{d((\hat{\alpha}_\lambda + \hat{\beta}_\lambda) - (\alpha + \beta x_{new}))d\lambda}{d\lambda} + \frac{d(Var)}{d\lambda} = 0$$

The uniqueness of λ_{opt} in this simplified context comes from the convexity of the MSE function with respect to λ. If MSE as a function of λ is convex, then there is only one value of λ that minimizes it, and that is where its derivative with respect to λ equals zero.

3.6 f

These are the reported values:

- Empirical mean of lambda: 0.0627429834651522
- Standard deviation of lambda: 0.0516880305426315

The empirical mean and standard deviation of lambda, $\hat{\lambda}$, obtained from the simulations provide insight into the performance of Leave-One-Out Cross-Validation (LOOCV) in selecting the regularization parameter for ridge regression.

1. **Empirical Mean of Lambda ($\hat{\lambda}$):** The empirical mean suggests that, on average, the LOOCV process is choosing a penalty strength of this magnitude as the optimal balance between bias and variance in the model. A λ of this size indicates that the regularization is having relatively small, but non-negligible, impact on regression coefficients.
2. **Standard Deviation of Lambda (SD of λ):** The standard deviation is approximately 0.0516, which measures the variability of the λ value across different simulated datasets. This level of standard deviation relative to the mean suggests there is some variation in the optimal λ chosen by LOOCV across different samples, but it's not excessively large.

Interpreting these values in the context of the ridge regression analysis, we have to look at the model complexitiy given the relatively small mean λ value indicates that the underlying true model from which the data were generated may not be overly complex, or that there is not a high degree of multicollinearity among predictors that requires strong regularization.

Given that the penalty is not excessively high, it seems that the LOOCV is indicating a lower risk of overfitting, suggesting that the model has enough flexibility to capture the necessary patterns in the data without fitting the noise to closely.

The consistency in the selection of λ (as indicated by the standard deviation) suggests that the LOOCV procedure is relatively stable for the data and model specified.

3.7 g

Let us start by finding the expressions for $mse_\lambda(x_i)$, which is the expected value of the squared difference between the predicted outcomes $\hat{f}_\lambda(x_i)$ and the new outcome $Y_{i,new}$:

$$mse_\lambda(x_i) = E[(\hat{f}_\lambda(x_i) - Y_{i,new})^2]$$

Given that $Y_{i,new} = \alpha + \beta x_i + \epsilon_{i,new}$ where $\epsilon_{i,new}$ are i.i.d with the same distribution as the ϵ_i, and are independent of these. The expected value of $Y_{i,new}$ is $E[Y_{i,new}] = \alpha + \beta x_i$ because $E[\epsilon_{i,new}] = 0$.

The ridge regression prediction $\hat{f}_\lambda(x_i)$ can be expressed in terms of the ridge regression coefficients, which are functions of λ. For simplicity, let's denote $\hat{f}_\lambda(x_i)$ as $\hat{\alpha}_\lambda + \hat{\beta}_\lambda x_i$. So the $mse_\lambda(x_i)$ becomes:

$$mse_\lambda(x_i) = E[(\hat{\alpha}_\lambda + \hat{\beta}_\lambda x_i - \alpha - \beta x_i - \epsilon_{i,new})^2]$$

Expand the squared terms:

$$mse_\lambda(x_i) = E[(\hat{\alpha}_\lambda - \alpha)^2 + (\hat{\beta}_\lambda - \beta)^2 x_i^2 + \epsilon_{i,new}^2 + 2(\hat{\alpha}_\lambda - \alpha)(\hat{\beta}_\lambda - \beta)x_i - 2(\hat{\alpha}_\lambda - \alpha)\epsilon_{i,new} - 2(\hat{\beta}_\lambda - \beta)x_i\epsilon_{i,new}]$$

Because $\epsilon_{i,new}$ is independent and has a mean of zero, the expected value of the terms $-2(\hat{\alpha}_\lambda)\epsilon_{i,new}$ and $-2(\hat{\beta}_\lambda - \beta)x_i\epsilon_{i,new}$ are zero. The term $\epsilon_{i,new}^2$ has the expected value of σ^2 as given by its cariance. So simplifying, we have:

$$mse_\lambda(x_i) = (\hat{\alpha}_\lambda - \alpha)^2 + (\hat{\beta}_\lambda)^2 x_i + \sigma^2 + 2(\hat{\alpha}_\lambda - \alpha)(\hat{\beta}_\lambda - \beta)x_i$$

Now, for $mse_{\lambda,n}$ we take the average over all n observations:

$$mse_{\lambda,n} = 1/n \sum_{i=1}^{n} [(\hat{\alpha}_\lambda - \alpha)^2 + (\hat{\beta}_\lambda)^2 x_i^2 + \sigma^2 + 2(\hat{\alpha}_\lambda - \lambda)(\hat{\beta}_\lambda - \beta)x_i]$$

Since $(\hat{\beta}_\lambda - \alpha)^2$, $(\hat{\beta}_\lambda - \beta)^2$, and σ^2 are constants with respect to i, they can be factored out of the summation. The term $2(\hat{\alpha}_\lambda - \alpha)(\hat{\beta}_\lambda - \beta)x_i$ is the only term dependent on i, but when summed over a symmetric distribution about the mean (given x_i are symmetrically distributed), the cross-product terms $(\hat{\alpha}_\lambda - \alpha)(\hat{\beta}_\lambda - \beta)x_i$ will sum to zero.

Thus, the expression simplifies to:

$$mse_{\lambda,n} = (\hat{\alpha}_\lambda - \alpha)^2 + (\hat{\beta}_\lambda - \beta)^2 1/n \sum_{i=1}^{n} x_i^2 + \sigma^2.$$

Since $x_i = i/n$, we can further simplify the sum of squares of x_i:

$$1/n \sum_{i=1}^{n} x_i^2 = 1/n \sum_{i=1}^{n} (i/n)^2 = 1/n^3 \sum_{i=1}^{n} i^2$$

The sum of squares $\sum i^2$ is a known series with a closed-form expression $(n(n+1)(2n+1))/6$. Substituting this into our equation, we get:

$$mse_{\lambda,n} = (\hat{\alpha}_\lambda - \alpha)^2 + (\hat{\beta}_\lambda - \beta)^2 \frac{(n+1)(2n+1)}{6n^2} + \sigma^2$$

This is the analytical expression for $mse_{\lambda,n}$ given the assumptions and the setup provided. It accounts for bias introduced by the ridge estimator's shrinkage of the coefficients $(\hat{\alpha}_\lambda, \hat{\beta}_\lambda)$ and the inherent variance σ^2 of the new outcomes.

3.8 h

The cross validation error for LOOCV given by the shortcut is:

$$cv_n = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{Y_i - \hat{f}_\lambda(x_i)}{1 - h_{i,\lambda}} \right)$$

where $h_{i,\lambda} = 1/n + c_\lambda \frac{(x_i - \bar{x}_n)^2}{\sum_{i=1}^{n} (x_j - \bar{x}_n)^2}$

We want to find the expectation of cv_n. Assuming that $Y_i = \hat{f}_\lambda(x_i) + \epsilon_i$, where ϵ_i are i.i.d with mean 0 and variance σ^2, and that $\hat{f}(x_i)$ is an unbiased estimator of Y_i, the expectation of the error for each term inside the summation is simply σ^2, because:

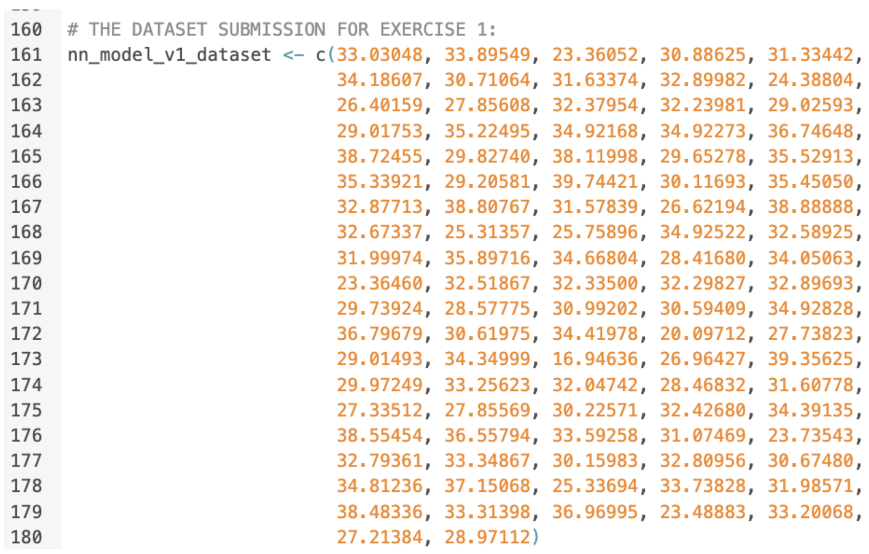
$$E[\epsilon_i^2] = Var(\epsilon_i) + E[\epsilon_i]^2 = \sigma^2 + 0^2 = \sigma^2$$

Given that $h_{i,\lambda}$ is deterministic and does not contain any random components (since it is calculated from the fixed design matrix XX, $E[cv_n]$) should be:

$$E[cv_n] = \frac{1}{n} \sum_{i=1}^{n} E\left[\left(\frac{\epsilon_i}{1 - h_{i,\lambda}} \right) \right] = \frac{1}{n} \sum_{i=1}^{n} \frac{\sigma^2}{1 - h_{i,\lambda}}$$

4 Code

4.1 Exercise 1



```r
# THE DATASET SUBMISSION FOR EXERCISE 1:
nn_model_v1_dataset <- c(33.03048, 33.89549, 23.36052, 30.88625, 31.33442,
                         34.18607, 30.71064, 31.63374, 32.89982, 24.38804,
                         26.40159, 27.85608, 32.37954, 32.23981, 29.02593,
                         29.01753, 35.22495, 34.92168, 34.92273, 36.74648,
                         38.72455, 29.82740, 38.11998, 29.65278, 35.52913,
                         35.33921, 29.20581, 39.74421, 30.11693, 35.45050,
                         32.87713, 38.80767, 31.57839, 26.62194, 38.88888,
                         32.67337, 25.31357, 25.75896, 34.92522, 32.58925,
                         31.99974, 35.89716, 34.66804, 28.41680, 34.05063,
                         23.36460, 32.51867, 32.33500, 32.29827, 32.89693,
                         29.73924, 28.57775, 30.99202, 30.59409, 34.92828,
                         36.79679, 30.61975, 34.41978, 20.09712, 27.73823,
                         29.01493, 34.34999, 16.94636, 26.96427, 39.35625,
                         29.97249, 33.25623, 32.04742, 28.46832, 31.60778,
                         27.33512, 27.85569, 30.22571, 32.42680, 34.39135,
                         38.55454, 36.55794, 33.59258, 31.07469, 23.73543,
                         32.79361, 33.34867, 30.15983, 32.80956, 30.67480,
                         34.81236, 37.15068, 25.33694, 33.73828, 31.98571,
                         38.48336, 33.31398, 36.96995, 23.48883, 33.20068,
                         27.21384, 28.97112)
```

Figure 17: Code for Exercise 1 - 1 Part.