Cross-Validation

Exercise 01

Using the function $x^2 - 2x^3 + 1$, generate random points from the relationship $Y = f(x) + \epsilon$, where ϵ follows a normal distribution with mean = 0 and standard deviation = 0.5. For x values, use uniformly sampled points within $[-1, 1]$. Create two tibbles: one with 50 pairs (xi, yi) for training and another with 100 pairs for testing.

Perform regression on the generated data using (i) linear regression; (ii) the k-nearest neighbors (kNN) method with k = 1; (iii) the kNN method with k = 10. Compare the errors on the 100 pairs used for testing and comment on the results in terms of the Bias-Variance tradeoff.

```
library(tibble)
library(tidymodels)
library(dplyr)

f <- function(x) {
  return(x^2 - 2 * x^3 + 1)
}

# Define the function that generates random points
set.seed(1)
x <- runif(150, -1, 1)
e <- rnorm(150, 0, 0.5)
y <- f(x) + e
data <- data.frame(x, y)


# Create training and testing datasets
test <- slice(data, 1:100)
train <- slice(data, 101:150)


# kNN with k = 10
knn_model_10 <- nearest_neighbor(
  neighbors = 10,
  weight_func = "rectangular",
  dist_power = 2
) %>%
  set_engine("kknn") %>%
  set_mode("regression")
knn_fit_10 <- knn_model_10 %>%
  fit(y ~ x, data = train)


test_pred_10 <- knn_fit_10 %>%
  predict(new_data = test) %>%
```

```
  bind_cols(test)

# kNN with k = 1
knn_model_1 <- nearest_neighbor(
  neighbors = 1,
  weight_func = "rectangular",
  dist_power = 2
) %>%
  set_engine("kknn") %>%
  set_mode("regression")
knn_fit_1 <- knn_model_1 %>%
  fit(y ~ x, data = train)


test_pred_1 <- knn_fit_1 %>%
  predict(new_data = test) %>%
  bind_cols(test)


# Linear Regression
lin_model <- linear_reg() %>%
set_engine("lm") %>%
set_mode("regression")

lin_fit <- lin_model %>%
fit(y ~ x, data = train)

test_pred_reg <- lin_fit %>%
  predict(new_data = test) %>%
  bind_cols(test)

print("KNN, k = 10")
```

```
## [1] "KNN, k = 10"
```

```
print(rmse(test_pred_10, y, .pred))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.527
```

```
print("KNN, k = 1")
```

```
## [1] "KNN, k = 1"
```

```
print(rmse(test_pred_1, y, .pred))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.690
```

```
print("Linear Regression")
```

```
## [1] "Linear Regression"
```

```
print(rmse(test_pred_reg, y, .pred))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.673
```

Note that for k=1, the variance is high because the model effectively 'memorizes' individual data points, resulting in poorer generalization and, consequently, higher error. This model lacks flexibility for the test data, leading to overfitting.

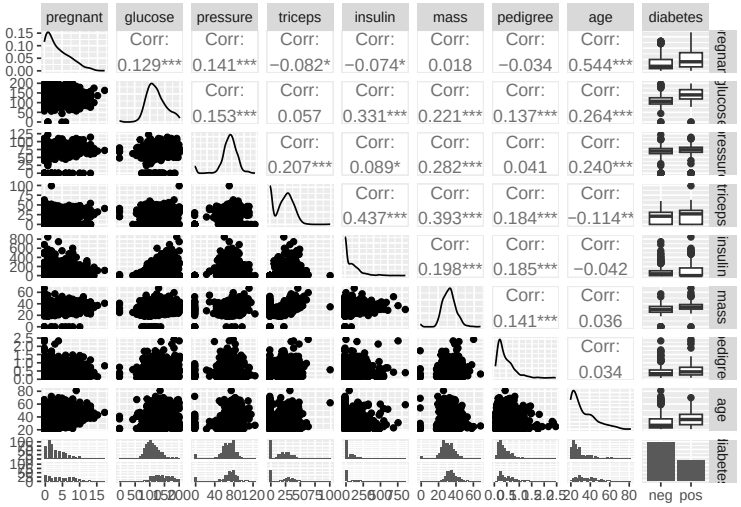
In contrast, linear regression is simpler and therefore has low variance but tends to have higher bias because it doesn't capture nonlinear relationships within the data, resulting in underfitting.

Finally, when k=10, the model strikes a better balance between bias and variance.

Exercise 02

Load the PimaIndiansDiabetes database in R and create a pair plot using ggpairs from the GGally library (or a similar function/library). Describe the insights that can be drawn from this plot.

```
library(tidyverse)
library(mlbench)
library(GGally)
data("PimaIndiansDiabetes")
pima_indians <- as_tibble(PimaIndiansDiabetes)

ggpairs(pima_indians)
```



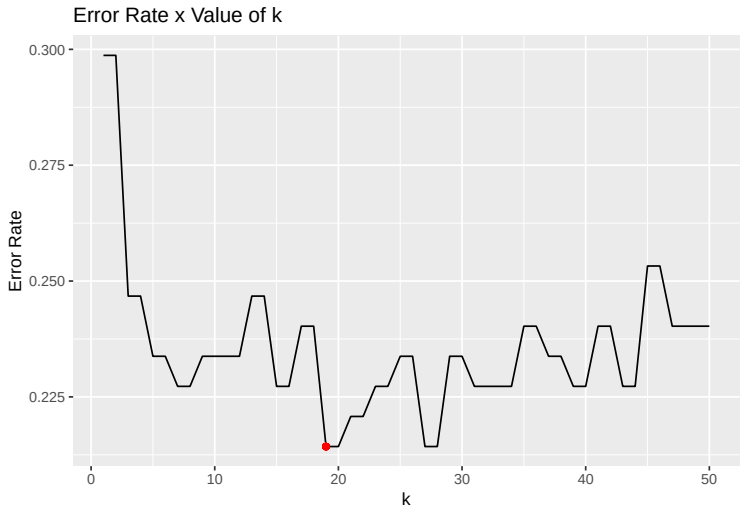
The pair plot visually represents the correlation between each pair of variables in the dataset. It includes scatter plots to visualize the correlations, along with numerical correlation coefficients ranging from -1 (strong negative correlation) to 1 (strong positive correlation). Additionally, the plot displays the distribution of each variable.

Exercise 03

Apply the kNN method to the PimaIndiansDiabetes dataset with different values of k. Employ the test set validation method to determine the optimal k value based on the classification error rate criterion.

```
library(tidyverse)
library(tidymodels)
library(mlbench)
library(yardstick)

# Function that takes the value of k and returns the error rate
accuracy_knn <- function(k, test, train) {

  # Create the kNN model
  knn.model <- nearest_neighbor(neighbors = k,
    weight_func = "rectangular",
    dist_power = 2) %>%
    set_engine("kknn") %>%
    set_mode("classification")

  # Train the model
  knn.fit <- knn.model %>%
    fit( diabetes ~ ., data = train)

  # Make predictions
  test.pred <- knn.fit %>%
    predict( new_data = test) %>%
    bind_cols( test )


  # Calculate the accuracy (acc = 1 - error_rate)
  acc <- accuracy(data = test.pred,
                  truth = diabetes,
                  estimate = .pred_class)

  # Calculate and return the error rate
  error_rate <- 1 - acc$.estimate

  return(error_rate)
}

# Load the PimaIndiansDiabetes dataset
data("PimaIndiansDiabetes")
pima_indians <- as_tibble(PimaIndiansDiabetes)

# Split into training and testing datasets
set.seed(1)
tt.split <- initial_split(pima_indians,prop = 0.8)
training <- training(tt.split)
testing <- testing(tt.split)

# Create a sequence with potential values for k
k_values <- seq(1, 50, by = 1)
```

```r
# Calculate the accuracy - using the function declared above - for each value of k
results <- lapply(k_values, function(k) accuracy_knn(k = k, test = testing, train = training))


# Plot a graph of the error rate vs. value of k
df_result <- data.frame(Parameters = k_values, Result = unlist(results))


best_k <- df_result %>%
  filter(Result == min(Result)) %>%
  pull(Parameters)

plot <- ggplot(df_result, aes(x = Parameters, y = Result)) +
  geom_line() +
  geom_point(x = 19, y = df_result$Result[19], color ="red" ) +
  xlab("k") +
  ylab("Error Rate") +
  ggtitle("Error Rate x Value of k")


print(plot)
```



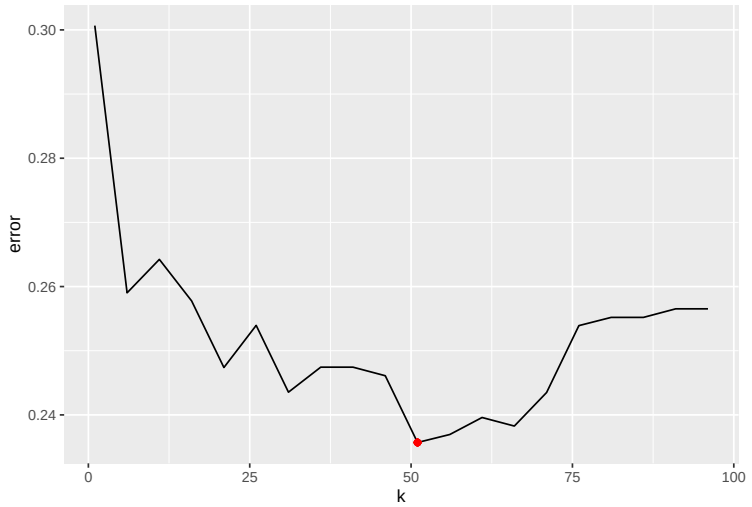

```r
print(paste("The optimal value of k is", min(best_k)))
```

```
## [1] "The optimal value of k is 19"
```

Exercise 04

Apply the kNN method to the PimaIndiansDiabetes dataset with different values of k. Use cross-validation to determine the optimal k value based on the classification error rate criterion. Create a graph that illustrates the classification error vs. 1/k, considering hardware constraints when choosing the appropriate k range (e.g., ranging from 1 to 20 with a step of 2 or 3).

```r
library(tidyverse)
library(tidymodels)
library(mlbench)
library(yardstick)

# Generate 10-fold cross-validation sets
set.seed(1)
cv.split <- vfold_cv(pima_indians,v=10)

run_knn <- function( k )
{
  # Prepare the model
  knn.model <- nearest_neighbor(
    neighbors = k,
    weight_func = "rectangular",
    dist_power = 2
  ) %>%
  set_engine("kknn") %>%
  set_mode("classification")

  # Run the model on cross-validation sets
  results <- fit_resamples( knn.model
  , diabetes ~ .
  , resamples = cv.split
  ) %>%
  collect_metrics() %>%
  filter( .metric == "accuracy" )

  # Store the results
  df <- results %>%
  mutate( k = k, error = 1 - mean ) %>%
  select( -.metric, -.estimator, -.config, -mean)
}

# Run the above function for k ranging from 1 to 100 with a step of 5
df_result <- map_dfr( seq(1, 100, 5), run_knn )

best_k <- df_result %>%
  filter(error == min(error)) %>%
  pull(k)

# Plot the graph
ggplot(df_result) +
aes( x = k, y = error ) +
geom_line() +
geom_point( x = 51, y = df_result$error[11], color ="red" )
```




```
print(paste("The optimal value of k is", best_k))
```

```
## [1] "The optimal value of k is 51"
```

Exercise 05

Repeat Exercise 01 using the function: $f(x) = 2x + 1$. Compare your results to what was obtained in Exercise 01.

```
library(tibble)
library(tidymodels)
library(dplyr)

f <- function(x) {
  return(2 * x + 1)
}

# Define the function that generates random points
set.seed(1)
x <- runif(150, -1, 1)
e <- rnorm(150, 0, 0.5)
y <- f(x) + e
data <- data.frame(x, y)

# Create training and testing datasets
test <- slice(data, 1:100)
train <- slice(data, 101:150)

# kNN with k = 10
knn_model_10 <- nearest_neighbor(
  neighbors = 10,
  weight_func = "rectangular",
  dist_power = 2
) %>%
  set_engine("kknn") %>%
```

```
  set_mode("regression")
knn_fit_10 <- knn_model_10 %>%
  fit(y ~ x, data = train)


test_pred_10 <- knn_fit_10 %>%
  predict(new_data = test) %>%
  bind_cols(test)

# kNN with k = 1
knn_model_1 <- nearest_neighbor(
  neighbors = 1,
  weight_func = "rectangular",
  dist_power = 2
) %>%
  set_engine("kknn") %>%
  set_mode("regression")
knn_fit_1 <- knn_model_1 %>%
  fit(y ~ x, data = train)


test_pred_1 <- knn_fit_1 %>%
  predict(new_data = test) %>%
  bind_cols(test)

# Linear Regression
lin_model <- linear_reg() %>%
set_engine("lm") %>%
set_mode("regression")

lin_fit <- lin_model %>%
fit(y ~ x, data = train)

test_pred_reg <- lin_fit %>%
  predict(new_data = test) %>%
  bind_cols(test)

print("KNN, k = 10")
```

```
## [1] "KNN, k = 10"
```

```
print(rmse(test_pred_10, y, .pred))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.498
```

```
print("KNN, k = 1")
```

```
## [1] "KNN, k = 1"
```

```r
print(rmse(test_pred_1, y, .pred))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.702
```

```r
print("Linear Regression")
```

```
## [1] "Linear Regression"
```

```r
print(rmse(test_pred_reg, y, .pred))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.465
```

Note that in Exercise 1, the test function was nonlinear, resulting in poor performance with the linear regression algorithm. In this case, the provided function is linear, and as a result, linear regression yielded the lowest error rate.