STA 5363, Homework 1

Carson Slater *Baylor University*

(1) In simple regression model, $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$, find $\hat{\beta}_0$ and $\hat{\beta}_1$. Show your work.

Theorem

Given a simple linear regression model with independent observations:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2), \quad i = 1, \ldots, n$$

the parameters minimizing the residual sum of squares are given by:

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x}$$
$$\hat{\beta}_1 = \frac{s_{xy}}{s_x^2}$$

where $\overline{x}$ and $\overline{y}$ are the sample means, s_x^2 is the sample variance of x, and s_{xy} is the sample covariance between x and y.

Proof

The residual sum of squares (RSS) is defined as:

$$\text{RSS}(\beta_0, \beta_1) = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

To find the parameters that minimize the RSS, we take the partial derivatives with respect to β_0 and β_1 and set them to zero.

The derivative with respect to β_0 is:

$$\frac{\partial \text{RSS}}{\partial \beta_0} = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)$$

The derivative with respect to β_1 is:

$$\frac{\partial \text{RSS}}{\partial \beta_1} = -2 \sum_{i=1}^{n} x_i (y_i - \beta_0 - \beta_1 x_i)$$

Setting these derivatives to zero gives the normal equations:

$$\sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$
$$\sum_{i=1}^{n} x_i (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

From the first normal equation, we can solve for $\hat{\beta}_0$:

$$\begin{aligned}\sum y_i - n\hat{\beta}_0 - \hat{\beta}_1 \sum x_i &= 0 \\ n\hat{\beta}_0 &= \sum y_i - \hat{\beta}_1 \sum x_i \\ \hat{\beta}_0 &= \overline{y} - \hat{\beta}_1 \overline{x}\end{aligned}$$

Substituting this expression for $\hat{\beta}_0$ into the second normal equation allows us to solve for $\hat{\beta}_1$:

$$\begin{aligned}\sum x_i y_i - \sum x_i(\overline{y} - \hat{\beta}_1 \overline{x}) - \hat{\beta}_1 \sum x_i^2 &= 0 \\ \sum x_i y_i - \overline{y} \sum x_i + \hat{\beta}_1 \overline{x} \sum x_i - \hat{\beta}_1 \sum x_i^2 &= 0 \\ \hat{\beta}_1 \left(\sum x_i^2 - \overline{x} \sum x_i \right) &= \sum x_i y_i - \overline{y} \sum x_i\end{aligned}$$

This simplifies to the final expression for $\hat{\beta}_1$:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} = \frac{s_{xy}}{s_x^2}$$

(2) In multiple regression model, $y = \mathbf{X}\beta + \epsilon$, find $\hat{\beta}$. Show your work.

Theorem

Given a linear regression model with independent observations, in matrix notation:

$$y = \mathbf{X}\beta + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

the parameter vector β that minimizes the residual sum of squares is given by the ordinary least squares (OLS) estimator:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$

Proof

The residual sum of squares (RSS) is defined as:

$$\text{RSS}(\beta) = \sum_{i=1}^{n} \epsilon_i^2 = \epsilon^T \epsilon = (y - \mathbf{X}\beta)^T (y - \mathbf{X}\beta)$$

This can be expanded as:

$$\begin{aligned} \text{RSS}(\beta) &= (y^T - (\mathbf{X}\beta)^T)(y - \mathbf{X}\beta) \\ &= (y^T - \beta^T\mathbf{X}^T)(y - \mathbf{X}\beta) \\ &= y^T y - y^T\mathbf{X}\beta - \beta^T\mathbf{X}^T y + \beta^T\mathbf{X}^T\mathbf{X}\beta \end{aligned}$$

Since $\beta^T\mathbf{X}^T y$ is a scalar, it is equal to its transpose, $(y^T\mathbf{X}\beta)^T = y^T\mathbf{X}\beta$. Therefore, the expression simplifies to:

$$\text{RSS}(\beta) = y^T y - 2\beta^T\mathbf{X}^T y + \beta^T\mathbf{X}^T\mathbf{X}\beta$$

To find the minimum, we take the derivative of the RSS with respect to β and set it to zero. Using matrix calculus rules, the derivative is:

$$\frac{\partial \text{RSS}(\beta)}{\partial \beta} = -2\mathbf{X}^T y + 2\mathbf{X}^T\mathbf{X}\beta$$

Setting the derivative to zero and replacing β with $\hat{\beta}$:

$$\begin{aligned} 0 &= -2\mathbf{X}^T y + 2\mathbf{X}^T\mathbf{X}\hat{\beta} \\ 2\mathbf{X}^T\mathbf{X}\hat{\beta} &= 2\mathbf{X}^T y \\ \mathbf{X}^T\mathbf{X}\hat{\beta} &= \mathbf{X}^T y \end{aligned}$$

Assuming that the matrix $\mathbf{X}^T\mathbf{X}$ is invertible, we can pre-multiply both sides by $(\mathbf{X}^T\mathbf{X})^{-1}$ to solve for $\hat{\beta}$:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$

This is the OLS estimator for the parameters of a multiple linear regression model.

(3) Create a new R function that allows you to perform K-nearest neighbor (KNN) classification when K is given.

```r
knn_classifier <- function(train_data, train_labels, test_data, k = 3) {
  # Ensure inputs are data frames
  train_data <- as.data.frame(train_data)
  test_data <- as.data.frame(test_data)

  # Sanity checks
  if (nrow(train_data) != length(train_labels)) {
    stop("Number of rows in train_data must match length of train_labels.")
  }
  if (ncol(train_data) != ncol(test_data)) {
    stop("Train and test data must have the same number of columns.")
  }

  # Identify numeric and categorical columns
  is_numeric <- sapply(train_data, is.numeric)
  is_factor <- sapply(train_data, is.factor)

  # Normalize numeric columns
  normalize <- function(x) (x - min(x)) / (max(x) - min(x))
  train_data[is_numeric] <- lapply(train_data[is_numeric], normalize)
  test_data[is_numeric] <- lapply(test_data[is_numeric], normalize)

  # Calculate distances
  calc_distance <- function(test_row) {
    distances <- numeric(nrow(train_data))

    for (i in 1:nrow(train_data)) {
      num_diff <- train_data[i, is_numeric] - test_row[is_numeric]
      num_dist <- sum(num_diff2)

      cat_diff <- train_data[i, is_factor] != test_row[is_factor]
      cat_dist <- sum(cat_diff)

      distances[i] <- sqrt(num_dist) + cat_dist  # Combine distances
    }

    distances
  }

  # Predict for each row in test_data
  predictions <- sapply(1:nrow(test_data), function(i) {
    dists <- calc_distance(test_data[i, ])
    nearest_indices <- order(dists)[1:k]
```

```
    nearest_labels <- train_labels[nearest_indices]
    pred <- names(sort(table(nearest_labels), decreasing = TRUE))[1]
    pred
  })

  bind_cols(test_data, tibble("preds" = predictions))
}
```

(4) Create a new R function to find an optimal K in K-nearest neighbor (KNN) classification.

```
find_optimal_k <- function(
    train_data,
    train_labels,
    k_range = 1:20,
    n_folds = 10,
    seed = 123,
    n_cores = NULL
) {

  # Ensure k_range contains valid integers
  k_range <- floor(k_range[k_range > 0])
  stopifnot(length(k_range) > 0)

  # Determine number of cores to use
  if (is.null(n_cores)) {
    n_cores <- max(1, availableCores() - 1)  # Leave one core free
  }

  # Set up parallel processing
  plan(multisession, workers = n_cores)

  # Create folds for cross-validation
  set.seed(seed) # for reproducibility of folds
  fold_indices <- sample(rep(1:n_folds, length.out = nrow(train_data)))

  # Function to evaluate a single k value
  evaluate_k <- function(k) {
    fold_accuracies <- numeric(n_folds)

    # Perform n-fold cross-validation for this k
    for (i in 1:n_folds) {
      # Split data into training and validation sets for this fold
      validation_idx <- which(fold_indices == i)
      cv_train_data <- train_data[-validation_idx, ]
      cv_validation_data <- train_data[validation_idx, ]
      cv_train_labels <- train_labels[-validation_idx]
      cv_validation_labels <- train_labels[validation_idx]

      # Get predictions using knn_classifier function
      results <- knn_classifier(
        cv_train_data,
        cv_train_labels,
        cv_validation_data,
        k
```

```
      )

      # Extract predictions from the results
      predictions <- results$preds

      # Calculate accuracy for this fold
      accuracy <- sum(predictions == cv_validation_labels) /
        length(cv_validation_labels)
      fold_accuracies[i] <- accuracy
    }

    # Return the mean accuracy for the current k value
    mean(fold_accuracies)
  }

  # Parallel computation across k values using furrr
  mean_accuracies <- future_map_dbl(
    k_range,
    evaluate_k,
    .options = furrr_options(seed = TRUE)
  )

  # Reset to sequential processing
  plan(sequential)

  # Print results for each k
  for (j in seq_along(k_range)) {
    cat(sprintf("k = %-2d, Mean CV Accuracy = %.4f\n", k_range[j], mean_accur
  }

  # Create a results data frame
  cv_results <- data.frame(k = k_range, accuracy = mean_accuracies)

  # Find the k that gave the highest mean accuracy
  optimal_k <- cv_results$k[which.max(cv_results$accuracy)]

  cat(
    sprintf("\nOptimal k found: %d with an accuracy of %.4f\n",
    optimal_k,
    max(cv_results$accuracy))
  )

  list(optimal_k = optimal_k, results = cv_results)
}
```

(5) Analyze the `Carseats` data set using the function built in (2). The outcome variable `High` takes on a value of Yes if the `Sales` variable exceeds 8, and takes on a value of No otherwise.

```
library("ISLR2")
data(Carseats)
carseats <- Carseats |>
  mutate(high = factor(ifelse(Sales <= 8, "No", "Yes")))
rm(Carseats)
```

(5a) Split the data randomly into training and test sets.

```
# Basic 80/20 split
data_split <- initial_split(carseats, prop = 0.8)

# Extract training and testing sets
train_data <- training(data_split) |> select(-Sales)
test_data <- testing(data_split)[, -1]
```

(5b) Consider KNN to predict `High` using all variables but `Sales`. Find the optimal value of K in KNN.

```
optim_k <- find_optimal_k(
  train_data = train_data |> select(-high),
  train_labels = train_data$high,
  k_range = 1:30
)
```

```
## k = 1 , Mean CV Accuracy = 0.6531
## k = 2 , Mean CV Accuracy = 0.6813
## k = 3 , Mean CV Accuracy = 0.7063
## k = 4 , Mean CV Accuracy = 0.6937
## k = 5 , Mean CV Accuracy = 0.7063
## k = 6 , Mean CV Accuracy = 0.7188
## k = 7 , Mean CV Accuracy = 0.6937
## k = 8 , Mean CV Accuracy = 0.7156
## k = 9 , Mean CV Accuracy = 0.7000
## k = 10, Mean CV Accuracy = 0.7094
## k = 11, Mean CV Accuracy = 0.6969
## k = 12, Mean CV Accuracy = 0.7125
## k = 13, Mean CV Accuracy = 0.7125
## k = 14, Mean CV Accuracy = 0.7250
```

```
## k = 15, Mean CV Accuracy = 0.7219
## k = 16, Mean CV Accuracy = 0.7063
## k = 17, Mean CV Accuracy = 0.7031
## k = 18, Mean CV Accuracy = 0.7188
## k = 19, Mean CV Accuracy = 0.7188
## k = 20, Mean CV Accuracy = 0.7125
## k = 21, Mean CV Accuracy = 0.7312
## k = 22, Mean CV Accuracy = 0.7219
## k = 23, Mean CV Accuracy = 0.7156
## k = 24, Mean CV Accuracy = 0.7063
## k = 25, Mean CV Accuracy = 0.7188
## k = 26, Mean CV Accuracy = 0.7156
## k = 27, Mean CV Accuracy = 0.6969
## k = 28, Mean CV Accuracy = 0.6969
## k = 29, Mean CV Accuracy = 0.7063
## k = 30, Mean CV Accuracy = 0.7031
##
## Optimal k found: 21 with an accuracy of 0.7312
```

(5c) Compute test error rate using the KNN with the optimal K.

```
preds <- knn_classifier(
  train_data = train_data |> select(-high),
  train_labels = train_data$high,
  test_data = test_data |> select(-high),
  k = optim_k$optimal_k
)
```

```
## Prediction accuracy with optimal k:  0.725
```

```
## Test error rate with optimal k:  0.275
```