Week11-Assignement

Surenther

2024-11-11

Import CSV

```
# Import CSV
binary_data <- read.table(file = "binary-classifier-data.csv", header = TRUE, sep = ",")
trinary_data <- read.table(file = "trinary-classifier-data.csv", header = TRUE, sep = ",")
```

Scatter Plot for Binary

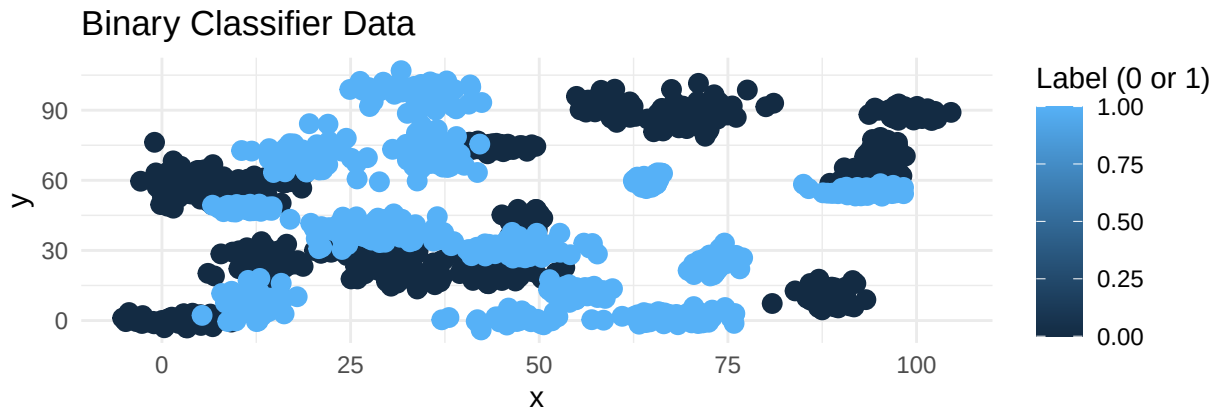
```
library(ggplot2)
binary_plot <- ggplot(binary_data, aes(x = x, y = y, color = label)) +
  geom_point(size = 3) +
  labs(title = "Binary Classifier Data", x = "x", y = "y", color = "Label (0 or 1)") +
  theme_minimal()
```

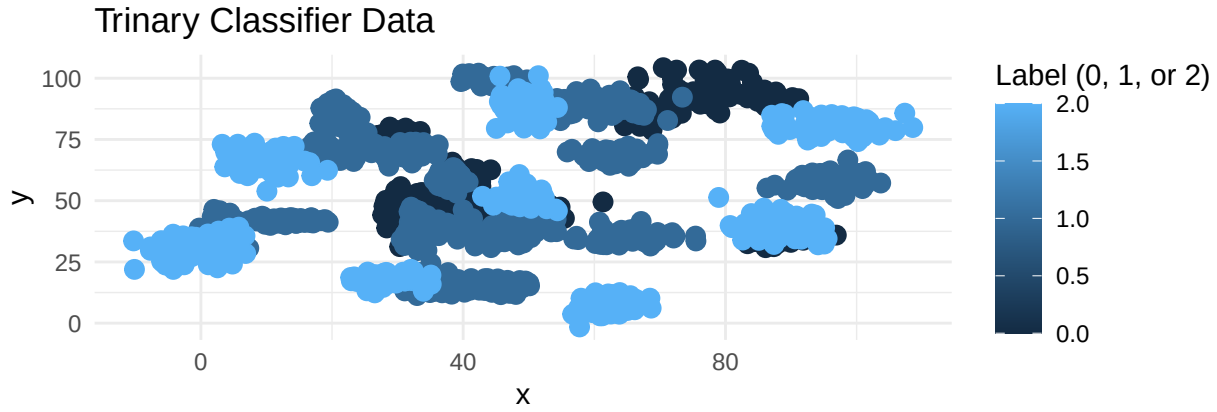
Scatter Plot for Trinary

```
library(ggplot2)
trinary_plot <- ggplot(trinary_data, aes(x = x, y = y, color = label)) +
  geom_point(size = 3) +
  labs(title = "Trinary Classifier Data", x = "x", y = "y", color = "Label (0, 1, or 2)") +
  theme_minimal()
```

Display Plots

```
# Display plots
library(gridExtra)
grid.arrange(binary_plot, trinary_plot, ncol = 1)
```





Euclidean function creation

```r
# Load necessary library
library(dplyr,warn.conflicts = FALSE)

# Define a function to calculate Euclidean distance
euclidean_distance <- function(point1, point2) {
  sqrt(sum((point1 - point2)^2))
}

# Define the k-NN function
knn_predict <- function(train_data, train_labels, test_point, k) {
  # Calculate Euclidean distances from the test point to all training points
  distances <- apply(train_data, 1, function(x) euclidean_distance(x, test_point))

  # Find the k nearest neighbors
  nearest_indices <- order(distances)[1:k]
  nearest_labels <- train_labels[nearest_indices]

  # Predict the label as the most common label among neighbors
  predicted_label <- names(which.max(table(nearest_labels)))
  return(as.numeric(predicted_label))
}
```

Eucliden distance for Binary data

```
# Set the training data and labels
train_data <- binary_data[, c("x", "y")]
train_labels <- binary_data$label

# Define a test point
test_point <- c(32, 95)  # This is the point we want to classify

# Predict the label using k-NN with k = 3
k <- 3
predicted_label <- knn_predict(as.matrix(train_data), train_labels, test_point, k)

# Output the predicted label for the test point
predicted_label
```

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

Findings

A predicted label of 1 means that, based on the 3 nearest points in the training dataset, the majority class label is 1. Therefore, the algorithm assigns the test point (32, 95) to class 1.

Eucliden distance for trinary data

```
# Set the training data and labels
train_data <- trinary_data[, c("x", "y")]
train_labels <- trinary_data$label

# Define a test point
test_point <- c(32, 95)  # This is the point we want to classify

# Predict the label using k-NN with k = 3
k <- 3
predicted_label <- knn_predict(as.matrix(train_data), train_labels, test_point, k)

# Output the predicted label for the test point
predicted_label
```

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

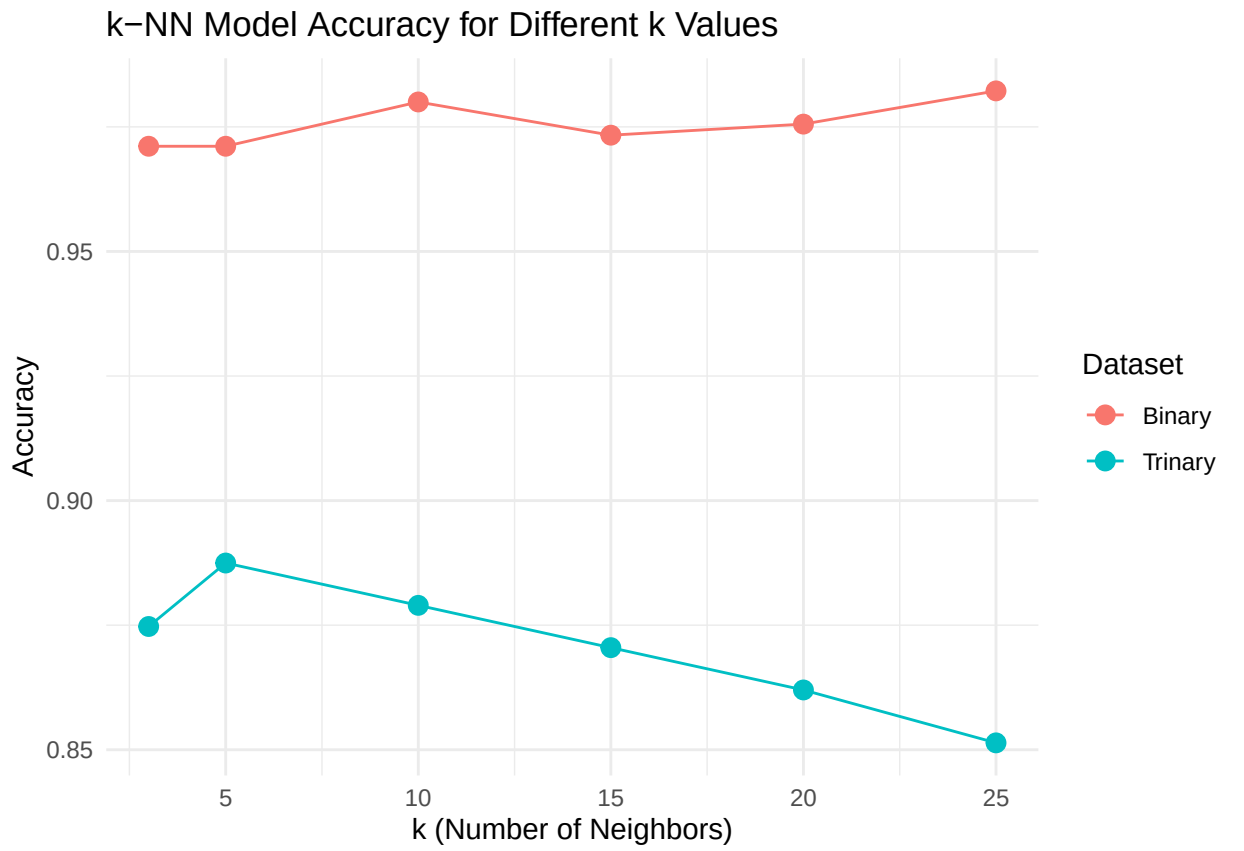
Findings

A predicted label of 1 for the trinary dataset means that, based on the k-nearest neighbors algorithm, the majority of the closest data points around the test point are labeled as 1.

k nearest neighbors' model fit

```r
library(class)
set.seed(123) #for reproducibility

# Split Binary Data
train_index_binary <- sample(1:nrow(binary_data), 0.7 * nrow(binary_data))
train_binary <- binary_data[train_index_binary, ]
test_binary <- binary_data[-train_index_binary, ]

# Split Trinary Data
train_index_trinary <- sample(1:nrow(trinary_data), 0.7 * nrow(trinary_data))
train_trinary <- trinary_data[train_index_trinary, ]
test_trinary <- trinary_data[-train_index_trinary, ]

# Function for K-N accuracy Calculation
calculate_knn_accuracy <- function(train_data, test_data, k_values) {
  accuracies <- c()  # Store accuracies for each k

  # Loop over each k value
  for (k in k_values) {
    # Run k-NN algorithm
    predicted_labels <- knn(
      train = train_data[, c("x", "y")],
      test = test_data[, c("x", "y")],
      cl = train_data$label,
      k = k
    )

    # Calculate accuracy
    accuracy <- mean(predicted_labels == test_data$label)
    accuracies <- c(accuracies, accuracy)
  }

  # Return the k values and their corresponding accuracies
  data.frame(k = k_values, accuracy = accuracies)
}

# Define k values to test
k_values <- c(3, 5, 10, 15, 20, 25)

# Calculate accuracies for each k on the binary dataset
binary_accuracy <- calculate_knn_accuracy(train_binary, test_binary, k_values)

# Calculate accuracies for each k on the trinary dataset
trinary_accuracy <- calculate_knn_accuracy(train_trinary, test_trinary, k_values)

# Add dataset information for easy plotting
binary_accuracy$dataset <- "Binary"
trinary_accuracy$dataset <- "Trinary"

# Combine the results
accuracy_data <- rbind(binary_accuracy, trinary_accuracy)
```

```r
# Plot the results
ggplot(accuracy_data, aes(x = k, y = accuracy, color = dataset)) +
  geom_line() +
  geom_point(size = 3) +
  labs(title = "k-NN Model Accuracy for Different k Values",
       x = "k (Number of Neighbors)",
       y = "Accuracy",
       color = "Dataset") +
  theme_minimal()
```



Findings about linear classifier

Looking at the accuracy plot and considering the data distributions, a linear classifier may not be ideal for these datasets, particularly for the trinary dataset. The lower accuracy trend for trinary data across different k-values suggests that the data may have complex, non-linear boundaries that are not well-captured by a simple linear decision boundary. In contrast, the binary dataset shows relatively high accuracy, which might be manageable by a linear classifier, though a non-linear method could still yield better results for capturing more intricate patterns.

Comparision with Last week accuracy

The k-Nearest Neighbors (k-NN) model achieved significantly higher accuracy on the binary dataset compared to the logistic regression model from last week, which had an accuracy of only 0.4165 (or around 41.7%).

This difference suggests that k-NN may be a better fit for this particular dataset.

The accuracy difference arises because k-NN and logistic regression are fundamentally different approaches.

k-NN is a non-parametric, instance-based method that classifies a new point based on the labels of the closest training points. It can capture non-linear relationships in the data since it doesn't assume any specific form for the decision boundary.

Logistic Regression is a linear model that fits a single linear decision boundary to separate classes. If the underlying pattern in the data is non-linear (as suggested by k-NN's higher accuracy), logistic regression may struggle, resulting in lower accuracy.

Clustering Data

```
# Import CSV
cluster_data <- read.table(file = "clustering-data.csv", header = TRUE, sep = ",")
```

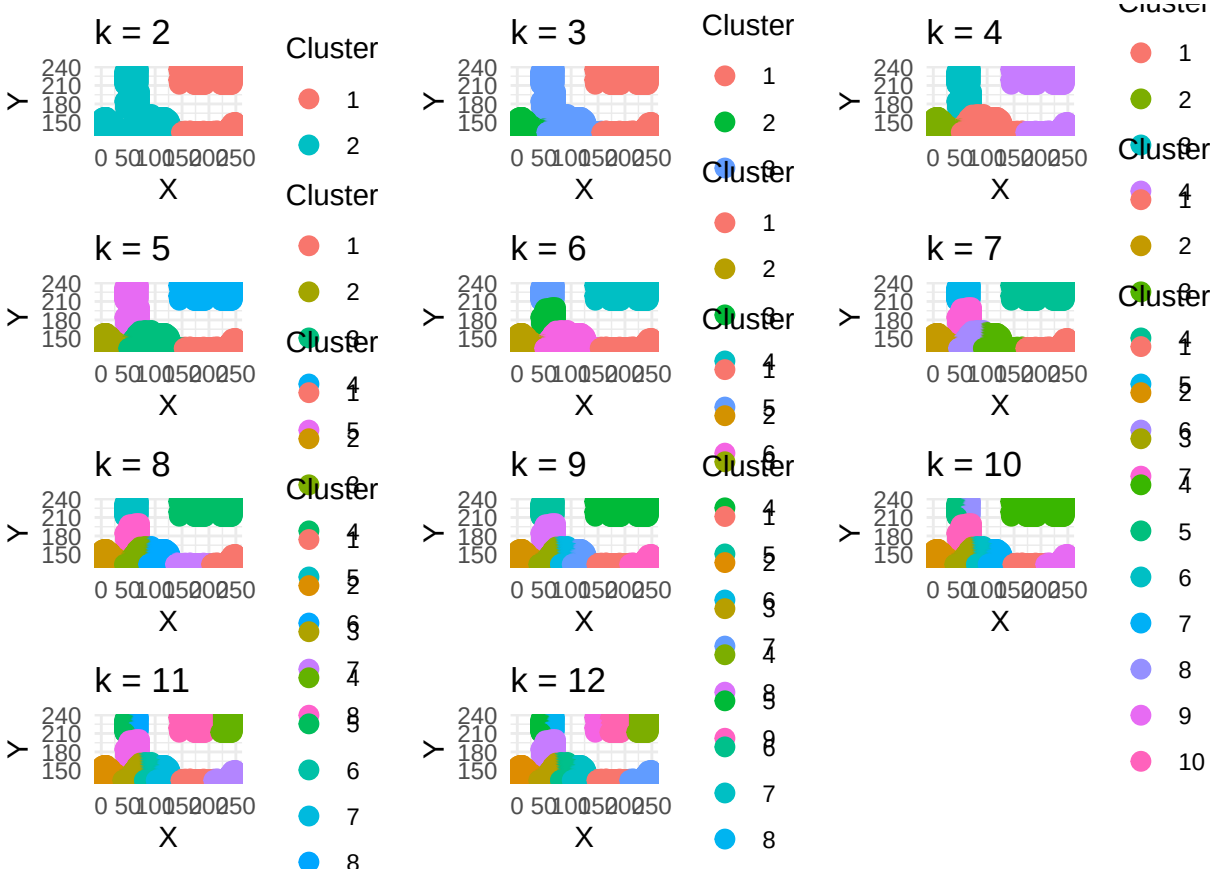
Scatter Plot for Cluster data

```
# Create scatter plot
ggplot(cluster_data, aes(x = x, y = y)) +
  geom_point(color = "blue", size = 3) +
  labs(title = "Scatter Plot of Cluster Data", x = "X Values", y = "Y Values") +
  theme_minimal()
```

Fit dataset using Kmeans

```r
# Prepare to store the plots
plots <- list()

# Fit k-means for k = 2 to k = 12 and store each plot
for (k in 2:12) {
  # Fit k-means
  set.seed(123)  # for reproducibility
  kmeans_result <- kmeans(cluster_data, centers = k)

  # Add cluster assignments to the dataset
  cluster_data$Cluster <- as.factor(kmeans_result$cluster)

  # Create scatter plot with clusters
  plot <- ggplot(cluster_data, aes(x = x, y = y, color = Cluster)) +
    geom_point(size = 3) +
    labs(title = paste("k =", k), x = "X", y = "Y") +
    theme_minimal()

  # Store each plot in the list
  plots[[k - 1]] <- plot
}
```

```r
# Arrange all plots in a grid
grid.arrange(grobs = plots, ncol = 3)
```



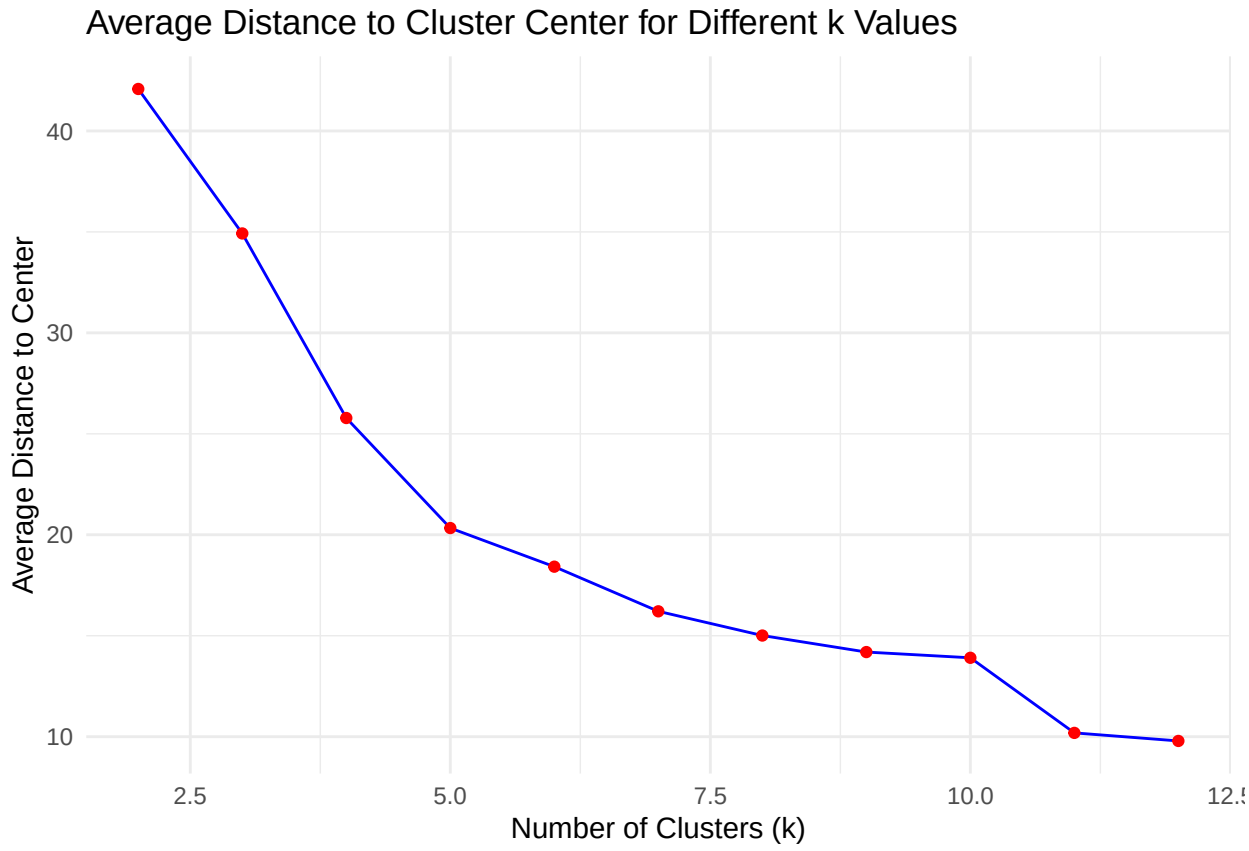
Findings

The chart shows scatter plots of clusters formed by the k-means algorithm for different values of kk (ranging from 2 to 12). As kk increases, the dataset is divided into more clusters, refining the segmentation of data points:

Finding Avg distance from the center

```r
# Explicitly convert x and y columns to numeric (in case they are factors)
cluster_data$x <- as.numeric(cluster_data$x)
cluster_data$y <- as.numeric(cluster_data$y)

# Initialize a data frame to store the average distances for each k
avg_distances <- data.frame(k = integer(), avg_distance = numeric())

# Calculate the average distance for each value of k
for (k in 2:12) {
  # Fit k-means
  set.seed(123)  # for reproducibility
  kmeans_result <- kmeans(cluster_data, centers = k)
```

```r
  # Calculate the distance of each point to its cluster center
  distances <- sqrt(rowSums((cluster_data[, c("x", "y")] - kmeans_result$centers[kmeans_result$cluster,

  # Calculate the average distance and store it
  avg_distance <- mean(distances)
  avg_distances <- rbind(avg_distances, data.frame(k = k, avg_distance = avg_distance))
}

# Plot the average distances as a line chart
library(ggplot2)
ggplot(avg_distances, aes(x = k, y = avg_distance)) +
  geom_line(color = "blue") +
  geom_point(color = "red") +
  labs(title = "Average Distance to Cluster Center for Different k Values",
       x = "Number of Clusters (k)",
       y = "Average Distance to Center") +
  theme_minimal()
```



Findings

The chart shows the average distance to the cluster center for different numbers of clusters (k) in a k-means clustering model. As we increase the number of clusters, the average distance to the center decreases, as expected. This happens because more clusters mean each cluster center is closer to the points it represents, reducing the overall distance.

Elbow Point

In k-means clustering, the "elbow point" is where adding more clusters begins to yield diminishing returns in terms of reducing the average distance. From this chart, the elbow point appears to be around k=4k=4 or k=5k=5. At this point, the rate of decrease in average distance slows down, suggesting that further increasing the number of clusters may not significantly improve the clustering quality. This is often a good choice for kk in practice.