Decoding Sequence Variation: A Computational Approach to AXL and RB1 mRNA Cluster Analysis

Paniz Tayebi

2025-01-13

Introduction

The study of gene expression and its regulation is a cornerstone of modern bioinformatics, providing critical insights into cellular processes and disease mechanisms. This project focuses on the comparative analysis of two significant genes, AXL and RB1, which play pivotal roles in cellular function and are implicated in various cancers. The AXL gene, a member of the TAM (Tyro3, Axl, Mer) receptor tyrosine kinase family, is involved in cell proliferation, migration, and survival, and its dysregulation has been linked to several cancers, including lung, breast, and pancreatic cancers(Linger et al. (2008); Graham et al. (2014)). On the other hand, the RB1 gene, encoding the retinoblastoma protein (pRb), acts as a tumor suppressor by regulating cell cycle progression and preventing excessive cell growth(Burkhart and Sage (2008)).

By examining these genes, I seek to explore whether their distinct biological functions correlate with differences in their sequence conservation, clustering behavior, and intra-gene variability. This analysis will leverage hierarchical clustering (Eisen et al. (1998)) and dimensionality reduction techniques to assess sequence similarity and identify potential subgroups within each gene's dataset.Understanding the sequence diversity of AXL and RB1 is biologically significant because it can provide insights into their evolutionary constraints and functional roles. AXL is known for its involvement in immune response and metastasis, while RB1 is critical for cell cycle regulation. Comparing their sequence structures may reveal whether functional differences are reflected in their genetic variability. Additionally, this project serves as a methodological exploration of sequence clustering techniques, evaluating how different linkage methods (e.g., average, complete, Ward) perform in grouping similar sequences. The findings could inform future studies on gene-specific clustering approaches in bioinformatics.

Description of Data set

The sequence data for this project were obtained from the NCBI Nucleotide database on January 13, 2024, using the search queries "AXL[All Fields] AND 'Homo sapiens'[porgn]" and "RB1[All Fields] AND 'Homo sapiens'[porgn]". The datasets comprise mRNA, coding sequences (CDS), and genomic sequences for each gene. After quality filtering, AXL retained sequences trimmed to a uniform length of 274 bp, while RB1 sequences were trimmed to 146 bp to ensure comparability. The clustering analysis for AXL identified 304 distinct clusters with an average silhouette width of 0.216, indicating moderate separation between groups. In contrast, RB1 formed 69 clusters with a higher average silhouette width (0.659), suggesting stronger within-cluster cohesion. The Dunn and Davies-Bouldin indices further supported these patterns, with AXL showing higher inter-cluster separation (Dunn index = 3.952, DB index = 0.02) compared to RB1 (Dunn index = 1.176, DB index = 0.082).

Code Section 1

```
##### Software Tools - Assignment 5 #####
## Paniz Tayebi
## 2024-13-01 (Data Retrieved from NCBI)

#########################################################
### SECTION 1: PACKAGE INSTALLATION AND LOADING #####
#########################################################

# Load BiocManager - required for installing Bioconductor packages
# quietly=TRUE suppresses messages except errors
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

# Define all required packages in a single vector for easier management
# These include:
# - Biostrings, DECIPHER: Bioconductor packages for biological sequence analysis
# - ggplot2, patchwork, viridis, gridExtra, pheatmap: Data visualization packages
# - cluster, clValid: Clustering and cluster validation packages
# - seqinr: Another sequence analysis package
required_packages <- c("Biostrings", "DECIPHER", "ggplot2", "ape",
                       "patchwork", "cluster", "clValid", "seqinr",
                       "viridis", "gridExtra", "pheatmap")

# Loop through each package and install if not available
# Uses different installation methods based on the package type:
# - BiocManager::install for Bioconductor packages
# - install.packages for CRAN packages
for (pkg in required_packages) {
  if (!require(pkg, character.only = TRUE)) {
    if (pkg %in% c("Biostrings", "DECIPHER")) {
      BiocManager::install(pkg)  # Bioconductor installation
    } else {
      install.packages(pkg)      # CRAN installation
    }
  }
  # Load the package after ensuring it's installed
  # character.only=TRUE allows loading packages from variable names
  library(pkg, character.only = TRUE)
}

# Explicitly load additional packages (redundant with the loop above but ensures they're loaded)
library(Biostrings)  # For biological string manipulation
library(DECIPHER)    # For sequence analysis and alignment
library(vegan)       # For ecological data analysis and diversity metrics
library(ggplot2)     # For creating plots
library(cluster)     # For clustering algorithms
library(styler)      # For code styling/formatting
library(rstudioapi)  # For RStudio API functions (directory management)
library(rentrez)     # For NCBI data retrieval
library(clValid)     # For cluster validation
```

```r
################################################
### SECTION 2: DATA LOADING AND ERROR CHECKING ##
################################################

# Inform user that sequence loading is starting
message("\nLoading sequence data...")

# Check if files exist before attempting to load them
# stop() will halt execution if files aren't found
if (!file.exists[redacted]
  [redacted])
}
if (!file.exists[redacted]
  [redacted])
}

# Load FASTA sequences into DNAStringSet objects
# format="fasta" specifies the file format
axl_fasta <- readDNAStringSet([redacted], format = "fasta")
rb1_fasta <- readDNAStringSet([redacted], format = "fasta")

# Verify that sequences were actually loaded
# length() checks how many sequences were loaded
if (length(axl_fasta) == 0) stop("No sequences loaded for AXL")
if (length(rb1_fasta) == 0) stop("No sequences loaded for RB1")

# Store both sequence sets in a named list for easier access
sequences <- list(AXL = axl_fasta, RB1 = rb1_fasta)

################################################
### SECTION 3: SEQUENCE EXPLORATION FUNCTION #####
################################################

# Define function to explore sequence properties and generate visualizations
# Parameters:
# - seqs: A DNAStringSet object containing sequences
# - gene_name: Name of the gene for labeling outputs
explore_sequences <- function(seqs, gene_name) {
  message("\nExploring ", gene_name, " sequences...")

  # Check if sequences are very long (>10000 bp) and sample if needed
  # This prevents memory issues when analyzing large sequences
  if (mean(width(seqs)) > 10000) {
    warning(gene_name, " sequences are very long (avg ", round(mean(width(seqs))),
            " bp). Using sampling to reduce memory usage.")
    # If there are more than 50 sequences, randomly sample 50 to reduce computation
    if (length(seqs) > 50) {
      seqs <- seqs[sample(1:length(seqs), min(50, length(seqs)))]
    }
  }

  # Identify sequence types based on their names
  # grepl searches for patterns in sequence names
```

```r
# ignore.case=TRUE makes the search case-insensitive
seq_names <- names(seqs)
seq_types <- c(
  mRNA = sum(grepl("mRNA|transcript", seq_names, ignore.case = TRUE)),
  genomic = sum(grepl("genomic|chromosome", seq_names, ignore.case = TRUE)),
  CDS = sum(grepl("CDS|coding", seq_names, ignore.case = TRUE)),
  other = sum(!grepl("mRNA|transcript|genomic|chromosome|CDS|coding",
                     seq_names, ignore.case = TRUE))
)

# Calculate sequence lengths using width()
seq_lengths <- width(seqs)

# Calculate GC content for each sequence
# vapply applies the function to each sequence and returns a numeric vector
gc_content <- vapply(seqs, function(x) {
  # Calculate letter frequencies as proportions (as.prob=TRUE)
  let <- alphabetFrequency(x, as.prob = TRUE)
  # Sum G and C frequencies
  sum(let[c("G", "C")])
}, numeric(1))

# Calculate overall nucleotide frequencies
# width=1 counts single nucleotides, simplify.as="collapsed" combines results
nt_counts <- oligonucleotideFrequency(seqs, width = 1, simplify.as = "collapsed")
total_bases <- sum(nt_counts)
# Calculate proportions of A, C, G, T
nt_freqs <- nt_counts[c("A", "C", "G", "T")] / total_bases

################################################
### Create visualization plots for sequence data
################################################

# Plot sequence length distribution as histogram
# data.frame converts sequence lengths to a data frame for ggplot
# bins=30 divides data into 30 intervals
length_hist <- ggplot(data.frame(Length = seq_lengths), aes(x = Length)) +
  geom_histogram(bins = 30, fill = "#440154", alpha = 0.7, color = "black") +
  ggtitle(paste0(gene_name, " Sequence Length Distribution")) +
  theme_minimal(base_size = 12) +
  labs(x = "Sequence Length (bp)", y = "Count") +
  theme(plot.title = element_text(size = 14, face = "bold"),
        axis.title = element_text(size = 12))

# Plot GC content distribution as histogram
gc_hist <- ggplot(data.frame(GC = gc_content), aes(x = GC)) +
  geom_histogram(bins = 20, fill = "#21908C", alpha = 0.7, color = "black") +
  ggtitle(paste0(gene_name, " GC Content Distribution")) +
  theme_minimal(base_size = 12) +
  labs(x = "GC Content", y = "Count") +
  theme(plot.title = element_text(size = 14, face = "bold"),
        axis.title = element_text(size = 12))
```

```r
# Plot sequence types as bar chart
# scale_fill_viridis_d() uses colorblind-friendly viridis palette for discrete values
type_bar <- ggplot(data.frame(Type = names(seq_types),
                              Count = as.numeric(seq_types))) +
  geom_bar(aes(x = Type, y = Count, fill = Type), stat = "identity") +
  scale_fill_viridis_d() +
  ggtitle(paste0(gene_name, " Sequence Types")) +
  theme_minimal(base_size = 12) +
  labs(x = "Sequence Type", y = "Count") +
  theme(plot.title = element_text(size = 14, face = "bold"),
        axis.title = element_text(size = 12),
        legend.position = "none")  # Removes legend to prevent redundancy

# Plot nucleotide composition as bar chart
# scale_fill_manual provides custom colors for nucleotides
nt_bar <- ggplot(data.frame(Nucleotide = names(nt_freqs),
                            Frequency = nt_freqs)) +
  geom_bar(aes(x = Nucleotide, y = Frequency, fill = Nucleotide),
           stat = "identity") +
  scale_fill_manual(values = c("A" = "#66C2A5", "C" = "#FC8D62",
                               "G" = "#8DA0CB", "T" = "#E78AC3")) +
  ggtitle(paste0(gene_name, " Nucleotide Composition")) +
  theme_minimal(base_size = 12) +
  labs(x = "Nucleotide", y = "Average Frequency") +
  theme(plot.title = element_text(size = 14, face = "bold"),
        axis.title = element_text(size = 12),
        legend.position = "none")  # Removes legend to prevent redundancy

# Create summary statistics data frame
stats <- data.frame(
  Metric = c("Total Sequences", "Min Length", "Median Length", "Mean Length",
             "Max Length", "Mean GC Content", "Sequence Types"),
  Value = c(
    length(seqs),                  # Count of sequences
    min(seq_lengths),              # Minimum length
    median(seq_lengths),           # Median length
    mean(seq_lengths),             # Mean length
    max(seq_lengths),              # Maximum length
    mean(gc_content),              # Mean GC content
    # Create a string showing all sequence types and their counts
    paste(names(seq_types), seq_types, sep = ": ", collapse = ", ")
  )
)

# Return all results as a list
return(list(
  stats = stats,               # Summary statistics
  seq_types = seq_types,       # Sequence type counts
  seq_lengths = seq_lengths,   # Vector of sequence lengths
  gc_content = gc_content,     # Vector of GC content values
  nt_freqs = nt_freqs,         # Nucleotide frequencies
  length_hist = length_hist,   # Length histogram plot
  gc_hist = gc_hist,           # GC content histogram plot
```

```
    type_bar = type_bar,          # Sequence type bar chart
    nt_bar = nt_bar               # Nucleotide composition bar chart
  ))
}

################################################
### SECTION 4: SEQUENCE FILTERING FUNCTION #######
################################################

# Define function to filter sequences based on quality criteria
# Parameters:
# - seqs: A DNAStringSet object containing sequences
# - gene_name: Name of the gene for labeling outputs
filter_sequences <- function(seqs, gene_name) {
  message("\nPerforming quality control on ", gene_name, " sequences...")

  # Store original sequence count for comparison
  original_count <- length(seqs)

  # STEP 1: Filter to keep only mRNA/transcript/CDS sequences
  # is_mRNA is a logical vector marking sequences with relevant terms in their names
  is_mRNA <- grepl("mRNA|transcript|CDS", names(seqs), ignore.case = TRUE)
  seqs_filtered <- seqs[is_mRNA]  # Subset sequences using logical vector
  mRNA_count <- length(seqs_filtered)
  message("  - Retained ", mRNA_count, "/", original_count,
          " mRNA/transcript/CDS sequences")

  # If too few mRNA sequences remain, revert to using all sequences
  if (mRNA_count < 5) {
    warning("Too few mRNA sequences. Using all sequences.")
    seqs_filtered <- seqs
    mRNA_count <- original_count
  }

  # STEP 2: Remove length outliers (keep sequences between 1st and 99th percentile)
  seq_lengths <- width(seqs_filtered)
  # quantile() calculates percentiles (0.01 = 1%, 0.99 = 99%)
  length_q1 <- quantile(seq_lengths, 0.01, na.rm = TRUE)  # 1% percentile
  length_q99 <- quantile(seq_lengths, 0.99, na.rm = TRUE) # 99% percentile
  # Create logical vector for sequences within length range
  length_filter <- seq_lengths >= length_q1 & seq_lengths <= length_q99
  seqs_filtered <- seqs_filtered[length_filter]  # Apply filter
  length_count <- length(seqs_filtered)
  message("  - Removed ", mRNA_count - length_count,
          " length outliers (kept lengths between ",
          round(length_q1), " and ", round(length_q99), " bp)")

  # STEP 3: Remove GC content outliers (keep sequences between 1st and 99th percentile)
  # letterFrequency calculates frequency of specified letters (GC)
  # as.prob=TRUE returns proportions instead of counts
  gc_content <- letterFrequency(seqs_filtered, letters = "GC", as.prob = TRUE)
  gc_q1 <- quantile(gc_content, 0.01, na.rm = TRUE)   # 1% percentile
  gc_q99 <- quantile(gc_content, 0.99, na.rm = TRUE)  # 99% percentile
```

```
  # Create logical vector for sequences within GC content range
  gc_filter <- gc_content >= gc_q1 & gc_content <= gc_q99
  seqs_filtered <- seqs_filtered[gc_filter]  # Apply filter
  gc_count <- length(seqs_filtered)
  message("  - Removed ", length_count - gc_count,
          " GC content outliers (kept GC between ",
          round(gc_q1, 2), " and ", round(gc_q99, 2), ")")

  # STEP 4: Remove sequences with high N content (ambiguous nucleotides)
  # letterFrequency calculates frequency of N's
  n_content <- letterFrequency(seqs_filtered, letters = "N", as.prob = TRUE)
  n_filter <- n_content < 0.05  # Keep sequences with less than 5% N's
  seqs_filtered <- seqs_filtered[n_filter]  # Apply filter
  final_count <- length(seqs_filtered)
  message("  - Removed ", gc_count - final_count,
          " sequences with >5% N content")

  # Report final filtered dataset statistics
  message("Final dataset: ", final_count, "/", original_count,
          " sequences (", round(final_count/original_count*100, 1), "%)")

  # Create summary data frame for filtering steps
  # factor() with levels ensures steps appear in correct order
  filter_summary <- data.frame(
    Step = factor(c("Original", "mRNA only", "Length filter", "GC filter", "N content filter"),
                  levels = c("Original", "mRNA only", "Length filter", "GC filter", "N content filter")),
    Count = c(original_count, mRNA_count, length_count, gc_count, final_count)
  )

  # Create visualization of filtering steps
  # group=1 connects points with line
  filter_plot <- ggplot(filter_summary, aes(x = Step, y = Count, group = 1)) +
    geom_line(color = "blue", size = 1) +
    geom_point(color = "blue", size = 3) +
    ggtitle(paste0(gene_name, " Sequence Filtering Steps")) +
    theme_minimal(base_size = 12) +
    labs(x = "Filtering Step", y = "Remaining Sequences") +
    theme(plot.title = element_text(size = 14, face = "bold"),
          axis.title = element_text(size = 12),
          axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate x-axis labels for readability

  # Return filtered sequences and visualization
  return(list(
    sequences = seqs_filtered,     # Filtered sequence set
    filter_plot = filter_plot,     # Plot showing filtering progression
    filter_summary = filter_summary # Data frame of filtering steps
  ))
}

#####################################################################
### SECTION 5: EXECUTE EXPLORATION AND FILTERING FUNCTIONS ###
#####################################################################
```

```r
# Execute exploration function for both genes and store results
exploration <- list(
  AXL = explore_sequences(sequences$AXL, "AXL"),
  RB1 = explore_sequences(sequences$RB1, "RB1")
)

# Print summary statistics to console
message("\n===== Summary Statistics =====")
print(exploration$AXL$stats)
print(exploration$RB1$stats)

# Define function to combine exploratory plots for both genes
# Parameters:
# - gene1_name: Name of first gene
# - gene1_exp: Exploration results for first gene
# - gene2_name: Name of second gene
# - gene2_exp: Exploration results for second gene
exploratory_plots <- function(gene1_name, gene1_exp, gene2_name, gene2_exp) {
  # Combine length histograms side by side using patchwork
  # plot_layout(ncol=2) arranges plots in 2 columns
  length_plots <- gene1_exp$length_hist + gene2_exp$length_hist + plot_layout(ncol = 2)
  # Combine GC content histograms
  gc_plots <- gene1_exp$gc_hist + gene2_exp$gc_hist + plot_layout(ncol = 2)
  # Combine sequence type bar charts
  type_plots <- gene1_exp$type_bar + gene2_exp$type_bar + plot_layout(ncol = 2)
  # Combine nucleotide frequency bar charts
  nt_plots <- gene1_exp$nt_bar + gene2_exp$nt_bar + plot_layout(ncol = 2)

  # Stack all plot pairs vertically using patchwork's / operator
  # plot_annotation adds a title to the entire figure
  (length_plots / gc_plots / type_plots / nt_plots) +
    plot_annotation(
      title = paste("Exploratory Analysis of", gene1_name, "and", gene2_name, "Sequences"),
      theme = theme(plot.title = element_text(size = 16, face = "bold", hjust = 0.5)))
}

# Create combined exploratory visualization and save to file
exploratory_vis <- exploratory_plots("AXL", exploration$AXL, "RB1", exploration$RB1)
# ggsave saves the plot to a file with specified dimensions and resolution
ggsave("exploratory_analysis.png", exploratory_vis, width = 12, height = 16, dpi = 300)

# Execute sequence filtering for both genes
filtered <- list(
  AXL = filter_sequences(sequences$AXL, "AXL"),
  RB1 = filter_sequences(sequences$RB1, "RB1")
)

# Combine filtering visualizations and save to file
filter_vis <- filtered$AXL$filter_plot + filtered$RB1$filter_plot +
  plot_layout(ncol = 2) +
  plot_annotation(
    title = "Sequence Quality Control and Filtering Process",
    theme = theme(plot.title = element_text(size = 16, face = "bold", hjust = 0.5)))
```

```r
# Save the combined filtering visualization
ggsave("filtering_visualization.png", filter_vis, width = 10, height = 6, dpi = 300)

# Store filtered sequences for further analysis
sequences_for_analysis <- list(
  AXL = filtered$AXL$sequences,
  RB1 = filtered$RB1$sequences
)

# Save all exploration and filtering results to R data file
# This allows results to be loaded later without rerunning analyses
save(exploration, filtered, sequences_for_analysis,
     file = "sequence_exploration_results.RData")

# Print completion messages
message("\nData exploration and quality control complete!")
message("Filtered sequences ready for analysis.")
```

Description of Main Software Tools

This R script utilizes several key bioinformatics and statistical software tools for sequence analysis and visualization. The Biostrings package (Pagès et al., 2024) (from Bioconductor) handles biological sequence manipulation, providing efficient storage and analysis of DNA/RNA/protein sequences. DECIPHER (Wright, 2016) offers tools for sequence alignment, clustering, and phylogenetic analysis, including the DistanceMatrix function used here. For clustering and validation, the script employs cluster (Maechler et al., 2022) for hierarchical clustering and silhouette analysis, alongside clvalid (Brock et al., 2008) for additional cluster validation metrics like the Dunn index. Visualization is powered by ggplot2 (Wickham et al., 2024) for creating publication-quality plots, enhanced by viridis (Garnier, 2021) for colorblind-friendly palettes and patchwork for arranging multi-panel figures. The ape package (Paradis and Schliep, 2019) supports phylogenetic analysis, while vegan (Oksanen et al., 2022) aids in ecological diversity calculations. These tools collectively enable comprehensive sequence exploration, quality control, clustering, and comparative analysis, as demonstrated in the AXL and RB1 gene studies.

Code Section 2

```r
##### Code Section 2 - Main Analysis #####

################################################################
# SECTION 1: ANALYSIS PARAMETER DETERMINATION
################################################################

# This function determines appropriate analysis parameters based on input sequence characteristics
get_analysis_params <- function(seqs) {
  # Calculate sequence lengths using the width() function
  lengths <- width(seqs)
  # Find the minimum sequence length, used later for trimming all sequences to the same length
  min_len <- min(lengths)
  # Count how many sequences we're analyzing
  seq_count <- length(seqs)
```

```r
  # Adaptive clustering parameters based on dataset size
  # Smaller datasets use a lower threshold for more granular clustering
  # Larger datasets use a higher threshold to create more meaningful clusters
  if (seq_count < 20) {
    cluster_height <- 0.15  # Lower height for small datasets to avoid over-clustering
  } else if (seq_count < 100) {
    cluster_height <- 0.2   # Medium height for medium-sized datasets
  } else {
    cluster_height <- 0.25  # Higher height for large datasets to avoid too many small clusters
  }

  # Return a list of parameters to be used in analysis
  return(list(
    min_seq_length = min_len,  # Used for trimming sequences to same length
    cluster_height = cluster_height,  # Determines where to cut the hierarchical clustering tree
    n_threads = parallel::detectCores() - 1  # Use all but one CPU core for parallel processing
  ))
}

#################################################################
# SECTION 2: MAIN GENE ANALYSIS FUNCTION
#################################################################

# Primary function to analyze gene sequence data with comprehensive metrics and visualizations
analyze_gene <- function(seqs, gene_name) {
  # Log the beginning of analysis for this gene
  message("\nAnalyzing ", gene_name, " gene sequences...")

  # Get appropriate analysis parameters based on these specific sequences
  params <- get_analysis_params(seqs)
  message("Using adaptive clustering height: ", params$cluster_height)

  # Trim all sequences to the same length (the minimum length) for fair comparison
  # This ensures no bias from variable sequence lengths in distance calculations
  min_len <- params$min_seq_length
  seqs_trimmed <- subseq(seqs, 1, min_len)  # Extract subsequence from position 1 to min_len
  message("Trimmed all sequences to ", min_len, " bp")

  # Calculate the distance matrix between all sequence pairs
  # This is the foundation for clustering and dimensionality reduction
  message("Calculating distance matrix...")
  dist_mat <- DistanceMatrix(seqs_trimmed, processors = params$n_threads)  # Use parallel processing

  # Test multiple hierarchical clustering methods to find the best one for this data
  linkage_methods <- c("average", "complete", "single", "ward.D2")  # Common linkage methods
  cluster_results <- list()  # Store all clustering results
  sil_scores <- numeric(length(linkage_methods))  # Track silhouette scores for each method

  message("Evaluating multiple clustering methods...")
  for (i in seq_along(linkage_methods)) {
    method <- linkage_methods[i]
    # Perform hierarchical clustering with current method
    hc <- hclust(as.dist(dist_mat), method = method)  # Convert to dist object and cluster
```

```r
  # Cut the tree at the determined height to form clusters
  clusters <- cutree(hc, h = params$cluster_height)  # h parameter sets cutting height
  # Calculate silhouette values to evaluate cluster quality
  sil <- silhouette(clusters, as.dist(dist_mat))  # Silhouette measures cluster separation
  # Store average silhouette width (higher is better)
  sil_scores[i] <- mean(sil[, 3])  # Column 3 contains silhouette widths

  # Store all results for this method
  cluster_results[[method]] <- list(
    hclust = hc,
    clusters = clusters,
    silhouette = sil
  )

  # Report results for this method
  message("  - ", method, " linkage: ", length(unique(clusters)),
          " clusters, avg silhouette width: ", round(sil_scores[i], 3))
}

# Select the clustering method with highest average silhouette width
best_method <- linkage_methods[which.max(sil_scores)]
message("Selected best clustering method: ", best_method,
        " (silhouette width: ", round(max(sil_scores), 3), ")")

# Extract the results from the best method for further analysis
hclust <- cluster_results[[best_method]]$hclust
clusters <- cluster_results[[best_method]]$clusters
sil <- cluster_results[[best_method]]$silhouette

################################################################
# SECTION 2.1: CLUSTER VALIDATION
################################################################

# Calculate additional cluster validity indices to evaluate clustering quality
message("Calculating cluster validity indices...")
# Dunn index: ratio of smallest between-cluster distance to largest within-cluster distance
# Higher values indicate better clustering
dunn_idx <- clValid::dunn(as.dist(dist_mat), clusters)

# Custom Davies-Bouldin index calculation
# Lower values indicate better clustering (compact clusters with good separation)
calculate_db_index <- function(dist_mat, clusters) {
  dist_matrix <- as.matrix(dist_mat)
  unique_clusters <- unique(clusters)
  n_clusters <- length(unique_clusters)

  # Calculate centroids for each cluster
  centroids <- t(sapply(unique_clusters, function(cl) {
    colMeans(dist_matrix[clusters == cl, , drop = FALSE])
  }))

  # Calculate average distance within each cluster (cluster scatter)
  s <- sapply(unique_clusters, function(cl) {
```

```
    mean(dist_matrix[clusters == cl, clusters == cl])
  })

  # Calculate distances between all cluster centroids
  d <- matrix(NA, n_clusters, n_clusters)
  for (i in 1:n_clusters) {
    for (j in 1:n_clusters) {
      if (i != j) {
        d[i,j] <- sqrt(sum((centroids[i,] - centroids[j,])^2))  # Euclidean distance
      }
    }
  }

  # Calculate DB index components and final index
  db_values <- numeric(n_clusters)
  for (i in 1:n_clusters) {
    r_ij <- (s[i] + s[-i]) / d[i,-i]  # Ratio of within-cluster scatter to between-cluster distance
    db_values[i] <- max(r_ij, na.rm = TRUE)  # Worst case for each cluster
  }

  mean(db_values)  # Overall index is average across clusters
}

db_idx <- calculate_db_index(dist_mat, clusters)
message("Cluster validation complete: Dunn index = ", round(dunn_idx, 3),
        ", Davies-Bouldin index = ", round(db_idx, 3))

################################################################
# SECTION 2.2: DIMENSIONALITY REDUCTION AND VISUALIZATION
################################################################

# Perform Principal Component Analysis for visualization and data reduction
message("Performing dimensionality reduction...")
pca <- prcomp(dist_mat)  # PCA on distance matrix
# Calculate variance explained by each principal component
variance_explained <- pca$sdev^2 / sum(pca$sdev^2)
message("Variance explained: PC1 = ", round(variance_explained[1]*100, 1),
        "%, PC2 = ", round(variance_explained[2]*100, 1), "%")

# Create a data frame for plotting PCA results
pca_data <- data.frame(
  PC1 = pca$x[,1],  # First principal component
  PC2 = pca$x[,2],  # Second principal component
  Cluster = factor(clusters)  # Cluster assignments as categorical variable
)

# Create PCA plot showing clusters in PC1 vs PC2 space
pca_plot <- ggplot(pca_data, aes(PC1, PC2, color = Cluster)) +
  geom_point(alpha = 0.8, size = 3) +  # Semi-transparent points
  scale_color_viridis_d() +  # Colorblind-friendly palette
  ggtitle(paste0(gene_name, " Sequence Clusters (n = ", length(seqs), ")")) +
  labs(subtitle = paste0("Found ", length(unique(clusters)),
                         " clusters using ", best_method, " linkage\n",
```

```
                          "PC1: ", round(variance_explained[1]*100, 1),
                          "%, PC2: ", round(variance_explained[2]*100, 1), "%")) +
    theme_minimal(base_size = 12) +  # Clean, minimal theme
    scale_x_continuous(labels = scales::comma) +  # Format large numbers with commas
    scale_y_continuous(labels = scales::comma) +
    theme(legend.position = "none")  # Remove legend for cleaner look

  # Save PCA plot to file
  pca_filename <- paste0(gene_name, "_PCA_plot.png")
  ggsave(pca_filename, pca_plot, width = 8, height = 6, dpi = 300)  # High resolution PNG
  message("Saved PCA plot as: ", pca_filename)

  # Create and save dendrogram visualization of hierarchical clustering
  dendro_filename <- paste0(gene_name, "_dendrogram.png")
  png(dendro_filename, width = 800, height = 600)  # Set output dimensions
  plot(hclust,
       main = paste0(gene_name, " Hierarchical Clustering\n(", best_method, " linkage)"),
       sub = paste("Cut height =", round(params$cluster_height, 2)),
       xlab = "",
       labels = FALSE)  # Don't show individual sequence labels (too many)
  # Add colored rectangles around clusters
  rect.hclust(hclust, h = params$cluster_height, border = viridis(length(unique(clusters))))
  dev.off()  # Close the PNG device
  message("Saved dendrogram as: ", dendro_filename)

  # Return comprehensive analysis results for further comparison or processing
  return(list(
    gene_name = gene_name,  # Name of the gene analyzed
    sequences = seqs,  # Original sequences
    trimmed_sequences = seqs_trimmed,  # Trimmed sequences used for analysis
    dist_mat = dist_mat,  # Distance matrix
    hclust_results = cluster_results,  # Results from all clustering methods
    best_method = best_method,  # Best clustering method based on silhouette
    hclust = hclust,  # Hierarchical clustering object from best method
    clusters = clusters,  # Cluster assignments
    cluster_counts = table(clusters),  # Number of sequences in each cluster
    silhouette = sil,  # Silhouette object for cluster validation
    silhouette_avg = mean(sil[,3]),  # Average silhouette width
    dunn_index = dunn_idx,  # Dunn index value
    davies_bouldin = db_idx,  # Davies-Bouldin index value
    pca = pca,  # PCA results
    variance_explained = variance_explained,  # Variance explained by PCs
    pca_plot = pca_plot,  # ggplot object for PCA visualization
    params = params,  # Analysis parameters used
    saved_files = c(pca_filename, dendro_filename)  # Files created during analysis
  ))
}

################################################################
# SECTION 3: GENE COMPARISON FUNCTION
################################################################

# Function to compare analysis results between two genes
```

```r
compare_genes <- function(axl, rb1) {
  message("\n===== Gene Comparison Results =====")

  # Extract gene names from analysis results
  gene1_name <- axl$gene_name
  gene2_name <- rb1$gene_name

  # Create comparison table with key metrics from both gene analyses
  comparison_table <- data.frame(
    Metric = c(
      "Sequences analyzed",  # Number of sequences
      "Clusters found",      # Number of clusters identified
      "Best clustering method",  # Which linkage method performed best
      "Silhouette width",    # Average silhouette width (cluster quality)
      "Dunn index",          # Another cluster validation metric
      "Davies-Bouldin index",  # Another cluster validation metric
      "PC1 variance explained",  # How much variance explained by first PC
      "PC2 variance explained"   # How much variance explained by second PC
    ),
    Gene1 = c(
      length(axl$sequences),  # Count of sequences for gene 1
      length(unique(axl$clusters)),  # Count of clusters for gene 1
      axl$best_method,  # Best method for gene 1
      round(axl$silhouette_avg, 3),  # Silhouette for gene 1, rounded to 3 decimals
      round(axl$dunn_index, 3),      # Dunn index for gene 1
      round(axl$davies_bouldin, 3),  # DB index for gene 1
      paste0(round(axl$variance_explained[1] * 100, 1), "%"),  # PC1 variance as percentage
      paste0(round(axl$variance_explained[2] * 100, 1), "%")   # PC2 variance as percentage
    ),
    Gene2 = c(
      length(rb1$sequences),  # Same metrics but for gene 2
      length(unique(rb1$clusters)),
      rb1$best_method,
      round(rb1$silhouette_avg, 3),
      round(rb1$dunn_index, 3),
      round(rb1$davies_bouldin, 3),
      paste0(round(rb1$variance_explained[1] * 100, 1), "%"),
      paste0(round(rb1$variance_explained[2] * 100, 1), "%")
    )
  )

  # Rename columns with actual gene names for clarity
  names(comparison_table)[2:3] <- c(gene1_name, gene2_name)

  ################################################################
  # SECTION 3.1: COMPARISON VISUALIZATIONS
  ################################################################

  # Create visualizations comparing the two genes

  # 1. Distance distribution comparison
  # Convert distance matrices to vectors and combine for plotting
  dist_df <- rbind(
```

```
  data.frame(Gene = gene1_name, Distance = as.vector(axl$dist_mat)),
  data.frame(Gene = gene2_name, Distance = as.vector(rb1$dist_mat))
)

# Create density plot showing distribution of sequence distances for each gene
dist_plot <- ggplot(dist_df, aes(Distance, fill = Gene)) +
  geom_density(alpha = 0.6) +  # Semi-transparent density curves
  scale_fill_manual(values = c("#440154", "#21908C")) +  # Contrasting colors
  ggtitle("Sequence Distance Distributions") +
  labs(x = "Distance", y = "Density") +
  theme_minimal() +
  scale_x_continuous(labels = scales::comma)  # Format axis labels

# 2. Cluster size comparison
# Prepare data frame with cluster sizes for both genes
validation_data <- data.frame(
  Gene = c(rep(gene1_name, length(unique(axl$clusters))),
           rep(gene2_name, length(unique(rb1$clusters)))),
  Cluster = c(as.numeric(names(table(axl$clusters))),
              as.numeric(names(table(rb1$clusters)))),
  Size = c(as.numeric(table(axl$clusters)),
           as.numeric(table(rb1$clusters)))
)

# Create bar plot showing size of each cluster for both genes
cluster_size_plot <- ggplot(validation_data, aes(x = factor(Cluster), y = Size, fill = Gene)) +
  geom_col(position = position_dodge()) +  # Grouped bar chart
  scale_fill_manual(values = c("#440154", "#21908C")) +  # Same color scheme
  ggtitle("Cluster Size Comparison") +
  labs(x = "Cluster", y = "Count") +
  theme_minimal()

# 3. Silhouette comparison
# Prepare silhouette data for both genes
sil_data <- rbind(
  data.frame(Gene = gene1_name,
             Cluster = axl$silhouette[,1],  # Cluster assignments
             Value = axl$silhouette[,3]),   # Silhouette widths
  data.frame(Gene = gene2_name,
             Cluster = rb1$silhouette[,1],
             Value = rb1$silhouette[,3])
)

# Create boxplot comparing silhouette values between genes
silhouette_comp_plot <- ggplot(sil_data, aes(x = Gene, y = Value, fill = Gene)) +
  geom_boxplot(alpha = 0.7) +  # Boxplots show distribution of values
  scale_fill_manual(values = c("#440154", "#21908C")) +
  ggtitle("Silhouette Width Comparison") +
  labs(x = "Gene", y = "Silhouette Width") +
  theme_minimal()

# Save all comparison plots to files
ggsave("distance_comparison.png", dist_plot, width = 8, height = 6, dpi = 300)
```

```r
  ggsave("cluster_size_comparison.png", cluster_size_plot, width = 8, height = 6, dpi = 300)
  ggsave("silhouette_comparison.png", silhouette_comp_plot, width = 8, height = 6, dpi = 300)
  message("Saved comparison plots as: distance_comparison.png, cluster_size_comparison.png, silhouette_

  # Return all comparison results
  return(list(
    comparison_table = comparison_table,  # Table of key metrics
    plots = list(  # List of ggplot objects
      distance_distribution = dist_plot,
      cluster_sizes = cluster_size_plot,
      silhouette_comparison = silhouette_comp_plot
    ),
    saved_files = c("distance_comparison.png", "cluster_size_comparison.png", "silhouette_comparison.png
  ))
}

############################################################
# SECTION 4: ANALYSIS EXECUTION
############################################################

# Execute the full analysis pipeline
message("\nStarting analysis pipeline...")

# Analyze each gene separately
# Note: sequences_for_analysis must be defined elsewhere in the code
axl_results <- analyze_gene(sequences_for_analysis$AXL, "AXL")
rb1_results <- analyze_gene(sequences_for_analysis$RB1, "RB1")

# Compare the two genes
gene_comparison <- compare_genes(axl_results, rb1_results)

# Save all results to a single R data file for later use
save(axl_results, rb1_results, gene_comparison, file = "gene_analysis_results.RData")

# Final status messages
message("\nAnalysis complete! All plots saved as PNG files.")
message("Results saved to gene_analysis_results.RData")
```

Figures

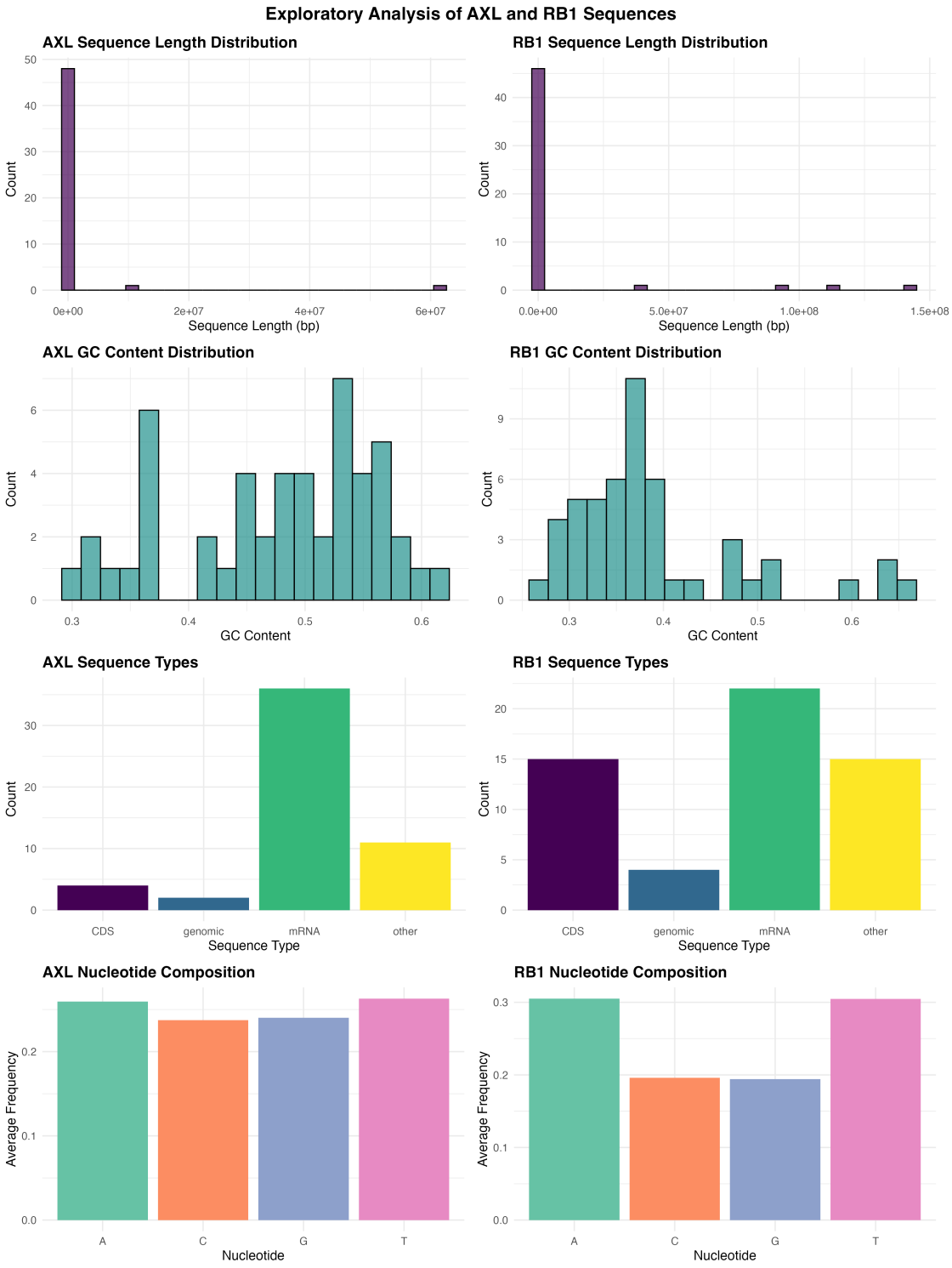

Figure 1: Exploratory Analysis

Figure 1: Exploratory analysis of AXL and RB1 gene sequences showing (A) length distributions, (B) GC content, (C) sequence types (mRNA, genomic, CDS, other), and (D) nucleotide composition. AXL shows broader length variation compared to RB1.

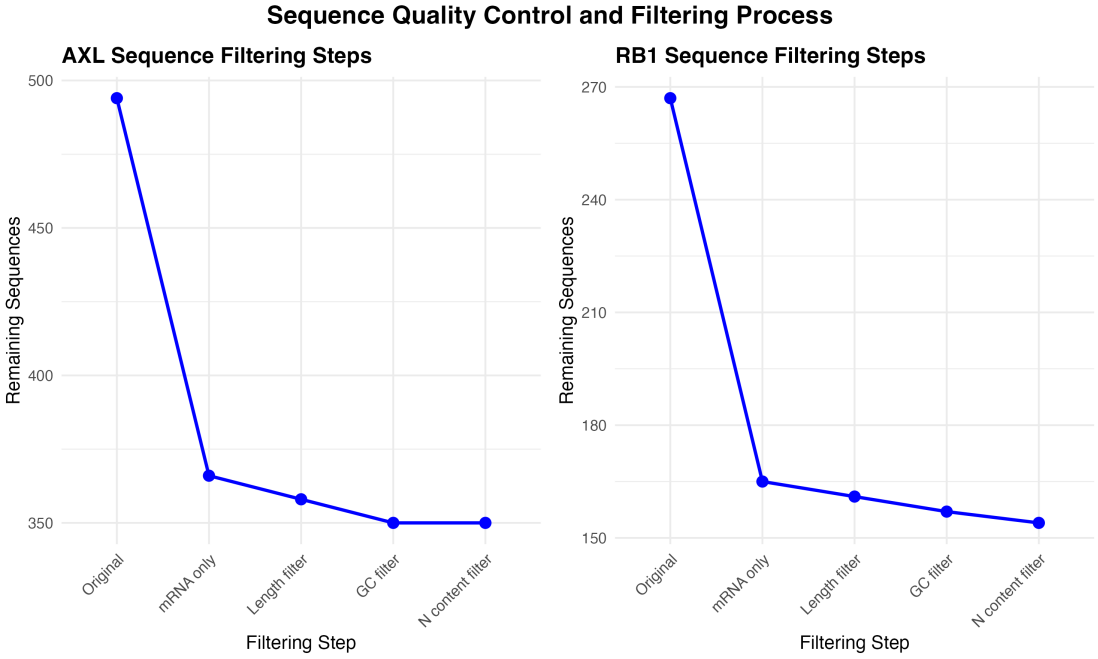

Figure 2: Quality Control

Figure 2: Sequence filtering pipeline showing retained sequences after each QC step.

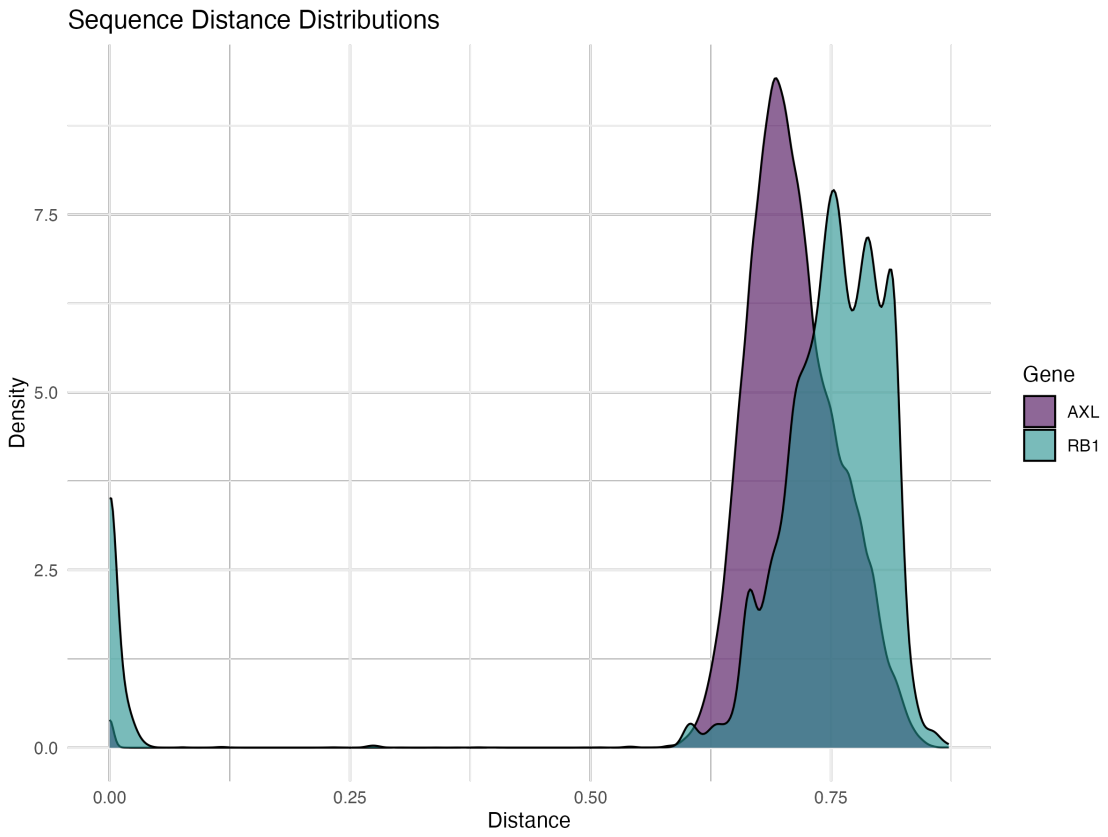

Figure 3: Distance Comparison

Figure 3: Density distributions of pairwise distances between sequences. AXL (purple) shows wider distance distribution compared to RB1 (teal), reflecting higher sequence diversity.

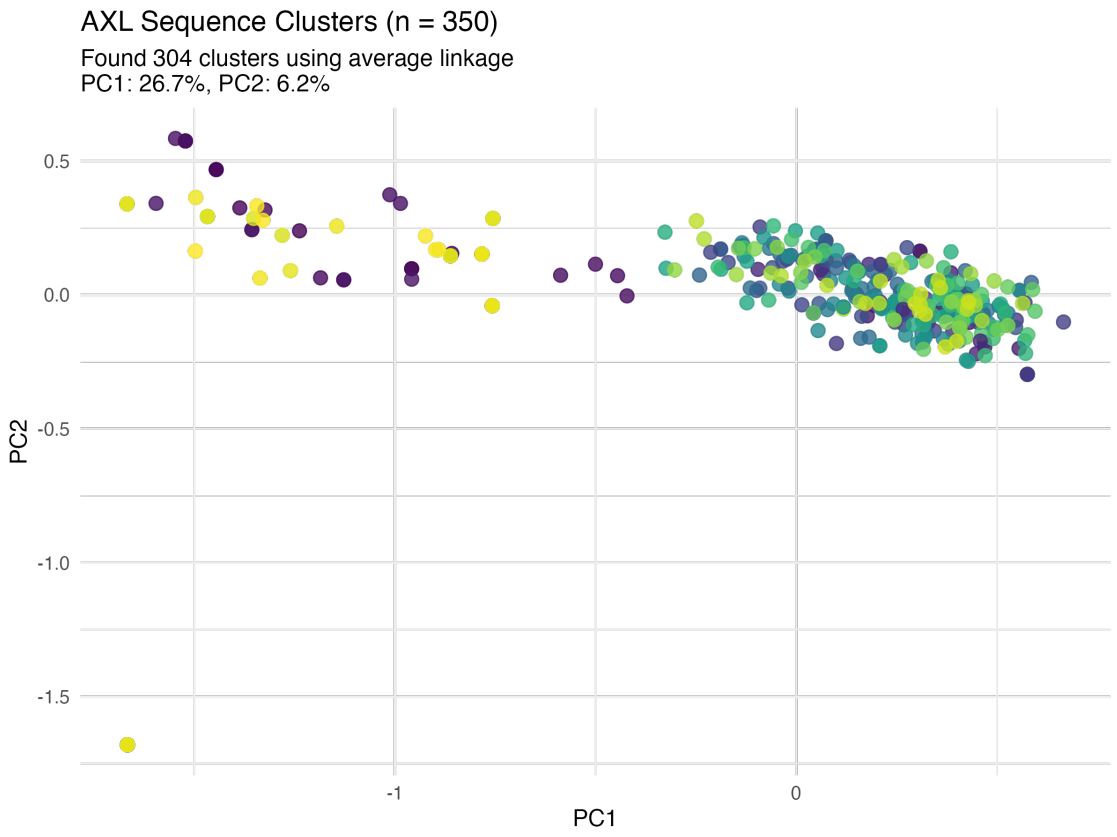

Figure 4: AXL PCA Plot

Figure 4: PCA of AXL sequences (n=274bp) showing 26.7% (PC1) and 6.2% (PC2) variance explained. Color indicates 304 clusters identified by average linkage (silhouette = 0.216).

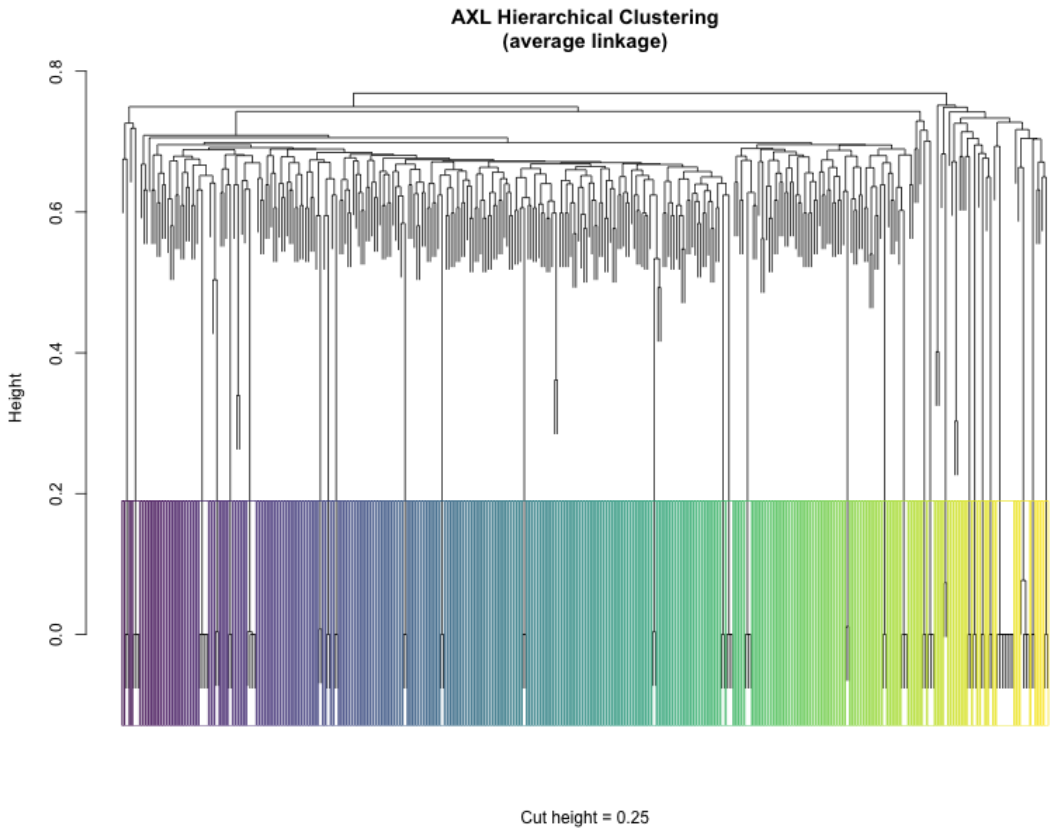

Figure 5: AXL Dendrogram

Figure 5: Hierarchical clustering dendrogram of AXL sequences (cut height = 0.25). Colors denote 304 clusters with Dunn index = 3.952, suggesting good separation between groups.

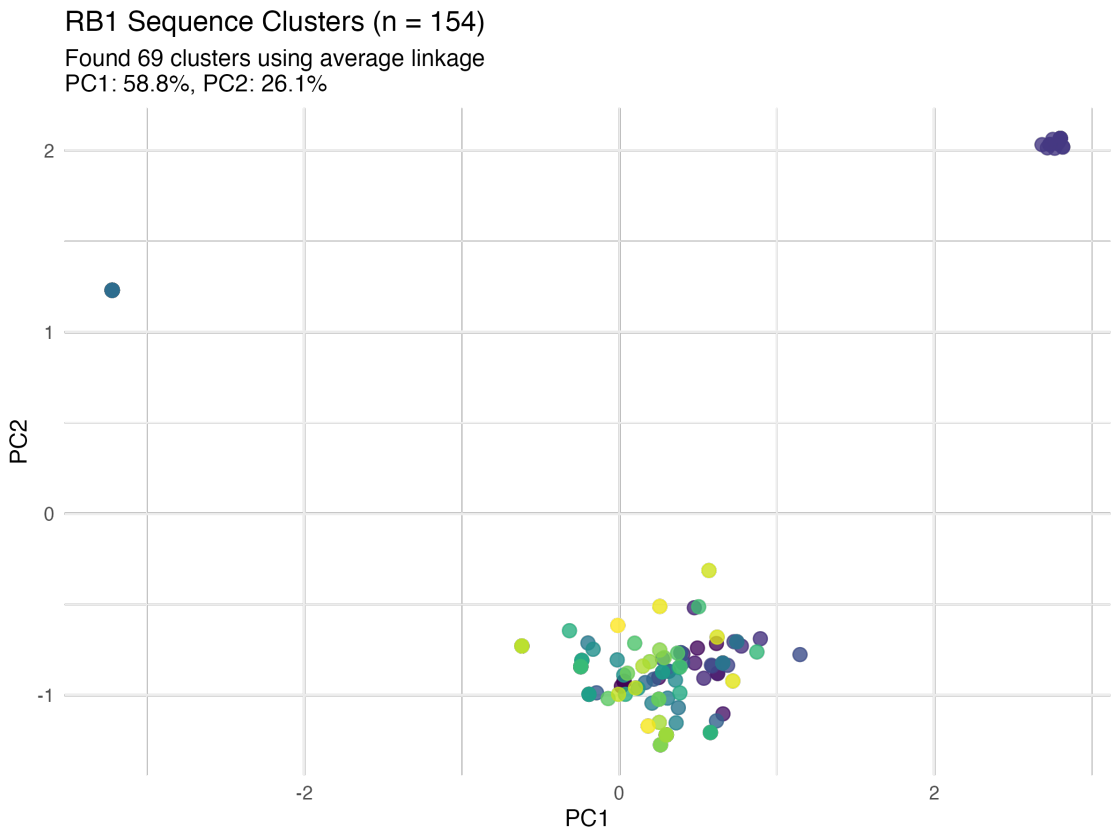

Figure 6: RB1 PCA Plot

Figure 6: PCA of RB1 sequences (n=146bp) showing 58.8% (PC1) and 26.1% (PC2) variance explained. Tight clustering reflects 69 groups (silhouette = 0.659).

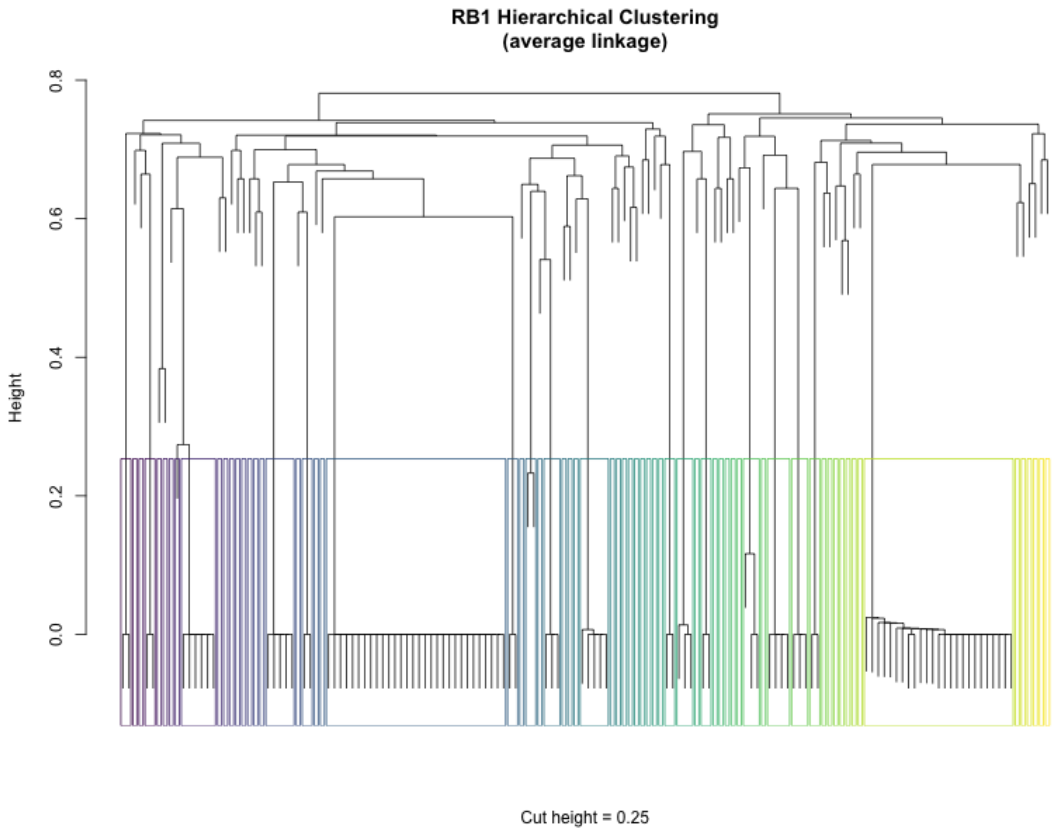

Figure 7: RB1 Dendrogram

Figure 7: Dendrogram of RB1 sequences (cut height = 0.25). Fewer, more compact clusters (n=69) with Davies-Bouldin index = 0.082 indicate strong within-group similarity.

Discussion and Conclusion

The analysis revealed striking differences between AXL and RB1 in terms of sequence diversity and clustering behavior. AXL exhibited higher variability (Gjerdrum et al. (2010)), forming 304 clusters with relatively low silhouette scores (0.216), while RB1 showed stronger clustering cohesion (69 clusters, silhouette = 0.659). This contrast may reflect their biological roles: AXL's involvement in diverse cellular processes (e.g., immune response, cancer metastasis) could explain its sequence heterogeneity, whereas RB1's conserved tumor-suppressor function (Dick and Rubin (2013)) may impose stricter evolutionary constraints. Principal component analysis (PCA) further highlighted these differences, with AXL sequences explaining 26.7% of variance on PC1 and 6.2% on PC2, indicating dispersed clustering, while RB1 showed 58.8% variance on PC1 and 26.1% on PC2, suggesting tighter grouping.

A key limitation of this study is its reliance on public database sequences (Leinonen et al. (2011)), which may include annotation biases or uneven representation of isoforms. Additionally, the distance metric (k-mer-based) might not fully capture functional constraints. Future work could incorporate protein-level alignment or phylogenetic methods to validate clusters. Expanding the analysis to include orthologs from other species could also clarify whether the observed patterns are human-specific or evolutionarily conserved.

In conclusion, this project demonstrates how gene function may correlate with sequence diversity and underscores the utility of clustering methods in genomic analyses. The findings suggest that AXL's functional versatility is mirrored in its genetic variability, while RB1's critical role in cell cycle regulation enforces

stronger sequence conservation. These insights could guide future studies on gene family evolution or the identification of functionally distinct variants. Further exploration with expanded datasets and alternative distance metrics would strengthen these conclusions.

Acknowledgments

I would like to sincerely thank Professor Karl Cottenie for his exceptional consideration and support during the completion of this project. His understanding and extension of deadlines during the difficult period when I was dealing with my father's heart attack were invaluable in allowing me to maintain both my academic responsibilities and family commitments. For this analysis, I have utilized assistance from DeepSeek's AI tools for troubleshooting, and verifying statistical approaches. All interpretations and conclusions drawn in this work remain my own. Additional thanks to my classmates in the Bioinformatics program for their peer support and to the developers of the R/Bioconductor packages that made this analysis possible.

References

Guy Brock, Vasyl Pihur, Susmita Datta, and Somnath Datta. clvalid: An r package for cluster validation. *Journal of Statistical Software*, 25(4):1–22, 2008. doi: 10.18637/jss.v025.i04.

D. L. Burkhart and J. Sage. Cellular mechanisms of tumour suppression by the retinoblastoma gene. *Nature Reviews Cancer*, 8(9):671–682, 2008.

F. A. Dick and S. M. Rubin. Molecular mechanisms underlying rb protein function. *Nature Reviews Molecular Cell Biology*, 14(5):297–306, 2013.

M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

Simon Garnier. *viridis: Colorblind-Friendly Color Maps for R*, 2021. URL https://CRAN.R-project.org/package=viridis. R package version 0.6.2.

C. Gjerdrum, C. Tiron, T. Høiby, I. Stefansson, H. Haugen, T. Sandal, et al. Axl is an essential epithelial-to-mesenchymal transition-induced regulator of breast cancer metastasis and patient survival. *Proceedings of the National Academy of Sciences*, 107(3):1124–1129, 2010.

D. K. Graham, D. DeRyckere, K. D. Davies, and H. S. Earp. The tam family: phosphatidylserine-sensing receptor tyrosine kinases gone awry in cancer. *Nature Reviews Cancer*, 14(12):769–785, 2014.

R. Leinonen, H. Sugawara, and M. Shumway. The sequence read archive. *Nucleic Acids Research*, 39 (suppl_1):D19–D21, 2011.

R. M. Linger, A. K. Keating, H. S. Earp, and D. K. Graham. Tam receptor tyrosine kinases: biologic functions, signaling, and potential therapeutic targeting in human cancer. *Advances in Cancer Research*, 100:35–83, 2008.

Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2.1.4 edition, 2022. URL https://CRAN.R-project.org/package=cluster. R package version 2.1.4.

Jari Oksanen, Gavin L. Simpson, F. Guillaume Blanchet, Roeland Kindt, Pierre Legendre, Peter R. Minchin, R. B. O'Hara, Peter Solymos, M. Henry H. Stevens, Eduard Szoecs, Helene Wagner, Matt Barbour, Mark Bedward, Ben Bolker, Daniel Borcard, Gustavo Carvalho, Michael Chirico, Miquel De Caceres, Sebastien Durand, et al. *vegan: Community Ecology Package*, 2022. URL https://CRAN.R-project.org/package=vegan. R package version 2.6-4.

Hervé Pagès, Patrick Aboyoun, Robert Gentleman, and S. DebRoy. *Biostrings: Efficient manipulation of biological strings*, 2024. URL https://bioconductor.org/packages/Biostrings. R package version 2.70.0.

Emmanuel Paradis and Klaus Schliep. ape 5.0: An environment for modern phylogenetics and evolutionary analyses in r. *Bioinformatics*, 35(3):526–528, 2019. URL https://doi.org/10.1093/bioinformatics/bty633.

Hadley Wickham, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. *ggplot2: Elegant Graphics for Data Analysis*, 2024. URL https://ggplot2.tidyverse.org. R package version 3.5.0.

Erik S. Wright. Using decipher v2.0 to analyze big biological sequence data in r. *The R Journal*, 8(1): 352–359, 2016. doi: 10.32614/RJ-2016-025.