# Gene module analysis using optimized parameters

Mikayla, Mustafa, Yibo

2023-05-07

## Table of Contents



## Introduction

This report is totally original without taking code from the github repo of the original paper. As this report also includes all the output for your references, we have also provided a table of content so that you might skip some of very tedious output and go straight to some code.

## InferCNV based on immune cells and reference genome.

```
library(infercnv)

library(Matrix)



# List of patient IDs

patient_ids <- c("CID3586", "CID3838", "CID3921", "CID3941", "CID3946", "CID3948", "CID3963", "CID4040", "CID4066", "CID4067", "CID4290A", "CID4398", "CID4461", "CID4463", "CID4465", "CID4471", "CID4495", "CID4513", "CID4515", "CID4523", "CID4530N", "CID4535", "CID44041", "CID44971", "CID44991", "CID45171")
```

```
# Function to create annotations file and generate raw count matrix

create_annotations_file_and_generate_matrix <- function(metadata_csv, count_matrix, barcodes_tsv, genes_tsv) {

 metadata <- read.csv(metadata_csv)

 annotations <- data.frame(metadata[,1], metadata$celltype_major)

 colnames(annotations) <- c("", "cell_group")


 # Read sparse count matrix

 sparse_count_matrix <- readMM(count_matrix)

 colnames(sparse_count_matrix) <- read.table(barcodes_tsv, col.names = c("barcode"))$barcode

 rownames(sparse_count_matrix) <- read.table(genes_tsv, col.names = c("gene"))$gene


 # Convert sparse matrix to dense matrix

 dense_count_matrix <- as.matrix(sparse_count_matrix)


 return(list(annotations = annotations, dense_count_matrix = dense_count_matrix))

}


calculate_genome_instability_score <- function(expr_data, reference_indices, observation_indices) {

 # Get reference and observation data

 references <- expr_data[, unlist(reference_indices)]

 observations <- expr_data[, unlist(observation_indices)]


 # Calculate changes

 changes <- observations - rowMeans(references)
```

```
  # Scale changes between -1 and 1
  scaled_changes <- 2 * (changes - min(changes)) / (max(changes) - min(changes)) - 1

  # Calculate instability scores
  instability_scores <- rowMeans(scaled_changes^2)
  return(instability_scores)
}

# Loop over patient IDs
genome_instability_scores_list <- list()

for (patient_id in patient_ids) {
  # Define file paths
  metadata_csv <- paste0("./GSE176078_RAW/", patient_id, "/metadata.csv")
  count_matrix <- paste0("./GSE176078_RAW/", patient_id, "/count_matrix_sparse.mtx")
  barcodes_tsv <- paste0("./GSE176078_RAW/", patient_id, "/count_matrix_barcodes.tsv")
  genes_tsv <- paste0("./GSE176078_RAW/", patient_id, "/count_matrix_genes.tsv")

  # Create annotations file and generate dense count matrix
  results <- create_annotations_file_and_generate_matrix(metadata_csv, count_matrix, barco
des_tsv, genes_tsv)
  annotations <- results$annotations
  dense_count_matrix <- results$dense_count_matrix
  annotations_file <- paste0("./GSE176078_RAW/", patient_id, "/", patient_id, "_annotations.
txt")
  write.table(annotations, file = annotations_file, sep = "\t", row.names = FALSE, col.names =
 FALSE, quote = FALSE)
  tryCatch({
```

```
  # Create inferCNV object with custom parameters
  infercnv_obj = CreateInfercnvObject(raw_counts_matrix = dense_count_matrix,
                                      annotations_file = annotations_file,
                                      delim = "\t",
                                      gene_order_file = "gene_order_file.txt",
                                      ref_group_names = c("T-cells", "B-cells", "Endothelial"))

  # Set scipen option before running inferCNV
  options(scipen = 100)

  # Filter genes with a mean count of less than 0.1 across all cells
  filtered_counts_matrix = dense_count_matrix[rowMeans(dense_count_matrix) >= 0.1, ]

  # Run inferCNV with custom parameters
  infercnv_obj = infercnv::run(infercnv_obj,
                               cutoff = 1.3, # dynamic threshold of 1.3 standard deviations from the mea
n
                               out_dir = paste0("infercnv_output_", patient_id),
                               cluster_by_groups = TRUE,
                               denoise = TRUE,
                               HMM = TRUE,
                               analysis_mode = "subclusters",
                               window_length = 100) # 100 gene sliding window
 },
  error = function(e) {
  cat("Error processing patient_id:", patient_id, "with 'T-cells', 'B-cells', and 'Endothelial' as
reference\n")
  cat("Error message:", conditionMessage(e), "\n")
```

```
  cat("Re-initializing with 'T-cells' and 'B-cells' as reference\n")

  tryCatch({
    infercnv_obj = CreateInfercnvObject(raw_counts_matrix = dense_count_matrix,
                                        annotations_file = annotations_file,
                                        delim = "\t",
                                        gene_order_file = "gene_order_file.txt",
                                        ref_group_names = c("T-cells", "B-cells"))

    # Set scipen option before running inferCNV
    options(scipen = 100)

    # Filter genes with a mean count of less than 0.1 across all cells
    filtered_counts_matrix = dense_count_matrix[rowMeans(dense_count_matrix) >= 0.1, ]

    # Run inferCNV with custom parameters
    infercnv_obj = infercnv::run(infercnv_obj,
                         cutoff = 1.3, # dynamic threshold of 1.3 standard deviations from the mea
n
                         out_dir = paste0("infercnv_output_", patient_id),
                         cluster_by_groups = TRUE,
                         denoise = TRUE,
                         HMM = TRUE,
                         analysis_mode = "subclusters",
                         window_length = 100) # 100 gene sliding window
  }, error = function(e) {
    cat("Error processing patient_id:", patient_id, "with 'T-cells' and 'B-cells' as reference\n"
)
```

```
    cat("Error message:", conditionMessage(e), "\n")
    cat("Re-initializing with 'T-cells' as reference\n")

    tryCatch({
     infercnv_obj = CreateInfercnvObject(raw_counts_matrix = dense_count_matrix,
                                         annotations_file = annotations_file,
                                         delim = "\t",
                                         gene_order_file = "gene_order_file.txt",
                                         ref_group_names = c("T-cells"))

     # Set scipen option before running inferCNV
     options(scipen = 100)

     # Filter genes with a mean count of less than 0.1 across all cells
     filtered_counts_matrix = dense_count_matrix[rowMeans(dense_count_matrix) >= 0.1, ]

     # Run inferCNV with custom parameters
     infercnv_obj = infercnv::run(infercnv_obj,
                                  cutoff = 1.3, # dynamic threshold of 1.3 standard deviations from the me
an
                                  out_dir = paste0("infercnv_output_", patient_id),
                                  cluster_by_groups = TRUE,
                                  denoise = TRUE,
                                  HMM = TRUE,
                                  analysis_mode = "subclusters",
                                  window_length = 100) # 100 gene sliding window
   }, error = function(e) {
```

```
    cat("Error processing patient_id:", patient_id, "with 'T-cells' as reference\n")
    cat("Error message:", conditionMessage(e), "\n")
    })
  })
 })

 # Calculate genome instability scores
 genome_instability_scores <- calculate_genome_instability_score(infercnv_obj@expr.data,
                                                                 infercnv_obj@reference_grouped_cell_indices,
                                                                 infercnv_obj@observation_grouped_cell_indices)
 genome_instability_scores_list[[patient_id]] <- genome_instability_scores
}
```

## Calculate genomic instability score for each cell

```
library(Matrix)


# Function to calculate genomic instability score
genomic_instability_score <- function(count_matrix, genomic_scores) {
  squared_genomic_scores <- genomic_scores^2
  weighted_counts <- count_matrix * squared_genomic_scores
  rowMeans(weighted_counts)
}


# Main implementation
patient_ids <- c("CID3586", "CID3838", "CID3921", "CID3941", "CID3946", "CID3
948", "CID3963", "CID4040", "CID4066", "CID4067", "CID4290A", "CID4398", "CID
4461", "CID4463", "CID4465", "CID4471", "CID4495", "CID4513", "CID4515", "CID
4523", "CID4530N", "CID4535", "CID44041", "CID44971", "CID44991", "CID45171")
```

```
instability_scores_all_patients <- list()



for (i in 1:length(patient_ids)) {

  patient_id <- patient_ids[i]

  cat("Dealing with", i, "of", length(patient_ids), "patients:", patient_id,
"\n")



  count_matrix_barcodes_path <- paste0("./GSE176078_RAW/", patient_id, "/coun
t_matrix_barcodes.tsv")

  count_matrix_genes_path <- paste0("./GSE176078_RAW/", patient_id, "/count_m
atrix_genes.tsv")

  count_matrix_sparse_path <- paste0("./GSE176078_RAW/", patient_id, "/count_
matrix_sparse.mtx")



  count_matrix <- readMM(file = count_matrix_sparse_path)

  barcodes <- read.delim(count_matrix_barcodes_path, header = FALSE, col.name
s = c("barcode"))

  genes <- read.delim(count_matrix_genes_path, header = FALSE, col.names = c(
"gene"))



  colnames(count_matrix) <- barcodes$barcode

  rownames(count_matrix) <- genes$gene



  # Get the genomic scores for the current patient

  genomic_scores <- genome_instability_scores_list[[patient_id]]



  # Subset count_matrix to have only the genes in genomic_scores

  count_matrix <- count_matrix[rownames(count_matrix) %in% names(genomic_scor
es), ]



  # Sort genomic_scores to match the order of genes in count_matrix
```

```
  genomic_scores <- genomic_scores[rownames(count_matrix)]


  # Calculate genomic instability scores
  instability_scores <- genomic_instability_score(t(count_matrix), genomic_sc
ores)


  # Store the results in the list
  instability_scores_df <- data.frame(instability_score = instability_scores)
  rownames(instability_scores_df) <- colnames(count_matrix)
  instability_scores_all_patients[[patient_id]] <- instability_scores_df
}
```

## Calculate the variance of cancer cell's genomic instability score as results

```
# Initialize a list to store the variance for each patient
variance_list <- list()


# Loop through each patient
for (patient_id in patient_ids) {
 # Define file paths for the patient's data
 count_matrix_barcodes_path <- paste0("./GSE176078_RAW/", patient_id, "/count_matrix_
barcodes.tsv")
 count_matrix_genes_path <- paste0("./GSE176078_RAW/", patient_id, "/count_matrix_gen
es.tsv")
 count_matrix_sparse_path <- paste0("./GSE176078_RAW/", patient_id, "/count_matrix_sp
arse.mtx")
 metadata_path <- paste0("./GSE176078_RAW/", patient_id, "/metadata.csv")


 # Load count matrix and metadata
 count_matrix <- readMM(file = count_matrix_sparse_path)
```

```
 barcodes <- read.delim(count_matrix_barcodes_path, header = FALSE, col.names = c("barc
ode"))
 genes <- read.delim(count_matrix_genes_path, header = FALSE, col.names = c("gene"))
 metadata <- read.csv(metadata_path, row.names = 1)

 # Filter cells based on the celltype_major column
 cells_to_keep <- metadata$celltype_major %in% c("Cancer Epithelial", "CAFs")

 # Set the colnames and rownames of the count matrix
 colnames(count_matrix) <- barcodes$barcode
 rownames(count_matrix) <- genes$gene

 # Get the indices of the cells to keep
 cells_to_keep_indices <- which(cells_to_keep)

 # Subset the count matrix using the cells_to_keep_indices
 count_matrix <- count_matrix[, cells_to_keep_indices]

 # Subset the metadata using cells_to_keep
 metadata <- metadata[cells_to_keep, , drop = FALSE]

 # Get the genomic instability scores for all cells in the current patient
 all_instability_scores <- instability_scores_all_patients[[patient_id]]

 # Subset the genomic instability scores using cells_to_keep_indices
 cancer_instability_scores <- all_instability_scores[cells_to_keep_indices, , drop = FALSE]
```

```
  # Calculate the variance of the cancer cell genomic instability scores for the current patien
t
  variance <- var(cancer_instability_scores$instability_score)

  # Store the variance in the variance_list
  variance_list[[patient_id]] <- variance
}

# Combine the variance_list into a data frame
variance_df <- data.frame(variance = unlist(variance_list))
rownames(variance_df) <- patient_ids
```

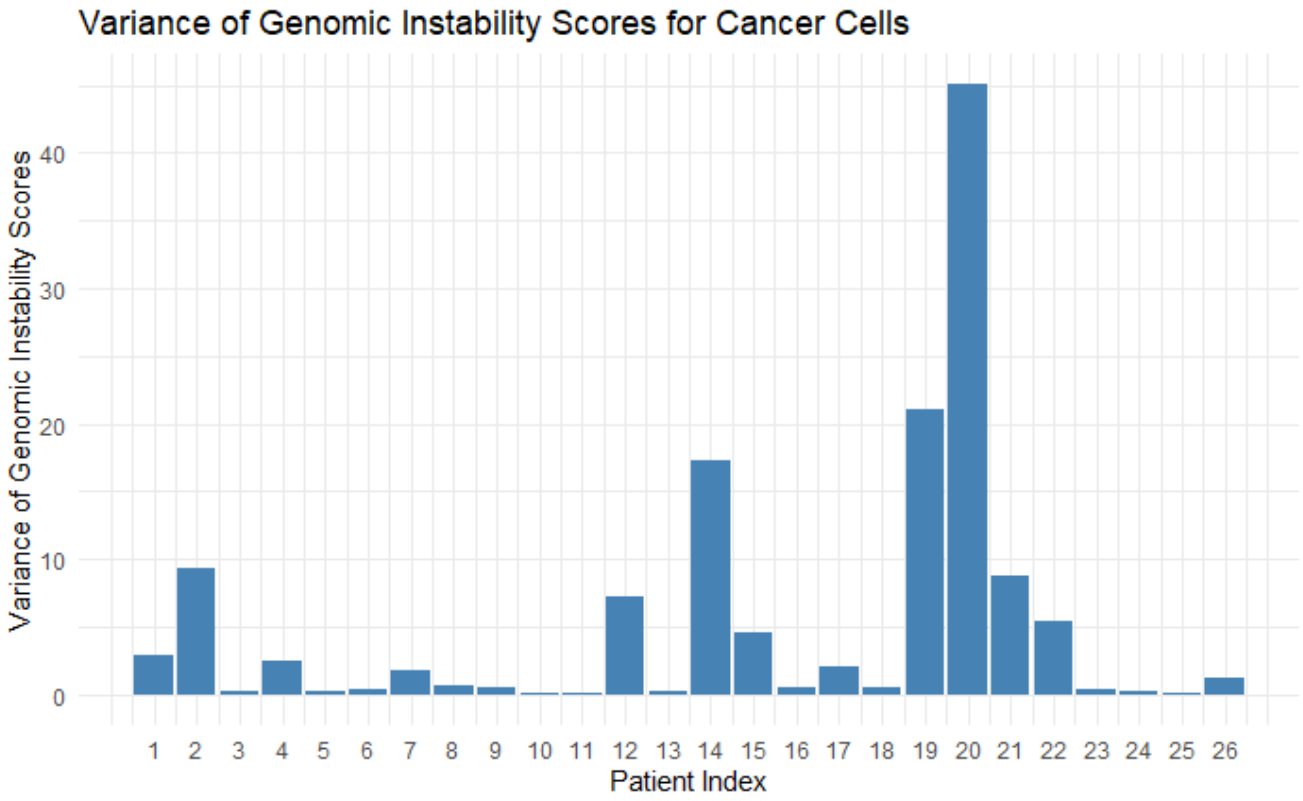
## Visualization

```
library(ggplot2)

# Convert the variance data frame into a format suitable for ggplot2
variance_df$patient_index <- 1:nrow(variance_df)

# Create a function to display a subset of patient IDs
display_subset_labels <- function(patient_ids, step = 2) {
  sapply(1:length(patient_ids), function(i) {
    if (i %% step == 0) {
      return(patient_ids[i])
    } else {
      return("")
    }
  })
}
```

```
# Create the bar plot
ggplot(variance_df, aes(x = patient_index, y = variance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  scale_x_continuous(breaks = 1:length(patient_ids), labels = 1:length(patient_ids)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5)) +
  labs(x = "Patient Index", y = "Variance of Genomic Instability Scores", title = "Variance of G
enomic Instability Scores for Cancer Cells") +
  theme_minimal()
```




## Store the result

```
# Convert the variance_list to a data frame
variance_df <- data.frame(
  patient_id = patient_ids,
  variance = variance_list
```

```
)


# Write the data frame to a CSV file
write.csv(variance_df, file = "CNV_variance.csv", row.names = FALSE)
```