# Preprocessing and Differential Expression

## Day 3

Miguel Casanova & Dany Mukesha
2025-11-25

# Preprocessing and Differential Expression

# Learning Objectives

By the end of Day 3, you will be able to:

- Apply different normalization methods to proteomic data
- Perform batch effect correction
- Conduct differential expression analysis using limma
- Interpret and visualize differential expression results
- Create volcano plots and MA plots

# Module 1: Data Preprocessing

# Why Normalize?

Normalization removes systematic technical variation to:

- Make samples comparable
- Reduce technical noise
- Preserve biological signal

# Loading Example Data

```
# Load the data components
protein_matrix ← read.csv("data/protein_matrix.csv",
                          row.names = 1, check.names = FALSE)
sample_metadata ← read.csv("data/sample_metadata.csv")
protein_annotations ← read.csv("data/protein_annotations.csv")

cat("Dataset structure:\n")
cat("Proteins:", nrow(protein_matrix), "\n")
cat("Samples:", ncol(protein_matrix), "\n")
cat("Missing values:", sum(is.na(protein_matrix)), "\n")
```

# Initial Data Exploration

```
# Experimental design
design_table ← table(
  sample_metadata$condition,
  sample_metadata$batch
)
print(design_table)
```

```
# Data summary
summary_stats ← data.frame(
  Statistic = c("Total proteins",
                "Total samples",
                "Missing values"),
  Value = c(
    nrow(protein_matrix),
    ncol(protein_matrix),
    sum(is.na(protein_matrix))
  )
)
print(summary_stats)
```

# Visualizing Raw Data

```
par(mfrow = c(1, 2))
boxplot(protein_matrix,
        main = "Raw Data - Before Preprocessing",
        las = 2, cex.axis = 0.7,
        ylab = "Log2 Abundance")
plot(density(protein_matrix[!is.na(protein_matrix)],
     na.rm = TRUE),
     main = "Distribution of Raw Values")
```

# Handling Missing Values

Strategy 1: Filter proteins with too many missing values

```
threshold ← 0.3  # Remove if >30% missing
missing_per_protein ← rowSums(is.na(protein_matrix)) /
                      ncol(protein_matrix)
filtered_proteins ← missing_per_protein ⩽ threshold

protein_matrix_filtered ← protein_matrix[filtered_proteins, ]
cat("Proteins after filtering:",
    nrow(protein_matrix_filtered), "\n")
```

# Missing Value Imputation

Strategy 2: Impute remaining missing values

```
handle_missing_values ← function(data_matrix,
                                 missing_threshold = 0.2) {

  # Filter proteins
  missing_per_protein ← rowSums(is.na(data_matrix)) /
                        ncol(data_matrix)
  filtered_data ← data_matrix[missing_per_protein ⩽
                              missing_threshold, ]

  # Impute with minimum value (common in proteomics)
  data_imputed ← filtered_data
  for (i in 1:nrow(data_imputed)) {
    row_vals ← data_imputed[i, ]
    if (any(is.na(row_vals))) {
      impute_val ← min(row_vals, na.rm = TRUE) - 0.5
      data_imputed[i, is.na(row_vals)] ← impute_val
    }
  }
```

# Normalization Methods

## 1. Median Normalization

```
# Calculate median for each sample
sample_medians ← apply(protein_matrix_imputed, 2,
                       median, na.rm = TRUE)
global_median ← median(sample_medians)

# Normalize
protein_matrix_median ← protein_matrix_imputed
for (i in 1:ncol(protein_matrix_median)) {
  protein_matrix_median[, i] ← protein_matrix_median[, i] -
    sample_medians[i] + global_median
}
```

# Normalization Methods

## 2. Quantile Normalization

```
library(limma)
protein_matrix_quantile ← normalizeBetweenArrays(
  protein_matrix_imputed,
  method = "quantile"
)
```

## 3. VSN Normalization

```
library(vsn)
vsn_fit ← vsn2(protein_matrix_imputed)
protein_matrix_vsn ← predict(vsn_fit, protein_matrix_imputed)
```

# Comparing Normalization Methods

```
plot_pca ← function(data, title, metadata) {
  pca_result ← prcomp(t(data), scale. = FALSE)
  var_exp ← summary(pca_result)$importance[2, 1:2] * 1

  pca_df ← data.frame(
    PC1 = pca_result$x[, 1],
    PC2 = pca_result$x[, 2],
    condition = metadata$condition,
    batch = metadata$batch
  )

  ggplot(pca_df, aes(x = PC1, y = PC2,
                     color = condition,
                     shape = batch)) +
    geom_point(size = 3) +
    labs(title = title,
         x = paste0("PC1 (", round(var_exp[1], 1), "%)"
         y = paste0("PC2 (", round(var_exp[2], 1), "%)"
}
```

Compare methods:

- Raw data
- Median normalized
- Quantile normalized
- VSN normalized

# Exercise 3.1: Apply Normalization

Apply all three normalization methods and:

1. Calculate CV for each method
2. Compare sample correlations
3. Choose the best method for your data

Try it yourself before looking at the solution!

# Exercise 3.1 Solution

```
calculate_mean_cv ← function(data) {
  cvs ← apply(data, 1, function(x)
    sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE) * 100)
  mean(cvs, na.rm = TRUE)
}

cat("Mean CV - Raw:",
    round(calculate_mean_cv(protein_matrix_imputed), 2), "%\n")
cat("Mean CV - Median:",
    round(calculate_mean_cv(protein_matrix_median), 2), "%\n")
cat("Mean CV - Quantile:",
    round(calculate_mean_cv(protein_matrix_quantile), 2), "%\n")

# Sample correlations
cor_raw ← mean(cor(protein_matrix_imputed)
               [upper.tri(cor(protein_matrix_imputed))])
cat("Mean correlation - Raw:", round(cor_raw, 3), "\n")
```

# Module 2: Batch Effect Correction

# Detecting Batch Effects

```
pca_result ← prcomp(t(protein_matrix_quantile), scale. = TRUE)
var_exp ← summary(pca_result)$importance[2, 1:2] * 100

pca_df ← data.frame(
  PC1 = pca_result$x[, 1],
  PC2 = pca_result$x[, 2],
  sample_id = colnames(protein_matrix_quantile)
) %>% merge(sample_metadata, by = "sample_id")

ggplot(pca_df, aes(x = PC1, y = PC2,
                   color = batch, shape = condition)) +
  geom_point(size = 4) +
  labs(title = "PCA - Batch Effect Detection",
       x = paste0("PC1 (", round(var_exp[1], 1), "%)"),
       y = paste0("PC2 (", round(var_exp[2], 1), "%)"))
```

# ComBat Batch Correction

```
library(sva)

# Prepare for ComBat
batch_vector ← sample_metadata$batch
condition_matrix ← model.matrix(~condition,
                                data = sample_metadata)

# Apply ComBat
protein_matrix_combat ← ComBat(
  dat = protein_matrix_quantile,
  batch = batch_vector,
  mod = condition_matrix,
  par.prior = TRUE,
  prior.plots = FALSE
)
```

# Before vs After Batch Correction

**Before ComBat:**

- Samples cluster by batch
- Technical variation visible
- Biological signal may be obscured

**After ComBat:**

- Batch effect reduced
- Biological groups clearer
- Improved downstream analysis

```
# Compare PCA plots before/after
pca_combat ← prcomp(t(protein_matrix_combat), scale. = TRUE)
# Plot both side by side...
```

# Scaling Methods

```r
# Z-score scaling (by protein)
protein_matrix_scaled ← t(scale(t(protein_matrix_combat)))

# Pareto scaling
protein_matrix_pareto ← t(scale(t(protein_matrix_combat))) /
  sqrt(apply(protein_matrix_combat, 1, sd, na.rm = TRUE))

# Heatmap with scaling
pheatmap(protein_matrix_combat[1:1000, ],
         scale = "row",
         main = "Heatmap with Row Scaling",
         show_rownames = FALSE)
```

# Exercise 3.2: Complete Preprocessing Pipeline

Create a complete preprocessing function that:

1. Filters proteins with >30% missing
2. Imputes missing values
3. Applies normalization
4. Corrects for batch effects

# Exercise 3.2 Solution

```
preprocess_proteomics ← function(raw_data, metadata,
                                 missing_threshold = 0.3,
                                 norm_method = "quantile",
                                 batch_correction = TRUE) {

  # Step 1: Filter missing values
  missing_per_protein ← rowSums(is.na(raw_data)) /
                        ncol(raw_data)
  filtered_data ← raw_data[missing_per_protein ⩽
                           missing_threshold, ]

  # Step 2: Impute
  data_imputed ← filtered_data
  for (i in 1:nrow(data_imputed)) {
    row_vals ← data_imputed[i, ]
    if (any(is.na(row_vals))) {
      impute_val ← min(row_vals, na.rm = TRUE) - 0.5
      data_imputed[i, is.na(row_vals)] ← impute_val
    }
  }
```

# Module 3: Differential Expression Analysis

# Introduction to limma

limma (Linear Models for Microarray Data) advantages:

- Empirical Bayes moderation
- Handles complex designs
- Works well with small sample sizes
- Robust and widely validated

# Basic Differential Expression Workflow

```
# Design matrix
design ← model.matrix(~0 + condition, data = sample_metadata)
colnames(design) ← c("Control", "Treatment")

# Fit linear model
fit ← lmFit(processed_data, design)

# Define contrast
contrast_matrix ← makeContrasts(
  TreatmentVsControl = Treatment - Control,
  levels = design
)

# Fit contrasts
fit2 ← contrasts.fit(fit, contrast_matrix)

# Empirical Bayes moderation
fit2 ← eBayes(fit2)
```

# Extract and Annotate Results

```
# Extract results
de_results ← topTable(fit2, coef = "TreatmentVsControl",
                      number = Inf)

# Add annotations
de_results$protein_id ← rownames(de_results)
de_results_annotated ← de_results %>%
  left_join(protein_annotations, by = "protein_id")

# View top results
cat("Top differentially expressed proteins:\n")
print(de_results_annotated[1:5,
      c("protein_id", "gene_symbol", "logFC", "P.Value")])
```

# Differential Expression Summary

```
cat("Differential Expression Summary:\n")
cat("Significant proteins (FDR < 0.05):",
    sum(de_results$adj.P.Val < 0.05), "\n")
cat("Upregulated (FC > 1.5, FDR < 0.05):",
    sum(de_results$adj.P.Val < 0.05 &
        de_results$logFC > log2(1.5)), "\n")
cat("Downregulated (FC < -1.5, FDR < 0.05):",
    sum(de_results$adj.P.Val < 0.05 &
        de_results$logFC < -log2(1.5)), "\n")
```

# Volcano Plot

```
# Prepare data
volcano_data ← de_results
volcano_data$significance ← "NS"
volcano_data$significance[volcano_data$adj.P.Val < 0.05 &
                          volcano_data$logFC > log2(1.5)] ← "Up"
volcano_data$significance[volcano_data$adj.P.Val < 0.05 &
                          volcano_data$logFC < -log2(1.5)] ← "Down"

ggplot(volcano_data, aes(x = logFC, y = -log10(adj.P.Val),
                         color = significance)) +
  geom_point(alpha = 0.6, size = 2) +
  scale_color_manual(values = c("Up" = "red",
                                "Down" = "blue",
                                "NS" = "grey")) +
  geom_hline(yintercept = -log10(0.05), linetype = "dashed") +
  geom_vline(xintercept = c(-log2(1.5), log2(1.5)),
             linetype = "dashed") +
  labs(title = "Volcano Plot: Treatment vs Control",
       x = "log2 Fold Change",
       y = "-log10 Adjusted P-value")
```

# MA Plot

```
volcano_data$AveExpr ← de_results$AveExpr

ggplot(volcano_data, aes(x = AveExpr, y = logFC,
                         color = significance)) +
  geom_point(alpha = 0.6, size = 2) +
  scale_color_manual(values = c("Up" = "red",
                                "Down" = "blue",
                                "NS" = "grey")) +
  geom_hline(yintercept = 0, linetype = "solid") +
  geom_hline(yintercept = c(-log2(1.5), log2(1.5)),
             linetype = "dashed") +
  labs(title = "MA Plot: Treatment vs Control",
       x = "Average Expression",
       y = "log2 Fold Change")
```

# Heatmap of DE Proteins

```r
# Select significant proteins
sig_proteins ← rownames(de_results[de_results$adj.P.Val < 0.05, ])

if (length(sig_proteins) > 1) {
  annotation_col ← data.frame(
    Condition = sample_metadata$condition
  )
  rownames(annotation_col) ← sample_metadata$sample_id

  pheatmap(processed_data[sig_proteins, ],
           scale = "row",
           annotation_col = annotation_col,
           show_rownames = FALSE,
           main = "Significant DE Proteins")
}
```

# Save Results for Day 4

```
# Create results directory
if (!dir.exists("results")) dir.create("results")

# Save processed data and results
saveRDS(processed_data, "results/day3_processed_data.rds")
saveRDS(de_results_annotated, "results/day3_de_results.rds")

# Save significant proteins
de_summary ← de_results_annotated %>%
  filter(adj.P.Val < 0.05) %>%
  select(protein_id, gene_symbol, logFC,
         P.Value, adj.P.Val, protein_name)

write.csv(de_summary, "results/day3_significant_proteins.csv",
          row.names = FALSE)
```

# Day 3 Summary

# What We Covered Today

- ✓ How to handle missing values in proteomic data
- ✓ Different normalization methods and their effects
- ✓ Batch effect detection and correction using ComBat
- ✓ Differential expression analysis with limma
- ✓ Visualization of DE results (volcano plots, MA plots, heatmaps)

# Key Takeaways

1. **Normalization** is crucial for making samples comparable
2. **Batch correction** can remove technical artifacts
3. **limma** provides robust differential expression analysis
4. **Visualization** helps interpret complex DE results

# Homework

1. Try different normalization methods and compare results
2. Experiment with different FDR and fold change thresholds
3. Create custom visualizations for your specific research questions

```
# Prepare for Day 4
install.packages(c("clusterProfiler", "enrichplot",
                   "org.Hs.eg.db"))
BiocManager::install(c("clusterProfiler", "enrichplot",
                       "org.Hs.eg.db"))
```

# Additional Resources

- limma User's Guide
- ComBat paper
- Proteomics normalization review

Questions?

See you tomorrow for Day 4!