QCB455/COS551 Homework 1

Sequencing, Statistical Concepts, and Population Genetics

William Svoboda (wsvoboda)

Last edited September 24, 2021

Contents

Collaboration Statement

I talked with Brendan McManamon (bm18, student), Debby Park (debbyp, student), and Danny Simpson (ds65, TA) about this homework.

1 Sequencing and Assembly

1. What is shotgun sequencing and how has this technique significantly contributed to the acquisition of the human genome?

- **Shotgun sequencing is a method for sequencing a desired genome. Shotgun sequencing works off random DNA strands after the target genome has been separated into individual fragments. Because this method requires less DNA material and does not need a mapping step, whole-genome sequencing could be accomplished faster and at a lower cost than with earlier techniques.**

2. What are the relative advantages and disadvantages of Sanger sequencing vs. next generation (Illumina) sequencing?

- **For sequencing a very small number of samples, Sanger sequencing and the equipment required are actually less expensive than NGS. On the other hand, while NGS might have a larger startup cost it is dramatically better at sequencing large amounts of DNA. This is because NGS can run millions of sequencing reactions in parallel. Sanger sequencing also requires multiple reactions to sequence a single fragment.**

3. What is paired-end sequencing? Give a cartoon example (i.e., a sample set of reads and/or a piece of a genome) where paired-end sequencing leads to improved assemblies.

- **Paired-end sequencing works by sequencing both ends of a DNA fragment. As an example, consider the fragment `GAAGA`. If read from only one end (say left-to-right), it would be easy to confuse `GAAGA` with a fragment like `GAAGG`. By sequencing both ends, it would become much easier to form the alignments since the opposite readings would help disambiguate.**

2 Computing q-values from p-values

1. Load the file **sim-pvals.txt**.

```
# Load sim-vals.txt using read.table
values <- read.table("./hw1data/sim-pvals.txt", header = TRUE)
```

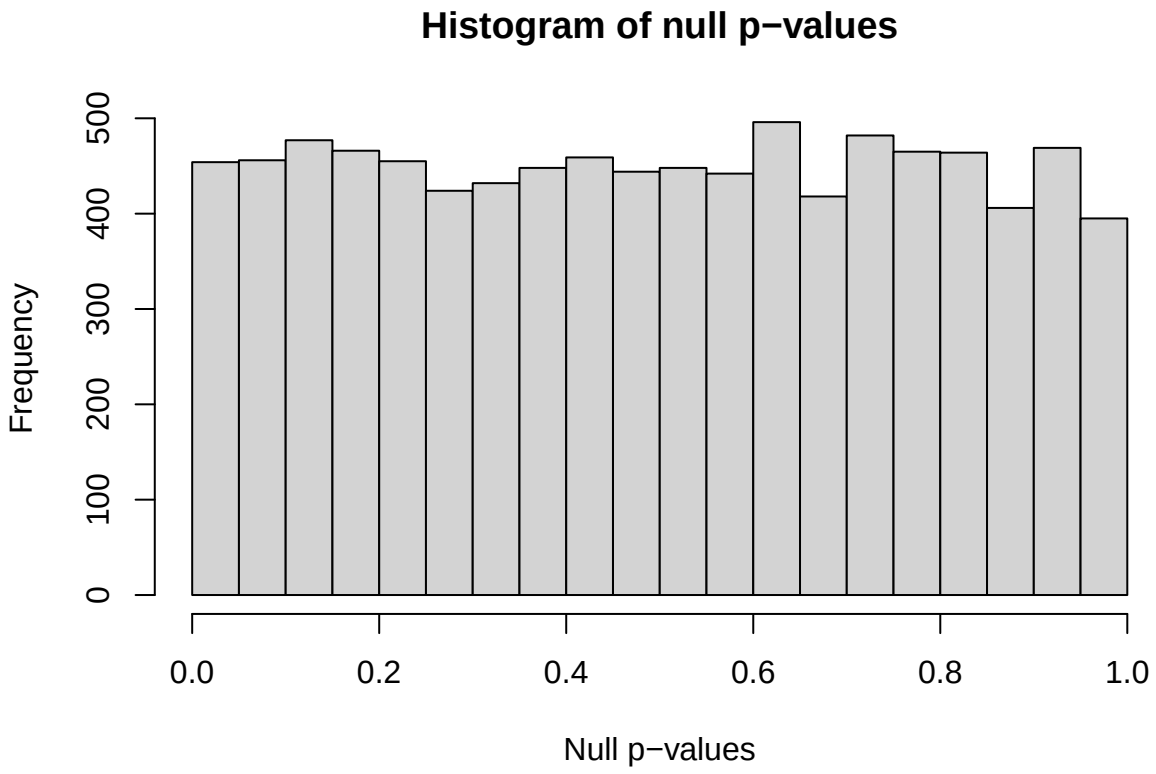
2. What is π_0 (the proportion of condition-negatives or null hypotheses) in this data?

```
# Divide number of null p-values by total number of rows
num_rows <- nrow(values)
proportion <- sum(values$null == 1)/num_rows
```

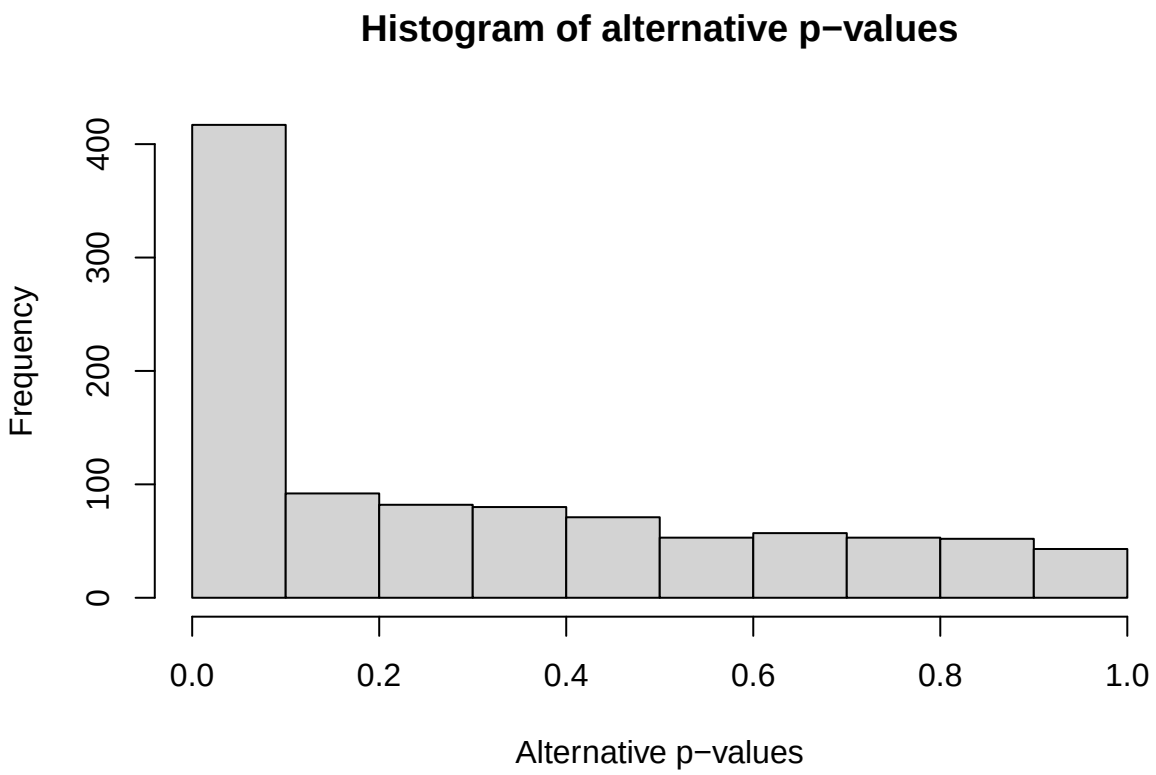
- π_0 **is 0.9**.

3. Plot the p-value distribution of the null and alternative p-values separately.

```
# Create a vector of null p-values and a vector of alternative p-values
null_values <- values %>%
    filter(null == 1) %>%
    pull(pval)
a_values <- values %>%
    filter(null == 0) %>%
    pull(pval)

# Plot each vector separately
hist(null_values, main = "Histogram of null p-values", xlab = "Null p-values")
```



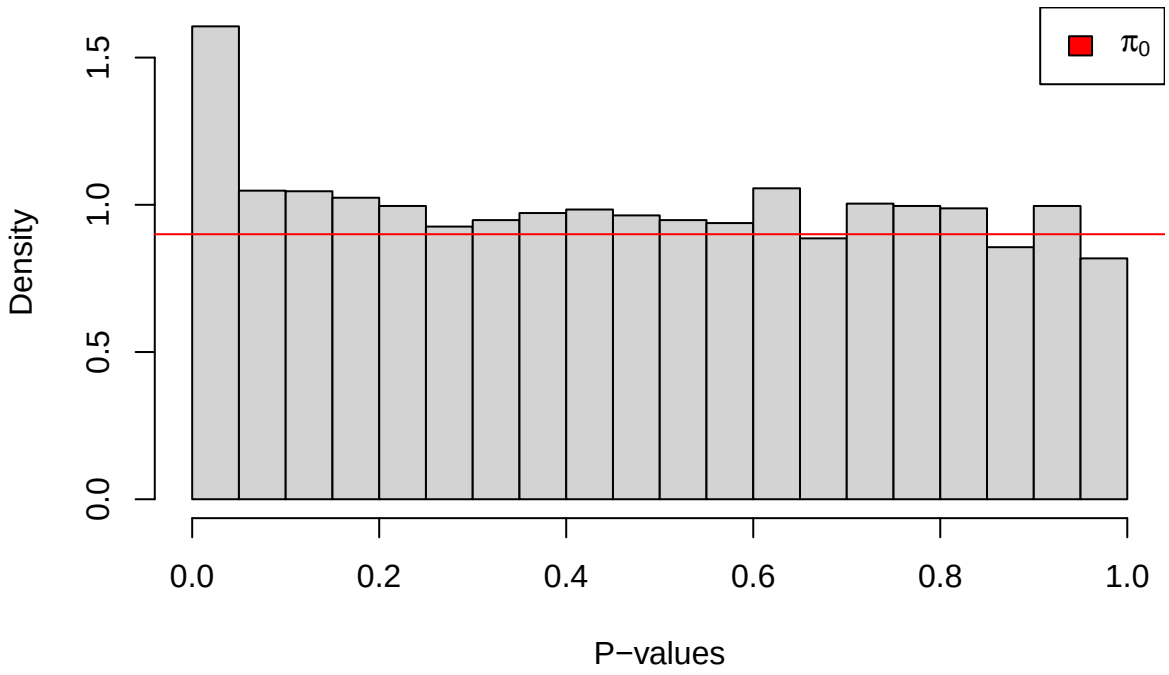
```
hist(a_values, main = "Histogram of alternative p-values", xlab = "Alternative p-values")
```



- **The distribution of null p-values is uniform by the definition of a Type I error. If the null-hypothesis is true, than the p-value is a random probability and must come from a uniform distribution. The distribution of alternative p-values is left-tailed because if the null hypothesis is false than p-values are not probabilities and should all fall close to 0.**

4. Now plot the combined p-value distribution.

```
# Select p-values in ascending order
pvals <- values[order(values$pval), ]$pval
hist(pvals, freq = FALSE, main = "Combined p-value distribution", xlab = "P-values")
# Horizontal line with height equal to null p-value proportion
abline(h = proportion, col = "red")
legend("topright", legend = c(expression(pi[0])), fill = c("red"))
```



- **In general, π_0 marks which part of the distribution comes from null p-values and which do not.**

5. Based on the last figure, how do you suggest estimating π_0 from the combined p-value distribution when the true condition of each case is unknown?

- **I would look at where the distribution flattens out, and draw a horizontal line at that point because we know that larger p-values (generally) come from the null distribution. This line would approximate the value of π_0.**

6. Calculate q-values manually.

```
# Preallocate vector to hold each FDR
fdr_vals <- vector(mode = "numeric", length = num_rows)

# For each p-value estimate the FDR
for (i in 1:length(pvals)) {
    fdr_vals[i] <- (pvals[i] * num_rows * proportion)/(sum(pvals <= pvals[i]))
}

# Ensure monotonicity, p-values are already in ascending order
for (i in 1:(length(fdr_vals) - 1)) {
    current_fdr <- fdr_vals[i]
    current_pval <- pvals[i]
    next_fdr <- fdr_vals[i + 1]
    next_pval <- pvals[i + 1]
    # If FDR estimate actually decreases for larger p-value
    if (next_fdr < current_fdr) {
        # Accept lowest FDR that includes given p-value
        min_fdr <- min(fdr_vals[i:length(fdr_vals)])
        fdr_vals[i] <- min_fdr
    }
    # Cap q-value estimates to 1
```

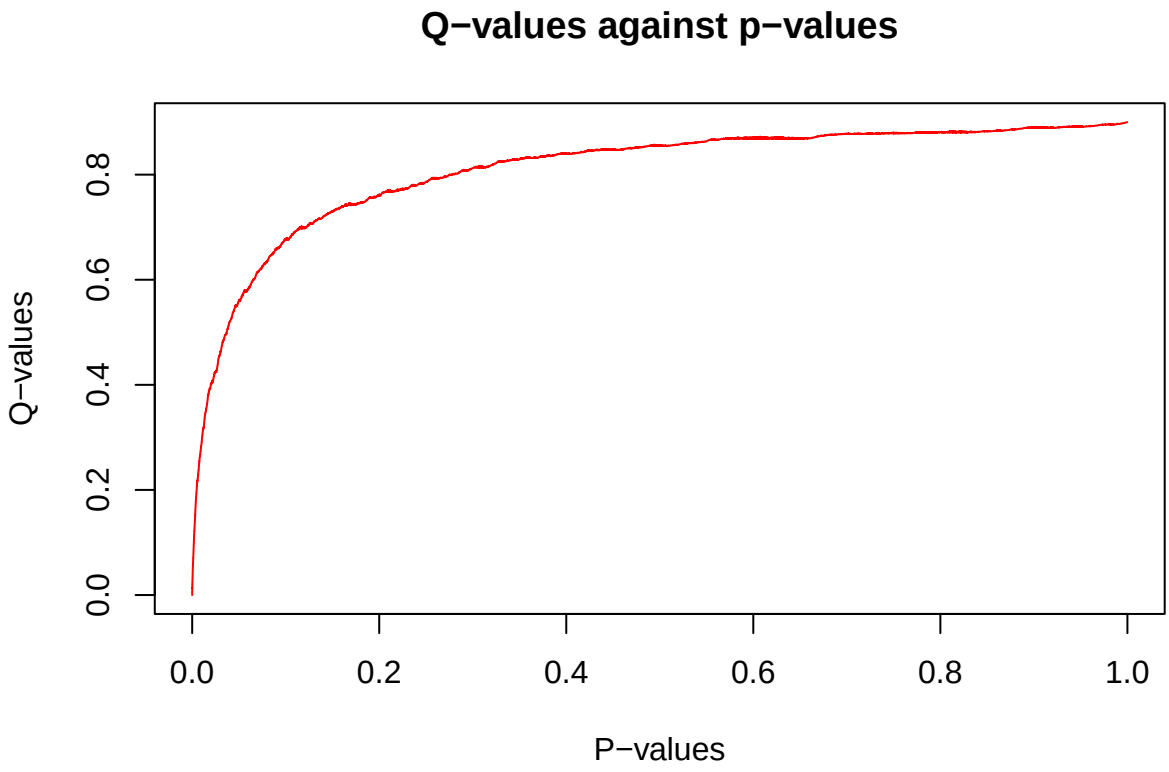
```
    fdr_vals[i] <- min(fdr_vals[i], 1)
}
```

7. Explain why $q\prime(p)$ above estimates the FDR.

- **$pm\pi_0$ represents the number of false discoveries given a p-value p, because it multiplies the total number of p-values m by the proportion of null hypotheses π_0 and then by the current p. The denominator, $\#\{p\prime \leq p\}$, is just the number of p-values smaller or equal to the given p. This means that their ratio will represent the proportion or rate of false discoveries out of the total number of positive events observed.**

8. Plot q-values against p-values and describe the relationship. What is the maximum q-value and why?

```
plot(pvals, fdr_vals, type = "l", col = "red", main = "Q-values against p-values",
    xlab = "P-values", ylab = "Q-values")
```



- **As q-values increase, p-values increase exponentially at first before leveling off (An "S" curve). The maximum q-value would approach π_0 since that would be the proportion of null hypotheses.**

9. Compute q-values with the R `qvalue` package. Use the original p-values and nothing else as input, with default options. Report the π_0 estimate of the package.

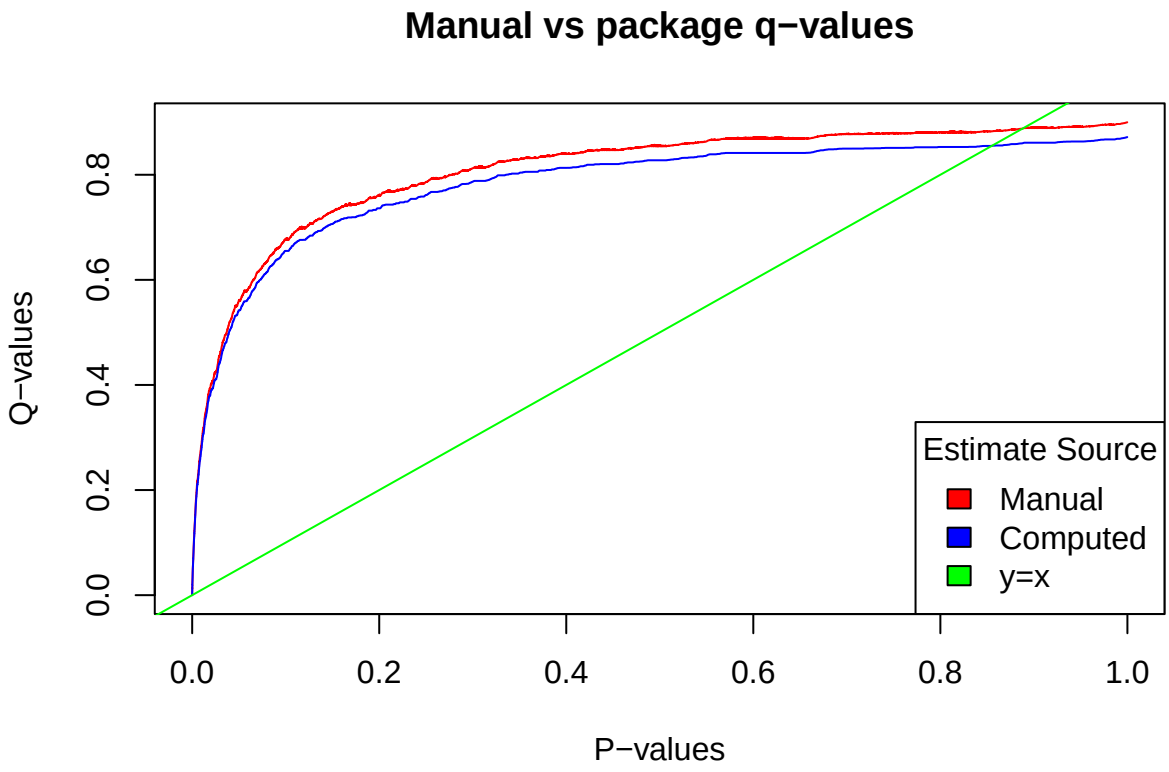
```
package_pi0est <- qvalue::pi0est(pvals)$pi0
```

- **The π_0 estimate of the package is 0.8717847.**

10. Show a plot that compares our manual q-value estimates with the q-value estimates of the package, and the $y = x$ line for comparison. Do they differ? Why?

```
# Compute q-values from package
package_qvals <- qvalue::qvalue(pvals)$qvalues

# Plot and compare both estimates Manual q-value estimates
plot(x = pvals, y = fdr_vals, col = "red", main = "Manual vs package q-values", xlab = "P-values",
```

```
    ylab = "Q-values", type = "l")
# Computed q-value estimates
lines(x = pvals, y = package_qvals, col = "blue")
abline(a = 0, b = 1, col = "green")
legend("bottomright", legend = c("Manual", "Computed", "y=x"), fill = c("red", "blue",
    "green"), title = "Estimate Source")
```



- **Each q-value estimate appears to be slightly greater than the corresponding estimate calculated from the package. This difference may be because the package uses a more sophisticated algorithm in order to estimate q-values than the method presented in this assignment.**

3 Precision-recall Analysis

3.1 Data import and cross-checks

Download the following datasets and import them into your working environment:

1. The list of genes that are differentially expressed in your experiment: **differentially_expressed_orfs.txt**

```
differentially_expressed_orfs <- read.table("./hw1data/differentially_expressed_orfs.txt",
    col.names = c("gene"))
```

2. The list of genes known to be involved in a specific biological process: **positive_orfs.txt**

```
positive_orfs <- read.table("./hw1data/positive_orfs.txt", col.names = c("gene"))
```

To verify that your data import was successful, perform the following cross-checks:

1. Is your list of differentially expressed genes unique? If not, which genes are repeated? How many times are they repeated? Remove the repeated ORFs, if any, keeping only their first occurrence.

```
# Find repeated genes and their count
dup_differentially_expressed_orfs <- differentially_expressed_orfs %>%
    group_by(gene) %>%
    filter(n() > 1) %>%
    summarise(count = n())

# Create table to show repeated genes
knitr::kable(dup_differentially_expressed_orfs, caption = "Repeated Genes")
```

Table 1: Repeated Genes

gene	count
YDL069C	2
YJL166W	2
YKL016C	2
YLL006W	3

```
# Remove repeated ORFs
differentially_expressed_orfs <- unique(differentially_expressed_orfs)
```

2. How long is the list of known genes? How many of them appear in your differentially expressed list?

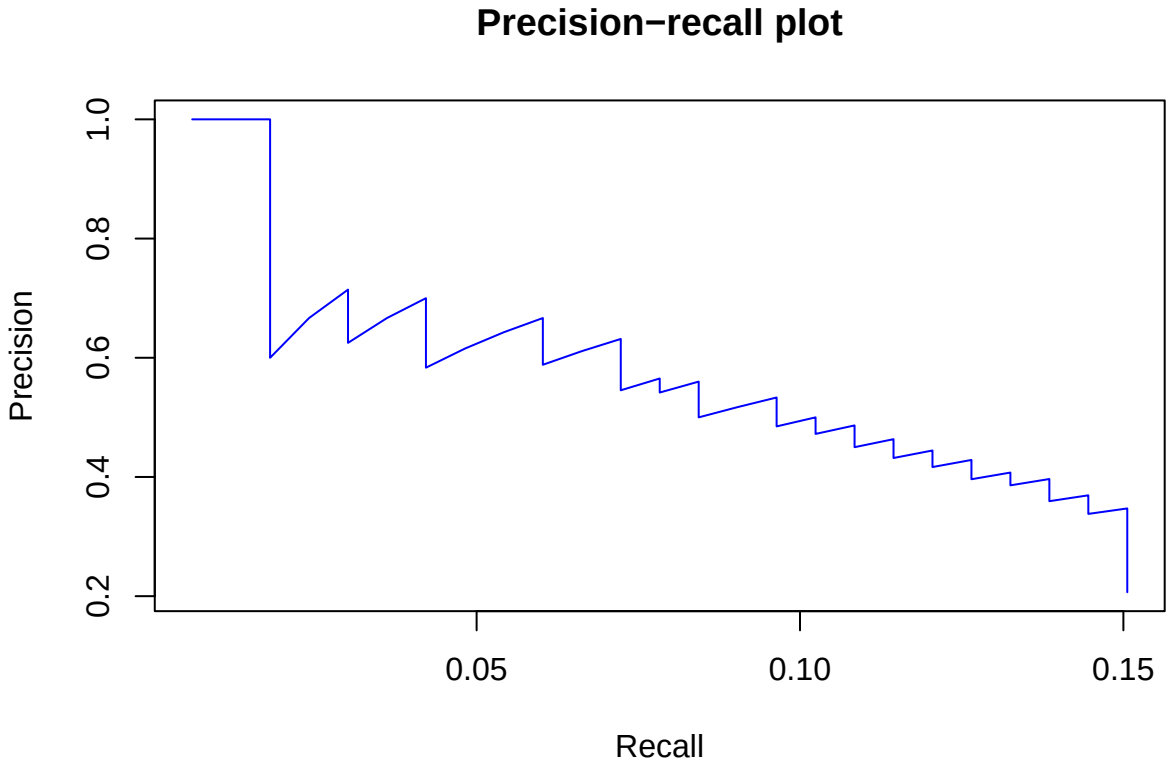
```
# Get length of known genes
length_positive_orfs <- nrow(positive_orfs)

# Find known genes in differentially expressed list
intersection_orfs <- intersect(positive_orfs, differentially_expressed_orfs)

# Get count of intersected genes
length_intersection_orfs <- nrow(intersection_orfs)
```

- **There are 166 genes in the list of known genes. There are 25 known genes that appear in the differentially expressed list.**

3.2 Precision-recall analysis

Assume that the order of genes in the differentially_expressed_orfs.txt file reflects their experimental ranking, which is based on the extent and confidence of their differential expression. At each position in this ranked list, calculate the precision and the recall of the data relative to the list of known genes. Plot the corresponding precision-recall plot.

```
# Convert dataframes to vector
vec_differentially_expressed_orfs <- differentially_expressed_orfs$gene
vec_positive_orfs <- positive_orfs$gene

# Preallocate vectors to hold corresponding precision and recalls
precision_vals <- vector(mode = "numeric", length = length(vec_differentially_expressed_orfs))
recall_vals <- vector(mode = "numeric", length = length(vec_differentially_expressed_orfs))

# Helper variables
tp <- 0
fp <- 0

for (i in 1:length(vec_differentially_expressed_orfs)) {
    # Update TP, FP, and FN counts
    if (vec_differentially_expressed_orfs[i] %in% vec_positive_orfs) {
        tp <- tp + 1
    } else {
        fp <- fp + 1
    }
    fn <- length(vec_positive_orfs) - tp

    # Calculate precision at current position
    precision_vals[i] <- tp/(tp + fp)

    # Calculate recall at current position
    recall_vals[i] <- (tp/(fn + tp))
}

# Plot precision-recall plot
plot(recall_vals, precision_vals, type = "l", col = "blue", main = "Precision-recall plot",
    xlab = "Recall", ylab = "Precision")
```



1. What is the precision of these data at 10% recall?

- **The precision is approximately 0.5 at 10% recall.**

2. Do you think it is reasonable to conclude that your experiment is recapitulating well previously known biology? Motivate your answer.

- **The graph shows that precision was very high for the genes at the top of the differentially expressed list. This makes sense, because genes were ranked based on the extent and confidence of their differential expression. In other words, the genes truly known to be associated with the chemical stress response were preferentially located at the top of the list (and thereby recapitulated previously known biology).**

4 Population Genetics

1. Write code to determine the number of segregating sites, S, in populations 1 and 2. Hint: the R function `readLines` can help read in the data files.

```
num_chromosomes <- 100
num_bps <- 10000

# Read population 1 data and convert it into a table
con_pop1 <- file("./hw1data/pop1.txt")
pop1 <- readLines(con = con_pop1, n = num_chromosomes)
close(con_pop1)
pop1_seq_length <- nchar(pop1[1])
pop1 <- read.fwf(file = textConnection(pop1), widths = rep(1, pop1_seq_length))

# Count number of segregating sites in population 1 and find frequency of
# alleles labled 1
pop1_ss_vec <- vector(mode = "numeric", length = pop1_seq_length)
for (i in 1:pop1_seq_length) {
    # Check if site at current position is a segregating site (does it vary
    # between chromosomes?)
    if (length(unique(pop1[, i])) > 1) {
        # Store frequency of alleles labeled 1 at current position
        pop1_ss_vec[i] <- sum(pop1[, i])
    }
}
# Count number of segregating sites
pop1_ss_count <- sum(pop1_ss_vec > 0)

# Read population 2 data and convert it into a table
con_pop2 <- file("./hw1data/pop2.txt")
pop2 <- readLines(con = con_pop2, n = num_chromosomes)
close(con_pop2)
pop2_seq_length <- nchar(pop2[1])
pop2 <- read.fwf(file = textConnection(pop2), widths = rep(1, pop2_seq_length))

# Count number of segregating sites in population 2 and find frequency of
# alleles labled 1
pop2_ss_vec <- vector(mode = "numeric", length = pop2_seq_length)
for (i in 1:pop2_seq_length) {
    # Check if site at current position is a segregating site (does it vary
    # between chromosomes?)
    if (length(unique(pop2[, i])) > 1) {
        # Store frequency of alleles labeled 1 at current position
        pop2_ss_vec[i] <- sum(pop2[, i])
    }
}
# Count number of segregating sites
pop2_ss_count <- sum(pop2_ss_vec > 0)
```

- **Population 1 has 15 segregating sites, while population 2 has 8 segregating sites.**

2. For each population, calculate a per nucleotide estimate of θ_W and π (i.e., divide θ_W and π by the number of bps sequenced in the region).

```
# Calculate per nucleotide estimate of nucleotide polymorphism for population 1
i <- 1:(num_chromosomes - 1)
pop1_theta_w <- pop1_ss_count/sum(1/i)
pop1_pne_theta_w <- pop1_theta_w/num_bps

# Calculate per nucleotide estimate of nucleotide diversity for population 1
pop1_ss_freq_sum <- 0
for (i in 1:pop1_seq_length) {
    if (pop1_ss_vec[i] > 0) {
        pop1_ss_freq_sum <- pop1_ss_freq_sum + (2 * (pop1_ss_vec[i]/num_chromosomes) *
            ((num_chromosomes - pop1_ss_vec[i])/num_chromosomes))
    }
}
pop1_pi <- (num_chromosomes/(num_chromosomes - 1)) * pop1_ss_freq_sum
pop1_pne_pi <- pop1_pi/num_bps

# Calculate per nucleotide estimate of nucleotide polymorphism for population 2
i <- 1:(num_chromosomes - 1)
pop2_theta_w <- pop2_ss_count/sum(1/i)
pop2_pne_theta_w <- pop2_theta_w/num_bps

# Calculate per nucleotide estimate of nucleotide diversity for population 2
pop2_ss_freq_sum <- 0
for (i in 1:pop2_seq_length) {
    if (pop2_ss_vec[i] > 0) {
        pop2_ss_freq_sum <- pop2_ss_freq_sum + (2 * (pop2_ss_vec[i]/num_chromosomes) *
            ((num_chromosomes - pop2_ss_vec[i])/num_chromosomes))
    }
}
pop2_pi <- (num_chromosomes/(num_chromosomes - 1)) * pop2_ss_freq_sum
pop2_pne_pi <- pop2_pi/num_bps
```

- **For population 1 the per nucleotide estimate of θ_W is 2.8972197×10^{-4} and for π it is 4.5371717×10^{-4}. For population 2 the per nucleotide estimate of θ_W is 1.5451838×10^{-4} and for π it is 2.9145455×10^{-4}.**

3. Calculate the statistic $D = \pi - \theta_W$ for each population.

```
# Calculate D for population 1
pop1_d <- pop1_pi - pop1_theta_w

# Calculate D for population 1
pop2_d <- pop2_pi - pop2_theta_w
```

- **For population 1, $D = 1.639952$. For population 2, $D = 1.3693616$.**

4. Use your estimates of θ_W to estimate N_e in each population. Note that for this problem, you can assume that the mutation rate μ is equal to 1×10^{-8} per site per generation.

```
mu <- 1e-08

# Estimate effect population size for population 1
pop1_ne <- pop1_theta_w/(4 * mu)

# Estimate effect population size for population 2
pop2_ne <- pop2_theta_w/(4 * mu)
```

- **For population 1, N_e is estimated to be 7.2430492×10^7. For population 2, N_e is estimated to be 3.8629596×10^7.**

5. Based on the data in 1-4, do you think N_e is the same or different between population's 1 and 2? What aspects of the data are most informative in arriving at your answer?

- **Based on the data in 1-4, I believe N_e is different between the two populations. Because we assume μ is the same per site per generation, the only variable that will increase or decrease the effective population size is the estimate of θ_w. Population 1 had greater nucleotide polymorphism (as a function of having more segregating sites), meaning we should expect N_e to be greater as well.**

```
sessionInfo(package = NULL)
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] formatR_1.11        knitr_1.34          qvalue_2.24.0
## [4] BiocManager_1.30.16 dplyr_1.0.7
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.7       pillar_1.6.2     compiler_4.1.1   plyr_1.8.6
##  [5] highr_0.9        tools_4.1.1      digest_0.6.27    evaluate_0.14
##  [9] lifecycle_1.0.0  tibble_3.1.4     gtable_0.3.0     pkgconfig_2.0.3
## [13] rlang_0.4.11     yaml_2.2.1       xfun_0.26        fastmap_1.1.0
## [17] stringr_1.4.0    generics_0.1.0   vctrs_0.3.8      grid_4.1.1
## [21] tidyselect_1.1.1 glue_1.4.2       R6_2.5.1         fansi_0.5.0
## [25] rmarkdown_2.11   purrr_0.3.4      ggplot2_3.3.5    reshape2_1.4.4
## [29] magrittr_2.0.1   scales_1.1.1     ellipsis_0.3.2   htmltools_0.5.2
## [33] splines_4.1.1    colorspace_2.0-2 utf8_1.2.2       stringi_1.7.4
## [37] munsell_0.5.0    crayon_1.4.1
```