Functions

Michael Tartamella

5/5/2023

The function, “pvals.manyGroups.anova.kw”, generates p-values for each biomarker in the situation where we have more than groups. Our function first reads in the data and next we find the total number of biomarkers our data contains. We then preallocate this amount of p-values. Next, I use a “For” loop which runs from 1 to the number of biomarkers we have. Inside this loop we iterate through all the biomarkers of our data and for each one we first preform Shapiro test to check if we can assume data to be normal. If the value of the p-value we get from Shapiro test is greater than , we assume data is normally distributed and we then use one-way Anova test to generate our p-value for the biomarker. If the value of the p-value we get from Shapiro test is less or equal to , we do not assume data is normally distributed and we use Kruskal Wallis test to generate our p-value for the biomarker. After we do this for every biomarker we then return both the p-values for each biomarker and the total number of observations for each biomarker in the form of a list.

pvals.manyGroups.anova.kw <- function(data)  
{  
 k <- ncol(data) - 1  
 p.values <- numeric(k)  
 for (i in 1:k)  
 {  
 nam <- paste("gene", i, sep = "")  
 result <- shapiro.test(data[[nam]])  
 if (result$p.value > .05)  
 {  
 anova.test <- aov(data[[nam]] ~ as.factor(data$group))  
 p.values[i] <- summary(anova.test)[[1]][["Pr(>F)"]][1]  
 }  
 else  
 {  
 kw.test <- kruskal.test(data[[nam]] ~ as.factor(data$group))  
 p.values[i] <- kw.test$p.value  
 }  
 }  
 return(list(p.value=p.values, n=nrow(data)))  
}

The function, “pvals.twoGroups.t.wilcoxon”, generates p-values for each biomarker in the situation where we have only groups. Our function first reads in the data and next we find the total number of biomarkers our data contains. We then preallocate this amount of p-values. Next, I use a “For” loop which runs from 1 to the number of biomarkers we have. Inside this loop we iterate through all the biomarkers of our data and for each one we first preform Shapiro test to check if we can assume data to be normal. If the value of the p-value we get from Shapiro test is greater than , we assume data is normally distributed and then we use function “var.test” in order to check if we can use equal variance assumption. If the p-value generated from “var.test” is greater than we preform t-test with equal variance assumption to generate our p-value for the biomarker. If the p-value generated from “var.test” is less or equal to we preform t-test with unequal variance assumption to generate our p-value for the biomarker. If the value of the p-value we get from Shapiro test is less or equal to , we do not assume data is normally distributed and we use Wilcoxon rank sum test to generate our p-value for the biomarker. After we do this for every biomarker, we return both the p-values for each biomarker and the total number of observations for each biomarker in the form of a list.

pvals.twoGroups.t.wilcoxon <- function(data)  
{  
 k <- ncol(data) - 1  
 p.values <- numeric(k)  
 for (i in 1:k)  
 {  
 nam <- paste("gene", i, sep = "")  
 result <- shapiro.test(data[[nam]])  
 if (result$p.value > .05)  
 {  
 var.equal <- var.test(data[[nam]], data$group)  
 if (var.equal$p.value > 0.05)  
 {  
 t <- t.test(data[[nam]], data$group, var.equal = TRUE)  
 p.values[i] <- t$p.value  
 }  
 else  
 {  
 t <- t.test(data[[nam]], data$group, var.equal = FALSE)  
 p.values[i] <- t$p.value  
 }  
 }  
 else  
 {  
 wilcox <- wilcox.test(data[[nam]], data$group)  
 p.values[i] <- wilcox$p.value  
 }  
 }  
 return(list(p.value=p.values, n=nrow(data)))  
}

The function, “pvals.categorical.fisher.chisquared”, generates p-values for each biomarker in the situation where we categorical data. Our function first reads in the data and a value for a threshold which defaults to if the user does not enter a value for this parameter. We next find the total number of biomarkers our data contains. We then preallocate this amount of p-values. Next, I use a “For” loop which runs from 1 to the number of biomarkers we have. Inside this loop we iterate through all the biomarkers of our data and for each one we first check if the minimum expected count for each group of the biomarker. If the value of the minimum expected count for the biomarker is less than or equal to the parameter, “threshold”, we use Fisher’s exact test to generate our p-value for the biomarker. If the value of the minimum expected count for the biomarker is greater than the parameter, “threshold”, we use chi-squared test to generate our p-value for the biomarker. After we do this for every biomarker we return both the p-values for each biomarker and the total number of observations for each biomarker in the form of a list.

pvals.categorical.fisher.chisquared <- function(data, threshold = 5)  
{  
 k <- ncol(data) - 1  
 p.values <- numeric(k)  
 for (i in 1:k)  
 {  
 nam <- paste("gene", i, sep = "")  
 min.expected.count <- min(table(data[[nam]], data$group))  
 if (min.expected.count <= threshold)  
 {  
 f.test <- fisher.test(data[[nam]], data$group)  
 p.values[i] <- f.test$p.value  
 }  
 else  
 {  
 chi.test <- chisq.test(data[[nam]], data$group)  
 p.values[i] <- chi.test$p.value  
 }  
 }  
 return(list(p.value=p.values, n=nrow(data)))  
}

The function, “fisher.pool”, reads in the p-values for a specific biomarker. First, we find the total number of p-values sent to the function and store this information into a variable, ‘k’. We then create a variable “stat” to hold the value . We generate our pooled p-value utilizing chi-squared distribution with degrees of freedom . Finally, we return the pooled p-value for the biomarker.

fisher.pool <- function(p.values)  
{  
 k <- length(p.values)  
 stat <- -2 \* sum(log(p.values))  
 p.pooled <- pchisq(stat, df = 2 \* k, lower.tail = FALSE)  
 return(p.pooled)  
}

The function, “stouffer.pool”, reads in the p-values for a specific biomarker. First, we find the total number of p-values sent to the function and store this information into a variable, ‘k’. We then create a variable “stat” to hold the value . We generate our pooled p-value utilizing normal distribution. Finally, we return the pooled p-value for the biomarker.

stouffer.pool <- function(p.values)  
{  
 k <- length(p.values)  
 stat <- sum(qnorm(p.values, lower.tail = FALSE)) / sqrt(k)  
 p.pooled <- pnorm(stat, lower.tail = FALSE)  
 return(p.pooled)  
}

The function, “weighted.stouffer.pool”, reads in the p-values for a specific biomarker and the number of observations which generated each of these p-values, which we store into variable ‘n’. First, we find the total number of p-values sent to the function and store this information into a variable, ‘k’. Next, we set variable, “weight” equal to the . I use method that pooled p-value equals . Finally, we return the pooled p-value for the biomarker.

weighted.stouffer.pool <- function(p.values, n)  
{  
 k <- length(p.values)  
 weight <- sqrt(n)  
 stat <- qnorm(p.values, lower.tail=FALSE)  
 p.pooled <- pnorm(sum(weight \* stat) / sqrt(sum(weight^2)), lower.tail=FALSE)  
 return(p.pooled)  
}

The function, “lancaster.pool”, reads in the p-values for a specific biomarker and the number of observations which generated each of these p-values, which we store into variable ‘n’. First, we find the total number of p-values sent to the function and store this information into a variable, ‘k’. Next, I generate pooled p-value by transforming our p-values to chi-square variables using inverse chi-square transformation with the degrees of freedom equal to the sum of the observations of the p-values we read in. Finally, we return the pooled p-value for the biomarker.

lancaster.pool <- function(p.values, n)  
{  
 stat <- sum(qchisq(p.values, n) \* (1 - p.values))  
 p.pooled <- pchisq(stat, sum(n))  
 return(p.pooled)  
}

The function, “min.p.pool”, reads in the p-values for a specific biomarker. First, we find the total number of p-values sent to the function and store this information into a variable, ‘k’. Next, we find the minimum value out of all of our p-values. Then, we find our pooled p-value using a beta distribution with degrees of freedom and . Finally, we return the pooled p-value for the biomarker.

minp.pool <- function(p.values)  
{  
 k <- length(p.values)  
 p.min <- min(p.values)  
 p.pooled <- pbeta(p.min, 1, k)  
 return(p.pooled)  
}

The function, “max.p.pool”, reads in the p-values for a specific biomarker. First, we find the total number of p-values sent to the function and store this information into a variable, ‘k’. Next, we find the maximum value out of all of our p-values. Then, we find our pooled p-value using a beta distribution with degrees of freedom and . Finally, we return the pooled p-value for the biomarker.

maxp.pool <- function(p.values)  
{  
 k <- length(p.values)  
 p.max <- max(p.values)  
 p.pooled <- pbeta(p.max, k, 1)  
 return(p.pooled)  
}

The function, “weighted.fisher.pool”, reads in the p-values for a specific biomarker. First, we find the total number of p-values sent to the function and store this information into a variable, ‘k’. Next, we set a variable “weight” equal to . Then, I use gamma distribution to find values for our test statistic and gamma to generate our pooled p-value. Finally, we return the pooled p-value for the biomarker.

weighted.fisher.pool <- function(p.values)  
{  
 k <- length(p.values)  
 weight <- sqrt(k)  
 stat <- qgamma(p.values, shape = weight, scale = 2)  
 p.pooled <- dgamma(sum(stat), shape = sum(weight), scale = 2)  
 return(p.pooled)  
}

The function, “adjust”, reads in all of our pooled p-values. We make a decision based on the number of p-values being sent to the function. If this number is less than or equal to , we use function “p.adjust” on our p-values using method “bonferroni”. If this number is greater than , we use function “p.adjust” on our p-values using method “fdr”. Finally, we return the adjusted p-values to the user.

adjust <- function(p.values)  
{  
 n <- length(p.values)  
 if (n <= 10)  
 {  
 adjusted.pvalues <- p.adjust(p.values, method = "bonferroni")  
 }  
 else  
 {  
 adjusted.pvalues <- p.adjust(p.values, method = "fdr")  
 }  
 return(adjusted.pvalues)  
}

References

Chang, L., Lin, H., Sibille, E., & Tseng, G. C. (2013). Meta-analysis methods for combining multiple expression profiles: comparisons, statistical characterization and an application guideline. BMC Bioinformatics, 14(1). <https://doi.org/10.1186/1471-2105-14-368>

Li, J., & Tseng, G. C. (2011). An adaptively weighted statistic for detecting differential gene expression when combining multiple transcriptomic studies. The Annals of Applied Statistics, 5(2A). <https://doi.org/10.1214/10-aoas393>

Zaykin, D. V. (2011). Optimally weighted Z-test is a powerful method for combining probabilities in meta-analysis. Journal of Evolutionary Biology, 24(8), 1836–1841. <https://doi.org/10.1111/j.1420-9101.2011.02297.x>