Minitask1

Florencia Luque and Seyed Amirhossein Mosaddad

2024-10-10

Part 1 - Apply Family

The apply Family is a way to apply a function to different types of data like a row, vector, matrix, etc. This functions have been replace or you should replace it with others libraries that do the same with an optimal result. This function is an alternative to a loop when you know waht you need to do to the data. #explain why or something about the general ways of the apply family

The family is created by this functions; apply(), lapply() , sapply(), vapply(), mapply() and tapply() functions.

apply

This function can be use to an arrays in different dimensions like a matrix. You will need a X the array, Margin 1 o 2 if you want to apply the function by row (1) and over columns (2). You can also apply on both using Margin=c(1,2) and obviously the fun that will be the function that you would like to apply like the sum or the mean. Example:

```r
mat = matrix(data = rnorm(20,20,4),nrow = 4,ncol = 5,byrow = TRUE)
print(mat)
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]
## [1,] 18.17144 23.57650 14.66258 28.17201 14.68590
## [2,] 17.17828 20.05414 29.63055 18.60789 19.74234
## [3,] 12.50154 24.02887 21.57654 25.29020 17.75962
## [4,] 14.27859 23.01254 22.06847 24.83915 18.63558
```

```r
sum_by_column = apply(mat,MARGIN = 2,FUN = sum)

print(sum_by_column)
```

```
## [1] 62.12984 90.67206 87.93814 96.90925 70.82344
```

tapply

Is very similar to apply but you can apply the function by groups. You can use this function to a dataframe or a matrix and you will need the data, the index that will be the factor that you want to group by and the fun. You can have more arguments but is not essential to make it work. We can transform or data to aggregate a factor variable and create and example.

```
data = data.frame(mat)
data$grupo = c("A","B","B","A")
colnames(data)[1] = "age"
colnames(data)[2] = "bmi"
data$grupo = as.factor(data$grupo)

tapply(data$age,data$grupo,mean)
```

```
##        A        B
## 16.22502 14.83991
```

lapply

This function is apply over a list or a vector and return a list of the same lenght of the original input. This is a way to apply a function to every element on a list. The function will need the list or vector and the function that you want to apply. This mean that if you give it a vector it will apply the function to each element and if you give it a list it will apply by elements of the list.

```
lista = list(tr = c(1:6),mn = c(5:8))
lista
```

```
## $tr
## [1] 1 2 3 4 5 6
##
## $mn
## [1] 5 6 7 8
```

```
lapply(lista, median)
```

```
## $tr
## [1] 3.5
##
## $mn
## [1] 6.5
```

```
vec = sample(1:40,5)
vec
```

```
## [1]  2 31  6  1 13
```

```
lapply(vec, FUN = function(x) 0.5*x)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 15.5
##
## [[3]]
## [1] 3
```

```
##
## [[4]]
## [1] 0.5
##
## [[5]]
## [1] 6.5
```

sapply

This function can by apply by a vector or list and will return a vector or a list. This function is very similar to lapply but it try to simplify it. This function is a wrapper of the lapply it can return not only a list it can return a vector, matrix or an array instead of a list. Let's do the same exercise to see the difference between both functions.

```r
lista = list(tr = c(1:6),mn = c(5:8))
lista
```

```
## $tr
## [1] 1 2 3 4 5 6
##
## $mn
## [1] 5 6 7 8
```

```r
sapply(lista, median)
```

```
##  tr  mn
## 3.5 6.5
```

```r
vec = sample(1:40,5)
vec
```

```
## [1] 12 39 36 23 26
```

```r
sapply(vec, FUN = function(x) 0.5*x)
```

```
## [1]  6.0 19.5 18.0 11.5 13.0
```

This function give you the same type that you give it in the input.

mapply

This is the multivariate version of sapply. Applying a function to multiple arguments.

```r
mapply(sum, 1:5, 6:10)
```

```
## [1]  7  9 11 13 15
```

vapply

This is a more strict version of sapply in where you need to specified the type of return that you would like.

```r
my_list <- list(a = 1:5, b = 6:10)
vapply(my_list, sum,numeric(1))
```

```
##  a  b
## 15 40
```

Part 2

MLE for gamma distribution

Differentiating with Respect to θ

Given a gamma distribution with parameters k (shape) and θ (scale), the probability density function is:

$$f(x \mid k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-x/\theta}$$

Log-Likelihood Function

The likelihood function for a sample $(x_1, x_2, \ldots, x_n)$ is given by:

$$L(x_1, \ldots, x_n \mid k, \theta) = \prod_{i=1}^{n} f(x_i \mid k, \theta) = \left(\frac{1}{\Gamma(k)\theta^k}\right)^n \prod_{i=1}^{n} x_i^{k-1} e^{-x_i/\theta}$$

Taking the logarithm of the likelihood function, we get the log-likelihood:

$$\log L(k, \theta) = -n \log \Gamma(k) - nk \log \theta + (k-1) \sum_{i=1}^{n} \log x_i - \frac{1}{\theta} \sum_{i=1}^{n} x_i$$

To find the maximum likelihood estimate of θ, we take the partial derivative of the log-likelihood with respect to θ and set it to zero:

$$\frac{\partial \log L(k, \theta)}{\partial \theta} = -\frac{nk}{\theta} + \frac{\sum_{i=1}^{n} x_i}{\theta^2}$$

Setting this equal to zero and solving for θ:

$$-\frac{nk}{\theta} + \frac{\sum_{i=1}^{n} x_i}{\theta^2} = 0$$

This simplifies to:

$$\theta = \frac{\sum_{i=1}^{n} x_i}{nk}$$

Final form for $\hat{\theta}$

Thus, the maximum likelihood estimate of the scale parameter θ is:

$$\hat{\theta} = \frac{\bar{x}}{k}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

Differentiating with Respect to k

The log-likelihood function for a sample $x_1, x_2, \ldots, x_n$ from a Gamma distribution with shape parameter k and scale parameter θ is:

$$\log L(k, \theta) = -n \log \Gamma(k) - nk \log \theta + (k-1)\sum_{i=1}^{n} \log x_i - \frac{1}{\theta}\sum_{i=1}^{n} x_i.$$

1. **First Term:** $-n \log \Gamma(k)$

   $$\frac{\partial}{\partial k}[-n \log \Gamma(k)] = -n\frac{\Gamma'(k)}{\Gamma(k)},$$

   where $\frac{\Gamma'(k)}{\Gamma(k)}$ is the digamma function $\psi(k)$.
2. **Second Term:** $-nk \log \theta$

   $$\frac{\partial}{\partial k}[-nk \log \theta] = -n \log \theta.$$
3. **Third Term:** $(k-1)\sum_{i=1}^{n} \log x_i$

   $$\frac{\partial}{\partial k}\left[(k-1)\sum_{i=1}^{n} \log x_i\right] = \sum_{i=1}^{n} \log x_i.$$
4. **Fourth Term:** $-\frac{1}{\theta}\sum_{i=1}^{n} x_i$
   This term does not depend on k, so its derivative with respect to k is zero.

Combining the Derivatives

Putting these together, the derivative of the log-likelihood with respect to k is:

$$\frac{\partial \log L(k, \theta)}{\partial k} = -n\psi(k) - n \log \theta + \sum_{i=1}^{n} \log x_i.$$

As we can see, we are not able to solve for k analytically as it does not have a closed-form solution. Hence, we need to use numerical methods to solve for k.

```r
# As we need to use the Digamma function, we need to install the required library
if (!requireNamespace("stats", quietly = TRUE)) {
  install.packages("stats")
}
```

Most optimisation algorithms are designed and implemented in a way to compute the minimisation the function rather than maximise it. But here we would like to estimate the maximum likelihood by maximising the log-likelihood function. So we need to convert our function into the negative log-likelihood.

```r
gamma_negativeLL = function(params, data) {
  k = params[1]
  theta = params[2]
  n = length(data)
  negativeLL = n*(log(gamma(k)) + (k*log(theta))) - (k - 1) * sum(log(data)) + (sum(data) / theta)
  return (negativeLL)
}
```

Given that:

$$X \sim \text{Gamma}(k, \theta)$$

- The mean of the gamma distribution is:

$$\mu = k\theta$$

- The variance of the gamma distribution is:

$$\sigma^2 = k\theta^2$$

From these, we can derive the following estimators using methods of moments:

- The sample mean:

$$\bar{X} = k\theta$$

Therefore,

$$\theta = \frac{S^2}{\bar{X}}$$

- The sample variance:

$$S^2 = k\theta^2$$

Thus, the shape parameter k can be estimated as:

$$k = \frac{\bar{X}^2}{S^2}$$

```r
mleGamma = function(data) {
  xBar = mean(data)
  xVar = var(data)
  initK = (xBar^2)/xVar
  initTheta = xVar/xBar

  # since we want to maximise a multivariate function (function with more than one parameter),
  # we use the function optim, which is used for multidimensional optimisation
  mleResult = optim(
    par = c(initK, initTheta), # Initial guesses for k and theta
    fn = gamma_negativeLL, # Objective function
    data = data,
    method = "L-BFGS-B", # optimisation method
    lower =  c(1e-6, 1e-6), # lower bounds to avoid invalid params
    upper = c(Inf, Inf) # upper bound
  )
  return (list(shape = mleResult$par[1], scale = mleResult$par[2]))
}
```

side note We specifically use the "L-BFGS-B" methods for optimisation as this method is useful for solving problems involving bound constraints on the parameters. The gamma distribution parameters θ and k must be strictly positive. Hence by using this methd we can set a lower bound to ensure they remain withtin a valid range.

Now we set a seed to be able to reproduce the same result and then we use the "*rgamma*" function to generate data with gamma distribution.

```r
set.seed(43)
sampleData = rgamma(1000, shape = 2, scale = 3)
```

Now we run our MLE estimator to see how close they are to our true parameters.

```r
results = mleGamma(sampleData)
cat("Estimated shape (k):", results$shape, "\n")
```

```
## Estimated shape (k): 2.067722
```

```r
cat("Estimated scale (theta):", results$scale, "\n")
```

```
## Estimated scale (theta): 2.916641
```