2018 Fall

Computational Statistics
HW#5

182STG18 이하경

Ⅰ. Description

EM 알고리즘은 Data 구조 내에서 관측되지 않은 파라미터의 값을 추정(Expectation)하여 완전한 데이터 구조를 가정하고, 이를 이용해 Complete Log-Likelihood 를 최대화(Maximization)하는 MLE 를 추정하는 데에 효과적으로 이용할 수 있는 반복적인 방법이다. HW#4 에서 Gaussian Mixture Model, missing value 를 포함한 Bivariate Normal Model 또는 Regression 등 다양한 경우의 Missing Case 에 대해 EM 알고리즘을 이용해 MLE 를 추정해보았다. 본 과제에서는 이어서 HW#4 의 Problem 4 와 유사한 문제로 복잡한 셀 구조를 가진 Multinomial Model 에서 Unobserved Frequency 를 추정하고 각각의 Cell Probability 의 MLE 를 추정하여 본다.

간단한 반복 과정을 통해 효과적으로 추정치를 계산할 수 있지만 EM 알고리즘은 추정치의 분산을 구하기 어렵다는 단점이 있다. 또한 Q function $Q(\theta, \theta^{(k)})$의 계산이 매우 힘들거나, M-step 에서 $\hat{\theta}$을 구할 때 Closed Form 을 이용한 계산이 불가능한 경우도 존재한다. 이러한 경우에 지난 step 의 값보다 향상되면 해당 값을 채택하는 GEM(Generalized EM)이나, 시뮬레이션을 이용해 계산하는 MCEM(Monte-Carlo EM)등의 방법을 이용하기도 한다.

위의 EM 알고리즘을 포함하여 통계학에서는 대부분의 경우에 기댓값(Expected Value)을 계산하고, 이를 위해서는 항상 확률분포를 이용한 적분 과정이 필요하다. (ex. $X_i s \sim$ (iid) f(x) 일 때 $\mathrm{E}[h(X)] = \int h(x)f(x)dx$) 그러나 실제로 적분 값을 직접적으로 구하기 어려운 경우가 많다. 따라서 수치 적분법(Numerical Integration Approximation), 또는 MC Simulation 으로 값을 근사적으로 계산하는 방법을 이용한다. 아래의 세 가지 수치적분 근사법은 구간 [a, b]에 대한 적분 $\int_a^b f(x)dx$을 계산하기 위해 전체 interval 을 n 개의 subinterval 들로 나누어 각 구간에서의 적분 값을 각각의 Rule 로 간단히 근사시켜 계산한 후, 이를 통합하여 전체 적분 값을 구한다.

■ Rieman Rule: $\widehat{R_{(n)}} = h \cdot \sum_{i=0}^{n-1} f(a+ih), \quad h = \frac{b-a}{n}$

■ Trapezoidal Rule: $\widehat{T_{(n)}} = \frac{h}{2} \cdot f(a) + h \cdot \sum_{i=1}^{n-1} f(a+ih) + \frac{h}{2} \cdot f(b), \quad h = \frac{b-a}{n}$

■Simpson's Rule: $\widehat{S_{(n)}} = \frac{h}{3} \cdot \sum_{i=1}^{\frac{n}{2}} [f(a+(2i-2)\cdot h) + f(a+(2i-1)\cdot h) + f(a+2i\cdot h)], \quad h = \frac{b-a}{n} \quad * n: even$

본 과제에서는 위의 세가지 근사법의 알고리즘을 R 에서 직접 구현해보고, 계산 결과를 비교해보려고 한다.

Ⅱ. Implementation

[Problem 1] E-M Algorithm: Multinomial with Complex Cell Structure: Peppered Moth's Genotypes

Moth's Color

(Phenotype)	frequency	prob.
C	$n_C = 85$	$\mathrm{p}^2 + 2\mathrm{pq} + 2\mathrm{pr}$
I	$n_I = 196$	$\mathrm{q}^2 + 2\mathrm{qr}$
T	$n_T = 341$	r^2
Total	622	1

→

Moth's Color

(Genotype)	frequency	prob.
CC	(unobserved)	p^2
CI	(unobserved)	$2\mathrm{pq}$
CT	(unobserved)	$2pr$
II	(unobserved)	q^2
IT	(unobserved)	$2qr$
TT	341	r^2
Total	622	

Goal 나방의 색에 대한 Phenotype Frequency Data 를 이용해 (Unknown) Genotype frequency $\boldsymbol{n_{CC}}$, $\boldsymbol{n_{CI}}$, $\boldsymbol{n_{CT}}$, $\boldsymbol{n_{II}}$, $\boldsymbol{n_{IT}}$ (, $n_{TT} = 341\ (known)$)을 각각 추정하고, Complete Log-Likelihood 를 가정하여 유전자 C(dominant to I, T), I(dominant to T), T 에서 $P_C = p, P_I = q, P_T = r$의 MLE $\hat{\theta} = (\hat{p}, \hat{q}, \hat{r})$을 추정하려고 한다.

① unobserved frequency 의 초기값을 각각 $n_{CC} = n_{CI} = n_{CT} = \frac{n_C}{3}$, $n_{II} = n_{IT} = \frac{n_I}{2}$ 으로 지정하여 $\mathrm{p} = \frac{2n_{CC}+n_{CI}+n_{CT}}{2n}$, $\mathrm{q} = \frac{2n_{II}+n_{IT}+n_{CI}}{2n}$, $r = 1 - p - q$ 의 초기값을 계산한다.
② (E-step) $\mathrm{E}[n_{CC}|n_C, n_I, n_T, p]$, $\mathrm{E}[n_{CI}|\sim]$, $\mathrm{E}[n_{CT}|\sim]$, $\mathrm{E}[n_{II}|n_C, n_I, n_T, q]$, $\mathrm{E}[n_{IT}|\sim]$을 각각 계산한다.
③ (M-step) $\hat{p}$, $\hat{q}$, $\hat{r}$의 값을 다시 update 한다.
④ $\mathrm{P} = (\hat{p}, \hat{q}, \hat{r})$ 의 값이 변하지 않을 때까지 반복한다. (Relative Convergence Criterion 사용)

Result

$\hat{p}$	$\hat{q}$	$\hat{r}$	$l(\hat{\theta})$	niter	comp.time
0.07084	0.18874	0.74043	-600.481	14	0.012

반복 수 14 회 만에 안정적으로 계산된 MLE 의 값은 각각 P(C)=0.07, P(I)=0.19, P(T)=0.74 이다. 열성 유전자인 T 유전자의 출현 확률이 제일 높은 것을 알 수 있다.

[Problem 2] Numerical Integration: 3 Approximations

Goal 함수 $f(x)$의 구간 [a, b]에 대한 적분 $\int_a^b f(x)\,dx$의 값을 구하려고 할 때 근사적으로 적분 값을 구하는 Rieman Rule, Trapezoidal Rule, Simpson Rule 3 가지 방법을 이용해 임의의 함수에 대해 값을 구해보고 실제 적분 값(손계산이 불가능한 경우 R 의 integrate 함수 이용)과 결과를 비교해본다.

Result

function 1. Simple Polynomial $\mathrm{f(x)} = -\mathrm{x}^3 + 1,\ [a, b] = [-1, 1]$

method	value	niter	comp.time
Rieman	2	27	28.83
Trapezoidal	2	2	0.00640
Simpson	2	2	0.00080
Integrate	2		0.00003

실제 적분 값은 $\int_{-1}^{1}(-x^3+1)\,dx = [-\frac{\mathrm{x}^4}{4} + \mathrm{x}]_{-1}^{1} = 2$으로, 간단한 다항함수에 대해 테스트 해보았을 때 세 가지 방법 모두 $\mathrm{Error} < 10^{-8}$에서 안정적이고 정확한 적분 근사를 하는 것을 확인하였다. Rieman Rule 을 이용한 계산의 수렴 속도가 다른 두 방법에 비해 눈에 띄게 느렸다.

function 2. $\mathrm{p.d.f.\ of}\ Normal(1,\ 3^2)$

$$\mathrm{f(x)} = \frac{1}{\sqrt{2\pi \cdot 3^2}} \exp\left(-\frac{(x-1)^2}{2 \cdot 3^2}\right), \qquad [a, b] = [1, 5]$$

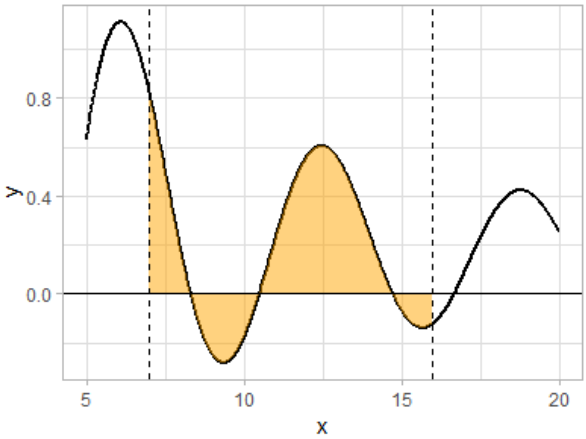
method	value	niter	comp.time
Rieman	0.40878878	26	26.097
Trapezoidal	0.40878878	13	0.00724
Simpson	0.40878878	7	0.00275
Integrate	0.40878878		0.00003

두번째 함수로는 정규분포 $N(1,3^2)$ 의 probability density function 을 이용하여 $P(1 \le X \le 5) = \int_1^5 f(x)dx$ 값을 계산해보았다. 세 가지 방법 모두 $Error < 10^{-8}$에서 누적확률 $P(1 \le X \le 5) = P\left(0 \le Z \le \frac{4}{3}\right) = \Phi\left(\frac{4}{3}\right) - 0.5 = 0.4088$ 을 잘 계산하였으며, 반복횟수와 계산 속도는 Simpson < Trapezoidal < Rieman 순으로 Simson 근사법이 가장 빨랐다.

function 3. Some Complicated Functions

1) $g(x) = \frac{\log(1+x) + 5\cos x}{\sqrt{1+x^2}}, \quad [a,b] = [7,16]$

method	value	niter	comp.time
Rieman	1.56501896	28*	155.58
Trapezoidal	1.56501895	15	0.01930
Simpson	1.56501895	9	0.00362
Integrate	1.56501895		0.00003



좀 더 복잡한 적분에도 적용해보기 위해 임의의 함수를 만들고 [7, 16] 범위에 대하여 각각 적분을 계산하였다. 이 경우에도 역시 Simson Rule 의 수렴 속도가 제일 빨랐으며, Rieman Rule 의 경우는 threshold $\epsilon = 10^{-8}$으로 설정 시에 반복 수 28 에서 n 의 크기가 매우 커져 R 에서 계산을 완료하지 못하는 경우가 발생하였다. 따라서 반복 수(niter)가 28 일 때 최종 계산된 적분 값을 확인한 결과 $Error = 1.016e-08$ 으로 threshold ϵ에 거의 근접하였으며 실제 적분 값이나 다른 방법에서의 결과와 거의 동일하게 계산되었다.

2) $\text{Calculate } E[g(X)] = \int_{-1}^{\infty} \frac{\log(1+x) + 5\cos x}{\sqrt{1+x^2}} \cdot f(x)dx, \quad X \sim \text{Normal}(1,3^2)$

method	value	niter	converged error
Rieman	0.8850265	25	5.17e-07
Trapezoidal	0.8850266	23	5.40e-07
Simpson	0.8850266	22	7.20e-07
Integrate	0.8850269		5.50e-05

앞에서 2 번째 함수인 Normal density 와 위의 g(X)를 이용해 기댓값 E[g(X)]을 계산해보았다.

여기서 X 의 가능한 범위는 $(-1,\infty)$으로, 구현한 함수를 이용하여 계산 시에는 누적확률이 1 에 수렴하는 20 을 오른쪽 끝 값으로 설정하고 (-1, 20) 범위에 대해 계산하였다. 함수의 식이 매우 복잡하여 $\epsilon = 10^{-8}$ 으로 설정하였을 때는 반복수가 증가하면서 계산을 완료하지 못하여 threshold 를 $\epsilon = 10^{-6}$으로 수정하고 계산을 완료하였다. R 의 Integrate 함수를 이용해 결과 값을 확인하였을 때 계산된 적분 값들이 소수 자리 7 번째 이상에서 거의 동일하여 올바른 계산을 한 것으로 판단하였다.

Ⅲ. Discussion

먼저 첫 번째 문제에서는 지난 과제와 동일하게 Unknown parameter 들의 기댓값을 계산하고 Complete Multinomial Log-Likelihood 를 최대화하는 MLE 를 추정하는 알고리즘을 활용해볼 수 있었다. MLE 계산이 불가능한 Cell 의 구조를 E-M 알고리즘을 이용해 계산 가능하도록 변환하고, unobservable parameter 와 Complete log-likelihood 에 대해 정의하는 과정에 대해 더 자세히 이해할 수 있었다.

다음으로 두 번째 주제인 수치적분 근사법을 직접 구현해보면서 방법 별로 계산 속도의 차이를 확인할 수 있었다. 간단한 함수의 계산에는 결과 값에 차이가 없었으나 복잡한 함수일수록 Rieman 의 경우 각 구간(subinterval)에서 subnode 가 없고(m=0), 각각 한 점에서의 함숫값 즉 상수로 근사하기 때문에 적분의 수렴 속도가 느리고 다른 방법에 비해 결과 값의 오차 또한 조금 더 발생하였다. 각각의 subinterval 에서의 적분을 m 차 다항함수(polynomial)로 근사하여 통합하는 Trapezoidal Rule(m=1)과 Simpson Rule(m=2)은 이에 비해 계산이 빨랐고, 서로 비슷한 속도를 보이거나 Simpson 이 좀 더 빨랐다. 따라서 고차의 다항함수로 근사할 수록 실제 적분 값에 가까워진다는 것을 알 수 있었다. 결과적으로 세 가지 방법 모두 적분 값을 직접 알기 어려운 경우에 효과적으로 정확한 값을 계산하므로, 통계학에서 적분이 필요한 다양한 경우에 이들을 이용하여 빠른 계산을 할 수 있음을 알게 되었다.

[Appendix] R Code

```
# Problem 1
  myem5 <- function(E=10^-10)
  {
    nc = 85 ; ni = 196 ; nt = 341 ; n = nc + ni + nt
  # initial
    ncc = nc/3 ; nci = nc/3 ; nct = nc/3 ; nii = ni/2 ; nit = ni/2
    p = (2*ncc+nci+nct)/(2*n) ; q = (2*nii+nci+nit)/(2*n) ; r = 1 - p - q

    niter = 0 ; maxiter = 1000
    error = 1

    while (error >= E & niter <= maxiter)
    { ncc_0 <- ncc ; nci_0 <-nci ; nct_0 <-nct ; nii_0 <- nii ; nit_0 <- nit
      p_0 <- p ; q_0 <- q ; r_0 <- r
    # E-step
      ncc <- nc*p^2 / (p^2 + 2*p*q + 2*p*r)
      nci <- nc*2*p*q / (p^2 + 2*p*q + 2*p*r)
      nct <- nc*2*p*r / (p^2 + 2*p*q + 2*p*r)
      nii <- ni*q^2 / (q^2 + 2*q*r)
      nit <- ni*2*q*r / (q^2 + 2*q*r)
    # M-step
      p <- (2*ncc + nci + nct) / (2*n) ; q <- (2*nii + nci + nit) / (2*n) ; r <- 1 - p - q

      error <- sqrt( sum( (c(p, q, r) - c(p_0, q_0, r_0))^2 ) ) / sqrt( sum(c(p_0, q_0, r_0)^2) )
      niter <- niter + 1

      print(paste("error = ", error, "niter = ", niter, sep = ""))
    }

    loglik <- nc*log(p^2 + 2*p*q + 2*p*r) + ni*log(q^2 + 2*q*r) + 2*nt*log(r)
    output <- list(MLE = c(p, q, r), loglik = loglik, freq = round(c(ncc, nci, nct, nii, nit)))
    return(output)
  }

  myem5() ; system.time( for (i in 1:100) myem5() )


# Problem 2. Numerical Integration

# (2-1) Rieman
  myrieman <- function(f, a, b, start=1, E=10^-8)
  {
    niter = 0 ; maxiter = 100 ; error = 1

  # initial value
    R = (b-a)*f(a)
    k <- start

    while (error >= E & niter <= maxiter)
    { R_0 <- R

      n <- 2^k ; h <- (b-a)/n
      i <- 0:(n-1)
      R <- h * sum( f(a+i*h) )
```

```
    error <- abs(R - R_0) / abs(R_0 + 10^-3)
    niter <- niter + 1
    k <- k + 1
    # if (k < 25) { k <- k + 1 } else { error <- 0 ; print("k is too large!") }

    print(paste("error = ", error, " niter = ", niter, sep = ""))
  }

  return(R)
}


# (2-2) Trapezoidal
mytrapezoid <- function(f, a, b, start=1, E=10^-8)
{
  niter = 0 ; maxiter = 100 ; error = 1

  # initial value
  t = (b-a)*f(a)
  k <- start

  while (error >= E & niter <= maxiter)
  { t_0 <- t

    n <- 2^k ; h <- (b-a)/n
    i <- 1:(n-1)
    t <- h/2*f(a) + h*sum( f(a+i*h) ) + h/2*f(b)

    error <- abs(t - t_0) / abs(t_0 + 10^-3)
    niter <- niter + 1
    k <- k + 1

    print(paste("error = ", error, " niter = ", niter, sep = ""))
  }

  return(t)
}


# (2-3) Simpson
mysimpson <- function(f, a, b, start=1, E=10^-8)
{
  niter = 0 ; maxiter = 100 ; error = 1

  # initial value
  S = (b-a)*f(a)
  k <- start

  while (error >= E & niter <= maxiter)
  { S_0 <- S

    n <- 2^k ; h <- (b-a)/n
    i <- 1:(n/2)
    S <- (h/3) * sum( f(a+(2*i-2)*h) + 4*f(a+(2*i-1)*h) + f(a+(2*i)*h) )

    error <- abs(S - S_0) / abs(S_0 + 10^-3)
    niter <- niter + 1
```

```
    k <- k + 1

    print(paste("error = ", error, " niter = ", niter, sep = ""))
  }

  return(S)
}


# test
myrieman(function(x) -x^3+1, -1, 1) ; system.time ( for (i in 1:3) myrieman(function(x) -x^3+1, -1, 1) )
mytrapezoid(function(x) -x^3+1, -1, 1) ; system.time ( for (i in 1:1000) mytrapezoid(function(x) -x^3+1, -1, 1) )
mysimpson(function(x) -x^3+1, -1, 1) ; system.time ( for (i in 1:1000) mysimpson(function(x) -x^3+1, -1, 1) )
integrate(function(x) -x^3+1, lower = -1, upper = 1) ; system.time( for (i in 1:1000) integrate(function(x) -x^3+1, lower = -1, upper = 1) )

# normal density
myfunc1 <- function(x) dnorm(x, 1, 3)

grim <- data.frame(x = seq(-8, 10, length = 1000)) %>% mutate(y = myfunc1(x))
ggplot(grim) + theme_light() + geom_line(aes(x, y), size = 1) +
  geom_area(data = filter(grim, (x>=1 & x<=5)), aes(x, y), fill = "lightblue", alpha = 0.5) +
  geom_vline(aes(xintercept = 1), linetype = "dashed") + geom_vline(aes(xintercept = 5), linetype = "dashed")

v11 <- myrieman(myfunc1, 1, 5) ; system.time ( for (i in 1:3) myrieman(myfunc1, 1, 5) )
v12 <- mytrapezoid(myfunc1, 1, 5) ; system.time ( for (i in 1:1000) mytrapezoid(myfunc1, 1, 5) )
v13 <- mysimpson(myfunc1, 1, 5) ; system.time ( for (i in 1:1000) mysimpson(myfunc1, 1, 5) )
v10 <- integrate(myfunc1, lower = 1, upper = 5)$value ; system.time ( for (i in 1:1000) integrate(myfunc1, 1, 5) )


# 험악한 함수
myfunc3 <- function(x) ( log(1+x) + 5*cos(x) ) / sqrt( 1 + x^2 )
x <- seq(5, 20, length = 1000)
plot(x, myfunc3(x), type = "l", lwd = 2)

grim <- data.frame(x = seq(5, 20, length = 1000)) %>% mutate(y = myfunc3(x))
ggplot(grim) + theme_light() + geom_line(aes(x, y), size = 1) +
  geom_area(data = filter(grim, (x>=7 & x<=16)), aes(x, y), fill = "orange", alpha = 0.5) +
  geom_hline(aes(yintercept = 0)) +
  geom_vline(aes(xintercept = 7), linetype = "dashed") + geom_vline(aes(xintercept = 16), linetype = "dashed")

v31 <- myrieman(myfunc3, 7, 16, E=0.00000002) ; system.time( myrieman(myfunc3, 7, 16, E=0.00000002) )
v32 <- mytrapezoid(myfunc3, 7, 16) ; system.time( for (i in 1:1000) mytrapezoid(myfunc3, 7, 16) )
v33 <- mysimpson(myfunc3, 7, 16) ; system.time( for (i in 1:1000) mysimpson(myfunc3, 7, 16) )
v30 <- integrate(myfunc3, 7, 16)$value ; system.time( for (i in 1:1000) integrate(myfunc3, 7, 16) )

v41 <- myrieman(function(x) myfunc3(x)*myfunc1(x), -1+10^-8, 20, E=10^-6)
v42 <- mytrapezoid(function(x) myfunc3(x)*myfunc1(x), -1+10^-8, 20, E=10^-6)
v43 <- mysimpson(function(x) myfunc3(x)*myfunc1(x), -1+10^-8, 20, E=10^-6)
v40 <- integrate(function(x) myfunc3(x)*myfunc1(x), -1, Inf)$value
```