Tutorial no. 1: Distribuciones de probabilidad en R

Borja Calvo

Resumen

R es un lenguaje de programación que se ha desarrollado fundamentalmente con la mente puesta en la probabilidad y la estadística. Por este motivo, la instalación base de R incluye un gran número de funciones para manejar distribuciones de probabilidad. En este tutorial veremos como usarlas para realizar las tareas típicas.

1 Funciones para manipular distribuciones

R permite trabajar, de base, con casi 20 distribuciones diferentes. Para cada distribución tendremos cuatro funciones que nos permitirán muestrear la distribución así como computar la densidad, la función de probabilidad acumulada y los cuantiles. La sintaxis de estas funciones es siempre la misma, comienza con una letra (`r` para las funciones de muestreo, `d` para la función de densidad, `p` para la función de probabilidad acumulada y `q` para los cuantiles); a continuación de esta letra viene un nombre representativo de la distribución. La lista de todas puede verse en la Tabla 1.

A modo de ejemplo, la función que nos permite muestrear una distribución hipergeométrica es `rhyper` y la que nos da la densidad de la distribución de Cauchy es `dcauchy`. En los siguiente apartados veremos ejemplos del uso de estas funciones. En cualquier caso, para obtener ayuda sobre el uso de las funciones basta con consultar la ayuda de R (ejecutando, por ejemplo, `?rnorm`).

2 Funciones de masa y densidad de probabilidad

Las funciones de densidad y de masa de probabilidad se pueden computar usando las funciones cuyo nombre empieza con la letra d. Veamos un ejemplo en el que representaremos la función de densidad de tres distribuciones Gausianas. El resultado de ejecutar el siguiente código puede verse en la Figura 1

```
> #Generamos los puntos en los que evaluar (entre -5 y 5)
> x<-seq(-5,5,0.01)
> #Obtenemos los valores de las 3 densidades
> densidad1<-dnorm(x,mean=0, sd=0.5)
> densidad2<-dnorm(x,mean=0,sd=1)
> densidad3<-dnorm(x,mean=2,sd=1)
> #Las representamos gráficamente
> plot(x,densidad1,type="l",main="Ejemplo de tres densidades Gausianas",
+      xlab="X", ylab="Densidad")
> lines(x,densidad2,col=2)
> lines(x,densidad3,col=3)
```

De igual forma, podemos representar las funciones de probabilidad acumulada; en este caso usaremos las funciones que comienzan con la letra p. Veamos un ejemplo similar, esta vez con la distribución log-normal.

Año: Master ICSI/KISA
Profesor(a): Borja Calvo, Aritz Pérez
Teléfono: 943 01 50 13
E-mail: [borja.calvo,aritz.perez]@ehu.es

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)

Distribución	Nombre	Distribución	Nombre
Beta	beta	Lognormal	lnorm
Binomial	binom	Negative Binomial	nbinom
Cauchy	cauchy	Normal	norm
Chi cuadrado	chisq	Poisson	pois
Exponencial	exp	T de Student	t
F	f	Uniforme	unif
Gamma	gamma	Tukey	tukey
Geométrica	geom	Weibull	weib
Hipergeométrica	hyper	Wilcoxon	wilcox
Logística	logis		

Tabla 1: Nombres de las distribuciones básicas disponibles en R

```
> #Obtenemos los valores de las 3 densidades
> #Generamos los puntos en los que evaluar (entre -5 y 5)
> x<-seq(0,100,0.1)
> cdf1<-plnorm(x,meanlog=0, sdlog=1)
> cdf2<-plnorm(x,meanlog=0,sdlog=2)
> cdf3<-plnorm(x,meanlog=3,sdlog=1)
> #Las representamos gráficamente
> library(ggplot2)
> df<-rbind(data.frame(X=x,Y=cdf1,Distribucion="CDF #1"),
+           data.frame(X=x,Y=cdf2,Distribucion="CDF #2"),
+           data.frame(X=x,Y=cdf3,Distribucion="CDF #3"))
>
> ggplot(df,aes(x=X, y=Y, col=Distribucion)) + geom_line(size=1.1) +
+   labs(x="X", y="CDF")
```

En este caso las funciones las hemos representado utilizando el paquete **ggplot2**; para entender como funciona este paquete puedes consultar el tutorial adicional sobre el uso de este paquete.

Dada una función de densidad, podemos querer computar la probabilidad de que la variable tome un valor dentro de un intervalo. Para ello debemos computar la integral en dicho intervalo, lo cual podemos hacer con la función `integrate`. Veamos como determinar la probabilidad de que una variable que sigue una distribución chi cuadrado con 5 grados de libertad tome un valor comprendido entre 4 y 6,

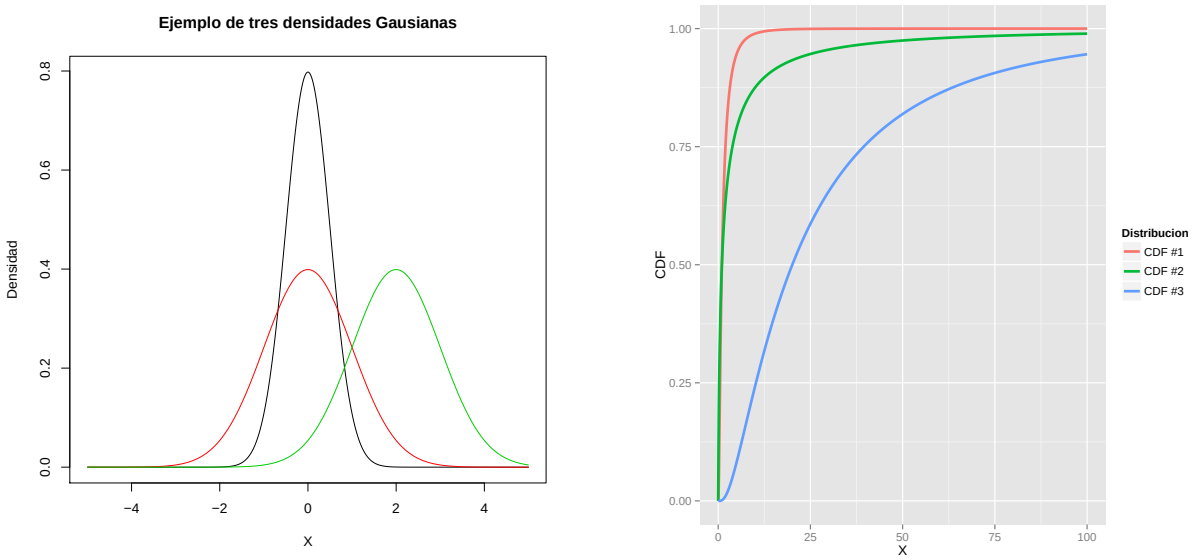
```
> #Para poder hacer la integral hay que definir la función a integrar
> f<-function (x){
+   dchisq(x,df=5)
+ }
>
> #Hecho esto la integral será ...
> integrate(f = f,lower = 4, upper = 6)

## 0.243197 with absolute error < 2.7e-15
```

Una forma alternativa de hacer esto es viendo la diferencia entre la función de densidad acumulada en 6 y en 4 (recordemos que esta función determina la probabilidad de tener un valor menor o igual al dado)

3 Muestreo de la distribución y cuantiles

la forma de muestrear una distribución en R es utilizando las funciones que comienzan con la letra r. Por ejemplo, podemos muestrear una distribución beta de parámetros $\alpha = 1, \beta = 10$ usando el siguiente código.



(a) Funciones de densidad

(b) Funciones de probabilidad acumulada

Figura 1: Ejemplo de funciones de densidad y funciones de probabilidad acumulada

```
> muestra<-rbeta(250,1,10)
```

Una forma de ver si una distribución se ajusta bien a una muestra es utilizando el llamado plot cuantil-cuantil. La idea es fijar una serie de cuantiles (los percentiles de 10 a 90, por ejemplo) y determinar su valor tanto en la distribución a evaluar como en la muestra. Si al representar uno frente a otro tenemos una diagonal recta, en ese caso el ajuste es bueno; en caso contrario la distribución no se ajusta bien a los datos. Veamos esto con en los datos generados en el código anterior.

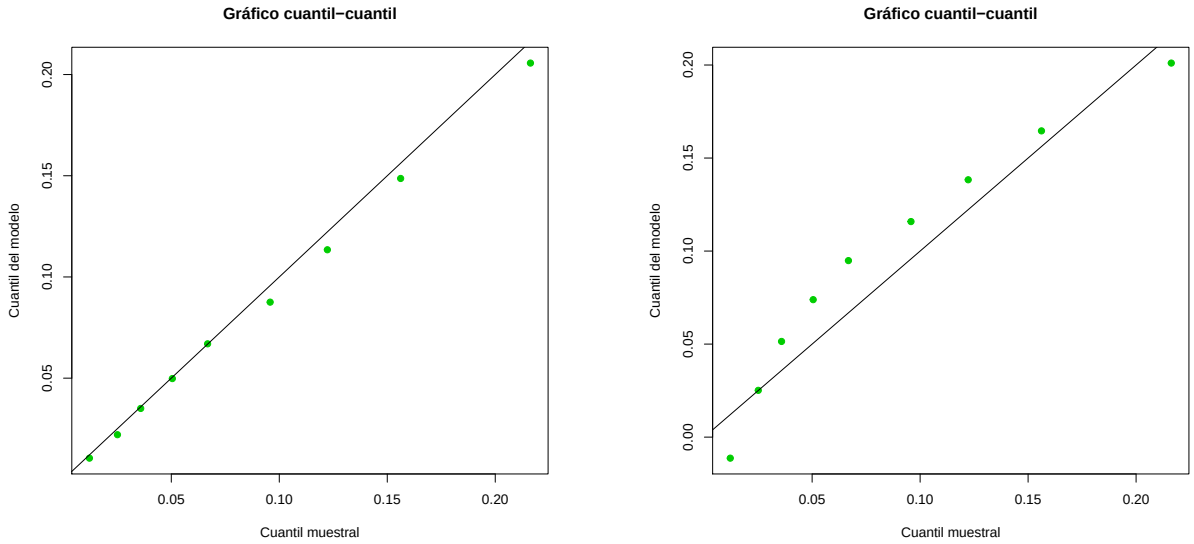
Los datos se han generado con una distribución Beta. Si queremos obtener sus cuantiles teóricos tenemos que usar la función `qbeta`.

```
> #Definimos los cuantiles a usar
> p<-seq(0.1,0.9,0.1)
> #Obtenemos los valores teóricos
> cuantiles_teoricos<-qbeta(p,1,10)
```

Para determinar los cuantiles muestrales basta con ordenar los valores de la muestra y buscar el valor para el cual el ratio entre el número de muestras a un lado y al otro coincide con el cuantil. Es decir:

```
> #Creamos una función para obtener los cuantiles muestrales
> qmuestral<-function(p, muestra){
+   muestra_ordenada<-sort(muestra)
+   posiciones<-p*length(muestra)
+   muestra_ordenada[posiciones]
+ }
> cuantiles_muestrales<-qmuestral(p,muestra)
```

Ahora podemos representar el gráfico:



(a) Comparación con la distribución generadora

(b) Comparación con una distribución Gausiana

Figura 2: Ejemlo de gráficos cuantil-cuantil

```
> plot(cuantiles_muestrales, cuantiles_teoricos,
+      main="Gráfico cuantil-cuantil", xlab="Cuantil muestral", ylab="Cuantil del modelo",
+      pch=19, col=3)
> lines(c(0,1),c(0,1))
```

A fin de comparar el resultado con una distribución que no se ajusta bien a los datos veremos que pasa si utilizamos una distribución normal con media y varianza obtenidas de los datos. Ambos resultados se muestran en la Figura 2

```
> cuantiles_normal<-qnorm(p,mean(muestra),sd(muestra))
> plot(cuantiles_muestrales, cuantiles_normal,
+      main="Gráfico cuantil-cuantil", xlab="Cuantil muestral", ylab="Cuantil del modelo",
+      pch=19, col=3)
> lines(c(0,1),c(0,1))
```

4 Verosimilitud

Un concepto muy importante a la hora de estimar los parámetros de una distribución a partir de una muestra es el de la verosimilitud, que no es más que la probabilidad de los datos bajo la distribución de probabilidad considerad; si asumimos que las muestras son independientes (e idénticamente distribuidas), entonces la probabilidad de la muestra completa es el producto de las probabilidades de cada muestra.

Para ilustrar este ejemplo generaremos una muestra de tamaño 1000 a partir de una distribución de Poison de parámetro $\lambda = 5$. A continuación determinaremos la verosimilitud de los datos bajo diferentes valores del parámetro.

```
> #Creamos la muestra
> sample<-rpois(1000,5)
>
```

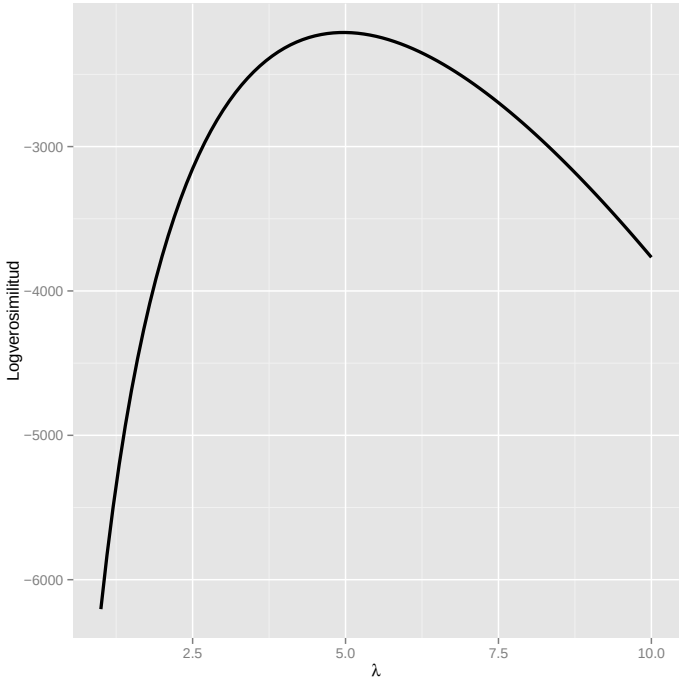


Figura 3: Evolución de la logverosimilitud con el parámetro Lambda

```
> #Determinamos la probabilidad asociada a cada valor muestreado
> probabilidades<-dpois(sample,5)
>
> #La probabilidad la computamos como el producto
> prod(probabilidades)

## [1] 0
```

En el código anterior tenemos que la verosimilitud es 0. Esto no es cierto, es debido a un problema con la representación de números muy pequeños (debemos tener en cuenta que la probabilidad asociada a cada valor es un valor menor que 1 y que tenemos que multiplicar la de los 1000 valores!). Este problema es recurrente al trabajar con probabilidades obtenidas como productos de un número elevado de probabilidades individuales. Afortunadamente, la solución es sencilla. En lugar de calcular la verosimilitud, podemos calcular la log-verosimilitud, es decir, su logaritmo. En ese caso el producto se convierte en una suma de logaritmos, evitando así el problema computacional.

```
> #El logaritmo del producto de probabilidades es la suma de los logaritmos
> sum(log(probabilidades))

## [1] -2209.838
```

Como vemos, la logverosimilitud es un valor negativo muy grande. Si tratamos de dehacer el logaritmo el resultado será 0.

Veamos ahora como evoluciona la verosimilitud en función del valor del parámetro que utilicemos.

```
> f<-function(lambda){
+   sum(log(dpois(sample,lambda)))
```

```
+ }
> lambdas<-seq(1,10,0.1)
> logveros<-sapply(lambdas, FUN = f)
> df<-data.frame(Lambda=lambdas, Logverosimilitud=logveros)
> ggplot(df,aes(x=Lambda, y=Logverosimilitud)) + geom_line(size=1.1) +
+   labs(x=expression(lambda))
```

Como podemos ver en la Figura 3, el valor máximo de la logverosimilitud se alcanza con el valor del parámetro que generó los datos. Esto es, los datos son más "creibles" si fijamos el parámetro λ a su valor real.