Simulation, échantillonnage, bootstrap, validation croisée avec R

1 Introduction

Cette scéance propose une introduction aux outils de simulation avec R et à leur application en statistique : sondage, bootstrap, validation croisée pour l'estimation d'une erreur.

2 Simulation de lois

La base de toute simulation est un générateur de nombres pseudo-aléatoires supposés distribués selon une loi discrète uniforme sur l'intervalle $[0, 1]$. Connaissant une valeur initiale (seed ou semence), une formule de récurrence à base de congruence (prime modulus multiplicative generator) est plus généralement utilisée à cette fin. Divers procédés permettent ensuite de générer une variable pseudo-aléatoire suivant une autre loi (binomiale, multinomiale, poisson, exponentielle, gamma, Cauchy, gaussienne...). C'est immédiat lorsque l'expression analytique de l'inverse de la fonction de répartition est connue. La méthode dite de Box et Muller est utilisée pour simuler une loi normale.

L'étude de différentes distributions, de leur estimation est abordée par des simulations et des outils non-paramétriques.

2.1 Quelques lois discrètes

Bernoulli

Commençons par un tirage de pile ou face :

```
>n=100
```

Tirage avec remise

```
>x=sample(c(-1,1), n, replace=T)
>plot(x, type='h', main="Variable de Bernoulli")
```

Le générateur est pseudo-aléatoire et dépend de l'initialisation. Faire plusieurs tirages et représentations dans le cas précédent puis plusieurs tirages en initialisant prélablement le générateur à chaque fois. Remarque : utiliser la touche du clavier (flèche vers le haut) pour rappeler une commande précédente.

```
>set.seed(123)
>x=sample(c(-1,1), n, replace=T)
>plot(x, type='h', main="Variable de Bernoulli")
```

Cela permet de retrouver la même séquence de nombres pseudo-aléatoires et donc de reproduire, si nécessaire, des simulations identiques.

Avec des probabilités différentes :

```
>x=sample(c(-1,1), n, replace=T, prob=c(.2,.8))
>plot(x, type='h',main="Bernoulli, cas général")
```

Binomiale et estimation de densité

On peut "programmer" une distribution binomiale $B(n, p)$ comme la somme de n variables de Bernoulli mais il existe aussi une fonction ad'hoc :

Génération de 1000 valeurs suivant une $B(10, 0.5)$

```
>x=rbinom(1000,10,0.5)
```

Tracé de l'histogramme

```
>hist(x,xlim=c(min(x),max(x)), probability=T, col='blue',nclass=max(x)-min(x))
```

Ajouter une estimation nom-paramétrique de la densité

```
>lines(density(x), col='red', lwd=2)
```

Un noyau gaussien est utilisé par défaut pour cette estimation avec une optimisation du paramètre de lissage (largeur de fenêtre).

Loi de Poisson

Les mêmes instructions sont reprises avec la génération d'une variable de Poisson de paramètre 1.

```
>x=rpois(1000,1)
>hist(x,xlim=c(min(x),max(x)), probability=T, col='blue',nclass=max(x)-min(x))
```

Changer le paramètre de la loi :

```
>x=rpois(1000,20)
>hist(x,xlim=c(min(x),max(x)), probability=T, col='blue',nclass=max(x)-m
```

D'autres lois sont disponibles : géométrique, hypergéométrique, négative binomiale, multinomiale.

2.2 Lois continues

Loi uniforme, de Cauchy

Quelques tirages suivant une loi uniforme :

```
>round(runif(10), digits=3)
```

Estimation des densités dans le cas d'une loi uniforme

```
>x=runif(1000)
>hist(x, probability=T, col='blue')
>lines(density(x), col='red', lwd=2)
```

Puis de celle d'un loi de Cauchy qui n'admet pas de moment.

```
>x=rcauchy(1000)
>hist(x, probability=T, col='blue')
>lines(density(x), col='red', lwd=2)
```

Autres lois

De nombreuses autres fonctions de simulation sont prévues pour des lois, gaussienne, exponentielle, du Chi2, de Student, de Fisher, lognormale, de Weibul, Gamma, Beta, de Dérichlet. En essayer quelques unes :

Loi exponentielle

```
>x=rexp(1000)
>hist(x, probability=T, col='blue')
>lines(density(x), col='red', lwd=2)
```

Loi normale

```
>x=rnorm(1000)
>hist(x, probability=T, col='blue')
>lines(density(x), col='red', lwd=2)
```

Lois théoriques

Les expressions théoriques des densités sont aussi connues de R. Voici les tracés des fonctions pour des lois du Chi2 avec différentes valeurs du paramètre :

```
>curve(dchisq(x,1), xlim=c(0,10), ylim=c(0,.6), col='red', lwd=3)
>curve(dchisq(x,2), add=T, col='green', lwd=3)
>curve(dchisq(x,3), add=T, col='blue', lwd=3)
>curve(dchisq(x,5), add=T, col='orange', lwd=3)
>abline(h=0,lty=3)
>abline(v=0,lty=3)
```

Loi des grands nombres

La moyenne empirique de n variables suivant une loi uniforme est calculée. Comparer la précision obtenue à travers plusieurs exécutions de

```
>mean(runif(10))
>mean(runif(1000))
```

Même chose avec la loi de Cauchy "pathologique" :

```
>mean(rcauchy(10))
>mean(rcauchy(1000))
>mean(rcauchy(100000))
```

2.3 Limite centrale et loi gaussienne

Estimer par un histogramme la densité d'une variable aléatoire, somme de 12 variables uniformes sur $[0, 1]$.

```
>x=rep(0,1000)
>for (i in 1 :1000) x[i]=sum(runif(12))
>hist(x, col='blue', probability=T) Ajouter une estimation non paramétrique de la densité
>lines(density(x), col='red', lwd=2) Comparer avec la densité théorique d'une loi $\mathcal{N}(6, 1)$.
```

`>curve(dnorm(x,mean=6,sd=1), add=T, col='green', lwd=2)` Commentaire sur la vitesse de convergence en loi de la moyenne empirique de n variables aléatoires i.i.d. d'espérance m et de variance σ^2 vers une gaussienne de moyenne m et de variance σ^2/n.

3 Échantillonnage

3.1 Aléatoire simple

L'étude de grandes bases de données nécessite d'extraire un sous-ensemble par un échantillonnage alétoire simple dans la base afin de réaliser des analyses ou mettre au point des programmes sur un échantillon représentatif de taille raisonnable. Ce sondage est très simple à réaliser en R ; il est utilisé pour diviser l'échantillon en deux parts aléatoires : échantillon d'apprentissage et échantillon test. Le générateur est initialisée avec une semence spécifique à chaque étudiant pour construire des échantillons différents.

Utiliser les trois derniers chiffre de son code INSEE comme initialisation du générateur

```
>set.seed(xx)
>npop=1000
```

tirage de 200 indices sans remise

```
>testi<-sample(1 :npop,200)
```

Liste des indices restant qui n'ont pas été tirés

```
>appri<-setdiff(1 :npop,testi)
```

Ces listes d'indices seront utilisées dans les prochains TPs pour extrire des échantillons d'apprentissage et de test afin de comparer entre elles les performances des différentes méthodes dans leur capacité de prédiction.

3.2 Avec remise ou *bootstrap*

Principe

Le *bootstrap* est une technique de rééchantillonnage permettant de simuler la distribution d'un estimateur quelconque pour en apprécier le biais, la variance donc le risque quadratique ou encore pour en estimer un intervalle de confiance même si la loi théorique est inconnue. Le principe consiste à estimer itérativement ce même estimateur sur des réalisations différentes de l'échantillon obtenus aléatoirement par n tirages avec remise dans l'échantillon initial.

Un échantillon bootstrap est facilement obtenu avec l'option `replace=TRUE` de la commande `sample`. Comparer :

```
>sample(1 :20,20)
>sample(1 :20, 20, replace=TRUE)
```

Régression simple

L'exemple élémentaire ci-dessous génère une enveloppe pouvant s'interpréter comme une région de confiance de la droite de régression du revenu en fonction du nombre d'appartements.

Lire les données si nécessaire :

```
>suit=read.table("suitincom.dat")
>names(suit)=c("revenu","nbappt")
```

Tracé du nuage de points

```
>plot(suit$nbappt,suit$revenu)
```

Itération du tirage de l'échantillon bootstrap et des estimations des régressions

```
>for (i in 1 :100) {
>suit.b=suit[sample(47,47,replace=TRUE),]
>reg=lm(revenu~nbappt,data=suit.b)
>abline(reg)}
```

La même chose est obtenue pour l'autre modèle :

```
>lsuit=data.frame(log(suit$nbappt),log(suit$revenu))
>names(lsuit)=c("Lrevenu","Lnbappt")
>plot(lsuit$Lnbappt,lsuit$Lrevenu)
>for (i in 1 :100) {
>suit.b=lsuit[sample(47,47,replace=TRUE),]
>reg=lm(Lrevenu~Lnbappt,data=suit.b)
>abline(reg)}
```

Ou encore pour des régressions non-paramétriques :

```
>plot(lsuit$Lnbappt,lsuit$Lrevenu)
>for (i in 1 :100) {
```

```
>suit.b=lsuit[sample(47,47,replace=TRUE),]
>lsuit.spl=smooth.spline(suit.b$Lnbappt,suit.b$Lrevenu,df=4)
>lines(lsuit.spl, col = "blue")}
```

Régression linéaire multiple

Cet outil permet d'obtenir des informations sur les distributions des paramètres.

Lecture des données

```
>ukcomp1=read.table('ukcomp1.datr',
+header=TRUE)
```

Le bootstrap construit "à la main" :

Initialisation de la matrice qui contiendra les différentes estimations

```
>stock=data.frame(matrix(0,100,5))
>names(stock)=c("CST","CFTDT","LOGSALE","NFATAST","CURRAT")
```

Itération des tirages des échantillons et des estimations

```
>for (i in 1 :100) {
>Ib=sample(40,40,replace=TRUE)
>stock[i,]=coef(lm(RETCAP∼WCFTDT+LOGSALE+NFATAST+CURRAT,data=ukcomp1[Ib,]))}
```

Dispersion des estimations des paramètres

```
>boxplot(stock,horizontal=TRUE)
```

Même chose pour le modèle optimal au sens du R^2 ajusté.

```
>stock=data.frame(matrix(0,100,9))
>names(stock)=c("CST","WCFTDT","LOGSALE","LOGASST","NFATAST",
+"FATTOT","INVTAST","QUIKRAT","CURRAT")
>for (i in 1 :100) {
>Ib=sample(40,40,replace=TRUE)
>stock[i,]=coef(lm(RETCAP∼WCFTDT+LOGSALE+LOGASST+NFATAST+FATTOT+
+INVTAST+QUIKRAT+CURRAT,data=ukcomp1[Ib,]))}
>boxplot(stock,horizontal=TRUE)
```

Comparer les dispersions des paramètres.

3.2.1 Librairie spécifique

Des fonctions sont prévues dans le package `boot` pour simplifier le travail et qui s'appliquent à toute statistique. Ainsi, pour estimer les biais et écarts-types "bootstrap" des paramètres en régression :

Chargement de la librairie

```
>library(boot)
```

Définition de la fonction qui calcule la statistique d'intérêt à bootstraper

Celle-ci comporte deux paramètres : les données et les indices d'un échantillon

```
>uk1.fun=function(d,i) coef(lm(RETCAP∼WCFTDT+LOGSALE+NFATAST+CURRAT,data=d[i,]))
```

Kancement du bootstrap

```
>uk1.b=boot(ukcomp1,uk1.fun,R=999)
```

Paramètres, biais et écart-type bootstrap desparamètres

```
>uk1.b
>boxplot(data.frame(uk1.b$t),horizontal=TRUE)
```

Même chose pour l'autre modèle.

```
>uk2.fun=function(d,i) coef(lm(RETCAP∼WCFTDT+LOGSALE+LOGASST+
+NFATAST+FATTOT+INVTAST+QUIKRAT+CURRAT,data=d[i,]))
>uk2.b=boot(ukcomp1,uk2.fun,R=999)
>uk2.b
>boxplot(data.frame(uk2.b$t),horizontal=TRUE)
```

4 Validation croisée

4.1 Principe

La validation croisée permet d'estimer, avec un biais réduit, un risque quadratique ou un taux de mal classés selon l'objectif de la méthode utilisée. Le principe consiste à estimer le modèle sur une partie de l'échantillon puis de calculer l'erreur commise sur une auitre partie de l'échantillon qui n'a pas participer à l'estimation des paramètres de ce modèle. Ce calcul

est itéré sur plusieurs échantillons de validation afin d'améliorer par moyennage la précision. Différentes façons de constituer des échantillons de validation peuvent être considérées. La validation croisée originale (PRESS de Allen) ou *leave-one-out* considère n échantillons d'apprentissage de taille $n-1$ obtenus en éliminant tour à tour chaque observation qui constitue l'échantillon test. La deuxième découpe aléatoirement l'échantillon en k (par défaut, $k=10$) parties approximativement de même taille. Après k itérations, chaque groupe a joué le rôle d'échantillon de validation. Enfin la dernière propose d'itérer B fois le tirage aléatoire simple d'un échantillon de validation avec un taux fixé. Chaque algorithme estime respectivement n, k, B fois le modèle sur la partie apprentissage de l'échantillon, prédit la part validation, puis estime l'erreur en moyennant les erreurs de prédictions.

Le choix de l'algorithme de validation croisée dépend principalement de la taille de l'échantillon : le premier si n est petit, le deuxième en situation intermédiaire en choisissant k, le dernier si n est grand.

Dans R, comme pour le bootstrap, des fonctions génériques ont été définies pour être appliquées à toute méthode de modélisation : fonction `crossval` du package `bootstrap` ou `cv.glm` du package `boot` adaptée au modèle linéaire général (gaussien, binomial...).

La validation est très largement utilisée comme estimation de l'erreur de prédiction à minimiser pour optimiser un choix de modèle parmi une famille définie par un paramètre de complèxité.

4.2 Application au modèle linéaire

L'utilisation de la validation croisée ne s'impose pas pour cette technique de modélisation car d'autres critères basés également sur une estimation de l'erreur de prédiction sont diponibles à moindre coût. Elle est mise en œuvre à titre illustratif sur les données précédentes.

Estimation du modèle

```
>uk1.glm = glm(RETCAP∼WCFTDT+LOGSALE+NFATAST+CURRAT,family=gaussian,data=ukcomp1)
```

Estimation de l'erreur leave-one-out

```
>cv.err = cv.glm(ukcomp1,uk1.glm)
```

Estimation sur 5 échantillons de taille 8

```
>cv.err.5 = cv.glm(ukcomp1,uk1.glm, K=5)
>cv.err$delta[1]
>cv.err.5$delta[1]
```

Même chose sur l'autre modèle.

```
>uk1.glm = glm(RETCAP∼WCFTDT+LOGSALE+LOGASST+NFATAST+FATTOT+
+INVTAST+QUIKRAT+CURRAT,family=gaussian,data=ukcomp1)
>cv.err = cv.glm(ukcomp1,uk1.glm)
>cv.err.5 = cv.glm(ukcomp1,uk1.glm, K=5)
>cv.err$delta[1]
>cv.err.5$delta[1]
```

Vérifier que les choix du TP précédent sont confirmés.

Pour mémoire, rappelons que dans le cas de la régression multiple, l'erreur *leave-one-out* est obtenue directement sans estimation itérative du modèle :

```
>muhat=uk1.glm$fitted
>uk1.diag= glm.diag(uk1.glm)
>mean((uk1.glm$y-muhat)^2/(1-uk1.diag$h)^2)
```