Scénario: Fraudes électriques

Résumé

—-

1 Problématique

2 Données

On considère un ensemble de relevés de consommation électrique mensuelles pour des individus à Medellin en Colombie. Pour chaque individu, on relève par mois sa consommation électrique, la consommation du nœud auquel il est rattaché, la consommation de l'éclairage public sur ce nœud et les pertes associées. On considère en fin deux variables booléennes : la présence d'irrégularité et la présence de fraude au mois.

3 Objectifs

Ces données sont très incomplètes ; il s'agira dans un premier temps de créer un jeu de données viable, quitte à faire des hypothèses fortes, pour le bon déroulement de l'étude. Les données manquantes devront ensuite être imputées. On se propose de tester différentes méthodes de complétion : LOCF, complétion par la moyenne, la médianne, KNN, régression locale, algorithme NIPALS, par décomposition en valeurs singulières, utilisation de forêts aléatoires, data augmentation ou enfin par un algorithme EMB qui combine de l'estimation du maximum de vraisemblance et du bootstrap. Le premier objectif étant de comparer ces méthodes d'imputation afin de déterminer la plus précise pour créer un jeu de données complété qui servira de base à une modélisation de la fraude.

Une fois les données complétées, on passera à la deuxième partie du problème : l'explication de la fraude. Dans un premier temps,

Penser à conserver dans des fichiers les commandes successives utilisées ainsi que les principaux résultats tableaux et graphes.

Toutes les opérations sont réalisées dans R avec l'appui de bibliothèques complémentaires éventuellement à télécharger : `zoo`, `VIM`, `missForest`, `Amelia`, `norm`, `imputation`, `ade4`

4 Exploration des données

4.1 Prise en charge des données

Les données initiales sont échantillonnées par mois sur deux années (soit 24 mois) pour 998 individus. On considère les variables suivantes :

- `Numero.cuenta` le numéro d'identifiant de l'individu (`Numero`$=1,\ldots,998$)
- `Mes` et `Ano` la date (resp. mois et année) du relevé
- `Consumo.Medidor` la consommation électrique de l'individu
- `Consumo.Macromedidor` la consommation totale sur le transformateur
- `Consumo.Alumbrado.Publico` la consommation de l'éclairage public sur le transformateur
- `Sumatoria.consumo.usuarios` la somme des consommations des usagers sur le transformateur divisée par 1000
- `Cantidad.de.usuarios` la quantité d'usagers sur le transformateur
- `Perdidas` les pertes d'électricité
- `Fraude` un booléen valant 1 s'il y a eu fraude et 0 sinon.

A noter que

$$\texttt{conso.macro} = 1000*\texttt{somme.conso}+\texttt{conso.eclairage}+\texttt{pertes}$$

```
# Chargement des données
conso=read.csv("consom_initial.csv",header=TRUE,sep="")
attach(conso)
summary(conso)
```

4.2 Création d'un jeu de données pour la complétion

4.2.1 Réduction du nombre de données manquantes

On s'intéresse dans un premier temps à la consommation individuelle. On remarquera que `conso` contient des doublons : un individu est parfois connecté à différents transformateurs le même mois. On crée donc une table de la consommation individuelle mensuelle sur deux ans, en additionnant les doublons éventuels

```
# Conso individuelle sur 24 mois
table.2ans=as.data.frame(with(conso,
        tapply(Consumo.Medidor,
        list(Numero.cuenta, Mes,Ano), sum)))
```

Quel est le pourcentage de données manquantes dans cette table ? Existe-t-il un individu pour lequel les 24 relevés sont disponibles ? Afin de se ramener à un nombre acceptable de données manquantes, nous ferons l'hypothèse que les deux années sont identiques. Si un relevé est disponible pour les deux années, on gardera la moyenne des consommations. Sinon, la consommation du mois est celle disponible, quelle que soit l'année.

```
# Conso individuelle sur 12 mois
table.1an=matrix(nrow=999,ncol=12)
for (i in 1:12) {
  table.1an[,i]=rowMeans(cbind(table.2ans[,i],
                table.2ans[,i+12]),na.rm=TRUE)
}
```

A présent, quel est le pourcentage de données manquantes ? Existe-t-il un individu pour lequel les 12 relevés sont disponibles ? Pour la suite de cette étude, on ne considèrera que les individus pour lesquels il y a moins de 3 données manquantes :

```
conso.1an={}
for (i in 1:max(Numero.cuenta)){
  if (length(which(is.na(table.1an[i,])))<4){
    conso.1an=rbind(conso.1an,c(i,table.1an[i,]))
  }
}
colnames(conso.1an)=c("Numero.cuenta",1:12)
conso.1an[which(is.nan(conso.1an),arr.ind=TRUE)]=NA
```

Combien reste-t-il d'individus ? Est-ce acceptable ? Quel est à présent le pourcentage de données manquantes ? Que peut-on dire de leur répartition ?

4.2.2 Fraude

On estimera la fraude non plus au mois mais par individu :

```
#Fraude par individu
fraude=matrix(data=NA,ncol=1,nrow=dim(conso.1an)[1])
ind=0
for (i in conso.1an[,1]){
  ind=ind+1
  if ("SI" %in% conso$Fraude[conso$Numero.cuenta==i]){
    fraude[ind]="SI"
  }else {
    fraude[ind]="NO"
  }
}
fraude=as.factor(fraude)
```

4.2.3 Transformation des données

Les données sont des entiers positifs. Certaines méthodes d'imputation font des hypothèses de normalité qu'il est important d'approcher. On procède donc à une transformation en log :

```
Numero.cuenta=conso.1an[,1]
lconso.1an=log(conso.1an[,-1]+1)
```

Les variables suivent-elles à présent une loi $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$? En estimer les paramètres.

4.3 Extraction des échantillons

Le programme ci-dessous réalise l'extraction du sous-ensemble des données d'apprentissage et de test. Attention, chaque étudiant ou binôme tire un échantillon différent ; il est donc "normal" de ne pas obtenir les mêmes modèles pour chaque méthode.

Utiliser un nombre au hasard, en remplacement de "42" ci-dessous, comme initialisation du générateur de nombres aléatoires.

```
set.seed(42) # initialisation du générateur
# Extraction des échantillons
test.ratio=0.1   # part de l'échantillon test
# indices des valeurs observées
IND=which(!is.na(lconso.1an),arr.ind=TRUE)
# taille de l'échantillon test
ntest=ceiling(dim(lconso.1an)[1]*test.ratio)
# indices de l'échantillon test
ind.test=IND[sample(1:dim(IND)[1],ntest),]
```

Construction des échantillons. La création de l'échantillon d'apprentissage consiste ici à remplacer les valeurs de l'échantillon test par des valeurs manquantes :

```
# construction de l'échantillon de test
conso.test=lconso.1an[ind.test]
# construction de l'échantillon d'apprentissage
conso.train=lconso.1an
conso.train[ind.test]=NA
```

5 Complétion

Cette section a pour objectif d'appliquer les procédures d'imputation. Les erreurs de complétion sont estimées sur l'échantillon test.

LOCF

```
library(zoo) # chargement de la bibliothèque
conso.locf=na.locf(conso.train,na.rm=FALSE)
conso.locf=na.locf(conso.locf,na.rm=FALSE,
        fromLast=TRUE) # dans l'autre sens
err.locf=abs(conso.test-conso.locf[ind.test])
hist(err.locf)
boxplot(err.locf,ylim=c(0,3))
```

Pourquoi doit on appliquer deux fois la commande `na.locf` ?

Remplacement par la moyenne

```
library(imputation) # chargement de la bibliothèque
conso.moy=meanImpute(conso.train)$x
err.moy=abs(conso.test-conso.moy[ind.test])
boxplot(data.frame(err.locf,err.moy),ylim=c(0,3))
```

Remplacement par la médiane

```
med=apply(conso.train,1,median,na.rm=TRUE)
conso.med=conso.train
ind.na=which(is.na(conso.med),arr.ind=TRUE)
conso.med[ind.na]=med[ind.na[,1]]
err.med=abs(conso.test-conso.med[ind.test])
```

k plus proches voisins (kNN)

```
library(VIM) # chargement de la bibliothèque
conso.kNN=kNN(conso.train, k=5, imp_var=FALSE)
err.kNN=abs(conso.test-conso.kNN[ind.test])
boxplot(data.frame(err.locf,err.moy,err.med,
        err.kNN),ylim=c(0,3))
```

Le choix de $k = 5$ voisins est arbitraire. Est-ce le meilleur choix ? La fonction `cv.kNNImpute` de la librairie `{imputation}` peut être utile.

LOESS

```
library(imputation) # chargement de la bibliothèque
conso.LOESS=lmImpute(conso.train)$x
err.loess=abs(conso.test-conso.LOESS[ind.test])
boxplot(data.frame(err.locf,err.moy,err.med,
        err.kNN,err.loess),ylim=c(0,3))
```

Quelles remarques pouvez-vous faire sur cette méthode ?

SVD

```
library(imputation) # chargement de la bibliothèque
```

```
conso.SVD=SVDImpute(conso.train,3)$x
err.svd=abs(conso.test-conso.SVD[ind.test])
boxplot(data.frame(err.locf,err.moy,err.med,
        err.kNN,err.loess,err.svd),ylim=c(0,3))
```

missForest

```
library(missForest) # chargement de la bibliothèque
conso.missForest=missForest(conso.train,
                maxiter = 10, ntree = 200,
                variablewise = TRUE)$ximp
err.mf=abs(conso.test-conso.missForest[ind.test])
boxplot(data.frame(err.locf,err.moy,err.med,
   err.kNN,err.loess,err.svd,err.mf),ylim=c(0,3))
```

Data Augmentation

```
library(norm) # chargement de la bibliothèque
rngseed(42)
pre =prelim.norm(as.matrix(conso.train))
#estimation du maximum de vraisemblance
xhat =em.norm(pre)
conso.da =imp.norm(pre,xhat,conso.train)
err.da=abs(conso.test-conso.da[ind.test])
boxplot(data.frame(err.locf,err.moy,err.med,
        err.kNN,err.loess,err.svd,err.mf,
        err.da),ylim=c(0,3))
```

AmeliaII

```
library(amelia) # chargement de la bibliothèque
conso.amelia=amelia(conso.train,m=1)$imputations$imp1
err.amelia=abs(conso.test-conso.amelia[ind.test])
boxplot(data.frame(err.locf,err.moy,err.med,
        err.kNN,err.loess,err.svd,err.mf,
        err.da,err.amelia),ylim=c(0,3))
```

Quelle méthode choisiriez-vous pour imputer le jeu de données avant de passer à la suite de l'étude ? Soit `conso.imp` le jeu imputée par cette méthode.

6 Analyse des données complétées

6.1 Consommation moyenne

Représentation de la consommation de l'individu moyen

A partir des données initiales, on calcule la consommation moyenne :

```
client.moyen=colMeans(conso.1an[,-1], na.rm = TRUE)
plot(client.moyen,type='l',xlab='Mes',lwd=2)
title("Profil de consommation moyen")
```

Comparer avec les moyennes obtenues après complétion. Que peut-on en dire ?

Test d'égalité des moyennes mensuelles

La répartition de chaque mois est visible par un `boxplot(conso.imp)`. Que peut-on en déduire ? Y a-t-il un effet du facteur "mois" sur la consommation mensuelle ? Quel(s) test(s) pourrions nous utiliser pour vérifier cette assertion ? Quelles en seraient les hypothèses ? L'exemple de code ci-dessous permet un test d'égalité des moyennes. Détaillez le test, commentez le code et les résultats.

```
Consumo.Medidor.Client={}
for (i in 1:12){
  Consumo.Medidor.Client=c(Consumo.Medidor.Client,
                           conso.imp[,i])
}
Mes.Client={}
for (i in 1:12){
  Mes.Client=c(Mes.Client,rep(i,dim(conso.imp)[1]))
}
FMes=as.factor(Mes.Client)
aov=aov(Consumo.Medidor.Client~FMes)
summary(aov)
TukeyHSD(aov,"FMes",ordered=T)
```

Que fait la fonction `TukeyHSD` ? Pourquoi l'utilise-t-on ? Expliquer les résultats obtenus.

6.2 ACP

Le code ci-dessous permet d'effectuer une analyse en composantes principales. Commenter le code et les résultats.

```
acp=princomp(conso.imp,center=colMeans(conso.imp))
plot(acp)
biplot(acp)
```

6.3 ACP fonctionnelle

Qu'est ce qu'une ACP fonctionnelle ? Que fait le code ci-dessous ?

```
acp=princomp(conso.imp,cor=TRUE)
plot(acp)
plot.ts(acp$loadings[,1:6],main="Fonctions propres")
```

Commentez les résultats.

7 Classification

Le but de cet section est de faire émerger des profils de comportement types en observant différentes classes d'usagers.

7.1 Classification Ascendante Hiérarchique

Quelle distance est la plus pertinente pour calculer l'écart entre les observations ? Quelle est celle donnée en exemple dans le code ci-dessous ?

```
d=dist(conso.imp)
dN=dimnames(conso.imp)[[1]]
hc.d =hclust(d,method="ward")
# choix du nombre de classes:
plot(hc.d$height[252 :240],type="b")
```

Que fait la fonction `hclust` ? Quel nombre de classes préconisez-vous ? Soit `K` ce nombre. On procède à la CAH :

```
class = cutree(hc.d,k=K)
mds=cmdscale(d,k=2)
plot(mds, type="p",col=class,pch=19,cex=1)
```

Qu'est ce que la représentation dans le MDS ? On affiche les traces dans chaque classe, avec la moyenne par classe :

```
par(mfcol=c(ceiling(K/2),ceiling(K/ceiling(K/2))))
for (k in 1:K){
  ts.plot(t(conso.imp[class==k,]),ylim=c(0,10),
        col=8)
  lines(apply(conso.imp[class==k,],2,mean),
        lwd=2,col=k)
  title(paste("Classe ",k,sep=""))
}
```

7.2 Classification par Réallocation Dynamique

L'algorithme k-means permet d'affiner la classification.

```
#initialisation
km.conso.init=matrix(nrow=K,ncol=12)
for (i in 1 :K){
  km.conso.init[i,]=apply(conso.imp[class==i,],
                          2,mean)
}
#classification
conso.km=kmeans(conso.imp,centers=km.conso.init)
mds=cmdscale(d,k=K)
plot(mds, type="p",col=conso.km$cluster,
        pch=19,cex=1, xlab="Dim1", ylab="Dim2",
        main="Classification par Kmeans")
#Homogénéité des classes
par(mfcol=c(ceiling(K/2),ceiling(K/ceiling(K/2))))
for (k in 1:K){
  ts.plot(t(conso.imp[conso.km$cluster==k,]),
        ylim=c(0,10),col=8)
  lines(apply(conso.imp[conso.km$cluster==k,],2,
        mean),lwd=2,col=k)
  title(paste("Classe ",k,sep=""))
}
```

Que peut-on dire de l'homogénéité des classes ? Dans quelle(s) classe(s) se trouvent les fraudeurs ? Afficher leurs consommations. Que peut-on en déduire ?

Pour trouver les classes auxquelles appartiennent les fraudeurs :

```
fraudeurs=conso.imp[which(fraude=="SI"),]
class=conso.km$cluster[fraude=="SI"]
```

Au lieu de modéliser directement la consommation des usagers on considèrera la table des fréquences, donnée par :

```
conso.freq=cbind(conso.imp/rowSums(conso.imp),
        rowSums(conso.imp))
```

Refaire les analyses précédentes sur `conso.freq`. Que peut-on en déduire ?

8 Modélisation de la fraude

8.1 Modèle LOGIT

```
myLogit<-function(conso.glm,fraude){
  IND=1:dim(conso.glm)[1]
  test.ratio=0.1
  ntest=ceiling(dim(conso.glm)[1]*test.ratio)

  ind.test=sample(IND,ntest)
  fraude.test=as.factor(fraude[ind.test])

  while (length(which(fraude.test=="SI"))<4){
    ind.test=sample(IND,ntest)
    fraude.test=as.factor(fraude[ind.test])
  }
  ind.train=setdiff(IND,ind.test)
  fraude.train=as.factor(fraude[ind.train])

  attach(conso.glm)
  mylogit<-glm(fraude.train ~ .,
        data=conso.glm[ind.train,],na.action=na.omit,
        family="binomial")
  print(summary(mylogit))
  #courbe ROC
  library(ROCR)
  fraude.logit=predict(mylogit,
        newdata=conso.glm[ind.test,],type="response")
  fitpred = prediction(fraude.logit,fraude.test)
  fitperf = performance(fitpred,"tpr","fpr")
  plot(fitperf,col=1,lwd=2)
  abline(a=0,b=1,lwd=2,lty=2,col="gray")
}
```

Appliquer cette fonction à `conso.imp` et `conso.freq`

8.2 Modèles d'arbres

```
conso=read.csv("consom_preprocessed.csv",
        header=TRUE,sep=",")
attach(conso)
Fraude=as.factor(Fraude)
Ha.tenido.Irregularidad=as.factor(Ha.tenido.Irregularida
Mes=as.factor(Mes)
Numero.cuenta=as.factor(Numero.cuenta)
```

8.2.1 CART

```
library(rpart)
fit=rpart(Fraude~., data=conso,method="class",
        na.action=na.omit)
printcp(fit)
plotcp(fit)
print(fit)
summary(fit)
plot(fit)
text(fit)
post(fit, file = "tree_CART.ps")
```

8.2.2 Ctree

```
library(party)
fit=ctree(Fraude~., data=conso)
plot(fit)
```

8.2.3 Random Forest

```
library(randomForest)
fit <- randomForest(Fraude~.,data=conso,
        na.action=na.omit,importance=TRUE,
        proximity=TRUE)
print(fit)
importance(fit)
```