Scénario: transcrits et identification de souches de légionnelle

Résumé

L'objectif est de savoir identifier une souche parmi 21 de légionelle à partir de l'expression mesurée de 2 fois 47 sondes plus ou moins spécifiques des génotypes des différentes souches. Les données expressions et identification de la souche sont connues pour 2239 analyses. Après ue étude exploratoire (analyse en composantes principales, discriminante, pls) qualifiant la bonne qualité générale des données, il s'agit d'estimer et comparer les meilleures méthodes de classification susceptibles de répondre au problème : réseau de neurones, k plus proches voisins, agrégation de modèle...

1 Introduction

1.1 Problématique

La légionelle est une bactérie naturellement présente dans les eaux stagnantes et les boues. Certaines souches sont très pathogènes et leur présence dans les systèmes de climatisation puis leur dissémination sous forme d'aérosol peut provoquer de graves pathologies (légionellose) : pneumopathie sévère, manifestations neurologiques, troubles digestifs, insuffisance rénale. C'est un problème important de santé publique. La start-up toulousaine Dendris met au point une puce dédiée munie de 2x47 sondes (les sondes sont dupliquées pour plus de fiabilité) visant à proposer aux laboratoires d'analyse biologique une chaîne d'analyse pour la détection et l'identification rapide de souches de légionelles. Chaque sonde est spécifique de la portion d'un gène pouvant caractériser quelques unes des souches, celles pathogènes. Le résultat de l'analyse biologique est un vecteur de 2x47 composantes mesurant l'intensité du signal pour chacune des 2x47 sondes présentes sur la puce. La complexité des processus biologiques est source de bruits de mesure dans la chaîne d'analyse. Aussi, l'identification d'une souche nécessite une phase d'apprentissage des modèles de prévision sur la base d'un ensemble de données fournies par la société Dendris. Pour ces analyses, les souches sont connues a priori. Nous disposons d'un premier échantillon de 2239 analyses de biopuces concernant les 21 souches pathogènes identifiées.

Après une approche exploratoire, la démarche proposée utilise systématiquement les ressources de la librairie `caret` afin de

1. tester et sélectionner les méthodes de modélisation pertinentes pour identifier au mieux les souches,
2. optimiser le choix en comparant les distributions des erreurs de prévision évaluées sur un ensemble d'échantillons aléatoires de test.

1.2 Données

Les données sont accessibles dans le fichier `legio_dat.csv` du répertoire data. L apremière ligne contient un identifiant suivie de 2239 observations d'échantillons des 21 différentes souches, une centaine de réplicats par souche. la première colonne contient l'identité de la souche suivie de deus fois 47 colonnes contenant les niveaux d'expression de chacune des sondes.

2 Approche exploratoire

2.1 Prise en charge des données

```
# lire les données
legiob=read.csv2("legio_dat.csv")
# isoler le nom de souches
Souche=legiob[,1]
legiob=legiob[,-1]
summary(legiob)
```

Chaque sonde étant répliquée, il est important de s'assurer de la bonne reproductibilié d'une même sonde. Si c'est bien le cas, il suffit alors de calculer les moyennes.

```
# tracer la heatmap
heatmap(cor(legiob),Rowv=NA,Colv=NA)
# calculer les moyennes.
legiom=matrix(0,nrow(legiob),47)
```

```
for (j in 1:47){
legiom[,j]=(legiob[,2*j-1]+legiob[,2*j])/2
}
summary(legiom)
```

2.2 Statistiques élémentaires

Distributions très dissymétriques, prendre le logarithme.

```
boxplot(legiom)
Llegio=log(8+legiom)
boxplot(Llegio)
```

2.3 Analyse en composantes principales

Une ACP classique permet de s'assurer de la bonne cohérence des données.

```
pca=prcomp(Llegio,scale=TRUE)
plot(pca)
biplot(pca)
# avec de la couleur
color=as.integer(Souche)
pt=as.integer(Souche)%%3+1
plot(pca$x[,1:2],type="p",col=color,pch=pt)
legend("topleft",legend=levels(Souche),
   text.col=c(1:21))
plot(pca$x[,3:4],type="p",col=color,pch=pt)
legend("topleft",legend=levels(Souche),
   text.col=c(1:21))
plot(pca$x[,5:6],type="p",col=color,pch=pt)
legend("topleft",legend=levels(Souche),
   text.col=c(1:21))
```

Les souches se distinguent-elles facilement les unes des autres ?

2.4 Analyse factorielle discriminante

L'analyse factorielle discriminante mettra mieux en évidence les possibles séparations des souches.

```
# chargement de la librairie
library(MASS)
# analyse discriminante linéaire
fda= lda(Llegio,Souche)
print(fda)
plot(fda,dimen=5)
# Il faut tracer des graphes en couleur
# Coordonnées des observations dans
# les axes discriminants
pred=predict(fda,data=Llegio)
# calcul des barycentres
m=matrix(rep(0,210),nrow=21,ncol=10)
for (i in 1:21){
 for (j in 1:10){
 m[i,j]=mean(pred$x[unclass(Souche)==i,j]) }}
# Graphes dans les axes discriminants (1,2)
plot( pred$x[,1],pred$x[,2],bg=color,pch=21)
abline(0,0,h=0); abline(0,0,v=0)
text(m[,1],m[,2],labels=levels(Souche),cex=2)
# Graphes dans les axes discriminants (3,4)
plot( pred$x[,3],pred$x[,4],bg=color,pch=21)
abline(0,0,h=0); abline(0,0,v=0)
text(m[,3],m[,4],labels=levels(Souche),cex=2)
```

Produire les mêmes graphiques sur les 10 premiers axes. Toutes les souches sont-elles bien distinctes ?

2.5 Régression PLS DA

La version "discriminant analysis" de la régression PLS propose une approche similaire.

```
library(mixOmics)
pls= plsda(Llegio,Souche,ncomp=10)
plotIndiv(pls,comp=1:2,col=color,ind.names=Souche)
# Pour associer sondes et souches :
x11()
plotVar(pls,comp=1:2,var.names=1:47,var.label=TRUE)
```

```
# possible aussi en 3d avec rotation de l'espace
plot3dIndiv(pls,col=color,ind.names=Souche)
# ou dans chaque plan
plotIndiv(pls,col=color,comp=c(3,4),
   ind.names=Souche)
plotIndiv(pls,col=color,comp=c(5,6),
   ind.names=Souche)
plotIndiv(pls,col=color,comp=c(7,8),
   ind.names=Souche)
plotIndiv(pls,col=color,comp=c(9,10),
   ind.names=Souche)
```

En analysant soigneusement les graphiques, il est donc possible de se faire une idée sur quelles souches se distinguent mieux des autres et quelles sondes sont le splus concernées.

2.6 Détection d'atypiques

Le modèle dégrégation "Random Forest" offre une option de détection d'observation atypiques ou ouliers.

```
library(randomForest)
rfFit2=randomForest(Llegio,Souche,proximity=TRUE)
plot(outlier(rfFit2),type="h",col=color)
legend("topleft",legend=levels(Souche),
   text.col=c(1:21))
```

Une dizaine d'échantillons de souches "jor" de distinguent particulièrmenet de leur groupe.

3 Classification supervisée

L'objectif n'est pas seulement exploratoire, il est prédictif : construire des modèles d'identification de la souche en fonction des réponses aux sondes. Différentes méthodes cont être comparées et testées car il n'est pas a priori sûr que des séparations linéaires soient les plus efficaces. C'est le taux d'erreur sur un échantillon test distinct de la partie apprentissage qui permet de comparer les méthodes.

3.1 Tests de différents modèles

L'étude se limite à une utilisation de la librairie `caret` (Kunh, 2008 [**?**]) pour quelques unes des méthodes de modélisation proposées.

Préparation des données

Extraction des échantillons d'apprentissage et de test.

```
# préparation des données
library(caret)
X=Llegio
summary(X)
Y=as.factor(Souche)
summary(Y)
# indices de l'échantillon d'apprentissage
set.seerd(11)
# échantillon équilibré selon les niveaux de Y
inTrain = createDataPartition(Y,
   p = 80/100, list = FALSE)
# Extraction des échantillons
trainDescr=X[inTrain,]
testDescr=X[-inTrain,]
trainY=Y[inTrain]
testY=Y[-inTrain]
```

Il est recommandé de centrer et réduire les variables dans plusieurs méthodes. C'est fait systématiquement et simplement en utilisant évidemment les mêmes transformations sur l'échantillon test que celles calculées sur l'échantillon d'apprentissage. D'autre part, l'erreur de prévision est estimée par la suite par validation croisée.

```
# Normalisation
xTrans=preProcess(trainDescr)
trainDescr=predict(xTrans,trainDescr)
testDescr=predict(xTrans,testDescr)
# Choix de la validation croisée
cvControl=trainControl(method="cv",number=10)
```

Estimation et optimisation des modèles

La librairie intègre beaucoup plus de méthodes mais celles sélectionnées ci-dessous semblent les plus pertinentes. Le choix est par ailleurs limité par le nombre de classes : 21. Les techniques à 2 classes comme la régression logistique ne sont pas utilisables. De même, les SVM multiplcasses sont complexes et peu performant. Une approche systématique proposant 21 modèles : 1 classe contre les autres sera mis en œuvre plus tard.

Exécuter successivement les blocs de commandes pour tracer séparamment chacun des graphes afin de contrôler le bon comportement de l'optimisation du paramètre de complexité de chaque modèle.

```
#1 analyse discriminante linéaire
set.seed(2)
ldaFit = train(trainDescr, trainY,
   method = "lda2", tuneLength = 10,
   trControl = cvControl)
ldaFit
#2 K plus proches voisins
set.seed(2)
knnFit = train(trainDescr, trainY,
   method = "knn", tuneLength = 10,
   trControl = cvControl)
knnFit
plot(knnFit)
#3 Arbre de décision
set.seed(2)
rpartFit = train(trainDescr, trainY,
   method = "rpart", tuneLength = 20,
   trControl = cvControl)
rpartFit
plot(rpartFit)
#4 Réseau de neurones
set.seed(2)
nnetFit = train(trainDescr, trainY,
   method = "nnet", tuneLength = 6,
   trControl = cvControl, trace=F)
nnetFit
plot(nnetFit)
#5 Random forest
set.seed(2)
rfFit = train(trainDescr, trainY,
   method = "rf", tuneLength = 10,
   trControl = cvControl)
rfFit
plot(rfFit)
# 6 Classifieur bayésien naïf
# Quel problème pose ce "classifieur" ?
set.seed(2)
nbFit = train(trainDescr, trainY,
    method = "nb", tuneLength = 10,
   trControl = cvControl)
nbFit
plot(nbFit)
```

Prévisions, graphes et erreurs

La librairie offre la possibilité de gérer directement une liste des modèles et donc une liste des résultats.

```
models=list(add=ldaFit,knn=knnFit,
  cart=rpartFit,nnet=nnetFit,rf=rfFit,Bayes=nbFit)
testPred=predict(models, newdata = testDescr)
# taux de mal classés
lapply(testPred,function(x)mean(x!=testY))
# matrice de confusion de random forest
lapply(testPred,function(x)table(x,testY))$rf
# Quelques résultas spécifiques
testPred.prob=predict(models, newdata = testDescr,
   type="prob")
plot(testPred.prob$rf[,"jor"],
   testPred.prob$rf[,"hac"])
testPred.prob$rf[,"pne"]
t(testPred.prob$rf[355,])
```

Commenter les résultats.

Importance des variables

Certaines méthodes (forêts aléatoires) sont très peu explicites (boîte noire) quant au rôle des variables sur la prévision des observations. Néanmoins, des indicateurs d'importance sont proposés.

```
# A faire en principe par souche
rfFit2=randomForest(trainDescr,trainY,
   importance=T,mtry=2)
imp.sond=sort(rfFit2$importance[,"pne"],
   decreasing=T)[1:10]
par(xaxt="n")
plot(imp.sond,type="h",ylab="Importance",
   xlab="Variables")
points(imp.sond,pch=20)
par(xaxt="s")
axis(1,at=1:10,labels=names(imp.sond),cex.axis=0.8,
   las=3)

imp.sond=sort(rfFit2$importance[,"hac"],
   decreasing=T)[1:10]
par(xaxt="n")
plot(imp.sond,type="h",ylab="Importance",
   xlab="Variables")
points(imp.sond,pch=20)
par(xaxt="s")
axis(1,at=1:10,labels=names(imp.sond),cex.axis=0.8,
   las=3)

imp.sond=sort(rfFit2$importance[,"mac"],
   decreasing=T)[1:10]
par(xaxt="n")
plot(imp.sond,type="h",ylab="Importance",
   xlab="Variables")
points(imp.sond,pch=20)
par(xaxt="s")
axis(1,at=1:10,labels=names(imp.sond),cex.axis=0.8,
   las=3)

imp.sond=sort(rfFit2$importance[,"gor"],
   decreasing=T)[1:10]
par(xaxt="n")
plot(imp.sond,type="h",ylab="Importance",
   xlab="Variables")
points(imp.sond,pch=20)
par(xaxt="s")
axis(1,at=1:10,labels=names(imp.sond),cex.axis=0.8,
   las=3)
```

3.2 Automatisation

L'échantillon est de taille raisonnable mais la variance de l'estimation de l'erreur peut rester importante. Les estimations des erreurs très dépendantes de l'échantillon test sont sujettes à caution et on peut s'interroger sur la réalité des différences entre les différentes méthodes. En regardant les graphiques, on observe qu'il suffit d'une observation pour influencer les résultats. Il est donc important d'itérer le processus sur plusieurs échantillons tests. Il suffit d'intégrer les instructions précédentes dans une boucle.

Exécuter la fonction en annexe en choisissant les méthodes semblant les plus performantes en se montrant patient :

```
models=c("rf","nnet","knn")
noptim=c(6,6,6)
Niter=10 ; Init=11 ;
pred.legio=pred.autom(X,Y,methodes=models,
   N=Niter,p=.8,xinit=Init,size=noptim)
```

Puis calculer :

```
obs=pred.legio$obs
prev.legio=pred.legio$pred
# remettre les noms des facteurs (les souches)
obs.lev=apply(obs, 2, function(x) levels(Y)[x])
prev.legio.lev=lapply(prev.legio,
```

```
   function(x)apply(x,2,function(x) levels(Y)[x]))
# taux de mal classés par échantillon et par méthode
res.legio=lapply(prev.legio.lev,
   function(x)apply(x!=obs.lev,2,mean))
# distributions des taux de mal classés
boxplot(data.frame(res.legio))
# Moyenne des taux de mal classés par méthode
lapply(res.legio,mean)
# Matrices de confusion globales pour random forest
mat.conf.rf=lapply(prev.legio.lev,
   function(x)table(x,obs.lev))$rf
# taux d'erreur global :
(sum(mat.conf.rf)-sum(diag(mat.conf.rf)))/
   sum(mat.conf.rf)
```

Commenter les résultats. Que dire de l'identification d'une souche non-pathogène ?

Annexe : Fonction d'automatisation

```
pred.autom=function(X,Y,p=1/2,methodes=c("knn",
   "rf"),size=c(10,2),xinit=11,N=10,typerr="cv",
   number=4,type="raw") {
# Fonction de prévision de N échantillons tests
# par une liste de méthodes de régression
# ou classification (uniquement 2 classes)
# Optimisation des paramètres par validation
# croisée (défaut) ou bootstrap ou... (cf. caret)
# X : matrice ou frame des variables explicatives
# Y : variable cible quantitative ou qualitative
# p : proportion entre apprentissage et test
# methodes : liste des méthodes de rdiscrimination
# size : e grille des paramètres à optimiser
# xinit : générateur de nombres aléatoires
# N : nombre de réplications apprentissage / test
# typerr : "cv" ou "boo" ou "oob"
# number : nombre de répétitions CV ou bootstrap
# pred : liste des matrices de prévision
# type d'erreur
Control=trainControl(method=typerr,number=number)
# initialisation du générateur
set.seed(xinit)
# liste de matrices stockant les prévisions
# une par méthode
inTrain=createDataPartition(Y,p=p,list=FALSE)
ntest=length(Y[-inTrain])
pred=vector("list",length(methodes))
names(pred)=methodes
pred=lapply(pred,function(x)x=matrix(0,
   nrow=ntest,ncol=N))
obs=matrix(0,ntest,N)
set.seed(xinit)
cat("Patienter ... i=")
for(i in 1:N) {  # N itérations
cat(".",i)
# indices de l'échantillon d'apprentissage
inTrain=createDataPartition(Y,p=p,list=FALSE)
# Extraction des échantillons
trainDescr=X[inTrain,]
testDescr=X[-inTrain,]
trainY=Y[inTrain]
testY=Y[-inTrain]
# stockage des observés de testY
obs[,i]=testY
# centrage et réduction des variables
xTrans=preProcess(trainDescr)
trainDescr=predict(xTrans,trainDescr)
testDescr=predict(xTrans,testDescr)
# estimation et optimisation des modèles
# pour chaque méthode de la liste
for(j in 1:length(methodes)) {
# modélisation
modFit = train(trainDescr, trainY,
```

```
   method = methodes[j], tuneLength = size[j],
   trControl = Control)
# prévisions
if (type=="prob")  pred[[j]][,i]=predict(modFit,
    newdata = testDescr,type=type)[,1]
else pred[[j]][,i]=predict(modFit,
   newdata = testDescr)
}}
list(pred=pred,obs=obs) # résultats
}
```