TP3: RÉSEAUX DE NEURONES AVEC LE PACKAGE NEURALNET

Le but de ce TP est d'apprendre à utiliser le package `NeuralNet` de `R` pour résoudre des taches d'apprentissage supervisé.

Rappelons qu'il est recommandé de taper toutes les instructions dans un fichier et les exécuter en les sélectionnant et en appuyant sur les touches `Ctrl` + `R`**. Pour cela, commencez par lancer** `Rstudio` **faites** `File -> Save as`**. Vous pouvez nommer le fichier** `TP3_Apprentissage_Votre nom.R`**.**

Les données

On va travailler avec les données `Boston`, qui se trouvent dans le package `MASS` :

```
library(MASS)
data <- Boston
```

Tapez `?Boston` pour lire le descriptif de ce jeux de données.

On commence par vérifier que ces données ne contiennent pas de valeur manquante :

```
apply(data,2,function(x) sum(is.na(x)))
```

Nous procédons en séparant de manière aléatoire les données en un échantillon d'apprentissage et un échantillon de test, puis nous entraînons un modèle de régression linéaire et le testons sur l'échantillon de test. Notez qu'on utilise la fonction `glm()` au lieu de `lm()`. Cela deviendra utile plus tard lors de la validation croisée du modèle linéaire.

```
index <- sample(1:nrow(data),round(0.75*nrow(data)))
train <- data[index,]
test <- data[-index,]
lm.fit <- glm(medv~., data=train)
summary(lm.fit)
pr.lm <- predict(lm.fit,test)
MSE.lm <- sum((pr.lm - test$medv)^2)/nrow(test)
```

Remplissez le tableau ci-dessous:

taille d'échantillon	
nombre de variables explicatives	
nature de variable à prévoir (quantitative/catégorielle)	
fonction de perte utilisée	
pourcentage d'observations utilisées dans l'échantillon de test	
erreur obtenue sur l'échantillon de test par le modèle linéaire	

Préparation à l'apprentissage d'un réseau de neurone

Avant de commencer l'entraînement d'un réseau de neurones, une préparation est nécessaire. Dans un premier temps, nous allons effectuer un prétraitement des données dont l'élément le plus important est la normalisation.

L'oubli de normalisation peut avoir plusieurs effets désagréables: non-convergence de l'algorithme d'apprentissage, convergence vers des valeurs inutiles à cause de l'explosion ou, plus souvent, l'annulation des gradients.

Vous pouvez choisir différentes méthodes pour mettre les données à l'échelle (normalisation z, échelle min-max, etc.). J'ai choisi d'utiliser la méthode min-max qui ramène toutes les valeurs dans l'intervalle [0,1] par une transformation affine. Habituellement, la mise à l'échelle dans les intervalles $[0, 1]$ ou $[-1, 1]$ conduit à de meilleurs résultats.

```
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)
train_ <- as.data.frame(scale(train, center = mins, scale = maxs - mins))
test_ <- as.data.frame(scale(train, center = mins, scale = maxs - mins))
```

A noter que `scale` retourne une matrice qui doit être convertie en `data.frame`.

Il y a une erreur dans le code ci-dessus, corrigez-la.

Paramètres

Il n'y a pas de règle générale quant au nombre de couches et de neurones à utiliser, bien qu'il existe plusieurs règles empiriques plus ou moins acceptées. Habituellement, une ou deux couches cachées suffisent pour un grand nombre d'applications.

Comme il s'agit d'un exemple jouet, nous allons utiliser 2 couches cachées avec cette configuration: 13 : 5 : 3 : 1. La couche d'entrée a 13 entrées, les deux couches cachées ont 5 et 3 neurones et la couche de sortie a, bien sûr, une seule sortie puisque nous effectuons une régression.

Passons à la construction du réseau. Il faut d'abord installer (si besoin) et charger le package `NeuralNet`:

```
install.packages("neuralnet")
library(neuralnet)
```

Nous devons ensuite déclarer la "formule", c'est-à-dire quelle est la variable à prévoir (l'étiquette) et quelles sont les variables explicatives (les features). Enfin, il faut déclarer les paramètres du réseau qu'on souhaite entraîner. **Remplacez dans le code ci-dessous `N1,N2` par les valeurs adéquates.**

```
n <- names(train_)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse = " + ")))
nn <- neuralnet(f,data=train_,hidden=c(N1,N2),linear.output=T)
```

Lisez la documentation de la commande `neuralnet` pour comprendre le rôle de l'argument `linear.output`.

Le package `neuralnet` a une fonction `plot` dédiée à l'affichage d'un réseau.

```
plot(nn)
```

Notez cependant que cette fonction n'est utile que pour des réseaux de petites tailles. Essayez les commandes ci-dessus en les modifiant pour fitter un réseau à 5 couches cachées pour vous en convaincre.

Prévoir `medv` en utilisant un réseau de neurones

Maintenant que l'architecture du réseau a été spécifiée, et les poids ont été appris, nous pouvons effectuer les prévisions sur les données de l'échantillon de test. Il faut se rappeler que le réseau va prévoir des valeurs normalisées. Il faudra donc les remettre à l'échelle avant de calculer l'erreur de prévision.

```
pr.nn <- compute(nn,test_[,1:13])

pr.nn_ <- pr.nn$net.result*(max(data$medv)-min(data$medv))+min(data$medv)

MSE.nn <- sum((test$medv - pr.nn_)^2)/nrow(test)
```

Nous pouvons maintenant comparer les deux prévisions: celles faites par le modèle linéaire et celles du réseau de neurones:

```
print(paste("Erreur de prév par Modèle linéaire   : ", MSE.lm))
print(paste("Erreur de prév par Réseau de Neurones: ", MSE.nn))
```

On constate que l'erreur obtenue par les réseaux de neurones est plus faible que celle du modèle linéaire. Il faut garder à l'esprit que ce résultat est aléatoire, car on a découpé les données aléatoirement en un échantillon d'apprentissage et un échantillon de test. De plus, l'apprentissage des réseaux de neurones se fait par une méthode itérative qui utilise une initialisation aléatoire. Pour s'en convaincre, effectuer 5 fois l'apprentissage du réseaux sans changer le découpage de l'échantillon et reporter les erreurs obtenues sur l'échantillon de test dans le tableau ci-dessous:

Erreur 1	Erreur 2	Erreur 3	Erreur 4	Erreur 5

Validation croisée

On souhaite utiliser la validation croisée pour avoir une valeur plus robuste de l'erreur de chacun des deux méthodes de prévision utilisée. Pour cela, on utilisera la validation croisée (10-fold).

On commence par effectuer la validation croisée sur le modèle linéaire:

```
library(boot)
lm.fit <- glm(medv~.,data=data)
cv.glm(data,lm.fit,K=10)$delta[1]
```

Vous pouvez exécuter ces commandes plusieurs fois pour vous assurer que le resultat est peu variable.

Pour faire de la validation croisée avec les réseaux de neurones il faut se fatiguer un peu plus.

```
cv.error <- NULL
k <- 10

library(plyr)
pbar <- create_progress_bar('text')
pbar$init(k)

scaled = as.data.frame(scale(data, center = mins, scale = maxs - mins))


for(i in 1:k){
    index <- sample(1:nrow(data),round(0.9*nrow(data)))
    train.cv <- scaled[index,]
    test.cv <- scaled[-index,]

    nn <- neuralnet(f,data=train.cv,hidden=c(5,2),linear.output=T)

    pr.nn <- # écrire ici la commande qui permet de calculer la prévision
    pr.nn <- # écrire ici la commande qui dénormalise la prévision

    test.cv.r <- (test.cv$medv)*(max(data$medv)-min(data$medv))+min(data$medv)

    cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)

    pbar$step()
}
```

Une fois le travail effectué, nous pouvons afficher la moyenne des erreurs de la validation croisée

```
mean(cv.error)
```

On peut également afficher le boxplot de différentes erreurs obtenues lors de la validation croisée afin de voir leur variabilité:

```
boxplot(cv.error,xlab='MSE CV',col='cyan',
     border='blue',names='CV error (MSE)',
     main='CV error (MSE) for NN',horizontal=TRUE)
```

A faire pour le compte-rendu du TP3

Utilisez votre éditeur de texte préféré (Word, LaTeX, Open Office) pour rédiger le compte-rendu. Convertissez le fichier final en **pdf** avant de le soumettre.

Il ne faut pas envoyer vos codes, vous pouvez les inclure dans votre compte-rendu sous forme d'annexe.

1. Dans le code de la page précédente, qui fait de la validation croisée pour les réseaux de neurones, peut-on remplacer l'argument `linear.output=T` par `linear.output=F` dans la commande `neuralnet(...)`? Expliquer la modification induite par ce changement et afficher le boxplot des erreurs obtenues. Comparer le au boxplot précédent et commenter.
2. La commande `neuralnet` a également un argument appelé `rep`.
   - Expliquer le rôle de cet argument.
   - Lancer la commande `neuralnet` avec l'argument `rep = 10` et calculer l'erreur sur l'échantillon de test. Répéter l'expérience 5 fois et reporter les erreurs obtenues. Comparer ce résultat à ceux du tableau de la page 3.
3. On souhaite maintenant utiliser la tangente hyperbolique comme fonction d'activation pour toutes les couches (chachées + sortie).
   - Ecrire la commande qui permet de spécifier que la fonction d'activation est la tangente hyperbolique.
   - Comment doit-on modifier la normalisation et la dénormalisation ?
4. Il est fortement recommandé de rédiger le compte-rendu en binôme. N'oubliez pas d'indiquer vos noms et vos prénoms dans le rapport ainsi que dans le nom du fichier pdf (exemple: `ENSAE2015_TP2_Dalalyan_Hebiri.pdf`).
5. Soignez la présentation!

La date limite pour l'envoi du compte-rendu est une semaine après la séance.