FregSigCom

2023-12-08

Nature des données

La nature des données est une liste de 9 variables. Chacune de ces dernières correspond à un tableau multidimensionnel de taille 216 par 1800. Les variables sont de type numérique. La plage de temps est de 1800 secondes et la longueur des points de discrétisation est de 216.

```
class(cycling)

## [1] "list"

str(cycling)

## List of 9
##  $ SECS : num [1:216, 1:1800] 0 0 0 0 0 0 0 0 0 0 ...
##  $ KM   : num [1:216, 1:1800] 0 0 0 0 0 0 0 0 0 0 ...
##  $ WATTS: num [1:216, 1:1800] 0 0 508 106 37 0 248 0 0 72 ...
##  $ CAD  : num [1:216, 1:1800] 0 0 101 74 72 0 81 0 0 54 ...
##  $ KPH  : num [1:216, 1:1800] 0 11.3 24.4 16.7 17.3 ...
##  $ HR   : num [1:216, 1:1800] 112 77 122 82 0 77 148 79 87 76 ...
##  $ ALT  : num [1:216, 1:1800] 305 201 199 221 163 ...
##  $ SLOPE: num [1:216, 1:1800] 0 0 0 0 0 0 0 0 0 0 ...
##  $ TEMP : num [1:216, 1:1800] 18 15 17 15 15 17 37 22 15 20 ...
```

Algorithme cv.ff.interaction

Cet algorithme fonctionne par validation croisée

```
#Extraire les données des variables explicatives
colonnes_a_extraire <- c("SECS", "KM", "CAD", "KPH", "HR", "ALT", "SLOPE",
"TEMP")
X <- lapply(colonnes_a_extraire, function(col) cycling[[col]])

#Extraction de la variables à expliquer
Y<-cycling$WATTS

t.x <- lapply(X, function(mat) seq_len(ncol(mat)))
t.y <- seq_len(ncol(cycling$WATTS))

#Choisir les effets principaux : ici on prend toutes nos variables
explicatives comme c'est le 1er modèle
main.effect <- seq_along(colonnes_a_extraire)
```

```
# Appeler la fonction avec les matrices X et t.x
modele1<-cv.ff.interaction(X, Y, t.x = t.x, t.y=t.y,main.effect =
main.effect)

## **CV procedure for nonadaptive fitting**
## **(used to determine the adaptive constants)**
```

Calcul de la prédiction

```
prediction=pred.ff.interaction(modele1,  X)
```

Calcul de l'erreur du modèle (RMSE)

```
error<- round(mean((prediction-Y)^2),0)
print(c(" erreur :", error))

## [1] " erreur :" "14143"
```

L'erreur de ce modèle est de 14143. Cela signifie qu'en moyenne l'écart entre les valeurs prédites et actuelles est de 14143.

Calcul du Mean Absolute Percentage of Error

```
mape_1<-mape(Y, prediction)
print(c("MAPE :",mape_1))

## [1] "MAPE :" "Inf"
```

Le résultat Inf est lié au fait qu'une division par 0 à lieu dans le calcul. Ce qui le rend impossible. Une alternative possible pour le MAPE est le Symmetric Mean Absolute Percentage.

Calcul du Symmetric Mean Absolute Percentage Error

```
smape(Y, prediction)

## [1] 0.5792561
```

En moyenne, l'erreur relative entre les valeurs actuelles et prédites est de 57%. On peut en conclure que le modèle n'est pas performant.

Affichage des courbes réelles vs prédites

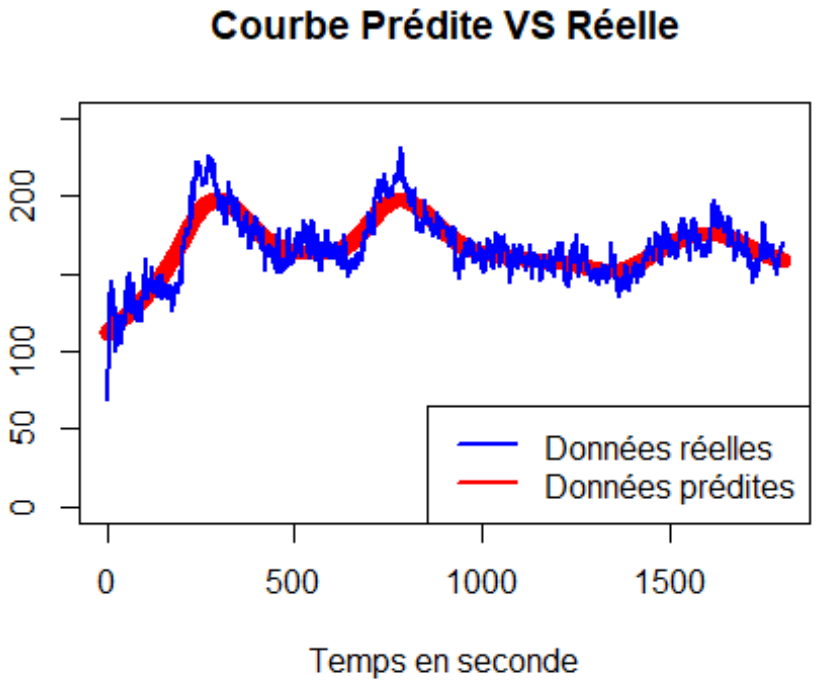
```
# Calculer la moyenne le long des colonnes
moy_pred <- colMeans(prediction)
moy_Y<-colMeans(Y)

#Tracer valeurs prédites
plot(moy_pred,
     xlim=c(0,1800),
     ylim=c(0,250),
     col = "red",
     lty = 1,
     lwd = 2,
```

```
     main="Courbe Prédite VS Réelle",
     xlab="Temps en seconde",
     ylab="")


# Tracer données réelles
lines(moy_Y, col = "blue", lty = 1, lwd = 2)

#Ajout de la légende
legend("bottomright", legend = c("Données réelles", "Données prédites"),
       col = c("blue", "red"), lty = 1, lwd = 2)
```



Algorithme cv.sigcom

Cet algorithme fonctionne par test/train. 70% Des données sont en train test

```
Y=cycling[[3]]
Y.train=Y[1:151,]
Y.test=Y[-(1:151),]
t.y=seq(0,1, length.out = ncol(Y))
X.list=list()
X.train.list=list()
X.test.list=list()
t.x.list=list()
for(i in 1:8)
```

```
{
  X.list[[i]]=cycling[[i+1]]
  X.train.list[[i]]=X.list[[i]][1:151,]
  X.test.list[[i]]=X.list[[i]][-(1:151),]
  t.x.list[[i]]=seq(0,1, length.out = ncol(X.list[[i]]))
}
fit.cv=cv.sigcom(X.train.list, Y.train, t.x.list, t.y)
```

Calcul de la prédiction

```
Y.pred=pred.sigcom(fit.cv, X.test.list)
```

Calcul du RMSE

```
error<- round(mean((Y.pred-Y.test)^2),0)
print(c(" erreur :", error))

## [1] " erreur :" "10027"
```

L'erreur de ce modèle est de 10027. Cela signifie qu'en moyenne l'écart entre les valeurs prédites et actuelles est de 10027. Comparé au modèle crée avec le 1er algorithme on peut conclure que celui ci est un peu mieux.

Calcule du MAPE

```
mape_2<-mape(Y.test,Y.pred)
print(c("MAPE :", mape_2))

## [1] "MAPE :" "Inf"
```

Comme pour l'algorithme précédent, le MAPE n'est pas calculable sur les données.Nous calculons donc le SMAPE

Calcul du Symmetric Mean Absolute Percentage Error

```
smape(Y.test, Y.pred)

## [1] 0.5294462
```

En moyenne, l'erreur relative entre les valeurs actuelles et prédites est de 52%. Avec cet algoithme, le modèle est un peu plus performant.

Affichage des courbes réelles vs prédites

```
# Calculer la moyenne le long des colonnes
moy_Y_pred <- colMeans(Y.pred)
moy_Y_test<-colMeans(Y.test)

#Tracer les données prédites
plot(moy_Y_pred,
     xlim=c(0,1800),
     ylim=c(0,250),
     col = "red",
     lty = 1,
     lwd = 2,
```

```
     main="Courbe Prédite VS Réelle",
     xlab="Temps en seconde",
     ylab="")


# Tracer les données réelles
lines(moy_Y_test, col = "blue", lty = 1, lwd = 2)

# Ajout de la légende
legend("bottomright", legend = c("Données réelles", "Données prédites"),
       col = c("blue", "red"), lty = 1, lwd = 2)
```