# Introduction_caret

Thomas Servant

06/12/2020

## Introduction du package CARET

Le package CARET (Classification And REgression Training) contient des fonctions permettant de rationaliser le processus de formation des modèles pour les problèmes complexes de régression et de classification.

Pour installer caret il faut utiliser :

**install.packages("caret", dependencies = c("Depends", "Suggests"))**

Si l'utilisation d'autres paquets est nécessaire, une invite de commande apparaitra pour les installer.

Les ressources de CARET sont disponibles a https://topepo.github.io/caret/.

CARET a plusieurs fonctions qui tentent de rationaliser le processus de construction et d'évaluation des modèles, ainsi que la sélection des caractéristiques et d'autre techniques.

---

Un de ces principaux outils est la fonction `train` qui peut être utilisé pour :
* Evaluer les effets d'un model
* Choisir le model le plus optimal
* Estimer la performance d'un model selon un set d'entrainement

---

Il y a des options pour customiser a peu près toutes les étapes du process. Nous allons ici utiliser les data Sonar de **mlbench**.

Ce dataset contient 208 point de donnée collecté a l'aide de 60 unité de prédiction. Le but est prédire les deux classes `M` pour les cylindres de metal et `R` pour les pierres.

Premièrement nous allons diviser les data en deux groupes : un set d'entrainement et un set de test. Pour cela nous utiliserons la fonction `createDataPartition`.

```
library(ggplot2)
library(lattice)
library(caret)
library(mlbench)

data(Sonar)

set.seed(107)
inTrain <- createDataPartition(
  y = Sonar$Class,
```

```
  ## les données sur les résultats sont nécessaires
  p = .75,
  ## Tle pourcentage de data
  ## dans le set d'entrainement
  list = FALSE
)
## Le format du résultat

## Le résultat est un set d'entier pour les lignes du Sonar
## cela fait partie du set d'entrainement
str(inTrain)
#>  int [1:157, 1] 1 2 3 4 5 7 10 11 12 13 ...
#>  - attr(*, "dimnames")=List of 2
#>   ..$ : NULL
#>   ..$ : chr "Resample1"
```

Par defaut, `createDataPartition` crée une répartition des données de façon aléatoire.

```
training <- Sonar[ inTrain,]
testing  <- Sonar[-inTrain,]

nrow(training)
#> [1] 157
nrow(testing)
#> [1] 51
```

Pour regler le modèle en utilisant l'algorythm au desssus, la fonction `train` peut être utilisé.

Ici, un modèle partiel d'analyse disciminante des moindres carrés sera accordé sur le nombre de composantes PLS qui devraient être retenues.

La syntaxe est la suivante :

```
plsFit <- train(
  Class ~ .,
  data = training,
  method = "pls",
  ## centre et adapte les unités de prédiction pour l'entrainement
  preProc = c("center", "scale")
)
```

Il est possible donc de modifier les valeurs candidates :

`train` permet de generer des un set de parametre `tuneLength` controle combien d'argument sont contrôlés `tuneGrid` est utilisé quand des valeurs spécifiques sont requises.

```
plsFit <- train(
  Class ~ .,
  data = training,
  method = "pls",
  preProc = c("center", "scale"),
  ## added:
  tuneLength = 15
)
```

Pour modifier la repartition des données on peut utiliser `trainControl`. la fonction `method`controle le type de données. La méthode par defaut est `"boost"` mais ici nous utiliserons la méthode `"repeatdcv"`.

```
ctrl <- trainControl(method = "repeatedcv", repeats = 3)

plsFit <- train(
  Class ~ .,
  data = training,
  method = "pls",
  preProc = c("center", "scale"),
  tuneLength = 15,
  ## added:
  trControl = ctrl
)
```

Finalement, pour choisir différentes méthodes de calcul de performance on peut ajouter des arguments a `trainControl`. `summaryFunction`est utilisé pour afficher une estimation du calcul de la performance. `classProbs = TRUE` est utilisé pour inclure les calculs : `defaultSummary` et `twoClassSummary`.

Enfin, la fonction choisira selon les meilleures résultat quel paramètre garder.

```
ctrl <- trainControl(
  method = "repeatedcv",
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

set.seed(123)
plsFit <- train(
  Class ~ .,
  data = training,
  method = "pls",
  preProc = c("center", "scale"),
  tuneLength = 15,
  trControl = ctrl,
  metric = "ROC"
)
plsFit
#> Moindre carrés
#>
#> 157 samples
#>  60 predictor
#>   2 classes: 'M', 'R'
#>
#>
#>   ncomp  ROC    Sens   Spec
#>    1     0.805  0.726  0.690
#>    2     0.848  0.750  0.801
#>    3     0.849  0.764  0.748
#>    4     0.836  0.765  0.736
#>    5     0.812  0.748  0.755
#>    6     0.789  0.724  0.699
#>    7     0.794  0.744  0.689
```

```
#>    8      0.801  0.739  0.698
#>    9      0.793  0.758  0.677
#>   10      0.790  0.741  0.690
#>   11      0.787  0.742  0.710
#>   12      0.777  0.737  0.715
#>   13      0.772  0.738  0.700
#>   14      0.768  0.718  0.690
#>   15      0.768  0.715  0.690
```

L'output de cette grille est une moyenne des estimation de performance.

```
ggplot(plsFit)
```