CONTRIBUTEURS

ACT-3114 Apprentissage statistique en actuariat

aut., cre. Alec James van Rassel
ctb. Alexis Picard
ctb. Gabriel Lord
src. Marie-Pier Côté

Fonctions R

Dupliqués et niveaux

Commande	Utilité	Résultat	Exemple
`duplicated()`	identifie les dupliqués	retourne un vecteur booléen identifiant les dupliqués	`duplicated(c(1, 1, 2, 4, 6, 4))` `## [1] FALSE TRUE FALSE FALSE FALSE TRUE`
`unique()`	extrais les valeurs uniques	retourne un vecteur contenant les valeurs uniques du vecteur donné en argument	`unique(c(1, 1, 2, 4, 6, 4))` `## [1] 1 2 4 6`
`dplyr::distinct()`	exclure les lignes dupliquées d'une base de données	retourne la BD conservant la première occurrence de dupliqués	`data %>%` `  distinct()`

`levels()`	identifie les niveaux d'un facteur	retourne une list des niveaux du facteur	`fac <- factor(x = c("one", "two", "two"))`
			`levels(fac)`
			`## [1] "one" "two"`

`droplevels()`	identifie les niveaux d'un facteur non utilisés	retourne le facteur en enlevant les niveaux sans observations	`fac_filt <- fac[-1]`
			`droplevels(fac_filt)`
			`## [1] two two`
			`## Levels: two`

`which()`	identifie la **position** d'objets rencontrant la condition	retourne les indices des objets	`data_wt <- c(76, 87, NA, 47, 55, 42, 666, NA)`
			`which(is.na(data_wt))`
			`## [1] 3 8`

Agrégation

`table()` et `prop.table()` Retourne un tableau de fréquence et de proportion.
À 2 dimensions, il faut s'assurer d'avoir les mêmes dimensions pour les 2 vecteurs.
À 3 dimensions, la fonction retourne un tableau par niveau du troisième argument.
`prop.table()` s'enchaîne à une table et ne pas être utilisé directement.
L'argument `margin` spécifie la dimension sur laquelle sommer (1 pour les rangées, 2 pour les colonnes).

```
facto <- factor(x = c("deux", "two", "one", "one", "un", "deux"))
table(facto)

## facto
## deux  one  two   un
##    2    2    1    1

age <- c(18, 20, 19, 18, 20)
weight <- c(200, 150, 175, 190, 220)
table(age, weight)

##     weight
## age  150 175 190 200 220
##   18   0   0   1   1   0
##   19   0   1   0   0   0
##   20   1   0   0   0   1

table(age, weight) %>%
    prop.table(margin = 1)

##     weight
## age  150 175 190 200 220
##   18 0.0 0.0 0.5 0.5 0.0
##   19 0.0 1.0 0.0 0.0 0.0
##   20 0.5 0.0 0.0 0.0 0.5
```

`aggregate()` Calcule une fonction par groupe pour une base de données.
Il faut faire attention à la fonction donnée en argument. Si l'on veut calculer une moyenne et la BD contient des facteurs, la fonction retourne `NA` pour cette colonne avec un message d'erreur.
Il faut donner l'argument à `by` en forme de liste même si nous n'avons qu'une seule variable par laquelle grouper.

```
BD <- data.frame(age = c(18, 20, 19, 18, 20),
                 weight = c(200, 150, 175, 190, 220))
sex <- factor(c("F", "F", "H", "F", "H"))
student <- as.factor(c(T, F, T, T, F))
aggregate(x = BD, by = list(sex), FUN = median)
```

```
##   Group.1  age weight
## 1       F 18.0  190.0
## 2       H 19.5  197.5

aggregate(x = BD, by = list(sex, student), FUN = median)

##   Group.1 Group.2 age weight
## 1       F   FALSE  20    150
## 2       H   FALSE  20    220
## 3       F    TRUE  18    195
## 4       H    TRUE  19    175
```

Modifications de variables

Commande	Utilité	Résultat	Exemple

`tolower()` et `toupper()`	convertis les chaînes de caractères en minuscules et majuscules	retourne le vecteur avec les chaînes de caractères modifiées	`tolower(c("ALec", "JAmES"))`
			`## [1] "alec" "james"`
			`toupper(c("ALec", "JAmES"))`
			`## [1] "ALEC" "JAMES"`

`element_text()`

Paramètre	Description	Possiblités	Exemple
`face`	type de police	"plain", "italic", "bold"	`face = "plain"`
`colour`	couleur du texte		`colour = "blue"`
`size`	taille du texte	pts	`size = 8`
`angle`	angle du texte	de 0 à 360 degrés	`angle = 90`
`hjust` et `vjust`	justification du text selon l'aire du graphique (et non selon les axes)	entre 0 et 1	`hjust = 0.5`

Fonctions R pour l'ajustement de modèles

caret::train()			
Paramètre	**Description**	**Possiblités**	**Exemple**
`form`	formule du modèle		
`data`	jeu de données		`data = dataset`
`method`	famille de modèle à entraîner	`Adaboost.M1`, `gbm` (boosting), `rf` (forêt aléatoire), `rpart` (CART)	`method = "gbm"`
`preProcess`	permet de transformer les données avant l'entraînement. Les données doivent être une matrice ou un tableau.		`preProcess = "scale"`
`tuneLength`	Nombre maximal de combinaisons des paramètres à tester dans l'optimisation.		`tuneLength = 8`
`tuneGrid`	Grille d'hyperparamètres à tester au lieu d'une recherche aléatoire.		`tuneGrid = expand.grid( maxdepth = c(3, 5, 7), mfinal = c(10, 50, 100), coeflearn = "Freund")`
`metric`	Critère à utiliser pour choisir le modèle optimal.	"Accuracy" (classification), "ROC" (Classification), "RSME" (régression)	`metric = "ROC"`
`trControl`	Spécifie les paramètres d'entraînement.	habituellement, on le défini séparément.	`trControl = trControlObjet`

`caret::trainControl()`			
Paramètre	**Description**	**Possiblités**	**Exemple**
`method`	méthode de rééchantillonnage	"cv" (validation croisée), "rcv" (validation croisée multiple), "LGOCV" (leave-group-out CV alias, échantillon de validation aléatoire)	`method = "LGOC"`
`p`	Proportion des données à utiliser par itération de l'entraînement.	chiffre.	`p = 0.6`
`number`	Nombre de plis (cv) ou de groupes d'entraînement (LGOCV).	chiffre.	`number = 2`
`repeats`	Nombre d'itérations complètes de l'entraînement (pour la validation croisée répétée rcv)	chiffre.	`repeats = 4`
`summaryFunction`	Précise le type de problème pour affficher les mesures de performance pertinentes.	twoClassSummary pour une classification binaire.	`summaryFunction = "twoClassSummary"`
`classProbs`	Si vrai, le modèle retourne la probabilité d'être dans chacune des classes ainsi que la prévision.	Booléen.	`classProbs = TRUE`
`search`	Spécifie la méthode de création de la grille d'hyperparamètres.	"grid" ou "random".	`search = "grid"`

`rpart::rpart()`			
Paramètre	**Description**	**Possiblités**	**Exemple**
`formula`	formule du modèle		
`data`	jeu de données		`data = dataset`
`method`	Spécifie le type d'arbre à entraîner.	typiquement, "class" pour la classification et "anova" pour la régression. Il y a aussi l'option "poisson" parmi d'autres.	`method = "class"`
`weights`	Facteurs de pondération des observations du jeu de données.	Vecteur appliqué aux **rangées** ; si un chiffre, sera appliqué à toutes les rangées.	`weights =`
`cost`	Spécifie un vecteur de coûts associés à chaque variable dans la séparation des données. Par exemple, on peut pénaliser une variable qu'on désire seulement utiliser si le pouvoir prédictif est très grand.	Vecteur appliqué aux **colonnes** ; si un chiffre, sera appliqué à toutes les colonnes.	`cost =`
`control`	Permet de spécifier les paramètres d'entraînement avec la fonction `rpart.control()`.	Typiquement défini séparément.	`control = rpart.controlObjet`

`rpart::rpart.control()`			
Paramètre	**Description**	**Possiblités**	**Exemple**
`cp`	Paramètre de complexité.	Poser `cp = 0` implique aucune pénalité.	`cp = 0.01`
`maxdepth`	Profondeur (nombre de branches) maximal de l'arbre.	Chiffre.	`maxdepth = 12`
`minbucket`	Nombre minimal d'observations dans une feuille de l'arbre.	Chiffre.	`minbucket = 5`
`minsplit`	Nombre minimal d'observations dans un nœud pour qu'une séparation des données soit tentée.	Chiffre.	`minsplit = 20`

`randomForest::randomForest()`

Paramètre	Description	Possiblités	Exemple
`formula`	formule du modèle		
`data`	jeu de données		`data = dataset`
`mtry`	Nombre de variables explicatives à considérer à chaque séparation.	Habituellement l'arrondi de la racine du nombre de variables en classification ou le tier du nombre en régression.	`mtry = 7`
`sampsize`	Nombre d'observations dans les sous-échantillons.	Chiffre.	`sampsize = 0.5`
`ntree`	Nombre d'arbres dans la forêt.	Chiffre.	`ntree = 300`
`nodesize`	Nombre minimal d'observations dans une feuille de l'arbre	Chiffre.	`nodesize = 5`
`importance`	Si vrai, calcule l'importance des variables.	Booléen.	`importance = TRUE`

`mod.iml` ← `Predictor$new(modeleTrain)` **puis** `FeatureImp$new(mod.iml, loss = "METRIC", compare = "difference", n.repetitions = 5)` Importance des variables selon l'erreur de classification "METRIC" (e.g., "mse") que l'on peut ensuite visualiser.

`FeatureEffect$new(mod.iml, "variable1", method = "pdp", grid.size = X)` Graphique de dépendance partielle univariée (PDP).

`FeatureEffect$new(mod.iml, feature = c("variable1", "variable2"), method = "pdp", grid.size = X)` Graphique de dépendance partielle bivariée (PDP).

NOTE Si l'on remplace `method = "pdp"` par `method = "ice"` on obtient un graphique d'espérance conditionnelle individuelle (ICE).

`Interaction$new(mod.iml, "variableImp")` Statistique H de Friedman pour quantifier les interactions entre la variable la plus importante et les autres.