CARET

Xueting YIN

2020/12/9

Introduction

Le package caret (abréviation de classification et regression training) est un ensemble de fonctions qui tentent de rationaliser le processus de création de modèles prédictifs. Le package contient des outils pour:

data splitting, pre-processing, feature selection, model tuning using resampling, variable importance estimation.

Dans ce minituto, nous avons choisi un data frame en ligne qui contient 4 variables (admit,gre,gpa,rank), nous allons utiliser le package caret pour construire un modèle prédictif (l'étudiant sera-t-il admis ?) et vérifier ce modèle.

data splitting

```
# install.packages("caret")
library(caret)  # chargement de package caret
```

```
AdmData <- read.csv("https://raw.githubusercontent.com/CGUIM-BigDataAnalysis/BigDataCGUIM/master/binary.csv")    # lire le fichier csv en ligne et mettre dans AdmData
AdmData$admit <- factor(AdmData$admit, levels = c(1,0))
# convertir la colonne admit en facteur, dites à la machine que cette colonne n'est pas des données numériques, mais des données de catégorie，levels est le niveau de facteur，1 pour positive, 0 pour négative
```

pre-processing

Diviser le groupe d'entraînement et le groupe de test, comment faisons-nous?

Une situation extrême est qu'il n'y a que 2 étudiants admis dans 100 étudiants. Nous tirons 75% des étudiants dans le groupe d'entraînement, y compris ces deux étudiants admis, mais nous ne voulons pas de ce résultat.

Nous espérons échantillonner de manière égale, les trois quarts des étudiants admis seront dans le groupe d'entraînement et les trois quarts des étudiants non admis seront dans le groupe de test. Donc nous avons besoin d'une fonction createDataPartition.

```
set.seed(3456)

# Définir la valeur de départ du nombre aléatoire. Une valeur de départ spécifique peut générer une séquence pseudo-aléatoire spécifique. Le but principal de cette fonction est de rendre notre simulation reproductible, car nous devons souvent prendre des nombres aléatoires, mais si nous l'exécutons à nouveau, le résultat est différent.

# Donc Si nous devons répéter le même résultat de simulation avec la fonction set.seed ().

# Les nombres entre parenthèses peuvent être définis à volonté, mais la partie décimale n'est pas valide. Tant que les nombres entre parenthèses sont identiques, le résultat du traitement aléatoire sera le même.
```

```
trainIndex <- createDataPartition(AdmData$admit, p = .75, list = FALSE, times = 1)
# AdmData$admit est le premier paramètre, Il indique à la machine comment échantillonner. Les personnes dont l'admission est 1, prennent 75%, est 0 prennent également 75%.
```

```
# .75 est pour 75%, list est une liste renvoyée, nous ne la voulons pas, times est pour échantionner combien de fois
```

```
head(trainIndex)
```

```
##      Resample1
## [1,]         3
## [2,]         5
## [3,]         6
## [4,]         7
## [5,]         8
## [6,]         9
```

Nous observons que 1, 2 et 4 ne sont pas les données du groupe d'entraînement, donc ce sont les données du groupe de test.

Après avoir index, nous utilisons maintenant la méthode du sous-ensemble pour extraire les données du groupe d'entraînement.

```
TrainData <- AdmData[trainIndex,]
```

Ensuite, nous utilisons la fonction de sous-ensemble inverse pour obtenir les données du groupe de test.

```
TestData <- AdmData[-trainIndex,]
```

feature selection

Ce n'est pas obligatoire. Pour avoir plus d'informations, consultez le document officiel.

model tuning using resampling

Avant de construire un modèle， nous pouvons ajuster les meilleurs paramètres.

La méthode "fold cross validation" est souvent utilisé.

Nous pouvons utiliser la fonction trainControl pour régler la méthode des paramètres et l'enregistrer dans fitControl

```
fitControl <- trainControl(method = "cv", number = 10)
fitControl
#cv est cross validation， donc 10-cross validation
```

Qu'est-ce que c'est 10 cross validation?

Prenons comme exemple 5 cross validation, 5 signifie que les données sont coupées en cinq, la première fois, prenons 1, 2, 3, 4 pour les entrainements, 5 est pour les tests. La deuxième fois, prenons 1,2,3,5 pour les entrainements, 4 est pour les tests. La troisème fois, prenons 1,2,4,5 pour les entrainements, 3 est pour les tests, et ainsi de suite.

Construire un model

```
model <- train(admit~., data = TrainData, method = "glm", trControl = fitControl)
model$finalModel
```

```
##
```

```
## Call:  NULL
##
## Coefficients:
## (Intercept)          gre          gpa         rank
##    3.244646    -0.001854    -0.840782     0.629771
##
## Degrees of Freedom: 300 Total (i.e. Null);  297 Residual
## Null Deviance:       376.9
## Residual Deviance: 342.7     AIC: 350.7
```

```
#train () est la fonction de modèle d'entrainement, y est avant ~,  x à l'arrière de ~. y indique si l'élève sera admis, et x indique quelle pré diction est utilisée.
# "." signifie les autres données sauf admit
# Le deuxième paramètre est de mettre les données, quelles données nous utilisons.
#Le troisième paramètre est quel algorithme est utilisé pour calculer le modèle
#Le quatrième paramètre est traincontrol, ce qui signifie comment ajuster les paramètres
```

Pour savoir les méthodes, nous pouvons consulter les documents officiels.

variable importance estimation

Vérifier l'efficacité du modèle avec la fonction predict().

Attention: Les données de groupe de test ne peuvent être utilisées qu' ici, et il ne peut pas apparaître avant.

```
Pred <- predict(model, newdata = TestData)
head(Pred)
```

```
## [1] 0 0 0 0 0 0
## Levels: 1 0
```

Vérifier la prediction avec la fonction confusionMatrix

```
confusionMatrix(data = Pred, reference =  TestData$admit)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1  6  5
##          0 25 63
##
##                Accuracy : 0.697
##                  95% CI : (0.5965, 0.7853)
##     No Information Rate : 0.6869
##     P-Value [Acc > NIR] : 0.4622241
##
##                   Kappa : 0.1456
##
##  Mcnemar's Test P-Value : 0.0005226
##
##             Sensitivity : 0.19355
##             Specificity : 0.92647
##          Pos Pred Value : 0.54545
##          Neg Pred Value : 0.71591
```

```
##              Prevalence : 0.31313
##          Detection Rate : 0.06061
##    Detection Prevalence : 0.11111
##       Balanced Accuracy : 0.56001
##
##        'Positive' Class : 1
##
```

```
#Le premier paramètre est la valeur qui vient d'être prédite, donc pred est un vecteur 0-1 que nous avons vu tout à l'heure.

#la référence est les données réelles, ce qui signifie que ces étudiants seront vraiment admis ou non, et les données sont stockées dans TestData$admit.
```

Analysez les résultats: la prédiction veut dire admise ou non, la plupart d'entre eux sont supposés qu'ils ne seront pas admis.

Parmis les étudiants qui sont réellement admis, ce modèle a bien prévu 6 étudiants et mal prévu 25 étudiants.

Parmis les étudiants qui ne sont pas réellement admis, ce modèle a bien prévu 63 étudiants et mal prévu 5 étudiants.

Donc ce modèle n'est pas le meilleur car le taux correct n'est que de 0,697.

Une question: la probabilité de la prédiction de chaque personne

Les résultats ci-dessus prédisent que la plupart d'entre eux ne seront pas admis, mais les chances de ne pas être admis sont également différentes: par exemple, certaines personnes ne seront pas admises à 100% et une autre ne sera pas admise qu'à 30%.

Pour voir la probabilité de la prédiction de chaque personne, nous pouvons rajouter un paramètre type = "prob"

```
PredProb <- predict(model, newdata = TestData, type = "prob")
head(PredProb)
```

```
##            1         0
## 1  0.1987224 0.8012776
## 2  0.3047470 0.6952530
## 4  0.1306509 0.8693491
## 12 0.4130344 0.5869656
## 14 0.3504792 0.6495208
## 16 0.2055675 0.7944325
```

Analyse le résultat: la probabilité d'admission de l'étudiant n°1 est de 0,19 et la probabilité de ne pas être admis est de 0,8.

Une autre méthode pour vérifier le résultat de la prédiction

AUC: area under the ROC curve

Comment calculer?

Fusionner la probabilité et le résulat avec la fonction twoClassSummary, l'ensemble de données de test pour cette

méthode doit avoir:

pred： stocker les prédictions, prédire si cette personne sera admise ou non

obs: stocker la bonne réponse, réellement admis ou non

En plus de ces deux colonnes, il faut avoir également une table de probabilité qui indique à la machine le pourcentage de chaque personne est admis ou non admis.

```
PerfData <- cbind(TestData, PredProb)
PerfData$pred <- Pred
PerfData$obs <- PerfData$admit # copier
twoClassSummary(PerfData, lev = levels(PerfData$admit)) # le deuxième paramètre lev n'est pas obligatoire，c'est pour dire à la machine dans PerfData, 1 signifie admis, 0 signifie non admis
```

```
##       ROC      Sens      Spec
## 0.6740987 0.1935484 0.9264706
```

Analyse le résultat: auc est 0.674， un bon modèle doit avoir plus.