Atelier de Machine Learning

Vous allez faire le même exercice de l’atelier à savoir étudier l’utilisation d’une méthode de classification supervisée pour prédire une variable Y où Y vaut 1 si spam et 0 sinon.

Vous devez :

## Utiliser la base de données spam de la libraire kernlab (les données utilisées dans l'atelier)

On note en bleu les lignes de commandes permettant d’effectuer l’analyse, en gris les commentaires et en noir l’analyse des résultats retournés.

|  |  |
| --- | --- |
| Nous chargeons les données spam de la library kernlab dans l’outil R :  #pour charger la lib  library(kernlab)  #pour charger les données  data(spam)  #visualiser les données  View(spam)[[1]](#footnote-0) |  |
| #information sur les données  summary(spam)  Cette commande nous permet de voir que la base contient 2788 non spams et 1813 spams |  |
| #on pose une graine qui fera qu'on aura toujours les mêmes lignes si on lance la fonction createDataPartition  #très important pour s'assurer qu'on travaille bien sur les mêmes données  set.seed(100)  #on fait un tirage au sort de 80% des lignes de nos données  indTrain<-createDataPartition(spam$type,p=0.8,list=F)  #nombre de lignes de ma partition / de mon vecteur de données de training  length(indTrain) |  |
| #Pour voir les valeurs au format valeurs discrètes  View(indTrain) |  |
| #on assigne la matrice spam qui contient toutes les lignes qui sont dans indTrain et toutes les colonnes de spam  #dim(Dtrain] donne la dimension du tableau  Dtrain<-spam[indTrain,]; dim(Dtrain)  #On crée la table des indices de test à partir de toutes les données qui ne sont pas dans Dtrain  Dtest<-spam[-indTrain,]; dim(Dtest)  On vient donc de créer les données d'entraînement (Dtrain) et les données de validation (Dtest). |  |

# Appliquer une autre méthode de classification supervisée pour prédire la variable Y (différente de la méthode utilisée dans l'atelier)

On choisit ici d’appliquer le modèle appelé **gradient boosting machine** (gbm), basé sur un algorithme de création d’arbres de décision.

|  |  |
| --- | --- |
| #On utilise la validation croisée pour découper les données en 10 blocs  paramTrain<-trainControl(method="cv",number=10)  #On lance la fonction “train” qui permet de prédire la variable "type" avec la méthode "gbm", en fonction de tous les attributs  fit\_gbm<-train(type~.,data=Dtrain,method="gbm",trControl=paramTrain, verbose=FALSE)  #Pour chaque bloc on donne le 1-taux d’erreur, appelé Accuracy  #Kappa c'est si on avait fait l'apprentissage avec des fold aléatoires --> on voit bien ici qu'utiliser la méthode donne de meilleurs taux de accuracy  fit\_gbm$resample |  |
| #donne le taux moyen d'erreur sur les 10 boîtes  fit\_gbm  On observe que le taux d’erreur moyen de tous les échantillons est de 1-0,9454 = 0,06 = 6%. |  |

# Donner l'erreur de prédiction estimée par validation croisée

|  |  |
| --- | --- |
| #On crée la variable qui contiendra la prédiction  pred\_gbm<-predict(fit\_gbm,newdata=Dtest)  #On affiche les 6 premières valeurs prédites  head(pred\_gbm) |  |
| #On affiche le résultat de la prédiction  table(pred\_gbm) |  |
| #Compter le nombre de fois où on se trompe quand on utilise les données de test avec mon algo entraîné (matrice de confusion)  table(predite=pred\_gbm,observee=Dtest$type)  On observe que que le prédicteur s’est trompé (20+33=) 53 fois sur 919 soit environ 6% d’erreurs |  |
| #Taux moyen d'échec du prédicteur  mean(pred\_gbm!=Dtest$type) |  |

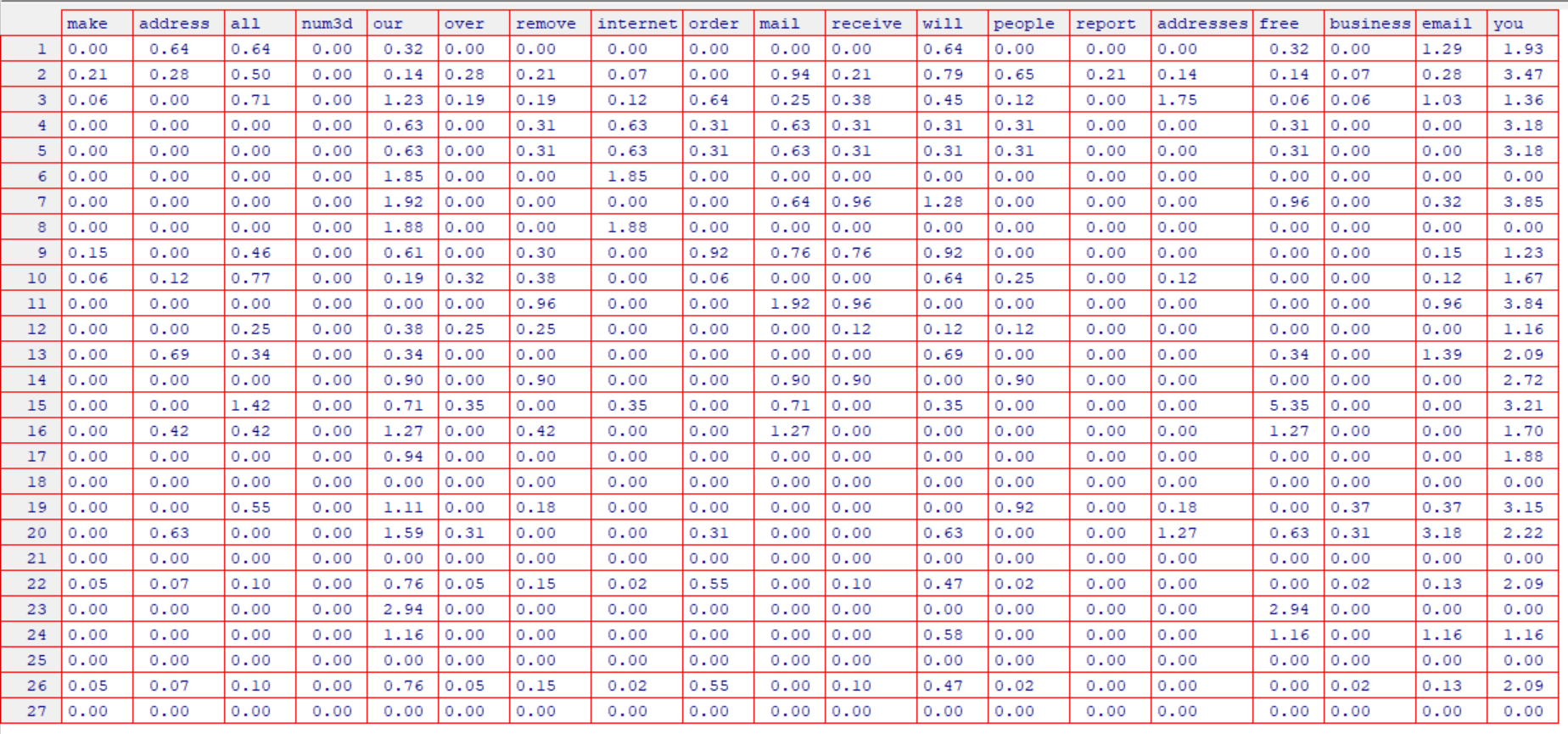
# Comparez le résultat obtenu avec celui de la méthode utilisée dans l'atelier.

Dans le premier exercice nous avons utilisé le modèle appelé General Linear Model. Nous avions obtenu un taux d’erreur lors de la phase de training de 7,5% et de 7,9% lors des prédictions.

Dans le modèle étudié ici, nous avons obtenu 5,6% d’erreur lors de la phase d’entraînement puis de 5,8% lors des prédictions.

On peut donc conclure que le modèle GBM est meilleur.

# Annexe



1. Voir annexe [↑](#footnote-ref-0)