Projet R de Classification binaire avec gestion de VMs

ZHANG Xiaopeng

Sorbonne Université

8 mai 2024

Vue d'ensemble

Spam E-mail Database

- `library("kernlab")`
- Le jeu de données comprend 4601 e-mails, classifiés comme spam ou non-spam, avec 57 variables liés à la fréquence des mots et symboles spécifiques.
- Les variables incluent des fréquences de mots clés, des symboles de ponctuation et des statistiques sur l'utilisation des majuscules.
- La dernière variable catégorise chaque e-mail en « non spam » ou « spam ».

Table – `head(spam)` avec les 3 types de variable

make	address	all	charSemicolon	charRoundbracket	charSquarebracket	capitalAve	capitalLong	capitalTotal	type
0.00	0.64	0.64	0.00	0.000	0.0	3.756	61	278	spam
0.21	0.28	0.50	0.00	0.132	0.0	5.114	101	1028	spam
0.06	0.00	0.71	0.01	0.143	0.0	9.821	485	2259	spam
0.00	0.00	0.00	0.00	0.137	0.0	3.537	40	191	spam
0.00	0.00	0.00	0.00	0.000	0.0	1.000	1	3	nonspam
0.00	0.00	0.00	0.00	0.000	0.0	1.000	1	3	nonspam
0.09	0.00	0.21	0.00	0.032	0.0	2.779	16	282	nonspam
0.00	0.00	0.00	0.00	0.000	0.0	2.000	3	7	nonspam

Pré-sélection des variables

- H_0 : Pour la variable testée, les espérances mathématiques de la fréquence de ce mot-clé dans les courriels **spam** et **non-spam** sont égales.
- H_1 : Négative de H_0, ce qui suggère que cette variable est significative pour déterminer si un mail est spam ou non.
- Le test ne suppose pas une variance homogène pour **spam** et **nonspam** comme le suggère le warning.

```
Breusch-Pagan Test of Heteroskedasticity suggests `var.equal = FALSE`
Breusch-Pagan Test of Heteroskedasticity suggests `var.equal = FALSE`
Breusch-Pagan Test of Heteroskedasticity suggests `var.equal = FALSE`
```

```
library(furniture)
```

	nonspam n = 2788	spam n = 1813	p-Value
make	0.1 (0.3)	0.2 (0.3)	<.001
address	0.2 (1.6)	0.2 (0.3)	0.013
all	0.2 (0.5)	0.4 (0.5)	<.001
num3d	0.0 (0.0)	0.2 (2.2)	0.002
our	0.2 (0.6)	0.5 (0.7)	<.001
over	0.0 (0.2)	0.2 (0.3)	<.001
remove	0.0 (0.1)	0.3 (0.6)	<.001
internet	0.0 (0.2)	0.2 (0.5)	<.001
order	0.0 (0.2)	0.2 (0.4)	<.001
mail	0.2 (0.6)	0.4 (0.6)	<.001

Gestion des VMs - Génération

Soient n et m, nrows et ncols, soit μ, le taux de VM.
la méthode implémentée dans la fonction `Fonction_MCAR()` consiste à choisir uniformément $\lfloor \mu nm \rfloor$ nombres sans remise entre $[\![1, n \times m]\!]$, les nombres choisis sont les indices des valeurs dans le jeu de donnée qui seront remplacé par *NA*.

Table – Partie du dataset spam avec 15% de VM

make	address	all	num3d	our	over	remove	internet	order	mail
	0.64	0.64		0.32	0.00	0.00		0.00	0.00
0.21	0.28	0.50	0.00	0.14	0.28	0.21	0.07	0.00	0.94
	0.00		0.00	1.23	0.19	0.19	0.12	0.64	0.25
0.00	0.00	0.00	0.00		0.00	0.31	0.63	0.31	0.63
0.00	0.00	0.00	0.00	0.63	0.00	0.31	0.63		0.63
0.00	0.00	0.00	0.00		0.00	0.00	1.85	0.00	0.00
0.00	0.00	0.00	0.00	1.92	0.00	0.00	0.00	0.00	0.64
	0.00	0.00	0.00	1.88	0.00	0.00	1.88	0.00	0.00
0.15		0.46	0.00	0.61	0.00	0.30	0.00		0.76
0.06	0.12		0.00	0.19	0.32	0.38		0.06	0.00

Imputation des valeurs manquants par 2 méthodes

Remplacement par la moyenne

- On remplace simplement les NAs par la moyenne empirique de chaque variable

```
if(method=='moyenne'){
  data_5pc_impute <- data_VM$taux_5pc[,-n] %>%
    mutate(across(everything(),
                  ~ifelse(is.na(.), mean(., trim = 0.5, na.rm = TRUE), .)))
  data_10pc_impute <- data_VM$taux_10pc[,-n] %>%
    mutate(across(everything(),
                  ~ifelse(is.na(.), mean(., trim = 0.5, na.rm = TRUE), .)))
  data_15pc_impute <- data_VM$taux_15pc[,-n] %>%
    mutate(across(everything(),
                  ~ifelse(is.na(.), mean(., trim = 0.5, na.rm = TRUE), .)))
```

`library(DMwR2)`

```
data_5pc_impute <- knnImputation(data_VM$taux_5pc[,-n],
                                 k = k_voisins)
data_10pc_impute <- knnImputation(data_VM$taux_10pc[,-n],
                                  k = k_voisins)
data_15pc_impute <- knnImputation(data_VM$taux_15pc[,-n],
                                  k = k_voisins)
```

Méthode kNN

- L'algorithme k plus proches voisins(kNN) est appliqué à l'ensemble des individus en prenant les individus qui ont au moins une valeur NA comme des centres, pour ces centres on trouve k plus proches voisins(complets) et on remplace les NAs par la moyenne empirique des valeurs respectives des k voisins.

Qualité d'imputation

Table – **métrique** moyenne de 100 expériences par imputation kNN

	5pc	10pc	15pc
rmse	2.59	4.71	5.99
mae	0.23	0.55	0.84
R2	0.95	0.89	0.83

Table – **métrique** moyenne de 100 expériences par imputation moyenne

	5pc	10pc	15pc
rmse	3.21	4.80	5.91
mae	0.27	0.55	0.81
R2	0.95	0.89	0.84

Table – **variance** moyenne de 100 expériences par imputation kNN

	5pc	10pc	15pc
rmse	0.96	1.24	1.29
mae	0.00	0.00	0.01
R2	0.00	0.00	0.00

Table – **variance** moyenne de 100 expériences par imputation moyenne

	5pc	10pc	15pc
rmse	0.88	1.26	1.36
mae	0.00	0.00	0.01
R2	0.00	0.00	0.00

Régression Logistique

- La régression logistique est une technique de modélisation statistique utilisée pour la classification binaire.
- Elle prédit la probabilité qu'une entrée donnée appartienne à une catégorie particulière.
- Le résultat est binaire (par exemple, Oui/Non, Succès/Échec).
- Elle estime les paramètres d'un modèle logistique, qui est un cas particulier des modèles linéaires.

Formulation

Le modèle logistique (ou modèle logit) est formulé comme suit :

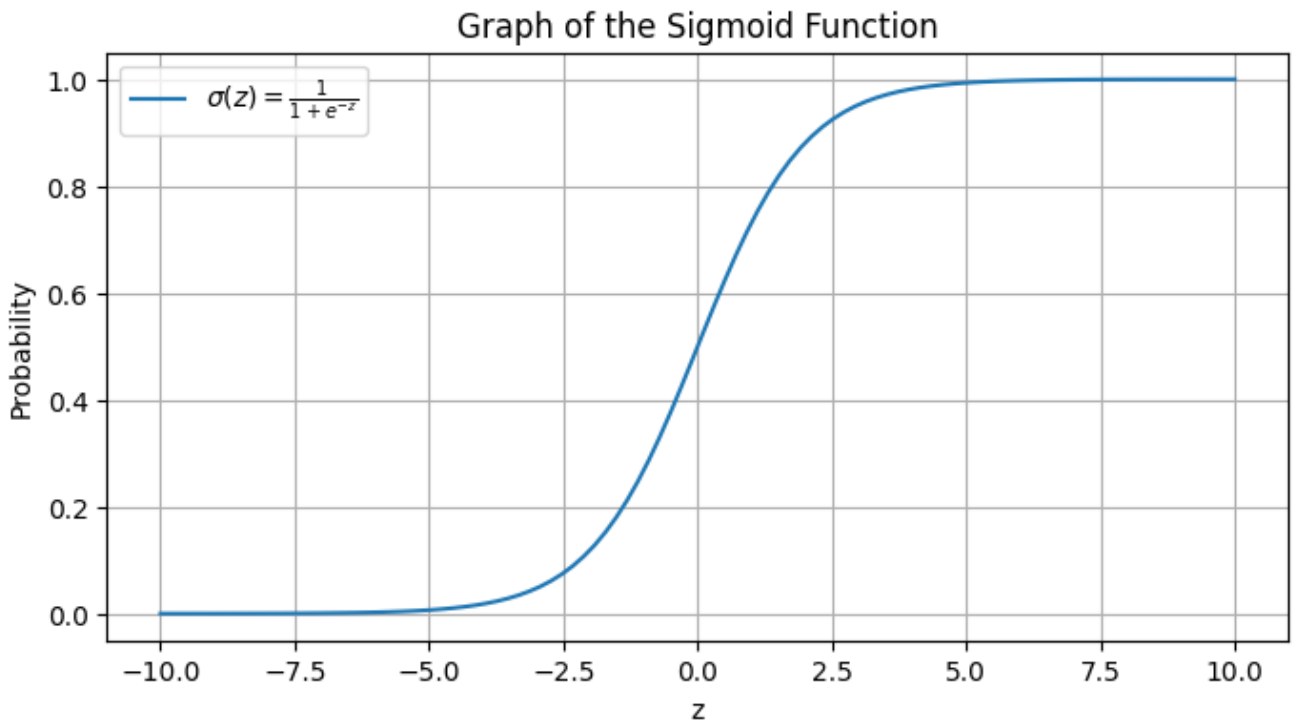
$$\mathbb{P}(Y = 1|X = x) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+\cdots+\beta_k x_k)}}$$

où $\mathbb{P}(Y = 1|X = x)$ est la probabilité de la classe positive.

La fonction sigmoïde

La fonction Sigmoïde $\sigma(z)$ est la pierre angulaire de la régression logistique. Elle mappe tout nombre réel dans l'intervalle (0, 1), la rendant idéale pour modéliser la probabilité :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Estimation des Coefficients β

- Les coefficients β de la régression logistique sont estimés en maximisant la fonction de vraisemblance.
- Cette méthode cherche à maximiser la probabilité que le modèle produise les classes observées, données les observations.

Estimation par la méthode du maximum de vraisemblance

La fonction de vraisemblance $L(\beta)$ pour le modèle logistique est définie par :

$$L(\beta) = \prod_{i=1}^{n} [\mathbb{P}(Y_i = 1 | X_i = x_i)]^{y_i} [1 - \mathbb{P}(Y_i = 1 | X_i = x_i)]^{1-y_i}$$

où y_i est la classe observée pour chaque observation i, et x_i sont les prédicteurs associés.

Modélisation par régression logistique en R

```r
# Data wrangling & Train test split
n<-nrow(spam)

# Using sample(n) for a specific set.seed(n=seed) deter
shuffled_indices<-sample(n)
split_point<-round(0.8*n)
indices_train<-shuffled_indices[1:split_point]
indices_test<-shuffled_indices[(split_point+1):n]
#indices_train <- sort(indices_train)
#indices_test <- sort(indices_test)
rm(n, shuffled_indices, split_point)

# Données train et train pour tableau initial
spam_train <- spam[indices_train,]
#spam_test <- spam[indices_test,]

# Modeling for spam_train
mod_spam_train <- glm(type~. , data = modeling_data$train$origin ,
family=binomial)
```

Figure – train-test split et modélisation par `glm()`

On a généré au total 7 modèles par le même processus en vue de comparer entre les jeux de données avec les valeurs manquantes imputées et celui complet ainsi que l'influence de différentes technique d'imputation

Comparaison des modèles

	origin				imputation_moyenne_5pc				imputation_knn_5pc			
Predictors	*Odds Ratios*	*std. Error*	*CI*	*p*	*Odds Ratios*	*std. Error*	*CI*	*p*	*Odds Ratios*	*std. Error*	*CI*	*p*
(Intercept)	0.20	0.03	0.14 – 0.27	**<0.001**	0.21	0.04	0.15 – 0.29	**<0.001**	0.23	0.04	0.16 – 0.31	**<0.001**
make	0.69	0.17	0.41 – 1.10	0.143	0.68	0.17	0.41 – 1.07	0.111	0.66	0.16	0.40 – 1.03	0.080
address	0.84	0.07	0.66 – 0.96	0.053	0.83	0.08	0.63 – 0.95	**0.041**	0.86	0.06	0.71 – 0.96	**0.031**
all	1.08	0.13	0.85 – 1.37	0.513	1.10	0.14	0.86 – 1.40	0.422	1.10	0.13	0.87 – 1.39	0.410
num3d	8.49	16.09	1.17 – 1000.46	0.259	14.89	28.84	1.66 – 1595.50	0.163	14.82	26.42	1.47 – 1320.79	0.130
our	1.81	0.22	1.45 – 2.30	**<0.001**	1.57	0.18	1.27 – 1.98	**<0.001**	1.56	0.17	1.27 – 1.95	**<0.001**
over	1.89	0.48	1.20 – 3.20	**0.011**	1.84	0.45	1.18 – 3.06	**0.012**	1.60	0.34	1.07 – 2.44	**0.028**

Figure – modèles établis à partir des données ayant 5pc de VMs

	origin				imputation_moyenne_15pc				imputation_knn_15pc			
Predictors	*Odds Ratios*	*std. Error*	*CI*	*p*	*Odds Ratios*	*std. Error*	*CI*	*p*	*Odds Ratios*	*std. Error*	*CI*	*p*
(Intercept)	0.20	0.03	0.14 – 0.27	**<0.001**	0.16	0.03	0.12 – 0.22	**<0.001**	0.19	0.03	0.14 – 0.27	**<0.001**
make	0.69	0.17	0.41 – 1.10	0.143	0.51	0.13	0.31 – 0.83	**0.009**	0.51	0.13	0.31 – 0.83	**0.009**
address	0.84	0.07	0.66 – 0.96	0.053	0.88	0.08	0.69 – 1.00	0.124	0.89	0.08	0.70 – 1.01	0.179
all	1.08	0.13	0.85 – 1.37	0.513	1.13	0.14	0.89 – 1.44	0.313	1.10	0.14	0.86 – 1.40	0.424
num3d	8.49	16.09	1.17 – 1000.46	0.259	5.43	8.42	1.09 – 499.61	0.275	9.26	13.87	1.44 – 620.08	0.137
our	1.81	0.22	1.45 – 2.30	**<0.001**	1.83	0.21	1.48 – 2.30	**<0.001**	1.77	0.18	1.46 – 2.18	**<0.001**

Figure – modèles établis à partir des données ayant 15pc de VMs

Qualité du modèle à seuil 0.5 sur train dataset

À présent, les résultats présentés sont ceux obtenus de l'échantillon original

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textbf{Sensitivity} = \frac{TP}{TP + FN}$$

$$\textbf{Specificity} = \frac{TN}{TN + FP}$$

Table – valeur numérique

Accuracy	Sensitivity	Specificity
0.93	0.93	0.93

Table – Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Table – Confusion matrix of fitted value

	nonspam fitted	spam fitted
nonspam réel	2131	148
spam réel	99	1303

Analyse ROC et optimisation du seuil

- Prédiction des probabilités
- Construction de la courbe ROC
- Calcul de AUC
- Détermination des seuils optimaux :
  Youden's index
  Closet top left

```r
ROC_AUC_Youden_Closet <- function(modele,data){
  predicted_probs<-predict(modele,data,type="response")
  roc_obj<-roc(response=data$type,predictor=predicted_probs,levels=c("nonspam","spam"))
  #plot(roc_obj,main="ROC Curve")
  auc_value<-auc(roc_obj)
  #print(paste("AUC:",auc_value))


  optimal_youden<-coords(roc_obj,"best",ret="threshold",best.method="youden")
  optimal_threshold_youden<-optimal_youden[1]
  sensitivity_youden<-coords(roc_obj,"best",ret="sensitivity",best.method="youden")
  specificity_youden<-coords(roc_obj,"best",ret="specificity",best.method="youden")


  optimal_closest<-coords(roc_obj,"best",ret="threshold",best.method="closest.topleft")
  optimal_threshold_closest<-optimal_closest[1]
  sensitivity_closest<-coords(roc_obj,"best",ret="sensitivity",best.method="closest.topleft")
  specificity_closest<-coords(roc_obj,"best",ret="specificity",best.method="closest.topleft")


  Youden <- data.frame(
    Seuil_optim = optimal_threshold_youden,
    Sensitivity = sensitivity_youden,
    Specificity = specificity_youden
  )


  Closet <- data.frame(
    Seuil_optim = optimal_threshold_closest,
    Sensitivity = sensitivity_closest,
    Specificity = specificity_closest
  )
  return(list(AUC = auc_value,
              ROC = roc_obj,
              Youden = Youden,
              Closet = Closet))
}
```

Figure – `ROC_AOC_Youden_Closet()`

Sélection du seuil de probabilité sur train dataset

Commentaire

On remarque qu'un seuil inférieur à 0.5 permet d'obtenir une meilleure performance du modèle

Figure – ROC original

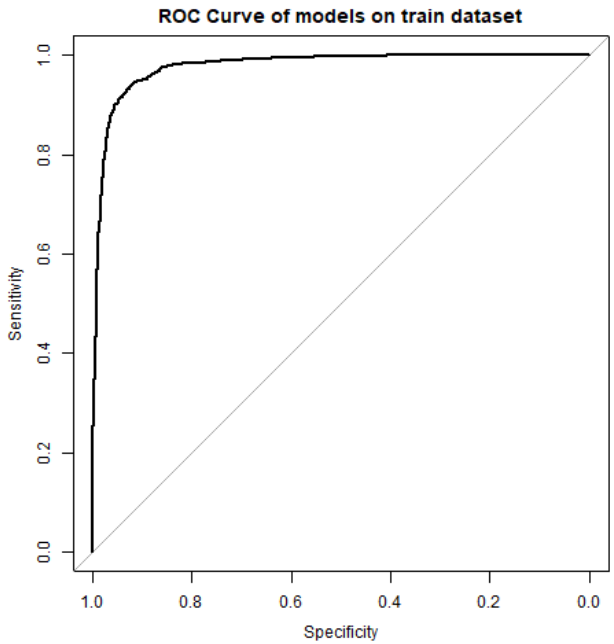


Figure – Sélection du seuil par Youden et par Closet

Name	Type	Value
appli_spam_train	list [4]	List of length 4
AUC	double [1] (S3: auc, numeric)	0.9773981
ROC	list [15] (S3: roc)	List of length 15
Youden	list [1 x 3] (S3: data.frame)	A data.frame with 1 row and 3 columns
threshold	double [1]	0.3447498
sensitivity	double [1]	0.9434873
specificity	double [1]	0.9197309
Closet	list [1 x 3] (S3: data.frame)	A data.frame with 1 row and 3 columns
threshold	double [1]	0.360984
sensitivity	double [1]	0.9379738
specificity	double [1]	0.9242152

Qualité du modèle à seuil 0.5 sur test dataset

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textbf{Sensitivity} = \frac{TP}{TP + FN}$$

$$\textbf{Specificity} = \frac{TN}{TN + FP}$$

Table – valeur numérique

Accuracy	Sensitivity	Specificity
0.94	0.94	0.93

Table – Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Table – Confusion matrix of predicted value

	nonspam fitted	spam fitted
nonspam réel	538	38
spam réel	20	324

Sélection du seuil de probabilité sur test dataset

Commentaire

On remarque aussi qu'un seuil inférieur à 0.5 permet d'obtenir une meilleure performance du modèle

Figure – ROC original

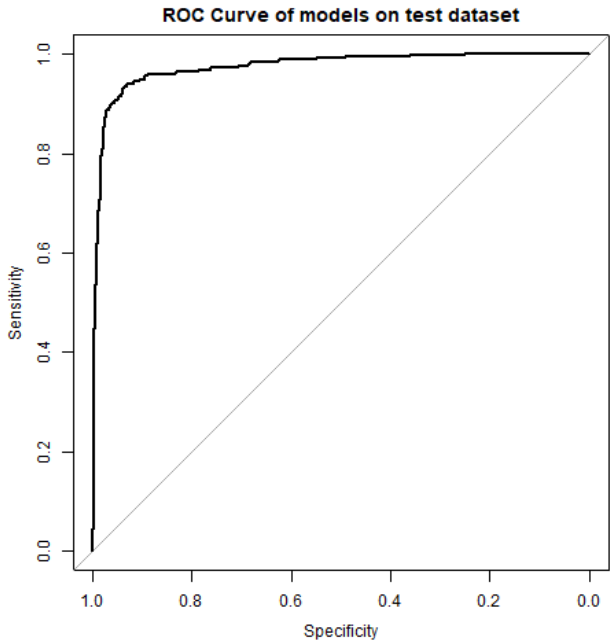


Figure – Sélection du seuil par Youden et par Closet

Name	Type	Value
appli_spam_test	list [4]	List of length 4
AUC	double [1] (S3: auc, numeric)	0.9741777
ROC	list [15] (S3: roc)	List of length 15
Youden	list [1 x 3] (S3: data.frame)	A data.frame with 1 row and 3 columns
threshold	double [1]	0.3714531
sensitivity	double [1]	0.9337017
specificity	double [1]	0.937276
Closet	list [1 x 3] (S3: data.frame)	A data.frame with 1 row and 3 columns
threshold	double [1]	0.3714531
sensitivity	double [1]	0.9337017
specificity	double [1]	0.937276

Non-consensus des méthodes backward, forward et both

```
consensus <- function(modele){
  name_backward <- names(modele$Backward$coefficients)
  name_forward <-  names(modele$Forward$coefficients)
  name_both <-  names(modele$Both$coefficients)

  name_commun <- Reduce(intersect, list(name_backward,name_forward,name_both))

  meme <- identical(name_backward,name_commun) &&
identical(name_forward,name_commun) && identical(name_both,name_commun)

  modele_name <- deparse(substitute(modele)) #Pour obtenir le nom du modèle
  if(meme){
    out <- sprintf("Les trois approches pour  modèle ** %s ** sélectionnent les
mêmes noms de variables ",modele_name)
    print(out)
  }else{
    out <- sprintf("Les trois approches pour  modèle ** %s **  n'ont pas de
consensus ",modele_name)
    print(out)
  }
}
```

Figure – function determine if there is a consensus

```
> minAIC_appli_spam_train <- minAIC_mod(AICmod_spam_train)
> sprintf("The model with least AIC is %s", minAIC_appli_spam_train[[2]])
[1] "The model with least AIC is Backward"
```

Figure – code finding the lowest AIC model

Qualité du modèle optimisant AIC à seuil 0.5

Table – train dataset

Accuracy	Sensitivity	Specificity
0.93	0.93	0.93

Table – Confusion matrix of train

	nonspam fitted	spam fitted
nonspam réel	2135	148
spam réel	95	1303

Table – test dataset

Accuracy	Sensitivity	Specificity
0.94	0.94	0.93

Table – Confusion matrix of test

	nonspam fitted	spam fitted
nonspam réel	538	38
spam réel	20	324

Seuil de probabilité pour modèle least AIC sur train set

Figure – ROC original

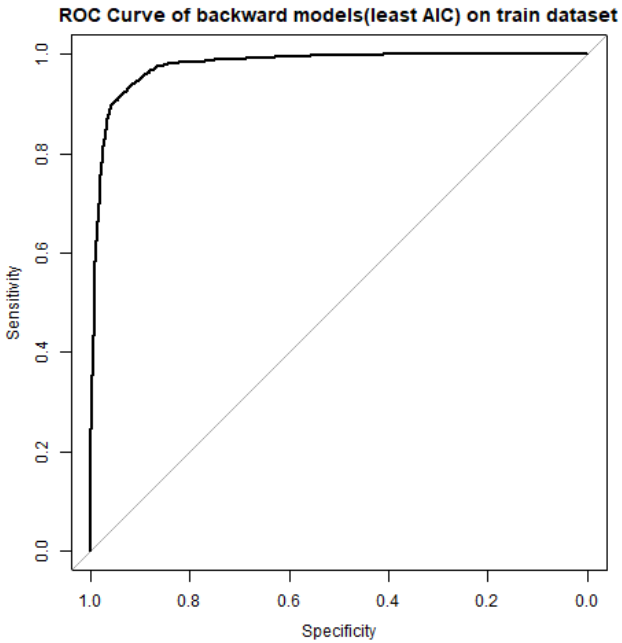


Figure – Sélection du seuil par Youden et par Closet

Name	Type	Value
minAIC_appli_spam_train	list [4]	List of length 4
AUC	double [1] (S3: auc, numeric)	0.9773132
ROC	list [15] (S3: roc)	List of length 15
Youden	list [1 x 3] (S3: data.frame)	A data.frame with 1 row and 3 columns
threshold	double [1]	0.3875034
sensitivity	double [1]	0.9228119
specificity	double [1]	0.9340807
Closet	list [1 x 3] (S3: data.frame)	A data.frame with 1 row and 3 columns
threshold	double [1]	0.3875034
sensitivity	double [1]	0.9228119
specificity	double [1]	0.9340807

Seuil de probabilité pour modèle least AIC sur test set

Figure – ROC original

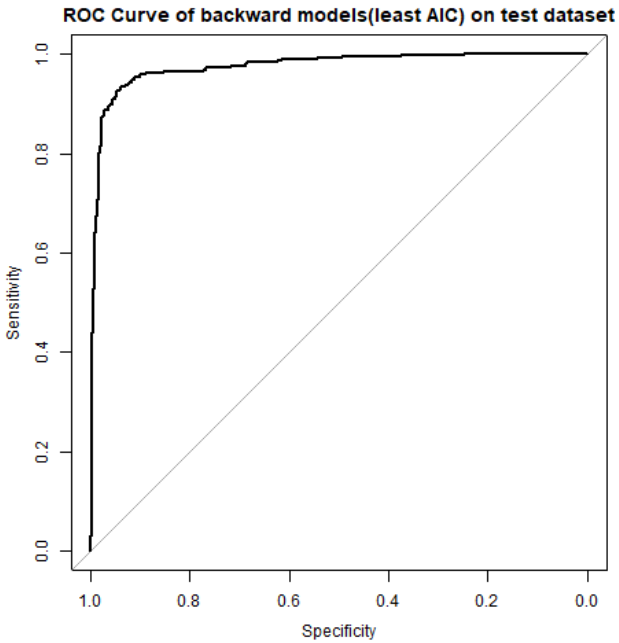


Figure – Sélection du seuil par Youden et par Closet

Name	Type	Value
minAIC_appli_spam_test	list [4]	List of length 4
AUC	double [1] (S3: auc, numeric)	0.9745639
ROC	list [15] (S3: roc)	List of length 15
Youden	list [1 x 3] (S3: data.frame)	A data.frame with 1 row and 3 columns
threshold	double [1]	0.3633923
sensitivity	double [1]	0.9337017
specificity	double [1]	0.9390681
Closet	list [1 x 3] (S3: data.frame)	A data.frame with 1 row and 3 columns
threshold	double [1]	0.3633923
sensitivity	double [1]	0.9337017
specificity	double [1]	0.9390681

Conclusion

- Imputation par méthode kNN a une meilleure performance sur les jeux de données ayant un faible taux de valeur manquante. Or si le taux de valeur manquante est élevé, aucun modèle n'a une performance satisfaisante.
- L'étude de la modélisation sur les jeux de données avec différents taux de VMs et différentes techniques d'imputation a aussi confirmé que kNN est une meilleur méthode en général comme les modèles établis à base des données imputés par kNN est plus cohérents aux modèles depuis les données originales.
- Enfin on remarque que la performance de modèle peut être améliorer en diminuant le seuil de probabilité et, on risque de diminuer la performance de modèle en réduisant les variables dû à la nature du jeu de donnée.

bibliotheques utilisées

```
library("kernlab") # For spam dataset
library(furniture) # For table1()
library(dplyr) # For mutate()
library(DMwR2) # For knnImputation()
library(pROC)
library(sjPlot) # For tab_model(), plot_model()
library(MASS)
```