Livret des Travaux Pratiques de Statistique en Grande dimension

M2 SSD-MIASHS

Statistiques et Sciences des Données

(Mathématiques et Informatique Appliquées aux Sciences de l'Homme et de la Société)

Intitulé du cours : Analyse des données 1

Chargé du cours

Professeur Mustapha RACHDI

Bureau : C008 du Bât. Michel Dubois

sur RDV

mustapha.rachdi@univ-grenoble-alpes.fr

Unité de Formation et de Recherche

Sciences de l'Homme et de la Société

Université Grenoble Alpes

UFR SHS, BP. 47

38040 Grenoble Cedex 09

Année universitaire

2019-2020

Table des matières

Table des matières **3**

1 Introduction à la Statistique en Grande Dimension **5**
1.1 Problème de l'estimateur du maximum de vraisemblance 5
1.2 Problème de sélection de modèle 6
1.3 Devoir 6

2 Sélection de modèle **9**
2.1 Les données 9
2.2 Modèle linéaire complet 10
2.2.1 Estimation du modèle et graphes des résidus 10
2.2.2 Erreur d'apprentissage 10
2.2.3 Erreur sur l'échantillon test 10
2.2.4 Nouvelle paramétrisation 10
2.3 Sélection de modèle par sélection de variables 11
2.3.1 Sélection par AIC et backward 11
2.3.2 Sélection par AIC et forward 11
2.3.3 Sélection par AIC et stepwise 11
2.3.4 Sélection par BIC et stepwise 11
2.3.5 Erreur sur l'échantillon d'apprentissage 11
2.3.6 Calcul de l'erreur sur l'échantillon test 11
2.4 Devoir 12

3 Pratique de la régression Ridge et et Elasticnet **13**
3.1 Données et régression logistique `glm()` 13
3.1.1 Base de données "adult" 13
3.1.2 Importation des bases d'apprentissage et de test – Quelques vérifications . . . 14
3.1.3 La librairie `glmnet` 15
3.1.4 Régression elasticnet 20
3.2 Les librairies " tensorflow/keras" 22
3.2.1 Regression logistique sous forme de perceptron simple 22
3.3 Recapitulatif des resultats 27
3.4 Conclusion 27
3.5 Devoir 28
3.6 Sélection de modèle par pénalisation Ridge 28
3.6.1 Comportement des coefficients 28

3.6.2 Pénalisation optimale par validation croisée . . . 28
3.6.3 Prévision et erreur d'apprentissage . . . 28
3.6.4 Prévision sur l'échantillon test . . . 29
3.7 Sélection de modèle par pénalisation Lasso . . . 29
3.7.1 Librairie Lasso2 . . . 30
3.7.2 Construction du modèle . . . 30
3.7.3 Visualisation des coefficients . . . 30
3.7.4 Sélection de la pénalité par validation croisée . . . 30
3.7.5 Erreur d'apprentissage . . . 30
3.7.6 Erreur sur l'échantillon test . . . 30
3.7.7 Librairie glmnet . . . 30
3.7.8 Mise en forme des variables . . . 31
3.7.9 Construction du modèle . . . 31
3.7.10 Visualisation des coefficients . . . 32
3.7.11 Sélection de la pénalité par validation croisée . . . 32
3.7.12 Erreur d'apprentissage . . . 32
3.7.13 Erreur sur l'échantillon test . . . 32
3.7.14 Elastic Net . . . 32
3.8 Sélection de modèle projection sur composantes orthogonales . . . 33
3.8.1 Régression PLS . . . 33
3.8.2 Régression sur composantes principales . . . 33

4 Pratique de la SVM **35**
4.1 Données et régression logistique `glm()` . . . 35
4.1.1 Base de données "adult" . . . 35
4.2 Devoir . . . 35

5 Analyse de données fonctionnelles (FDA) & Sélection de variables **37**
5.1 Données et régression logistique `glm()` . . . 37
5.1.1 Base de données "adult" . . . 37
5.1.2 Importation des bases d'apprentissage et de test – Quelques vérifications . . . 38
5.1.3 La librairie `glmnet` . . . 39
5.2 Devoir . . . 44

Chapitre 1

Introduction à la Statistique en Grande Dimension

Nous allons mettre le point sur deux problèmes, parmi d'autres (qui seront donnés, de façon non exhaustive, en devoir), qui sont rencontrés lors du passage de la statistique classique à la Statistique en Grande dimension (SGD).

— Estimateur du maximum de vraisemblance (mle)
— Sélection de modèle

1.1 Problème de l'estimateur du maximum de vraisemblance

On considère une matrice de données indépendantes X. Cette matrice $n \times p$ contient les variables prédictives. L'élément x_{ij} enregistrant la valeur de la $j$ème variable pour le $i$ème individu. On désigne par y une réalisation de la variable réponse Y : vecteur de longueur n. On a

$$y_i = \beta_0 + \beta_j x_{ij} + \varepsilon_i$$

où les variables $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ sont indépendantes et identiquement distribuées.

On va essayer de montrer empiriquement que :

— On constate qu'il y a plusieurs inconvénients a utiliser le maximum de vraisemblance pour estimer β quand p est grand.
— De plus, quand $p > n$ la matrice ${}^tXX$ n'est pas inversible, le MLE n'est pas unique.
— Néanmoins, même si ${}^tXX$ peut être inversé et un maximum unique identifié, comme p augmente et X approche la singularité, la surface de vraisemblance devient très plate.
— Cela signifie qu'une large gamme de valeurs de β est consistante avec les données, et de larges intervalles de confiance sont requis afin d'atteindre, disons, une confiance de 95%.

Par exemple, si X est une matrice avec $n = 20$ lignes et dont les éléments sont constitués de nombres aléatoires indépendants et normalement distribués, tracer la plus grande variance des estimations β_j en augmentant le nombre de colonnes de X de 1 à p.

1. Générer dans une matrice X, $p = 19$ échantillons de taille $n = 19$ de la loi normale standard. On a donc un échantillon de taille $n = 20$ pour $p = 19$ co-variables.
2. Tracer le maximum des variances des estimateurs des coefficients, du modèle linéaire, en fonction du nombre de colonnes i.e., des co-variables (ici, de 1 à p).
3. que se passe-t-il quand p s'approche de n.
4. Interpréter.

1.2 Problème de sélection de modèle

1. On considère une matrice de données X de $n = 25$ observations sur $p = 100$ co-variables distribuées aléatoirement et une matrice XX de même dimension mais distribuée uniformément sur $(0, 1)$. Puis une variable d'intérêt y constituée de n nombres aléatoires indépendants et issus d'une loi normale centrée-réduite. On selectionne un modéle avec seulement 5 variables par BIC et on répète ceci 500 fois et on observe les estimations $\widehat{\beta}$ de β. Compiler et interpréter la procédure suivante correspondant au Slide 16 :

```
set.seed(1)
n <- 25
p <- 100
xnam <- paste0("V", 1 :p)
form <- as.formula(paste("y ~ ", paste(xnam, collapse= "+")))
N <- 100


res <- cover <- pred <- NULL
pb <- txtProgressBar(1, N, style=3)
for (i in 1 :N) {
X <- (matrix(runif(n*p), n, p))
Data <- as.data.frame(X)
y <- rnorm(n)
j <- which.max(abs(crossprod(X, y - mean(y))))
XX <- (matrix(runif(n*p), n, p))
pData <- as.data.frame(XX)
yy <- rnorm(n)
fit0 <- lm(y~1, data=Data)
fit <- step(fit0, scope=form, direction="forward", trace=0, k=log(n), steps=5)
res <- rbind(res, summary(fit)$coef[-1,])
pred <- c(pred, yy - predict(fit, pData))
cover <- c(cover, apply(confint(fit)[-1,,drop=FALSE], 1, prod) <= 0)
setTxtProgressBar(pb, i)
                  }
```

2. Tracer l'histogramme du Slide 17 et interpréter.
3. Calculer les indicateurs du Slide 19.

1.3 Devoir

1. Constater les problème pouvant surgir dans une ACP lorsque l'on est en grande dimension.
2. Y a t il des problèmes liés à la grande dimension lorsque l'on effectue un test multiple.

3. Que signifient/révèlent les expressions suivantes : Family-wise error rates, False discovery rates, Local false discovery rates,

Chapitre 2

Sélection de modèle

Nous allons faire le tour des méthodes connues permettant de sélectionner le modèle. Ce TP sera donc l'occasion de traiter de façon presque exhaustive un problème concret en utilisant ces dites méthodes. Ce TP est repris du Scénario "sélection de modèles" de WikiStat.

Afin d'accompagner ce TP, nous considérons le jeu de données `Prostate` sous `R`. Nous allons procéder à une comparaison sur ce meme jeu de données des qualités de prévision de plusieurs modèles obtenus par :

- Modèle linéaire
- Algorithmes de sélection de modèles (critères AIC, BIC)
- Dans la séance suivante et sur les mêmes données : Régularisation (ridge, Lasso, Elastic Net) et Régression sur composantes (PCR, PLS)

2.1 Les données

Nous allons utiliser les données Prostate du package `lasso 2` de `R`.

Ces données [1] proviennent d'une étude qui a examiné la corrélation entre le niveau d'antigène spécifique de la prostate et un certain nombre de mesures cliniques chez les hommes sur le point de subir une prostatectomie radicale. Il sagit dune matrice de données comportant 97 lignes et 9 de colonnes.

- Télecharger les données.
- Vérfier bien que les données correspondent au tableau de description suivant :
- Faites quelques statistiques descriptives : dim, names, summary, cor, hist
- La suite du travail nécessite un échantillon d'apprentissage pour estimer les modèles et un échantillon test pour comparer les erreurs de prévision. Les valeurs de `lpsa` sont rangées par ordre croissant. On conserve 1/4 des données pour l'échantillon test (Prostate.test).
- Donner quelques statistiques descriptives des données : Prostate.app et Prostate.test

1. Stamey, T.A., Kabalin, J.N., McNeal, J.E., Johnstone, I.M., Freiha, F., Redwine, E.A. and Yang, N. (1989). Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate : II. radical prostatectomy treated patients, Journal of Urology 141(5), 10761083.

2.2 Modèle linéaire complet

Une fonction utile de graphe des résidus :

```
plot.res=function(x,y,titre="")

plot(x,y,col="blue",ylab="Résidus", xlab="Valeurs predites",main=titre)
abline(h=0,col="green")
```

2.2.1 Estimation du modèle et graphes des résidus

```
modlin=lm(lpsa∼., data=Prostate.app)
summary(modlin) # noter les p-valeurs
#Residus
res=residuals(modlin)
#Regroupement des graphiques sur la meme page
par(mfrow=c(1,2))
hist(residuals(modlin))
qqnorm(res)
qqline(res, col = 2)
# retour au graphique standard
par(mfrow=c(1,1))
plot.res(predict(modlin),res)
```

2.2.2 Erreur d'apprentissage

Calculer l'erreur d'apprentissage : `mean(res**2)`

2.2.3 Erreur sur l'échantillon test

```
pred.test=predict(modlin, newdata=Prostate.test)
res.test=pred.test-Prostate.test$lpsa
mean(res.test**2)
```

2.2.4 Nouvelle paramétrisation

Afin de faciliter l'interprétation des résultats concernant les variables qua- litatives, on introduit une nouvelle paramétrisation à l'aide de contrastes. Par défaut, la référence est prise pour la valeur 0 de svi et 6 de gleason, qui sont les plus petites valeurs. Les paramètres indiqués pour les variables svi1, gleason 7, 8 et 9 indiquent l'écart estimé par rapport à cette référence. Il est plus intéressant en pratique de se référer à la moyenne des observations sur toutes les modalités des variables qualitatives, et d'interpréter les coefficients comme des écarts à cette moyenne.

```
contrasts(Prostate.app$svi)= contr.sum(levels(Prostate.app$svi))
contrasts(Prostate.app$gleason)= contr.sum(levels(Prostate.app$gleason))
modlin2=lm(lpsa∼., Prostate.app)
summary(modlin2)
```

Remarque 2.2.1. Attention au nom des variables : gleason1 =6, gleason2 =7, gleason3 =8, gleason4 =9 (pas affiché). La somme des coefficients associés à ces variables est nulle
svi1=0, svi2=1 (pas affiché). La somme des deux coefficients est nulle

2.3 Sélection de modèle par sélection de variables

2.3.1 Sélection par AIC et backward

```
library(MASS)
modselect_b=stepAIC(modlin2,∼.,trace=TRUE, direction=c("backward"))
summary(modselect_b)
```

2.3.2 Sélection par AIC et forward

```
mod0=lm(lpsa 1,data=Prostate.app)
modselect_f=stepAIC(mod0,lpsa∼lcavol+lweight
+age+lbph+svi+lcp+gleason+pgg45,data=
Prostate.app,trace=TRUE,direction=c("forward"))
summary(modselect_f)
```

2.3.3 Sélection par AIC et stepwise

```
modselect=stepAIC(modlin2,∼.,trace=TRUE, direction=c("both"))
#both est l'option par défaut
summary(modselect)
#On retrouve ici le meme modèle qu'avec l'agorithme backward
```

2.3.4 Sélection par BIC et stepwise

```
# k=log(napp) pour BIC au lieu de AIC.
modselect_BIC=stepAIC(modlin2,∼.,trace=TRUE, direction=c("both"),k=log(napp))
summary(modselect_BIC)
#Le modèle sélectionné est plus parcimonieux
```

2.3.5 Erreur sur l'échantillon d'apprentissage

```
#Modèle stepwise AIC
mean((predict(modselect)-Prostate.app[,"lpsa"])**2)
#valeur trouvée 0.49

#Modèle stepwise BIC
mean((predict(modselect_BIC)-Prostate.app[,"lpsa"])**2)
# valeur trouvée 0.53
```

2.3.6 Calcul de l'erreur sur l'échantillon test

```
#Modèle stepwise AIC
mean((predict(modselect,newdata=Prostate.test)-Prostate.test[,"lpsa"])**2)
#valeur trouvée 0.46
#Modèle stepwise BIC
mean((predict(modselect_BIC,newdata=Prostate.test)-Prostate.test[,"lpsa"])**2)
# valeur trouvée 0.40
```

Remarque 2.3.1. Les modèles sélectionnés ont une erreur plus grande que le modèle linéaire comprenant toutes les variables sur l'échantillon d'apprentissage (c'est normal !). Sur l'échantillon test, le modèle qui minimise le critère BIC a de meilleures performances que le modèle initial. Les deux modèles sélectionnés sont beaucoup plus parcimonieux que le modèle initial.

2.4 Devoir

Chapitre 3

Pratique de la régression Ridge et et Elasticnet

Nous allons exposer un tutoriel de la pratique des régressions ridge et elasticnet sous la logiciel `R` via les packages `glmnet` et `tensorflow/keras`. Ce tutoriel fait suite au support de cours consacré a la régression régularisée.

Nous nous situons dans le cadre de la régression logistique avec une variable cible qualitative binaire. Les propriétés de régularisation de ridge et elasticnet devraient se révéler décisives. Encore faut-il savoir/pouvoir déterminer les valeurs adéquates des paramètres de ces algorithmes. Ils pèsent fortement sur la qualité des résultats.

Nous verrons comment faire avec les outils a notre disposition. Nous utiliserons les packages `glmnet` et `tensorflow/keras`. Il faut s'y référer notamment pour la partie installation qui n'est pas triviale.

3.1 Données et régression logistique `glm()`

3.1.1 Base de données "adult"

Nos données sont dérivées de la base "Adult Data Set" accessible sur le serveur UCI [1]. Elle a été retravaillée puis publiée sur le site LIBSVM [2] : les variables quantitatives ont été discrétisées, puis transformées en variables indicatrices via un codage disjonctif complet ; les variables catégorielles ont également été binarisées via le meme procédé. En définitive, nous disposons de $p = 123$ variables explicatives, toutes binaires. La variable cible est également binaire, définie dans *positive, negative*.

Nous avons réservé $ntrain = 200$ observations (extraits aléatoirement de la base d'apprentissage de "a1a") pour la modélisation, $ntest = 30956$ pour l'évaluation.

1. https ://archive.ics.uci.edu/ml/datasets/adult
2. https ://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html ; base "a1a"

3.1.2 Importation des bases d'apprentissage et de test – Quelques vérifications

Echantillon d'apprentissage. Nous importons l'échantillon d'apprentissage dans un premier temps.

```
#changer le dossier courant
setwd("... votre dossier ...")
#charger les données d'apprentissage
DTrain <- read.table("adult_train.txt",sep=’’ t’’,header=TRUE) print(dim(DTrain))
## [1] 200 124
```

Nous vérifions la répartition des classes. Le modèle par défaut consisterait a prédire systématiquement la classe majoritaire *negative*.

Répartition de la classe

```
print(prop.table(table(DTrain$classe)))
## ## negative           positive
## 0.765            0.235
```

Nous plaçons les variables explicatives et la variable cible dans deux structures distinctes, une matrice pour les premières, un vecteur pour la seconde.

```
#typage
XTrain <- as.matrix(DTrain[,-1])
yTrain <- as.matrix(DTrain[,1])
```

Notons une particularité qui ne va certainement pas faciliter le processus de modélisation : 32 colonnes sont composées de la valeur constante 0. Logiquement, il faudrait exclure d'emblée ces variables.

Elles sont sources de problèmes et ne peuvent en rien contribuer a la qualité prédictive des modèles. Dans notre cas, nous les conservons quand-meme pour corser l'affaire et voir justement comment se comporteront les différentes techniques et outils que nous examinerons dans ce tutoriel.

```
#particularité - variables remplies de 0 (somme de la colonne = 0)
print(length(which(colSums(XTrain)==0)))
## [1] 32
```

Échantillon test. Nous chargeons l'échantillon test dans un second temps. Il est composé de $ntest = 30956$ observations.

```
#charger les données test
DTest <- read.table("adult_test.txt",sep="",header=TRUE) print(dim(DTest))
## [1] 30956      124


#répartition des classes
print(prop.table(table(DTest$classe)))
## ## negative     positive
## 0.759465        0.240535
```

La répartition des classes est très similaire a celle de l'échantillon d'apprentissage, heureusement. Avec le modèle par défaut, prédiction systématique de la classe majoritaire *négative*, le taux d'erreur serait de 24.05%. Il s'agit de faire mieux avec les différentes variantes de la régression logistique que nous mettrons en uvre dans ce qui suit.

Nous placons également la matrice des descripteurs dans une structure dédiée.

```
# matrice en test
XTest <- as.matrix(DTest[,-1])
```

Régression logistique avec `glm()`

La fonction `glm()` du package `stats`, chargé automatiquement au démarrage, est l'outil privilégié pour la régression logistique sous `R`. Nous l'appliquons a l'échantillon d'apprentissage. Un message d'avertissement indiquant la non convergence de l'algorithme apparait. Ce n'est pas bon signe.

```
#rég. logistique usuelle
reg1 <- glm(classe ∼., data = DTrain, family = "binomial")
## Warning : glm.fit : algorithm did not converge
## Warning : glm.fit : fitted probabilities numerically 0 or 1 occurred
```

Affichons quand-meme les résultats par acquit de conscience. `print(summary(reg1))`

Le modèle est inutilisable. Certains coefficients n'ont pas été estimés et correspondent a la valeur `NA` (Not Available). Nous ne pouvons ni les interpréter, ni les déployer sur des individus supplémentaires. De fait, nous n'avons meme pas essayé d'appliquer le modèle sur l'échantillon test pour en mesurer les performances.

3.1.3 La librairie `glmnet`

La librairie `glmnet` a été développée par des grands noms de la statistique (on a le vertige rien qu'en lisant la liste des personnalités associées au package). Elle est efficace (rapidité des calculs, qualité des résultats), et surtout, elle propose toute une panoplie d'outils qui se révèlent précieux dans l'analyse exploratoire, lors de la recherche des combinaisons adéquates des paramètres λ et α. Cet aspect est d'autant plus important que, nous ne verrons dans ce tutoriel, le gap de performances entre les modèles selon les valeurs attribuées a ces paramètres peut etre important.

Pour rappel, la fonction a optimiser en régression logistique régularisée s'écrit (voir Hastie & Qian, 2016) :

$$J = \left[\frac{1}{ntrain}\sum_{i=1}^{ntrain} y_i\left(\beta_0 + x_i^t\beta\right) - \log\left(1 + \mathrm{e}^{\beta_0 + x_i^t\beta}\right)\right] + \lambda\left[\frac{1-\alpha}{2}||\beta||_2^2 + \alpha||\beta||_1\right]$$

où β_0 est la constante, β est le vecteur des coefficients appliqués aux variables. On observe que la constante de la régression n'est pas concernée par la régularisation.

λ est le coefficient de pénalité ; α permet d'arbitrer entre la contrainte portant sur les normes L^2 (ridge, $\alpha = 0$) et L^1 (lasso, $\alpha = 1$) des coefficients.

Régression logistique non-régularisée

Après avoir installé `glmnet` (a faire une seule fois), nous la chargeons (`library`) ...

```
#librairie glmnet
library(glmnet)
## Loading required package : Matrix ## Loading required package : foreach ## Loaded
glmnet 2.0-16
```

... puis nous lancons la régression logistique sans paramètre de régularisation ($\lambda = 0$).

Remarque 3.1.1. Nous désactivons la standardisation (standardize = `FALSE`) des variables explicatives parce que nous savons qu'elles sont toutes binaires, définies de facto sur la meme échelle. Sur une base quelconque, en l'absence d'informations précises sur les variables, il est plus prudent d'activer l'option.

```
#régression
reg2 <- glmnet(XTrain,yTrain,family="binomial",standardize=FALSE,lambda=0) print(reg2)
```

Tres curieusement, de par lalgorithme doptimisation utilisé, le processus semble converger malgré tout contrairement a `glm()` de `stats`. La valeur zéro est attribuée aux coefficients associés aux colonnes de constante. Ces variables sont inactives pour la prédiction.

Nous appliquons le modele sur léchantillon test avec `predict()`.

```
# prédiction
yp2 <- predict(reg2,XTest,type="class",s=c(0)) print(table(yp2))
```

```
## yp2
## negative      positive
## 19568           11388
```

Les prédictions sont reparties entre les deux classes, il ny a pas de prédiction systématique...

```
#taux d'erreur
print(sum(DTest$classe != yp2)/nrow(DTest))
```

```
   ## [1] 0.3150924
```

... mais le modele fait pire que la prédiction par défaut avec un taux derreur de 31.5%.

Conclusion. Manifestement, les difficultés (le ratio $p/ntrain$ élevé et, surtout, les variables constituées de constantes) ont eu raison de lalgorithme dapprentissage en labsence de contraintes sur les coefficients. Cela montre combien linspection des variables préalablement a lapprentissage est tres importante. Il aurait fallu exclure demblée ces variables a probleme. Nous continuons en létat néanmoins en espérant que les mécanismes de régularisation des régressions ridge et elasticnet nous sortiront daffaire.

Régression Ridge

Paramétrage et résultats de la régression. Pour `glmnet()`, la régression ridge correspond a ($\lambda > 0$) et ($\alpha = 0$). Lors de lappel de la fonction, nous indiquons bien ($\alpha = 0$) et nous laissons loutil déterminer une séquence de ($nlambda = 100$, paramétrable) valeurs de λ_i a explorer. Le mécanisme de détermination des extremums de la plage $(\lambda_{min}, \lambda_{max})$ est décrit dans un des articles fondateurs des auteurs du package (Friedman et al. (2010), section 2.5 pour la régression [3]). Lautre approche consiste a fixer nous-meme la liste des valeurs de λ a tester, soit parce que nous en avons une idée précise (ca reste difficile quand meme), soit parce que nous souhaitons comparer des solutions alternatives pour différentes valeurs de α.

```
#Régression Ridge
ridge2 <- glmnet(XTrain,yTrain,family="binomial",standardize=FALSE,alpha=0)
plot(ridge2,xvar="lambda")
```

A lissue de lapprentissage, nous disposons dun vecteur de coefficients β^i pour chaque λ_i. Plus λ_i augmente, plus la norme de β^i diminue. Tous les coefficients sont nuls lorsque $\lambda = \lambda_{max}$. Nous pouvons afficher une "Ridge path coefficients" permettant de juger de la dispersion des coefficients au regard de λ (Figure 5.1). Rappelons qua la différence de la régression Lasso, Ridge ne sait pas

3. Les corrélations avec la cible, le ratio nombre de variables/nombre dobservations, *la valeur de* α, la taille de léchantillon, le nombre de valeurs a explorer... entrent en jeu.

fixer sélectivement les coefficients a la valeur 0 et ne permet donc pas de réaliser une sélection de variables.

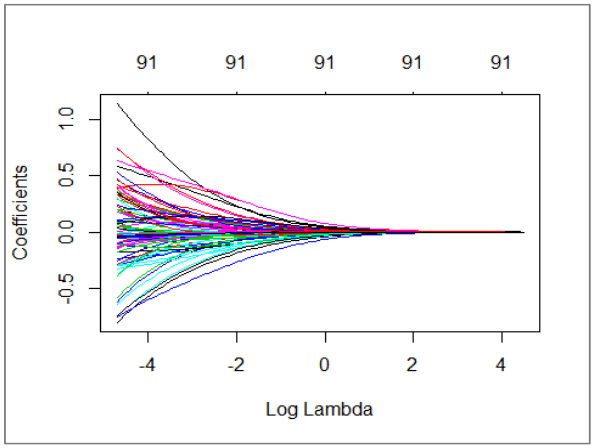

FIGURE 3.1 – Ridge path coefficients

Identification de la valeur "optimale" de λ. Nous avons les logarithmes des λ_i en abscisse de notre graphique. Ils varient de $\log(0.008982) = -4.7124$ a $\log(89.825) = 4.4978$. Une regle empirique consiste a choisir la valeur de λ a partir de laquelle les coefficients commencent "a se stabiliser". On se rend compte que la lecture en ce sens du graphique nest pas aisé (Figure 5.1). De plus, cette démarche ne nous assure pas de trouver la solution qui optimise les qualités prédictives du modele.

On lui préfere une démarche plus pragmatique visant a optimiser explicitement une mesure dévaluation des performances. Notre échantillon test devant jouer le role de juge impartial, il est hors de question quil intervienne dans ce processus. Et, malheureusement, la taille de notre échantillon dapprentissage est trop faible ($ntrain = 200$) pour que nous puissions le scinder en deux parties encore. La solution viable consiste a passer par la validation croisée, qui ne fait intervenir que léchantillon dapprentissage. Nous faisons appel a la fonction `cv.glmnet()`

```
#Optimisation du parametre lambda
set.seed(1)
cv.ridge2 <- cv.glmnet(XTrain,yTrain,family="binomial",type.measure="class",
                       nfolds=10,alpha=0,keep=TRUE)
```

Par rapport a `glmnet()`, de nouveaux parametres apparaissent :

- (`type.measure="class"`) indique que nous traitons dun probleme de classement et que le critere utilisé sera le taux derreur (misclassification error).
- (`nfolds = 10`) pour indiquer une validation croisée en 10 blocs (folds).
- (`keep = TRUE`) pour que lon conserve en mémoire les groupes de la validation croisée afin de pouvoir reconduire a lidentique lexpérimentation si daventure nous souhaitons la relancer avec dautres jeux de parametres. Nous reviendrons sur cette question ci-dessous.

Un graphique permet de mettre en relation les valeurs de $\log(\lambda)$ avec le taux derreur moyen en validation croisée (Figure 5.2, points rouges). Un intervalle de confiance est proposé, défini par ± 1 écart-type de lerreur en validation croisée.

```
plot(cv.ridge2)
```

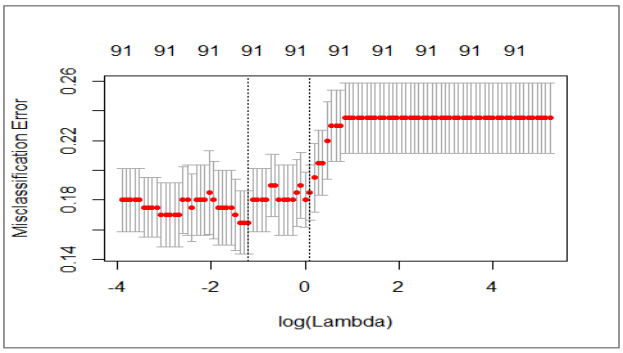

FIGURE 3.2 – Ridge–Taux d'erreur en validation croisée versus $\log(\lambda)$

Remarque 3.1.2. Subdivision en blocs des observations. La validation croisée subdivise les données en groupes pour réitérer les processus dapprentissage et test. Loption `keep` permet didentifier les blocs dappartenance de chaque individu et pouvoir ainsi répéter a lidentique la démarche pour dautres modeles. Cette forme dappariement rend plus puissant les comparaisons des performances, ce dont on ne se privera pas dans la suite de notre étude ci-dessous. Les indicatrices de blocs sont conservées dans le champ `$foldid`. Nous voyons ainsi que le premier individu a été affecté au bloc $n°4$, le second au $n°5$, le troisieme au $n°4$, etc.

```
#subidivision en folds (blocs) des données
print(cv.ridge2$foldid)
```

Revenons aux résultats de la validation croisée. Nous avons acces aux détails avec les propriétés de lobjet généré par la fonction. Nous affichons ci-dessous la suite de λ_i ; le taux derreur associé ; le nombre de coefficients non-nuls (qui névolue pas puisque nous utilisons une régression ridge).

```
#affichage
print(cbind(cv.ridge2$lambda,cv.ridge2$cvm,cv.ridge2$nzero))
```

Identifier visuellement les éléments importants dans cette liste nest pas aisé. Nous le faisons par le calcul. Tout dabord, nous affichons lerreur minimale.

```
#min de l'erreur en cross-validation
print(min(cv.ridge2$cvm))
```

```
## [1] 0.165
```

Puis nous cherchons le λ^* correspondant a cette erreur.

```
#lambda qui minimise l'erreur
print(cv.ridge2$lambda.min)
```

```
## [1] 0.2998318
```

Et nous calculons son logarithme.

```
#son logarithme
print(log(cv.ridge2$lambda.min))
```

```
## [1] -1.204534
```

Cette coordonnée est matérialisée par le premier trait pointillé (a gauche) dans le graphique de la validation croisée (Figure 5.2).

Regle de lécart-type. On percoit un second trait dans la Figure 5.2 (a droite). Il caractérise la plus grande valeur λ^{**} de λ telle que son erreur moyenne en validation croisée (point rouge) est inférieure a la borne haute de lintervalle de confiance de lerreur optimale (pour λ^{*}). Quel est son intéret ? Un peu comme le principe de parcimonie et la préférence a la simplicité des modeles, cette solution témoigne dune préférence a la régularisation : “ a performances similaires sur notre échantillon dapprentissage, on préfere la solution la plus fortement régularisée, la moins dépendante des données dapprentissage”.

Remarque 3.1.3. Cette “regle de lécart-type” est un héritage de la méthode dinduction darbres `CART` (Breiman et al., 1984) ou lon essaie de déterminer le plus petit arbre avec un niveau de performances satisfaisant.

Nous accédons a ce λ^{**} et a son logarithme avec les instructions suivantes.

```
#lambda le plus élevé en 1-se rule print(cv.ridge2$lambda.1se)
## [1] 1.102895

#son log print(log(cv.ridge2$lambda.1se))
## [1] 0.09793863
```

Inspection des coefficients β ***pour*** $\lambda = \lambda^{*}$. Nous avons acces au vecteur de coefficients de la régression pour λ fixé. Nous affichons ci-dessous les coefficients β^{*} pour$\lambda = \lambda^{*}$.

```
#coefficients de la régression pour le min
print(coef(cv.ridge2,s="lambda.min"))

## 124 x 1 sparse Matrix of class "dgCMatrix"
##
```

Les coefficients nuls `(.)` correspondent aux variables composées de 0.

Remarque 3.1.4. Nous ne le faisons pas ici, mais, les variables explicatives étant exprimées dans les memes unités, les trier selon la valeur absolue décroissante des coefficients permettrait de les hiérarchiser et distinguer celles qui sont les plus influentes dans la régression.

Prédiction sur léchantillon test. Nous pouvons produire plusieurs prédictions a partir des jeux de coefficients β correspondant a différentes valeurs de λ. Ci-dessous, nous appliquons les coefficients $\beta^{*}(\lambda^{*})$ et $\beta^{**}(\lambda^{**})$.

```
#prédiction
yr2 <- predict(cv.ridge2,XTest,s=c(cv.ridge2$lambda.min,cv.ridge2$lambda.1se),type="class")
```

Nous avons une matrice avec autant de colonnes que de valeurs de λ essayé. Le nombre de lignes correspond a `ntest`, leffectif de léchantillon test. Voici les 6 premieres lignes.

```
print(head(yr2))
```

Nous calculons les taux derreur pour les deux prédictions.

```
#erreur : lambda.min
print(sum(DTest$classe != yr2[,1])/nrow(DTest))
## [1] 0.1769932

#erreur : lambda.1se
```

```
print(sum(DTest$classe != yr2[,2])/nrow(DTest))
## [1] 0.2110738
```

Les deux font mieux que la prédiction systématique. Malgré les embuches, la régression ridge a su tirer parti des données. On note également que le modele avec λ^* est meilleur (17.69%). La préférence a la régularisation nest pas payante dans notre configuration.

3.1.4 Régression elasticnet

Régression elasticnet –Etude des coefficients. Pour la régression elasticnet, nous devons paramétrer ($\lambda > 0$ et $\alpha > 0$). Avec `glmnet()`, nous devons fixer la valeur de α et la fonction se charge de déterminer la plage de λ a explorer

Remarque 3.1.5. On peut spécifier explicitement la valeur ou la plage de valeurs de λ a essayer si nous le souhaitons.

```
#Régression Elasticnet
enet3 <- glmnet(XTrain,yTrain,family="binomial",standardize=FALSE,alpha=0.8)
plot(enet3,xvar="lambda")
```

Nous disposons dun graphique "Elasticnet path" (Figure 3.3)...

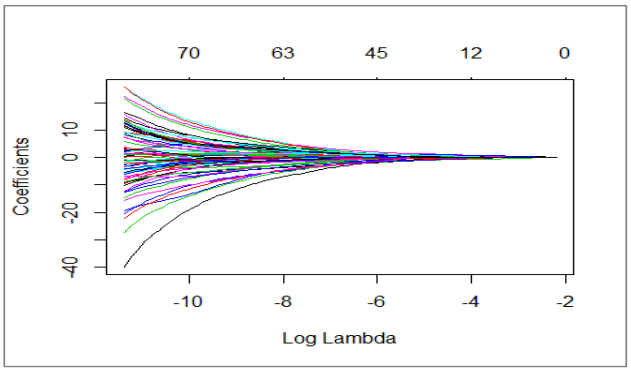

FIGURE 3.3 – Elasticnet path

... avec les λ_i suivants :

```
#liste des valeurs de lambda testes
print(enet3$lambda)
```

Lorsque nous affichons les λ_i avec le nombre de coefficients non-nuls, nous constatons quelasticnet, a la difference de ridge, procede bien a une selection de variables. Pour la premiere valeur de $\lambda_1 = 0.1228$, aucune variable nest selectionnee (tous les coefficients sont nuls, a lexclusion de la constante), pour la 2nde ($\lambda_2 = 0.1023$), 2 variables sont actives, ..., pour la 15eme ($\lambda_{15} = 0.03052465$), il y en a 7, etc.

```
#affichage - nombre de variables selectionnees vs. lambda (α = 0.8)
print(cbind(enet3$lambda,enet3$df))
```

Nous pouvons afficher le jeu de coefficients β^{15} correspondant a $\lambda_{15} = 0.03052465$.

```
#coefficients pour la 15e valeur de λ
```

```
print(enet3$beta[,15])
```

Si lon sen tient aux coefficients non-nuls, nous identifions les 7 variables selectionnees pour ($\alpha = 0.8$ et $\lambda = 0.03052465$) :

```
#coefficients non-nuls pour la 15e valeur de lambda
print(enet3$beta[abs(enet3$beta[,15])>0,15])
```

Optimisation de λ. Ici aussi, nous avons recours a la validation croisee pour detecter la valeur optimale de λ pour $\alpha = 0.8$.

```
#validation croisee
cv.enet3 <- cv.glmnet(XTrain,yTrain,family="binomial",type.measure="class",
nfolds= 10,alpha=0.8,foldid=cv.ridge2$foldid)
plot(cv.enet3)
```

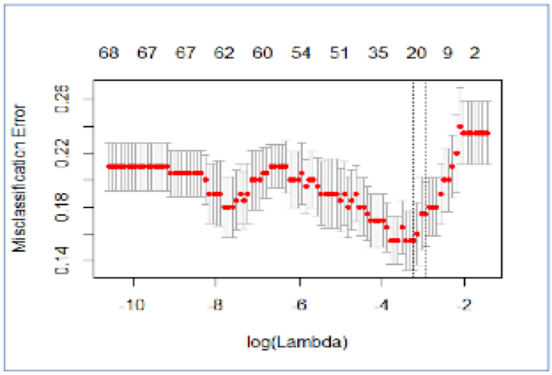

FIGURE 3.4 – Elasticnet - Taux d'erreur en validation croisee vs. log(λ)

$\lambda^* = 0.03926356$ minimise lerreur en validation croisee. Le taux derreur du modele applique sur lechantillon test est de 18.26%.

```
#lambda min
print(cv.enet3$lambda.min)

## [1] 0.03926356

#prediction
ye3 <- predict(cv.enet3,XTest,s=c(cv.enet3$lambda.min),type="class")
print(table(ye3))

## ye3 ## negative                positive
## 26202                4754

#taux d'erreur
print(sum(DTest$classe != ye3)/nrow(DTest))
```

```
## [1] 0.1826463
```

Optimisation conjointe de λ ***et*** α. Nous avons travaille a fixe dans cette section consacree a elasticnet ($\alpha = 0.8$). Une extension possible de letude serait de tenter doptimiser conjointement λ et α de maniere a obtenir le taux derreur le plus faible en validation croisee. La tache est simple, il suffit denglober dans une boucle sur α le code ci-dessus en veillant a utiliser des blocs identiques dans la validation croisee pour que les taux derreur soient directement comparables. Le resultat sest pourtant revele decevant en deploiement sur lechantillon test, parce qua force dacharnement sur lechantillon dapprentissage, meme en validation croisee, on finit par faire du surapprentissage. Le modele ń optimal ż avec ce processus devient trop specifique lorsquon multiplie les parametres a manipuler simultanement. Il faudrait introduire une forme de lissage dans lexploration des solutions pour eviter cet ecueil. Laffaire nest pas triviale.

3.2 Les librairies " tensorflow/keras"

Pour rappel, "Keras" est une surcouche qui permet dacceder relativement facilement aux fonctionnalites de "tensorflow", particulierement puissantes, mais particulierement esoteriques egalement (pour ne pas dire hermetiques).

3.2.1 Regression logistique sous forme de perceptron simple

La regression logistique proprement dite nexiste pas sous "keras". Nous passons par limplementation dun perceptron simple avec une fonction de transfert sigmoide et une fonction de perte correspondant a la log-vraisemblance. En definitive, notre processus dapprentissage est assimilable a celui de la regression logistique, avec un algorithme doptimisation specifique tout simplement.

Nous codons en binaire 0/1 la variable cible, puis nous construisons la structure de reseau.

```
#recoder la cible
y01Train <- ifelse(DTrain$classe=="positive",1,0)
#regression avec descente du gradient, sans coefficient de penalite
#librairie Keras
library(keras)
#initialiser la structure
kreg <- keras_model_sequential()
#ajouter une couche (entree -> sortie)
kreg %>% layer_dense(units=1,input_shape=c(ncol(XTrain)),activation="sigmoid")
#verification
print(summary(kreg))
```

Nous avons un perceptron simple, sans couche cachee. Le nombre de coefficients a estimer est 124 c.-a-d. $p = 123$ explicatives + la constante.
Linstruction `compile()` permet de specifier la fonction de perte a utiliser (loss) et lalgorithme (`optimizer = sgd` : descente de gradient stochastique). Le taux de reconnaissance (accuracy) sera utilise pour suivre levolution du processus dapprentissage.

```
#configuration de l'apprentissage
kreg %>% compile(
loss = "binary_crossentropy",
optimizer = "sgd",
```

```
metrics = "accuracy" )
```

La commande `fit()` lance lapprentissage proprement dit. Nous indiquons le nombre diterations sur la base complete (epochs).

```
#lancer les calculs
kreg %>% fit( x = XTrain,
y = y01Train,
epochs = 100 )
```

Deux graphiques de suivi de loptimisation apparaissent durant lapprentissage : lun pour la fonction de cout (loss), lautre pour la mesure de performance (metrics) (Figure 3.5).

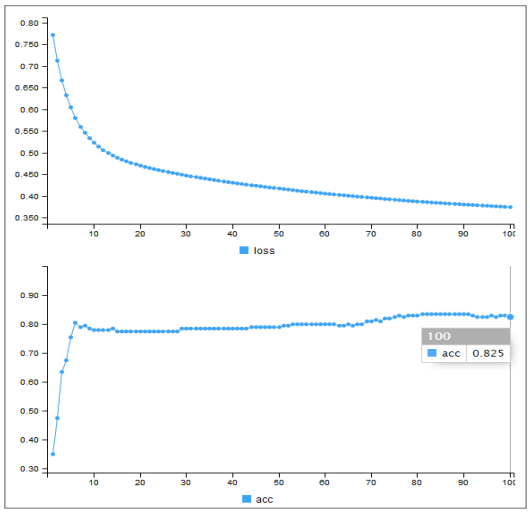

FIGURE 3.5 – Suivi du processus d’apprentissage - Package Keras

A lissue du processus, le taux derreur, calcule sur lechantillon dapprentissage, est de 17.5% (taux de reconnaissance = 82.5%). Nous verrons ce quil en est sur lechantillon test. Pour lheure, inspectons un peu les resultats. Nous recuperons le vecteur des poids synaptiques estimes, qui representent les coefficients de la regression logistique en fait. #recuperer poids.kreg <- get_weights(kreg)[[1]][,1]

Nous calculons le carre de la norme L2 des coefficients. A priori, avec ridge et elasticnet, nous devrions obtenir une valeur plus faible. Cest pour cela quon parle de ń shrinkage ż (retrecissement) dans la litterature. Notre valeur de reference pour la regression sans penalite est 2 = 3.192218 (Remarque : il y a une part daleatoire dans les calculs, il se peut que vous obteniez des resultats legerement differents). #afficher le carre de leur norme print(sum(poids.kreg$\hat{2}$)) ## [1] 3.192218 #histogramme des coefficients hist(poids.kreg) Une autre maniere de considerer les coefficients est dafficher leur histogramme de frequence. Nous devrions observer un retrecissement pour ridge, idem pour elasticnet avec, pour ce dernier, des coefficients nuls.

La prediction fournir les scores dappartenance aux classes. Il sagit de valeurs reelles comprises entre 0 et 1 puisque nous avons utilise une fonction de transfert sigmoide dans le reseau. Lhistogramme nous le confirme (Figure 7).

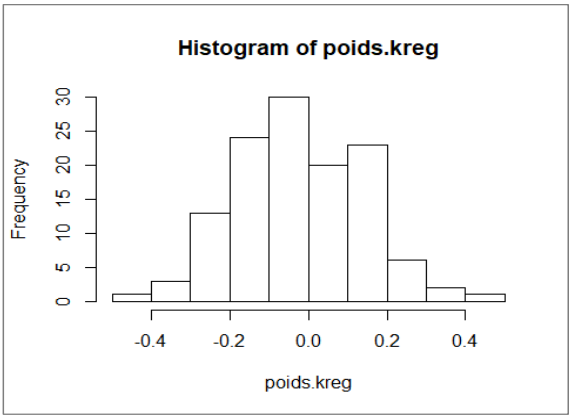

FIGURE 3.6 – Histogramme de distribution des coefficients - Regression sans penalites

#prediction -> score kscore <- kreg %>% predict(XTest) hist(kscore)

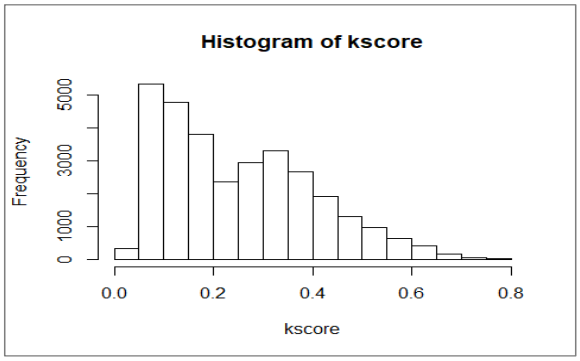

FIGURE 3.7 – Histogramme de distribution des scores - Echantillon test - Regression logistique

Il faut comparer le score a la valeur seuil 0.5 pour le convertir en prediction. Nous constatons quil ny a pas de prediction systematique dune des classes... #conversion en classe ypk <- ifelse(kscore > 0.5,"positive","negative") print(table(ypk)) ... et le taux derreur est de 19.68%. Contrairement a glm() et glmnet(), avec une regression non regularisee, sans precautions particulieres, loutil arrive a engranger de linformation utile pour produire un modele qui fait mieux que le classifieur par defaut. Lalgorithme est solide ! ## ypk ## negative positive ## 28711 2245 #taux d'erreur print(sum(DTest$classe != ypk)/nrow(DTest))
[1] 0.1968278 4.2 Regression Ridge sous ń Tensorflow / Keras ż La fonction de cout avec penalite ressemblerait a ceci sous Keras :

$$= [1.(+)log(1+(+))] + .2 + .002211 = 1$$

Jutilise le conditionnel parce que le parametrage est quand meme un peu ambigu sous Keras. On peut definir une fonction de cout global pour lapprentissage avec la commande compile(). En revanche

les coefficients de penalites sont introduites au niveau des couches avec linstruction layer_dense(). Nayant quune couche reliant la couche a la sortie dans notre reseau, on peut penser que notre equation decrit correctement la grandeur a optimiser. Je suis autrement plus sceptique si nous en avions plusieurs et quil nous venait la fantaisie dintroduire des coefficients de penalite differents dune couche a lautre. Pour linstant, nous sommes dans un schema simple, nous specifions 2 = 0.05 avec loption kernel_regularizer2 de la commande layer_dense().

#initialiser la structure kridge <- keras_model_sequential() #ajouter une couche (entree -> sortie) kridge %>% layer_dense(units=1,input_shape=c(ncol(XTrain)),activation="sigmoid", kernel_regularizer = regularizer_l2(0.05)) #configuration de l'apprentissage kridge %>% compile(loss = "binary_crossentropy", optimizer = "sgd", metrics = "accuracy") #lancer les calculs kridge %>% fit(x = XTrain, y = y01Train, epochs = 100) #recuperer les poids poids.kridge <- get_weights(kridge)[[1]][,1]
#afficher le carre de leur norme
print(sum(poids.kridge2̂))
[1] 1.213311

Le ń shrinkage ż (retrecissement) porte bien son nom, le carre de la norme a ete reduit a 2 = 1.213311 contre 2 = 3.192218. Letendue des coefficients est reduite 2=0.05 2 2 comme nous le constatons avec lhistogramme de distribution des coefficients (Figure 8). #histogramme des coefficients hist(poids.kridge)

2 La documentation de Keras nest pas tres claire non plus a ce sujet. Il y a deux types de regularisations possibles sur un layer_dense() : "kernel", ou lon semble chercher a harmoniser les poids synaptiques, cela correspond a ce que nous souhaitons faire ; activation, ou il sagirait plutot dharmoniser les sorties des neurones de la meme couche (sil y en a plusieurs). Il nest nullement fait mention de la regression ridge ou lasso dans la documentation... actuelle, pas lancienne. En cherchant attentivement sur le web, jai eu acces a la documentation de Keras 1.2.2 (https ://faroit.github.io/keras-docs/1.2.2/regularizers/), ou une autre option etait proposee, weight regularizer. Elle semble obsolete aujourdhui. On ne sait pas tres bien sil y a une correspondance, et jusqua quel point, entre cette derniere et kernel regularizer que nous utilisons (Version 2.1.5, Mai 2018).

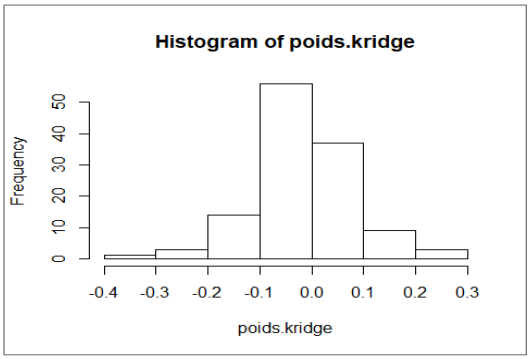

FIGURE 3.8 – Histogramme de distribution des coefficients - Regression ridge

Mais est-ce a juste titre ? Voyons si les predictions sont meilleures en deploiement.

#prediction -> score
ridge_score <- kridge %>% predict(XTest)
#conversion en classe
ypkridge <- ifelse(ridge_score > 0.5,"positive","negative")
print(table(ypkridge))

```
#taux d'erreur
print(sum(DTest$classe != ypkridge)/nrow(DTest))
## [1] 0.2168239
```

Non finalement. Le modele nest pas meilleur. Remarque : Bien sur, a linstar du travail effectue a laide de ń glmnet ż, nous pouvons essayer doptimiser 2 en validation croisee. Je nai pas trouve doutil dedie sous Keras (on nen parle pas dans la documentation), mais on peut y palier sans difficulte avec un peu de programmation. 4.3 Regression elasticnet sous ń tensorflow / keras ż Nous specifions 1 = 0.01 et 2 = 0.005 pour la regression elasticnet. Nous donnons plus dimportance a 1 pour avoir un comportement proche de Lasso. ## ypkridge

```
## negative positive
## 30012 944
#initialiser la structure
kenet <- keras_model_sequential()
#ajouter une couche (entree -> sortie) – elasticnet
kenet %>% layer_dense(units=1,input_shape=c(ncol(XTrain)),activation="sigmoid",
kernel_regularizer = regularizer_l1_l2(l1=0.01,l2=0.005))
#configuration de l'apprentissage
kenet %>% compile(
loss = "binary_crossentropy", optimizer = "sgd",
metrics = "accuracy" )

    #lancer les calculs
kenet %>% fit( x = XTrain,
y = y01Train, epochs = 100 )
#recuperer les poids
poids.kenet <- get_weights(kenet)[[1]][,1]
#afficher le carre de leur norme
print(sum(poids.kenet2̂)) ## [1] 1.562286
#histogramme des coefficients
hist(poids.kenet)
```

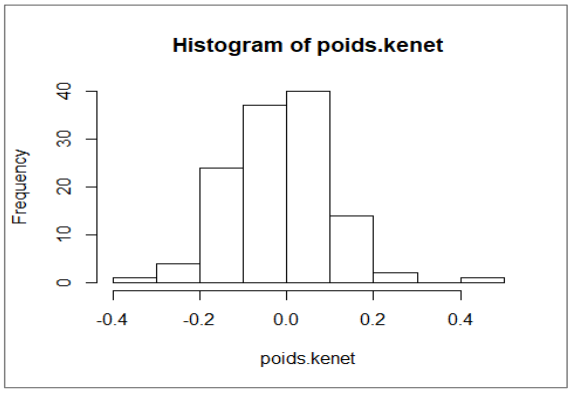

FIGURE 3.9 – Distribution des coefficients - Regression elasticnet

```
    #prediction -> score
enet_score <- kenet %>% predict(XTest)
#conversion en classe
ypkenet <- ifelse(enet_score > 0.5,"positive","negative")
```

```
print(table(ypkenet))
## ypkenet
## negative positive
## 29172 1784

   #taux d'erreur
print(sum(DTest$classe != ypkenet)/nrow(DTest))
## [1] 0.2038377
```

Avec le parametrage specifie (1 = 0.01 et 2 = 0.005), le taux derreur est intermediaire entre lapproche ridge (2 = 0.05) et le modele sans penalite (1 = 0 et 2 = 0).

3.3 Recapitulatif des resultats

Nous avons beaucoup tente avec de multiples outils dans ce tutoriel. Voici un tableau recapitulatif des performances en test selon les packages et parametrages utilises, sachant que le taux derreur du classifieur par defaut est 24%.

Outil	Sans régularisation	Ridge	Elasticnet
glm (« stats »)	Echec	-	-
« glmnet »	31.5%	17.7% ($\lambda = 0.29$)	18.3% ($\lambda = 0.04$, $\alpha = 0.8$)
Tensorflow / keras	19.6%	21.7% ($\lambda_2 = 0.05$)	20.4% ($\lambda_1 = 0.01$, $\lambda_2 = 0.005$)

FIGURE 3.10 – Tableau recapitulatif des performances en test selon les packages et parametrages utilises

Les ecarts ne sont absolument pas negligeables sur un echantillon test de 30956 observations. Manifestement, les parametres pesent fortement sur les resultats. Les outils tels que ńglmnetż qui proposent des solutions cles en main pour determiner semi- automatiquement les bonnes plages de valeurs des parametres de regularisation se revelent decisifs dans ce contexte.

3.4 Conclusion

Lobjectif de ce tutoriel etait de montrer la mise en uvre des regressions ridge et elasticnet sous R, un precedent document etant consacre a la regression lasso sous Python. Les outils etudies nous ont permis de mener a bien les taches que nous nous sommes assignees. Nous nous sommes focalises sur la manipulation des coefficients de penalites qui pesent directement sur les proprietes de regularisation des approches.

Mais letude de la documentation montre que les fonctions sont en realite bardees de tous un tas de parametres censes peser sur la qualite et vitesse de convergence. Malgre une lecture assidue, on a du mal a cerner concretement leur influence sur la qualite predictive des modeles. Et je nai pas vu de tutoriels, meme en anglais, qui en fassent le tour et explicitent leurs roles sur des exemples concrets (un peu quand meme dans les annexes du site de ń glmnet ż). Il reste du travail encore de ce cote-la...

References
(HASTIE & QIAN, 2016) Hastie T., Qian J., ń Glmnet Vignette ż, Septembre 2016 ; https ://web.stanford.edu/ hastie/glmnet/glmnet_beta.html

Friedman J., Hastie T., Tibshirani R., Simon N., Narasimhan B., Qian J., ń glmnet : Lasso and Elastic-Net Regularized Generalized Linear Models ż, version 2.0-16, Avril 2018 ; https ://cran.r-project.org/package=glmnet

(FRIEDMAN et al., 2010) Friedman J., Hastie T., Tibshirani R., ń Regularization Paths for Generalized Linear Models via Coordinate Descent ż, in Journal of Statistical Software, Volumne 33, Issue 1, January 2010.

(RAK, 2018) ń Ridge, Lasso, Elasticnet Diapos ż, Mai 2018 ; http ://tutoriels-data- mining.blogspot.fr/2018/05/ridge-lasso-elasticnet.html

Keras Documentation. ń Keras : The Python Deep Learning Library ż ; https ://keras.io/

J.J. Allaire, F. Chollet, RStudio, Google, Y. Tang, D. Falbel, W. Van Der Bijl, M. Studer, ń keras : R Interface to 'Keras' ż, version 2.1.6, Avril 2018 ; https ://cran.r-project.org/package=keras

3.5 Devoir

3.6 Sélection de modèle par pénalisation Ridge

3.6.1 Comportement des coefficients

```
library(MASS)
mod.ridge=lm.ridge(lpsa∼.,data=Prostate.app, lambda=seq(0,20,0.1))
par(mfrow=c(1,1))
plot(mod.ridge)

# évolution des coefficients
matplot(t(mod.ridge$coef),lty=1 :3,type="l",col=1 :10)
legend("top",legend=rownames(mod.ridge$coef),col=1 :10,lty=1 :3)
```

3.6.2 Pénalisation optimale par validation croisée

```
elect(mod.ridge)
# noter la valeur puis estimer
mod.ridgeopt=lm.ridge(lpsa∼.,data=Prostate.app, lambda=10.4)
```

3.6.3 Prévision et erreur d'apprentissage

Pour des raisons obscures, la fonction `predict.ridgelm` n'existe pas, il faut calculer les valeurs prédites à partir des coefficients. `#Coefficients du modèle sélectionné :`

```
coeff=coef(mod.ridgeopt)
#On crée des vecteurs pour les variables qualitatives
svi0.app=1*c(Prostate.app$svi==0)
svi1.app=1-svi0.app gl6.app=1*c(Prostate.app$gleason==6)
gl7.app=1*c(Prostate.app$gleason==7)
gl8.app=1*c(Prostate.app$gleason==8)
gl9.app=1*c(Prostate.app$gleason==9)
```

```
#variables quantitatives
lcavol.app=Prostate.app$lcavol
lweight.app=Prostate.app$lweight
age.app=Prostate.app$age
lbph.app=Prostate.app$lbph
lcp.app=Prostate.app$lcp
pgg45.app=Prostate.app$pgg45
#Calcul des valeurs prédites
fit.rid=rep(coeff[1],napp)+coeff[2]*lcavol.app+ coeff[3]*lweight.app+coeff[4]*age.app+
coeff[5]*lbph.app+coeff[6]*svi0.app- coeff[6]*svi1.app+coeff[7]*lcp.app+ coeff[8]*gl6.app+coeff[9
coeff[10]*gl8.app-(coeff[8]+coeff[9]+ coeff[10])*gl9.app+coeff[11]*pgg45.app
#Tracé des valeurs prédites en fonctions des valeurs observées
plot(Prostate.app$lpsa,fit.rid)
abline(0,1)
#Calcul et tracé des résidus
res.rid=fit.rid-Prostate.app[,"lpsa"]
plot.res(fit.rid,res.rid,titre="")
#Erreur d'apprentissage mean(res.rid**2)
#0.478
```

3.6.4 Prévision sur l'échantillon test

```
#Variables qualitatives
vi0.t=1*c(Prostate.test$svi==0)
svi1.t=1-svi0.t gl6.t=1*c(Prostate.test$gleason==6)
gl7.t=1*c(Prostate.test$gleason==7)
gl8.t=1*c(Prostate.test$gleason==8)
gl9.t=1*c(Prostate.test$gleason==9)
#Variables quantitatives
lcavol.t=Prostate.test$lcavol
lweight.t=Prostate.test$lweight
age.t=Prostate.test$age
lbph.t=Prostate.test$lbph
lcp.t=Prostate.test$lcp
pgg45.t=Prostate.test$pgg45


prediction=rep(coeff[1],ntest)+coeff[2]* lcavol.t+coeff[3]*lweight.t+coeff[4]*age.t+
coeff[5]*lbph.t+coeff[6]*svi0.t-coeff[6]*svi1.t+ coeff[7]*lcp.t+coeff[8]*gl6.t+coeff[9]*gl7.t+
coeff[10]*gl8.t-(coeff[8]+coeff[9]+coeff[10])* gl9.t+coeff[11]*pgg45.t
#Erreur sur l'échantillon test
mean((Prostate.test[,"lpsa"]-prediction)^ 2) # 0.424
```

Remarque 3.6.1. L'erreur d'apprentissage est légèrement plus élévée que pour le modèle linéaire sans pénalisation (c'est normal !) l'erreur de test est plus faible. Les perfor- mances sont comparables sur l'échantillon test au modèle sélectionné par le critère BIC. En terme d'interprétation, les modèles sélectionnés par AIC et BIC sont préférables.

3.7 Sélection de modèle par pénalisation Lasso

Les résultats sont obtenus avec la librairie lasso2 ou avec la librairie glmnet

3.7.1 Librairie Lasso2

3.7.2 Construction du modèle

```
library(lasso2)
l1c.P <- l1ce(lpsa∼., Prostate.app, bound=(1 :100)/100,absolute.t=FALSE)
```

Remarque 3.7.1. La borne est ici relative, elle correspond à une certaine proportion de la norme L^1 du vecteur des coefficients des moindres carrés. Une borne égale à 1 correspond donc à l'absence de pénalité, on retrouve l'estimateur des moindres carrés.

3.7.3 Visualisation des coefficients

```
coefficients=coef(l1c.P)
plot(l1c.P,col=1 :11,lty=1 :3,type="l")
legend("topleft",legend=colnames(coefficients), col=1 :11,lty=1 :3)

#On supprime le terme constant
penalite_relative=c(1 :100)/100
matplot(penalite_relative,coefficients[,-1], lty=1 :3,type="l",col=1 :10)
legend("topleft",legend= colnames(coefficients[,-1]),col=1 :10,lty=1 :3)
```

3.7.4 Sélection de la pénalité par validation croisée

```
vc=gcv(l1c.P)
crit.vc=vc[,"gcv"]
bound_opt=vc[which.min(crit.vc),"rel.bound"]
#0.95

l1c.opt <- l1ce(lpsa ∼ ., Prostate.app, bound=bound_opt, absolute.t=FALSE)
coef=coef(l1c.opt)
```

3.7.5 Erreur d'apprentissage

```
#erreur apprentissage
fit=fitted(l1c.opt)
mean((fit-Prostate.app[,"lpsa"])^2)

#0.46
```

3.7.6 Erreur sur l'échantillon test

```
prediction=predict(l1c.opt,newdata=Prostate.test)
mean((prediction-Prostate.test[,"lpsa"])^ 2)
#0.44
```

3.7.7 Librairie glmnet

L'utilisation de la librairie glmnet fournit des résultats plus rapides, ce qui peut s'avérer important pour des données de grande dimension. Par contre, on ne peut pas traiter à priori des variables

qualitatives. Nous allons donc devoir créer des vecteurs avec des variables indicatrices des diverses modalités pour les variables qualitatives. Nous ne prendrons pas en compte les contrastes.

3.7.8 Mise en forme des variables

```
on construit une matrice xx.app d'apprentissage et #xx.test de test
data(Prostate)
Prostate.app=Prostate[-ind.test,] Prostate.test=Prostate[c(ind.test),]

x.app=Prostate.app[,-9]
y.app=Prostate.app[,9]
x.app=as.matrix(x.app)

# construction des indicatrices
svi1.app=1*c(Prostate.app$svi==1)
gl7.app=1*c(Prostate.app$gleason==7)
gl8.app=1*c(Prostate.app$gleason==8)
gl9.app=1*c(Prostate.app$gleason==9)

#matrice avec les vecteurs indicatrices
xx.app=matrix(0,ncol=10,nrow=nrow(x.app)) xx.app[,1 :6]=x.app[,1 :6]
xx.app[,7 :9]=cbind(gl7.app,gl8.app,gl9.app)
xx.app[,10]=x.app[,8]

#on nomme les colonnes avec le noms des variables
colnames(xx.app)=c("lcavol","lweight", "age", "lbph","svi1","lcp","gl7","gl8","gl9","pgg45")
#on fait de meme pour l'echantillon test
x.test=Prostate.test[,-9]
y.test=Prostate.test[,9]
x.test=as.matrix(x.test)
svi1.test=1*c(Prostate.test$svi==1)
gl7.test=1*c(Prostate.test$gleason==7)
gl8.test=1*c(Prostate.test$gleason==8)
gl9.test=1*c(Prostate.test$gleason==9)

#on construit une matrice avec les vecteurs indicatrices
xx.test=matrix(0,ncol=10,nrow=nrow(x.test))
xx.test[,1 :6]=x.test[,1 :6]
xx.test[,7 :9]=cbind(gl7.test,gl8.test,gl9.test)
xx.test[,10]=x.test[,8]
colnames(xx.test)=colnames(xx.app)
```

3.7.9 Construction du modèle

```
library(glmnet)
out.lasso = glmnet(xx.app,y.app)
l=length(out.lasso$lambda)
b=coef(out.lasso)[-1,1 :l]
```

3.7.10 Visualisation des coefficients

```
# chemin de régularisation du lasso
matplot(t(as.matrix(out.lasso$beta)),type="l", col=1 :10,lty=1 :3)
legend("topleft",legend=colnames(xx.app), col=1 :10,lty=1 :3)
title("Lasso")
```

3.7.11 Sélection de la pénalité par validation croisée

```
y.app=as.matrix(y.app)
a=cv.glmnet(xx.app,y.app)
#le resultat est dans
lambda.opt=a$lambda.min
app=glmnet(xx.app,y.app,lambda=lambda.opt)
```

3.7.12 Erreur d'apprentissage

```
appr=predict(app,newx=xx.app)
mean((appr-Prostate.app[,9])^2)
#0.48
```

3.7.13 Erreur sur l'échantillon test

```
pred=predict(app,newx=xx.test)
mean((pred-Prostate.test[,9])^2)
#0.39
```

Remarque 3.7.2. On n'obtient pas exactement le meme modèle qu'avec Lasso2. La procédure de validation croisée conduit ici à un modèle plus parcimonieux.

3.7.14 Elastic Net

```
#on peut jouer avec le paramètre alpha, de glmnet
out.elnet <- glmnet(xx.app,y.app,alpha=0.5)
a.elnet=cv.glmnet(xx.app,y.app,alpha=0.5)
lambda.opt=a.elnet$lambda.min
app=glmnet(xx.app,y.app,lambda=lambda.opt)

#erreur apprentissage
app.elnet=predict(a.elnet,newx=xx.app) mean((app.elnet-Prostate.app[,9])^2)
#0.60
#erreur de prédiction
predi.elnet=predict(a.elnet,newx=xx.test)
mean((predi.elnet-Prostate.test[,9])^2)
#0.37
```

3.8 Sélection de modèle par projection sur composantes orthogonales

3.8.1 Régression PLS

```
data(Prostate)
Prostate$svi=as.factor(Prostate$svi)
Prostate$gleason=as.factor(Prostate$gleason)
Prostate.app=Prostate[-ind.test,]
Prostate.test=Prostate[c(ind.test),]
library(pls)

#nombre optimal de composantes par validation croisée
simpls= mvr(lpsa∼.,data=Prostate.app, ncomp=10, validation="CV", method="simpls")
summary(simpls)

#graphique
plot(simpls)
abline(0,1)

#noter le nombre optimal de composantes
plot(simpls,"val")

#Avec leave-one-out
simplsloo= mvr(lpsa∼.,data=Prostate.app, ncomp=10, validation="LOO", method="simpls")
summary(simplsloo)

#On sélectionne 6 composantes
#Calcul des prévisions
predapp.pcr=predict(simpls,ncomp=6)
resapp.pcr=predapp.pcr-Prostate.app$lpsa

#Erreur d'apprentissage
mean(resapp.pcr**2)
#0.47

#Valeurs prédites sur l'échantillon test
pred.pls=predict(simpls,newdata=Prostate.test, ncomp=6)

#Erreur de test
mean((pred.pls-Prostate.test[,"lpsa"])**2)
#0.46
```

3.8.2 Régression sur composantes principales

```
mod.pcr = pcr(lpsa∼.,data=Prostate.app, ncomp=9, validation="CV")
summary(mod.pcr)

# noter le nombre optimal
plot(mod.pcr,"val")
#On sélectionne 8 composantes
```

```
#Calcul des prévisions
predapp.pls=predict(mod.pcr,ncomp=8)
resapp.pls=predapp.pls-Prostate.app$lpsa

#Erreur d'apprentissage
mean(resapp.pls**2)
#0.47

#Valeurs prédites sur l'échantillon test
pred.pcr=predict(mod.pcr,newdata=Prostate.test, ncomp=8)

#Erreur de test
mean((pred.pcr-Prostate.test[,"lpsa"])**2) #0.44
```

Remarque 3.8.1. Il peut arriver que la régression sur composantes principales ne soit pas adaptée, si les premières composantes principales trouvées n'ont que peu de rapport avec la variable Y . Le nombre de composantes PCR sélectionnées est ici égal à 8 (on ne réduit pas beaucoup la complexité par rapport au modèle initial). Avec la régression PLS, on sélectionne 6 composantes. Dans les 2 cas, on n'a pas beaucoup réduit la complexité des modèles par rapport au modèle linéaire.

Chapitre 4

Pratique de la SVM

Nous allons exposé un tutoriel de la pratique des régressions ridge et elasticnet sous la logiciel `R` via les packages `glmnet` et `tensorflow/keras`. Ce tutoriel fait suite au support de cours consacré a la régression régularisée.

Nous nous situons dans le cadre de la régression logistique avec une variable cible qualitative binaire. Les propriétés de régularisation de ridge et elasticnet devraient se révéler décisives. Encore faut-il savoir/pouvoir déterminer les valeurs adéquates des paramètres de ces algorithmes. Ils pèsent fortement sur la qualité des résultats.

Nous verrons comment faire avec les outils a notre disposition. Nous utiliserons les packages `glmnet` et `tensorflow/keras`. Il faut s'y référer notamment pour la partie installation qui n'est pas triviale.

4.1 Données et régression logistique `glm()`

4.1.1 Base de données "adult"

	Mort.à	Années.de.carrière	Nombre.de.films	Prénom	Nom	Date.du.décès
1	93	66	211	Michel	Galabru	04-01-2016
2	53	25	58	André	Raimbourg	23-09-1970
3	72	48	98	Jean	Gabin	15-10-1976
4	68	37	140	Louis	De Funès	27-01-1983
5	68	31	74	Lino	Ventura	22-10-1987
6	53	32	81	Jacques	Villeret	28-01-2005

4.2 Devoir

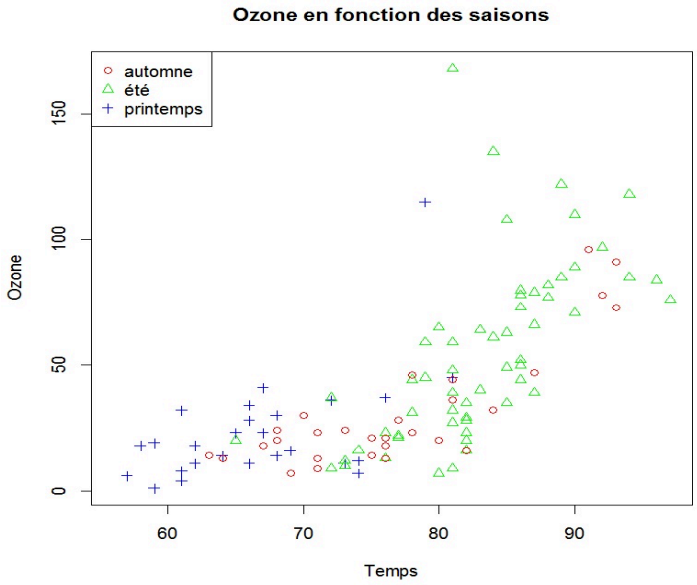

FIGURE 4.1 – Ozone en fonction des saisons

Chapitre 5

Analyse de données fonctionnelles (FDA) & Sélection de variables

Nous allons exposé un tutoriel de la pratique des régressions ridge et elasticnet sous la logiciel `R` via les packages `glmnet` et `tensorflow/keras`. Ce tutoriel fait suite au support de cours consacré a la régression régularisée.

Nous nous situons dans le cadre de la régression logistique avec une variable cible qualitative binaire. Les propriétés de régularisation de ridge et elasticnet devraient se révéler décisives. Encore faut-il savoir/pouvoir déterminer les valeurs adéquates des paramètres de ces algorithmes. Ils pèsent fortement sur la qualité des résultats.

Nous verrons comment faire avec les outils a notre disposition. Nous utiliserons les packages `glmnet` et `tensorflow/keras`. Il faut s'y référer notamment pour la partie installation qui n'est pas triviale.

5.1 Données et régression logistique `glm()`

5.1.1 Base de données "adult"

Nos données sont dérivées de la base "Adult Data Set" accessible sur le serveur UCI [1]. Elle a été retravaillée puis publiée sur le site LIBSVM [2] : les variables quantitatives ont été discrétisées, puis transformées en variables indicatrices via un codage disjonctif complet ; les variables catégorielles ont également été binarisées via le meme procédé. En définitive, nous disposons de $p = 123$ variables explicatives, toutes binaires. La variable cible est également binaire, définie dans *positive, negative.*

Nous avons réservé $ntrain = 200$ observations (extraits aléatoirement de la base d'apprentissage de "a1a") pour la modélisation, $ntest = 30956$ pour l'évaluation.

1. https ://archive.ics.uci.edu/ml/datasets/adult
2. https ://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html ; base "a1a"

5.1.2 Importation des bases d'apprentissage et de test – Quelques vérifications

Echantillon d'apprentissage. Nous importons l'échantillon d'apprentissage dans un premier temps.

```
#changer le dossier courant
setwd("... votre dossier ...")
#charger les données d'apprentissage
DTrain <- read.table("adult_train.txt",sep=’’ t’’,header=TRUE) print(dim(DTrain))
## [1] 200 124
```

Nous vérifions la répartition des classes. Le modèle par défaut consisterait a prédire systématiquement la classe majoritaire *negative*.

Répartition de la classe

```
print(prop.table(table(DTrain$classe)))
## ## negative           positive
## 0.765            0.235
```

Nous plaçons les variables explicatives et la variable cible dans deux structures distinctes, une matrice pour les premières, un vecteur pour la seconde.

```
#typage
XTrain <- as.matrix(DTrain[,-1])
yTrain <- as.matrix(DTrain[,1])
```

Notons une particularité qui ne va certainement pas faciliter le processus de modélisation : 32 colonnes sont composées de la valeur constante 0. Logiquement, il faudrait exclure d'emblée ces variables.

Elles sont sources de problèmes et ne peuvent en rien contribuer a la qualité prédictive des modèles. Dans notre cas, nous les conservons quand-meme pour corser l'affaire et voir justement comment se comporteront les différentes techniques et outils que nous examinerons dans ce tutoriel.

```
#particularité - variables remplies de 0 (somme de la colonne = 0)
print(length(which(colSums(XTrain)==0)))
## [1] 32
```

Échantillon test. Nous chargeons l'échantillon test dans un second temps. Il est composé de $ntest = 30956$ observations.

```
#charger les données test
DTest <- read.table("adult_test.txt",sep="",header=TRUE) print(dim(DTest))
## [1] 30956      124


#répartition des classes
print(prop.table(table(DTest$classe)))
## ## negative     positive
## 0.759465        0.240535
```

La répartition des classes est très similaire a celle de l'échantillon d'apprentissage, heureusement. Avec le modèle par défaut, prédiction systématique de la classe majoritaire *négative*, le taux d'erreur serait de 24.05%. Il s'agit de faire mieux avec les différentes variantes de la régression logistique que nous mettrons en uvre dans ce qui suit.

Nous placons également la matrice des descripteurs dans une structure dédiée.

```
# matrice en test
XTest <- as.matrix(DTest[,-1])
```

Régression logistique avec glm()

La fonction glm() du package stats, chargé automatiquement au démarrage, est l'outil privilégié pour la régression logistique sous R. Nous l'appliquons a l'échantillon d'apprentissage. Un message d'avertissement indiquant la non convergence de l'algorithme apparait. Ce n'est pas bon signe.

```
#rég. logistique usuelle
reg1 <- glm(classe ∼., data = DTrain, family = "binomial")
## Warning : glm.fit : algorithm did not converge
## Warning : glm.fit : fitted probabilities numerically 0 or 1 occurred
```

Affichons quand-meme les résultats par acquit de conscience. `print(summary(reg1))`

Le modèle est inutilisable. Certains coefficients n'ont pas été estimés et correspondent a la valeur NA (Not Available). Nous ne pouvons ni les interpréter, ni les déployer sur des individus supplémentaires. De fait, nous n'avons meme pas essayé d'appliquer le modèle sur l'échantillon test pour en mesurer les performances.

5.1.3 La librairie glmnet

La librairie glmnet a été développée par des grands noms de la statistique (on a le vertige rien qu'en lisant la liste des personnalités associées au package). Elle est efficace (rapidité des calculs, qualité des résultats), et surtout, elle propose toute une panoplie d'outils qui se révèlent précieux dans l'analyse exploratoire, lors de la recherche des combinaisons adéquates des paramètres λ et α. Cet aspect est d'autant plus important que, nous ne verrons dans ce tutoriel, le gap de performances entre les modèles selon les valeurs attribuées a ces paramètres peut etre important.

Pour rappel, la fonction a optimiser en régression logistique régularisée s'écrit (voir Hastie & Qian, 2016) :

$$J = \left[\frac{1}{ntrain}\sum_{i=1}^{ntrain} y_i\left(\beta_0 + x_i^t\beta\right) - \log\left(1 + \mathrm{e}^{\beta_0 + x_i^t\beta}\right)\right] + \lambda\left[\frac{1-\alpha}{2}||\beta||_2^2 + \alpha||\beta||_1\right]$$

où β_0 est la constante, β est le vecteur des coefficients appliqués aux variables. On observe que la constante de la régression n'est pas concernée par la régularisation.

λ est le coefficient de pénalité ; α permet d'arbitrer entre la contrainte portant sur les normes L^2 (ridge, $\alpha = 0$) et L^1 (lasso, $\alpha = 1$) des coefficients.

Régression logistique non-régularisée

Après avoir installé glmnet (a faire une seule fois), nous la chargeons (library) ...

```
#librairie glmnet
library(glmnet)
## Loading required package : Matrix ## Loading required package : foreach ## Loaded
glmnet 2.0-16
```

... puis nous lancons la régression logistique sans paramètre de régularisation ($\lambda = 0$).

Remarque 5.1.1. Nous désactivons la standardisation (standardize = FALSE) des variables explicatives parce que nous savons qu'elles sont toutes binaires, définies de facto sur la meme échelle. Sur une base quelconque, en l'absence d'informations précises sur les variables, il est plus prudent d'activer l'option.

```
#regression
reg2 <- glmnet(XTrain,yTrain,family="binomial",standardize=FALSE,lambda=0) print(reg2)
```

Tres curieusement, de par lalgorithme doptimisation utilise, le processus semble converger malgre tout contrairement a `glm()` de `stats`. La valeur zero est attribuee aux coefficients associes aux colonnes de constante. Ces variables sont inactives pour la prediction.

Nous appliquons le modele sur lechantillon test avec `predict()`.

```
# prediction
yp2 <- predict(reg2,XTest,type="class",s=c(0)) print(table(yp2))
```

```
## yp2
## negative      positive
## 19568           11388
```

Les predictions sont reparties entre les deux classes, il ny a pas de prediction systematique...

```
#taux d'erreur
print(sum(DTest$classe != yp2)/nrow(DTest))
```

```
   ## [1] 0.3150924
```

... mais le modele fait pire que la prediction par defaut avec un taux derreur de 31.5%.

Conclusion. Manifestement, les difficultes (le ratio $p/ntrain$ eleve et, surtout, les variables constituees de constantes) ont eu raison de lalgorithme dapprentissage en labsence de contraintes sur les coefficients. Cela montre combien linspection des variables prealablement a lapprentissage est tres importante. Il aurait fallu exclure demblee ces variables a probleme. Nous continuons en letat neanmoins en esperant que les mecanismes de regularisation des regressions ridge et elasticnet nous sortiront daffaire.

Regression Ridge

Parametrage et resultats de la regression. Pour `glmnet()`, la regression ridge correspond a ($\lambda > 0$) et ($\alpha = 0$). Lors de lappel de la fonction, nous indiquons bien ($\alpha = 0$) et nous laissons loutil determiner une sequence de ($nlambda = 100$, parametrable) valeurs de λ_i a explorer. Le mecanisme de determination des extremums de la plage $(\lambda_{min}, \lambda_{max})$ est decrit dans un des articles fondateurs des auteurs du package (Friedman et al. (2010), section 2.5 pour la regression [3]). Lautre approche consiste a fixer nous-meme la liste des valeurs de λ a tester, soit parce que nous en avons une idee precise (ca reste difficile quand meme), soit parce que nous souhaitons comparer des solutions alternatives pour differentes valeurs de α.

```
#Regression Ridge
ridge2 <- glmnet(XTrain,yTrain,family="binomial",standardize=FALSE,alpha=0)
plot(ridge2,xvar="lambda")
```

A lissue de lapprentissage, nous disposons dun vecteur de coefficients β^i pour chaque λ_i. Plus λ_i augmente, plus la norme de β^i diminue. Tous les coefficients sont nuls lorsque $\lambda = \lambda_{max}$. Nous pouvons afficher une "Ridge path coefficients" permettant de juger de la dispersion des coefficients au regard de λ (Figure 5.1). Rappelons qua la difference de la regression Lasso, Ridge ne sait pas

3. Les correlations avec la cible, le ratio nombre de variables/nombre dobservations, *la valeur de* α, la taille de lechantillon, le nombre de valeurs a explorer... entrent en jeu.

fixer selectivement les coefficients a la valeur 0 et ne permet donc pas de realiser une selection de variables.

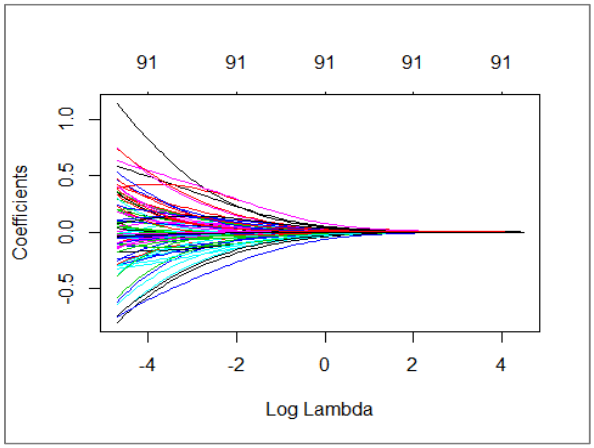

FIGURE 5.1 – Ridge path coefficients

Identification de la valeur "optimale" de λ. Nous avons les logarithmes des λ_i en abscisse de notre graphique. Ils varient de $\log(0.008982) = -4.7124$ a $\log(89.825) = 4.4978$. Une regle empirique consiste a choisir la valeur de λ a partir de laquelle les coefficients commencent "a se stabiliser". On se rend compte que la lecture en ce sens du graphique nest pas aise (Figure 5.1). De plus, cette demarche ne nous assure pas de trouver la solution qui optimise les qualites predictives du modele.

On lui prefere une demarche plus pragmatique visant a optimiser explicitement une mesure devaluation des performances. Notre echantillon test devant jouer le role de juge impartial, il est hors de question quil intervienne dans ce processus. Et, malheureusement, la taille de notre echantillon dapprentissage est trop faible ($ntrain = 200$) pour que nous puissions le scinder en deux parties encore. La solution viable consiste a passer par la validation croisee, qui ne fait intervenir que lechantillon dapprentissage. Nous faisons appel a la fonction `cv.glmnet()`

```
#Optimisation du parametre lambda
set.seed(1)
cv.ridge2 <- cv.glmnet(XTrain,yTrain,family="binomial",type.measure="class",
                       nfolds=10,alpha=0,keep=TRUE)
```

Par rapport a `glmnet()`, de nouveaux parametres apparaissent :

- (`type.measure="class"`) indique que nous traitons dun probleme de classement et que le critere utilise sera le taux derreur (misclassification error).
- (`nfolds = 10`) pour indiquer une validation croisee en 10 blocs (folds).
- (`keep = TRUE`) pour que lon conserve en memoire les groupes de la validation croisee afin de pouvoir reconduire a lidentique lexperimentation si daventure nous souhaitons la relancer avec dautres jeux de parametres. Nous reviendrons sur cette question ci-dessous.

Un graphique permet de mettre en relation les valeurs de $\log(\lambda)$ avec le taux derreur moyen en validation croisee (Figure 5.2, points rouges). Un intervalle de confiance est propose, defini par ± 1 ecart-type de lerreur en validation croisee.

```
plot(cv.ridge2)
```

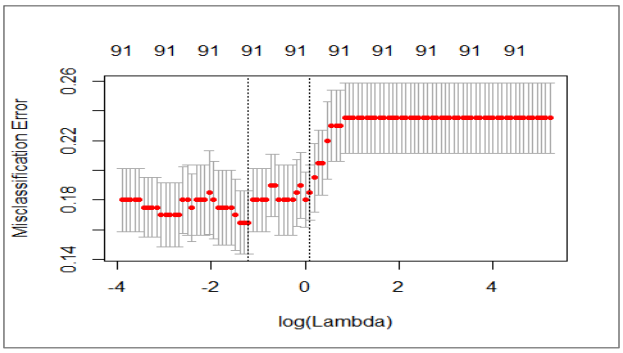

FIGURE 5.2 – Ridge–Taux d'erreur en validation croisée versus $\log(\lambda)$

Remarque 5.1.2. Subdivision en blocs des observations. La validation croisee subdivise les donnees en groupes pour reiterer les processus dapprentissage et test. Loption `keep` permet didentifier les blocs dappartenance de chaque individu et pouvoir ainsi repeter a lidentique la demarche pour dautres modeles. Cette forme dappariement rend plus puissant les comparaisons des performances, ce dont on ne se privera pas dans la suite de notre etude ci-dessous. Les indicatrices de blocs sont conservees dans le champ `$foldid`. Nous voyons ainsi que le premier individu a ete affecte au bloc $n°4$, le second au $n°5$, le troisieme au $n°4$, etc.

```
#subidivision en folds (blocs) des donnees
print(cv.ridge2$foldid)
```

Revenons aux resultats de la validation croisee. Nous avons acces aux details avec les proprietes de lobjet genere par la fonction. Nous affichons ci-dessous la suite de λ_i ; le taux derreur associe ; le nombre de coefficients non-nuls (qui nevolue pas puisque nous utilisons une regression ridge).

```
#affichage
print(cbind(cv.ridge2$lambda,cv.ridge2$cvm,cv.ridge2$nzero))
```

Identifier visuellement les elements importants dans cette liste nest pas aise. Nous le faisons par le calcul. Tout dabord, nous affichons lerreur minimale.

```
#min de l'erreur en cross-validation
print(min(cv.ridge2$cvm))
```

```
## [1] 0.165
```

Puis nous cherchons le λ^* correspondant a cette erreur.

```
#lambda qui minimise l'erreur
print(cv.ridge2$lambda.min)
```

```
## [1] 0.2998318
```

Et nous calculons son logarithme.

```
#son logarithme
print(log(cv.ridge2$lambda.min))
```

```
## [1] -1.204534
```

Cette coordonnee est materialisee par le premier trait pointille (a gauche) dans le graphique de la validation croisee (Figure 5.2).

Regle de lecart-type. On percoit un second trait dans la Figure 5.2 (a droite). Il caractérise la plus grande valeur λ^{**} de λ telle que son erreur moyenne en validation croisee (point rouge) est inferieure a la borne haute de lintervalle de confiance de lerreur optimale (pour λ^{*}). Quel est son interet ? Un peu comme le principe de parcimonie et la preference a la simplicite des modeles, cette solution temoigne dune preference a la regularisation : “ a performances similaires sur notre echantillon dapprentissage, on prefere la solution la plus fortement regularisee, la moins dependante des donnees dapprentissage”.

Remarque 5.1.3. Cette “regle de lecart-type” est un heritage de la methode dinduction darbres `CART` (Breiman et al., 1984) ou lon essaie de determiner le plus petit arbre avec un niveau de performances satisfaisant.

Nous accedons a ce λ^{**} et a son logarithme avec les instructions suivantes.

```
#lambda le plus eleve en 1-se rule print(cv.ridge2$lambda.1se)
## [1] 1.102895

#son log print(log(cv.ridge2$lambda.1se))
## [1] 0.09793863
```

Description	Unité ou Codage	Variable
Sexe	`F` pour fille ; `G` pour garçon	`SEXE`
Ecole située en zone d'éducation prioritaire	`O` pour oui ; `N` pour non	`zep`
Poids	`Kg` (arrondi à 100 g près)	`poids`
Age à la date de la visite	`Année`	`an`
Age à la date de la visite	`Mois`	`mois`
Taille	`Cm` (arrondi à 0.5 cm près)	`taille`

TABLE 5.1 – Variables et codage du jeu de données : IMC-Enfant (imcenfant.txt, xls, ...)

Instruction `R`	Description
`plot(Y∼X)`	Graphe du nuage de points
`lm(Y∼ X)`	Estimation du modèle linéaire
`summary(lm(Y∼ X))`	Description des résultats du modèle
`abline(lm(Y∼ X))`	Trace la droite estimée
`confint(lm(Y∼ X))`	Intervalle de confiance des paramètres de régression
`predict()`	Fonction permettant d'obtenir des prédictions
`plot(lm(Y∼ X))`	Analyse graphique des résidus

TABLE 5.2 – Liste des principales fonctions `R` permettant l'analyse d'une régression Iinéaire simple

Remarque 5.1.4. nt.

	Mort.à	Années.de.carrière	Nombre.de.films	Prénom	Nom	Date.du.décès
1	93	66	211	Michel	Galabru	04-01-2016
2	53	25	58	André	Raimbourg	23-09-1970
3	72	48	98	Jean	Gabin	15-10-1976
4	68	37	140	Louis	De Funès	27-01-1983
5	68	31	74	Lino	Ventura	22-10-1987
6	53	32	81	Jacques	Villeret	28-01-2005

5.2 Devoir

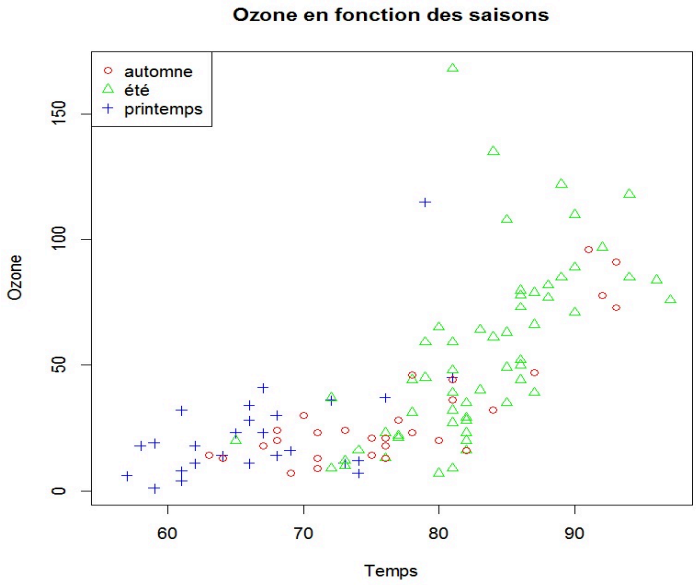

FIGURE 5.3 – Ozone en fonction des saisons