Rapport TP3

Léos Coutrot - Colin Coërchon

09-10-2023

Contents

II - Cookies Study

1 - Logistic regression model using features

On nous donne dans cette partie un fichier `Cookies.csv` :

```
tab = read.csv(file="cookies.csv",sep=',',dec='.',header=TRUE);
Y <- tab$fat

head(tab[, 1:8])
```

```
##     fat    X1100    X1102    X1104    X1106    X1108    X1110    X1112
## 1 12.57 0.259748 0.259482 0.259535 0.259270 0.259376 0.259270 0.259642
## 2 15.13 0.267625 0.267320 0.267117 0.266864 0.266965 0.267016 0.267422
## 3 12.63 0.251753 0.251654 0.251851 0.251900 0.252047 0.252342 0.252735
## 4 13.85 0.278077 0.277877 0.277827 0.278077 0.277777 0.278027 0.278177
## 5 15.25 0.288900 0.288484 0.288328 0.288328 0.288328 0.288796 0.289316
## 6 13.66 0.285423 0.284891 0.284625 0.284625 0.284678 0.284731 0.284785
```

Le fichier `Cookies.csv` comprend exactement 700 variables allant de “X1100” à “X2498”. Et nous avons seulement 32 observations (cookies) dans ce jeu de données.

Et voilà en qualité d’exemple le spectre des valeurs pour les 5 premiers cookies pour les 700 variables de notre problème.

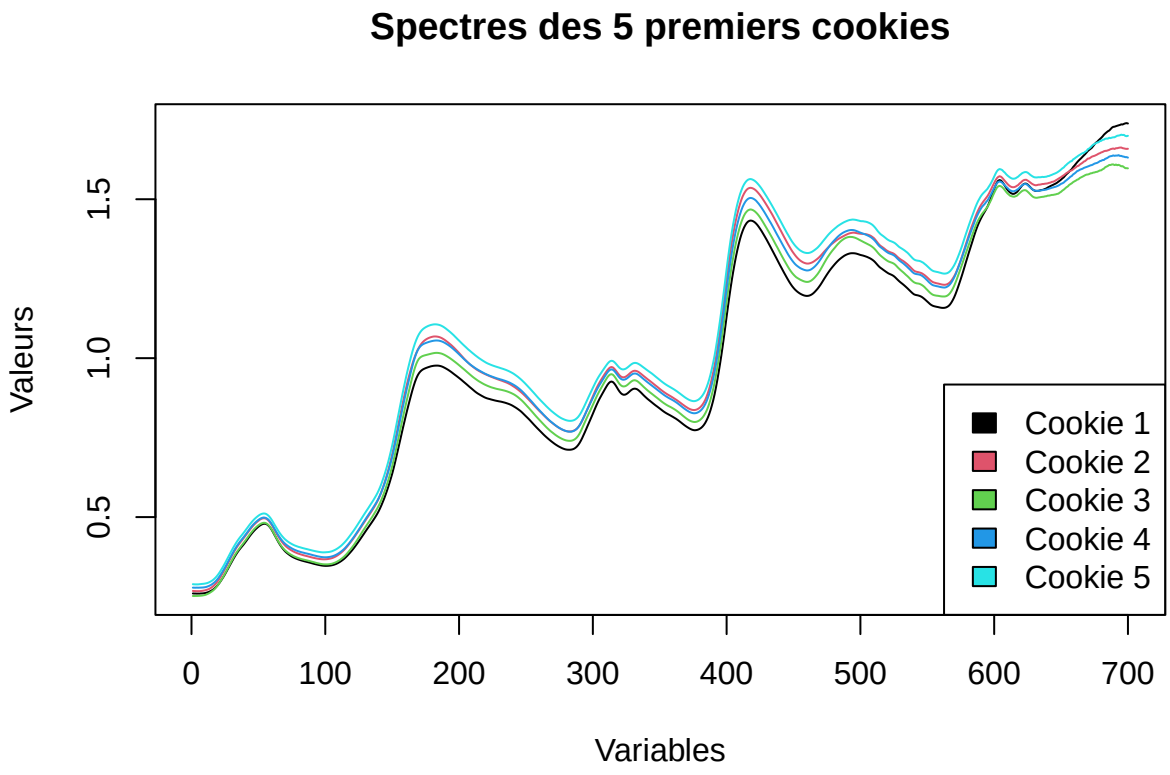
```
# On sélectionne les 5 premiers cookies
spectres <- as.matrix(tab[1:5, -1])

# Le premier spectre
plot(spectres[1,], type='l', ylim=range(spectres), main="Spectres des 5 premiers cookies",
```

```r
     xlab="Variables", ylab="Valeurs", col=1)

# Le tracé des 4 autres spectres
for (i in 2:5) {
    lines(spectres[i,], col=i)
}

legend("bottomright", legend=paste("Cookie", 1:5), fill=1:5)
```



Le but de cette étude est d'expliquer si, pour un cookie, la valeur "fat" est supérieure ou non à la médiane des valeurs "fat" pour chacun des cookies (que l'on indiquera dans la cible binaire `Ybin`).

```r
YBin = as.numeric(tab$fat>median(tab$fat)); # Donné par la prof dans le TP
median_fat <- median(tab$fat)
tab$YBin <- YBin
```

Dans notre situation, il nous est conseillé de transformer les données de 700 variables en 5 caractéristiques qui sont **la moyenne, l'écart-type, la "pente" (*détaillé plus tard*), le minimum et le maximum**, en plus de notre nouvelle variable binaire `Ybin`. Et ceci pour chaque donnée de Data Set.

On se contentera donc du modèle suivant :

$$Y = \beta_1 + \sum_{j=2}^{6} \beta_j X_j + \varepsilon$$

Avec :

- $X_1 = (1, \ldots, 1)$.
- X_2 la moyenne.
- X_3 l'écart-type.
- X_4 la "pente" des valeurs.
- X_5 le minimum.
- X_6 le maximum.

Voici la manière dont nous y sommes pris en R pour obtenir ce nouveau "Data Set" :

```
tab$mean <- rowMeans(tab[,2:701])
tab$std <- apply(tab[,2:701], 1, sd)
tab$min <- apply(tab[,2:701], 1, min)
tab$max <- apply(tab[,2:701], 1, max)
tab$slope <- apply(tab[,2:701], 1, function(x) coef(lm(x ~ seq_along(x)))[2])
features <- c('mean', 'std', 'min', 'max', 'slope')
model1_data <- tab[, c(features, 'YBin')]

head(model1_data)
```

```
##        mean       std      min     max       slope YBin
## 1 0.9851499 0.4111868 0.259270 1.73946 0.001914311    0
## 2 1.0355417 0.4123933 0.266864 1.66273 0.001898164    1
## 3 1.0010620 0.4025158 0.251654 1.60960 0.001860203    0
## 4 1.0280481 0.4040351 0.277777 1.63881 0.001861782    0
## 5 1.0655011 0.4158252 0.288328 1.70320 0.001910926    1
## 6 1.0840236 0.4262425 0.284625 1.74356 0.001967228    0
```

À noter que la "pente", pour chaque ligne de donnée (chaque cookie), est calculée en effectuant une régression linéaire simple des valeurs spectrales par rapport à leur ordre séquentiel. Autrement dit, on trace les données spectrales et on calcule la pente de la ligne qui s'ajuste le mieux à ces points.

À présent, il est enfin temps de réaliser notre régression logistique. Pour ça, on utilise comme préconisé dans le cours la fonction `glm` de R pour effectuer notre régression logistique avec comme "Binary Target Variable" la colonne `YBin` $\in \{0, 1\}$.

Voici le résultat sur RStudio :

```
res <- glm(YBin ~ ., data = model1_data, family = "binomial")
model_summary <- summary(res)
model_summary
```

```
##
## Call:
## glm(formula = YBin ~ ., family = "binomial", data = model1_data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.565e+01  2.440e+01  -1.051   0.2931
## mean        -4.440e+01  6.208e+01  -0.715   0.4745
## std          1.129e+03  6.271e+02   1.801   0.0717 .
```

```
## min          1.371e+02  1.288e+02   1.065   0.2870
## max          1.894e+00  2.294e+01   0.083   0.9342
## slope       -2.284e+05  1.186e+05  -1.925   0.0542 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 44.361  on 31  degrees of freedom
## Residual deviance: 26.927  on 26  degrees of freedom
## AIC: 38.927
##
## Number of Fisher Scoring iterations: 6
```

Regardons d'un peu plus près les coefficients qui sont importantes dans cette régression logistique, et ceux qui sont les plus insignifiants :

```
coefficients_data <- model_summary$coefficients
sorted_coefficients <- coefficients_data[order(coefficients_data[, "Pr(>|z|)"]), ]
most_significant_coefficient <- head(sorted_coefficients, 2)
least_significant_coefficient <- tail(sorted_coefficients, 2)

print("Les coefficients les plus significatifs :")
```

```
## [1] "Les coefficients les plus significatifs :"
```

```
print(most_significant_coefficient)
```

```
##          Estimate  Std. Error   z value   Pr(>|z|)
## slope -228413.515 118633.6505 -1.925369 0.05418325
## std      1129.463    627.1115  1.801056 0.07169398
```

```
print("Les coefficients les moins significatifs :")
```

```
## [1] "Les coefficients les moins significatifs :"
```

```
print(least_significant_coefficient)
```

```
##        Estimate Std. Error     z value  Pr(>|z|)
## mean -44.400257   62.07662 -0.71524932 0.4744550
## max    1.893735   22.93550  0.08256785 0.9341952
```

Ainsi, nous avons d'abord créé un modèle de **régression logistique** en utilisant toutes les variables disponibles dans `model1_data`, en ciblant `YBin` comme **variable dépendante**.

On remarque que la plupart des variables ont des p-values élevées, suggérant qu'elles ne sont pas statistiquement significatives dans ce modèle, à l'exception de **l'écart type** et de **la pente** qui approchent la significativité ($p < 0.1$).

De plus, la réduction de la **déviance résiduelle** par rapport à la **déviance nulle** indique une amélioration par rapport au modèle nul, mais il est encore difficile de juger de la qualité de l'ajustement avec notre modèle actuel.

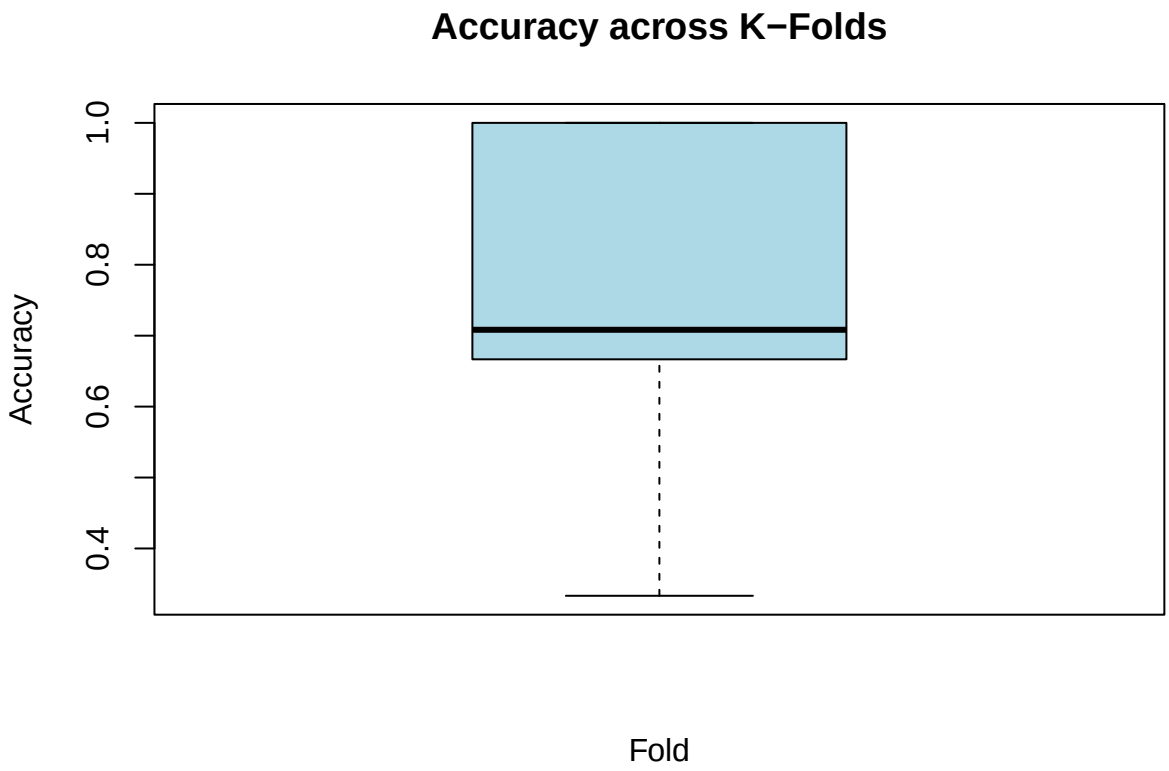
Après avoir construit le modèle initial, il est crucial de vérifier sa fiabilité. C'est là qu'intervient **la validation croisée K-fold** :

```
k <- 10
n <- nrow(tab)
fold_indices <- sample(cut(seq(n), breaks=k, labels=FALSE))
accuracies <- numeric(k)

for(i in 1:k){
  test_indices <- which(fold_indices == i)
  train_data <- tab[-test_indices, ]
  test_data <- tab[test_indices, ]

  fitted_model <- glm(YBin ~ mean + std + min + max + slope, data = train_data, family = "binomial")
  predictions <- predict(fitted_model, test_data, type = "response")
  predicted_classes <- ifelse(predictions > 0.5, 1, 0)
  confusionMatrix <- table(test_data$YBin, predicted_classes)
  accuracies[i] <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
}

boxplot(accuracies, main = "Accuracy across K-Folds", ylab = "Accuracy", xlab = "Fold", col = "lightblu
```



Avec **un AIC de 38.927** et une performance de validation croisée K-fold variant **de 0.7 à 1.0**, le modèle de régression logistique montre une capacité raisonnable à prédire si la teneur en matières grasses d'un cookie dépasse la médiane.

Bien que l'ajustement du modèle soit acceptable (comme le suggère l'AIC), la variabilité dans les exactitudes

de validation croisée indique une certaine instabilité du modèle (puisque seules les variables `std` et `slope` sont un minimum significatives).

Bref, **on ne peut pas encore donner une conclusion précise** avec ces seules 5 informations sur les 700 variables initiales.

2 - Logistic regression model using the spectra

Nous reprenons les données de la première partie. Dans l'objectif de tester nos modèles plus tard avec des méthodes de validation croisée, nous allons séparer notre jeu de données en deux groupes bien distincts. Nous allons prendre 80% des données pour construire notre modèle (`train_data`) et 20% pour les tester par la suite (`test_data`).

```
tab=read.csv(file="cookies.csv",sep=',',dec='.',header=TRUE);
YBin <- as.numeric(tab$fat > median(tab$fat))
tab$YBin <- YBin

split <- sample(c(TRUE, FALSE), nrow(tab), replace = TRUE, prob = c(0.8, 0.2))
train_data <- tab[split, ]
test_data <- tab[!split, ]
```

Ridge regression

On souhaite ici minimiser la fonction Φ donnée par :

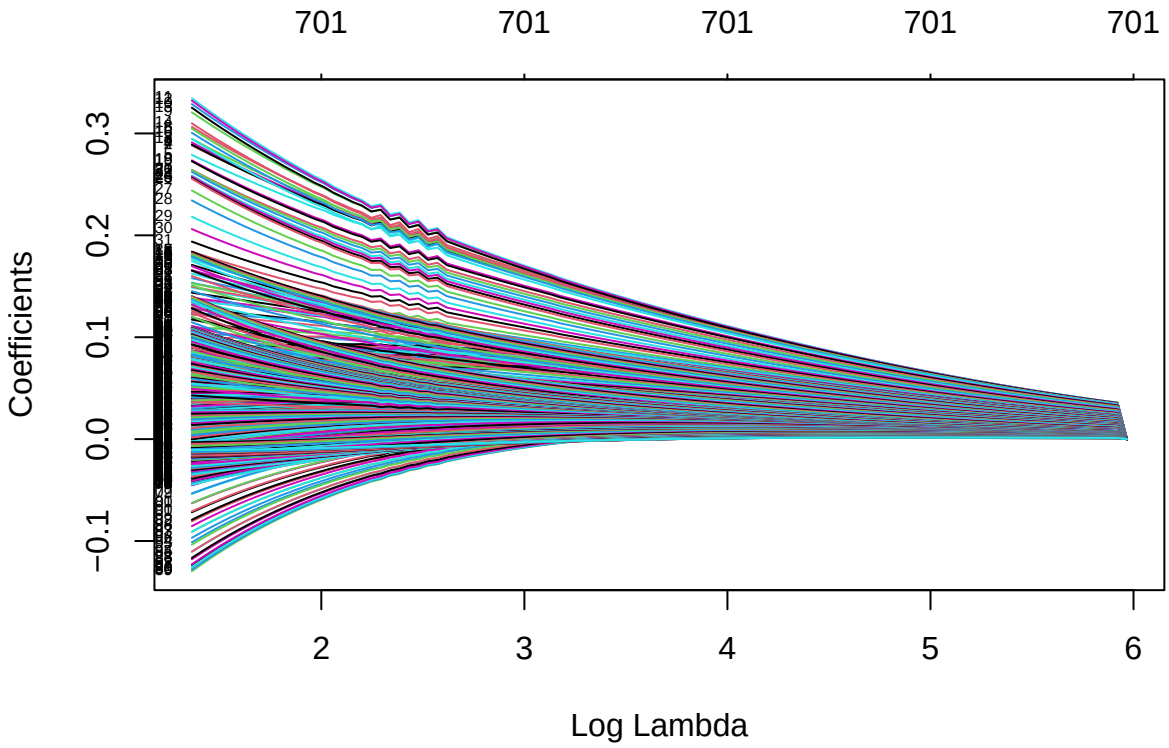
$$\Phi(\beta) = \|Y - X\beta\|_2^2 + k\,\|\beta\|_2^2 \qquad (\text{avec } k \in \mathbb{R}_+^*)$$

$$\implies \quad \Phi(\beta) = (Y - X\beta)^T(Y - X\beta) + k\sum_{j=1}^{p}\beta_j^2 \qquad (\text{avec } k \in \mathbb{R}_+^*)$$

On utilise pour cela (comme recommandé dans le TP) la bibliothèque `glmnet` pour réaliser cette régression pénalisée. Et on peut alors tracer le joli chemin de régularisation pour les différentes valeurs de k lorsqu'il augmente (de gauche à droite) :

```
X_train <- as.matrix(train_data[, -which(names(train_data) %in% c("YBin"))])
Y_train <- train_data$YBin

ridge_model <- glmnet(X_train, Y_train, alpha = 0, family = "binomial")
plot(ridge_model, xvar = "lambda", label = TRUE)
```



On s'intéresse alors à trouver le **k optimal**.

La fonction `cv.glmnet()` trouve empiriquement la valeur qui minimise l'erreur de validation croisée (c'est-à-dire, une minimisation de notre fonction Φ).

Dans ce TP, comparé au TP2, on s'intéresse aussi à la valeur de k (ou λ)qui vérifie la règles « 1 erreur standard » (le modèle le plus pénalisé à la 1ère distance du modèle avec le moins d'erreur).

```
cv_ridge <- cv.glmnet(X_train, Y_train, alpha = 0, family = "binomial", grouped=FALSE)

# Affiche le meilleur lambda (k) trouvé
print(cv_ridge$lambda.min)
```

```
## [1] 28.8641
```

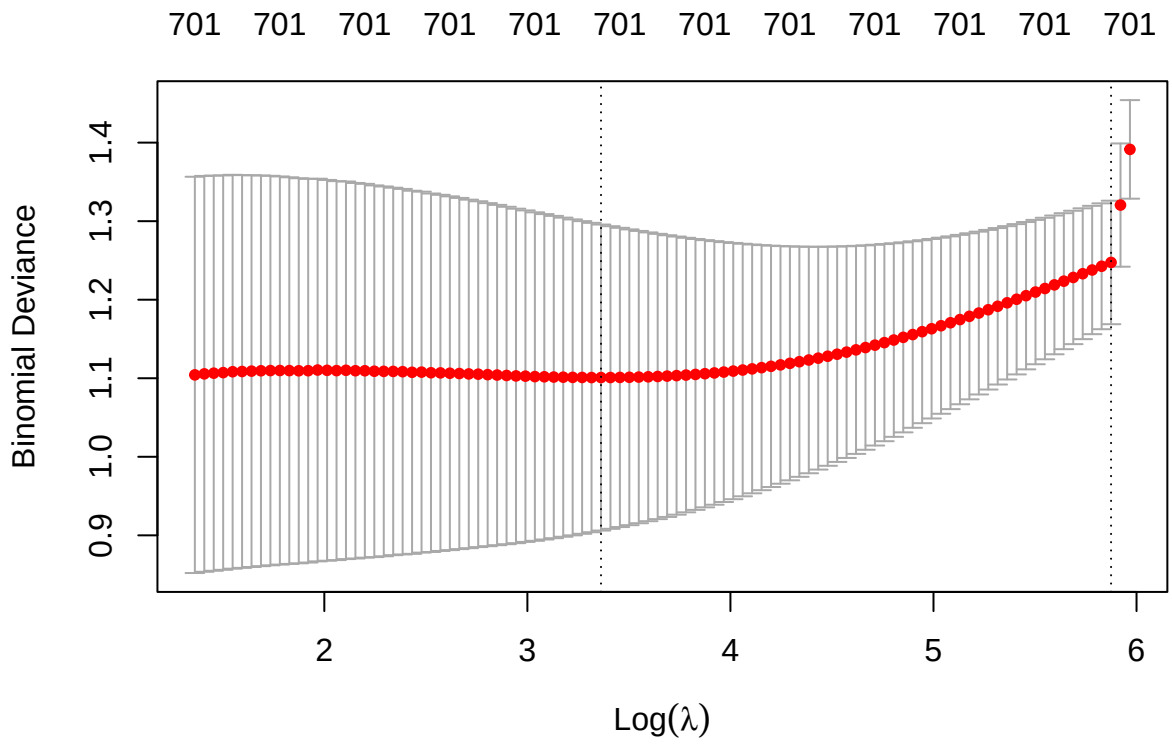
```
# Affiche le lambda 1sE trouvé
print(cv_ridge$lambda.1se)
```

```
## [1] 355.8501
```

Les deux modèles finaux sont ajustés avec ces k différents.

La fonction `cv.glmnet()` a donc utilisé une méthode de validation croisée, et cette librairie permet l'affichage très élégant de cette méthode là.

```r
plot(cv_ridge)
```




```r
ridge_min <- glmnet(X_train, Y_train, alpha = 0, family = "binomial", lambda = cv_ridge$lambda.min)
ridge_1se <- glmnet(X_train, Y_train, alpha = 0, family = "binomial", lambda = cv_ridge$lambda.1se)
```

Après avoir déterminé les meilleurs valeurs de lambda, les coefficients correspondants (les β) sont extraits. Les variables les plus importantes, c'est-à-dire celles dont les coefficients ont les valeurs absolues les plus élevées, sont ensuite identifiées. Cela fournit un aperçu des prédicteurs qui ont le plus d'impact dans le modèle Ridge.

On construit ainsi notre ensemble $\mathcal{M} \subset [\![1, p]\!]$.

Pour λ_{min} :

```r
coef_ridge <- predict(ridge_min, type = "coefficients", s = cv_ridge$lambda.min)
important_vars_ridge_min <- order(abs(coef_ridge), decreasing = TRUE)

head(important_vars_ridge_min)
```

```
## [1]  1 12 11 13 10 14
```

Pour λ_{1se} :

```r
coef_ridge <- predict(ridge_1se, type = "coefficients", s = cv_ridge$lambda.1se)
important_vars_ridge_1se <- order(abs(coef_ridge), decreasing = TRUE)

head(important_vars_ridge_1se)
```

```
## [1]  1  3  4 11 12 10
```

Lasso Regression

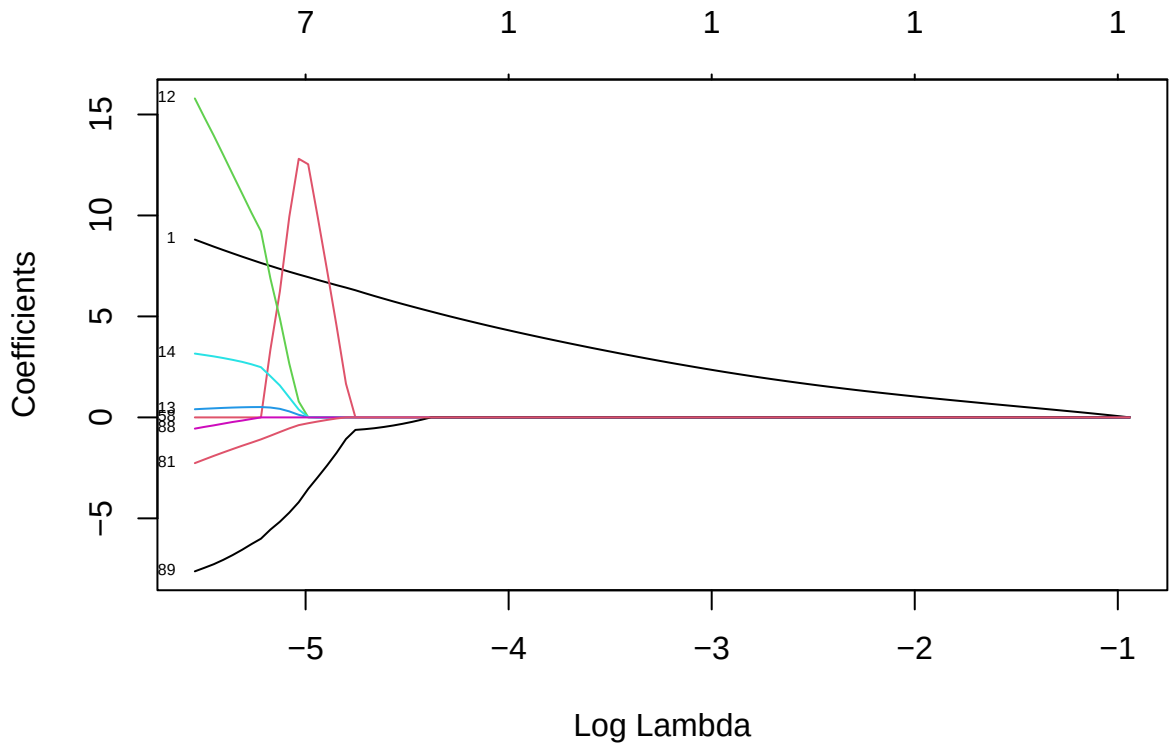
On souhaite ici minimiser la fonction Φ donnée par :

$$\Phi(\beta) = \|Y - X\beta\|_2^2 + k\,\|\beta\|_1 \qquad (\text{avec } k \in \mathbb{R}_+^*)$$

$$\implies \quad \Phi(\beta) = (Y - X\beta)^T(Y - X\beta) + k\sum_{j=1}^{p}|\beta_j| \qquad (\text{avec } k \in \mathbb{R}_+^*)$$

Les données sont chargées de la même manière que pour Ridge. Comme pour la régression Ridge, les données pour la régression Lasso doivent également être centrées et réduites.

```r
lasso_model <- glmnet(X_train, Y_train, alpha = 1, family = "binomial")
plot(lasso_model, xvar = "lambda", label = TRUE)
```



On s'intéresse alors à trouver le **k optimal**.

La fonction `cv.glmnet()` trouve empiriquement la valeur qui minimise l'erreur de validation croisée (c'est-à-dire, une minimisation de notre fonction Φ).

Dans ce TP, comparé au TP2, on s'intéresse aussi à la valeur de k (ou λ)qui vérifie la règles « 1 erreur standard » (le modèle le plus pénalisé à la 1ère distance du modèle avec le moins d'erreur).

```
cv_lasso <- cv.glmnet(X_train, Y_train, alpha = 1, family = "binomial", grouped = FALSE)

# Affiche le meilleur lambda (k) trouvé
print(cv_lasso$lambda.min)
```

```
## [1] 0.00390545
```

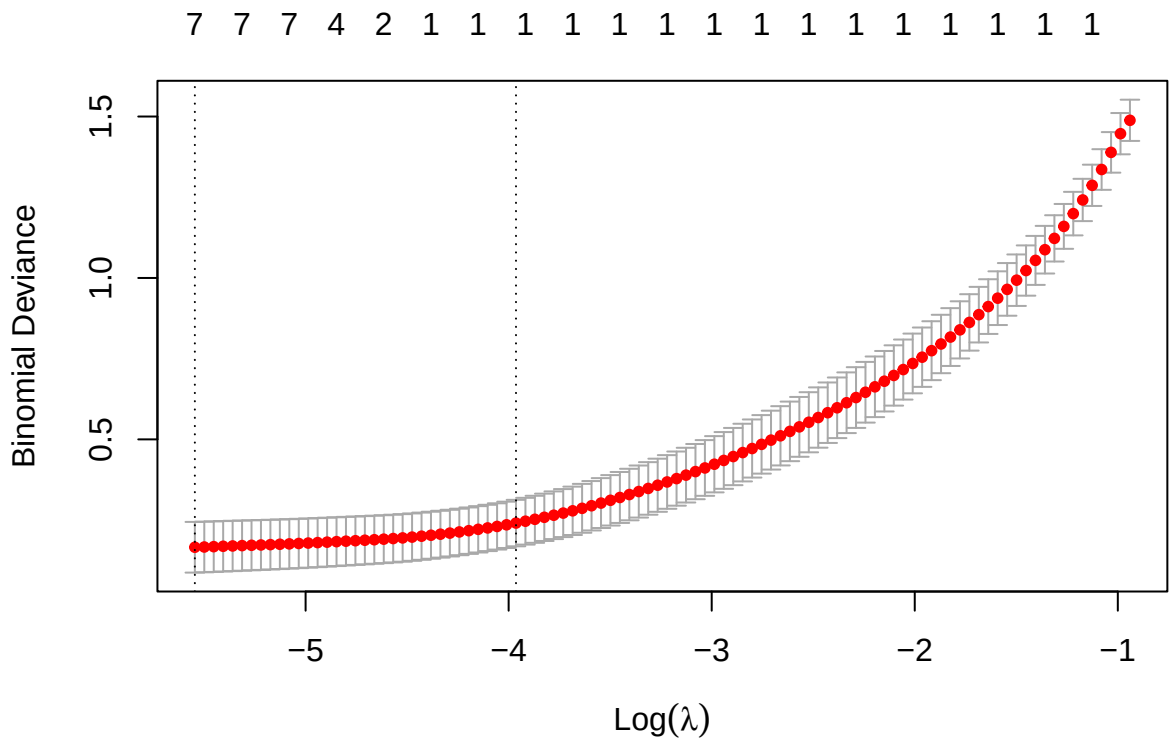
```
# Affiche le lambda 1sE trouvé
print(cv_lasso$lambda.1se)
```

```
## [1] 0.01899065
```

Les deux modèles finaux sont ajustés avec ces k différents.

La fonction `cv.glmnet()` a donc utilisé une méthode de validation croisée, et cette librairie permet l'affichage très élégant de cette méthode là.

```
plot(cv_lasso)
```



```r
lasso_min <- glmnet(X_train, Y_train, alpha = 1, family = "binomial", lambda = cv_lasso$lambda.min)
lasso_1se <- glmnet(X_train, Y_train, alpha = 1, family = "binomial", lambda = cv_lasso$lambda.1se)
```

Après avoir déterminé les meilleurs valeurs de lambda, les coefficients correspondants (les β) sont extraits. Les variables les plus importantes, c'est-à-dire celles dont les coefficients ont les valeurs absolues les plus élevées, sont ensuite identifiées. Cela fournit un aperçu des prédicteurs qui ont le plus d'impact dans le modèle Ridge.

On construit ainsi notre ensemble $\mathcal{M} \subset [\![1, p]\!]$.

- Pour λ_{min} :

```r
coef_lasso <- predict(lasso_min, type = "coefficients", s = cv_lasso$lambda.min)
important_vars_lasso_min <- order(abs(coef_lasso), decreasing = TRUE)

head(important_vars_lasso_min)
```

```
## [1]   1 313   2 489 682 490
```

- Pour λ_{1se} :

```r
coef_lasso <- predict(lasso_1se, type = "coefficients", s = cv_lasso$lambda.1se)
important_vars_lasso_1se <- order(abs(coef_lasso), decreasing = TRUE)

head(important_vars_lasso_1se)
```

```
## [1] 1 2 3 4 5 6
```

3 - Conclusion

Pour comparer efficacement nos différents modèles, nous proposons d'estimer la précision en utilisant la méthode des K Folds. Nous examinerons donc les modèles suivants :

- **Le modèle Ridge** (min et 1se)
- **Le modèle Lasso** (min et 1se)
- Notre modèle avec les différentes caractéristiques identifiées dans la partie 1 (pente, minimum, maximum, etc.)

Nous avons déjà codé cette procédure pour le modèle incluant les différentes caractéristiques dans la partie 1. Il nous reste donc à coder cette version pour les modèles restants.

```
k_fold_cv_glmnet <- function(data, k, alpha, lambda) {
  fold_indices <- sample(cut(seq(nrow(data)), breaks=k, labels=FALSE))
  accuracies <- numeric(k)

  for (i in 1:k) {
    test_indices <- which(fold_indices == i)
    train_data <- data[-test_indices, ]
    test_data <- data[test_indices, ]

    X_train <- as.matrix(train_data[, -which(names(train_data) %in% c("YBin"))])
    Y_train <- train_data$YBin
    X_test <- as.matrix(test_data[, -which(names(test_data) %in% c("YBin"))])
    model <- glmnet(X_train, Y_train, alpha = alpha, lambda = lambda, family = "binomial")
    predictions <- predict(model, newx = X_test, type = "response", s = lambda)
    predicted_classes <- ifelse(predictions > 0.5, 1, 0)
    confusionMatrix <- table(test_data$YBin, predicted_classes)
    accuracies[i] <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
  }

  return(accuracies)
}
```

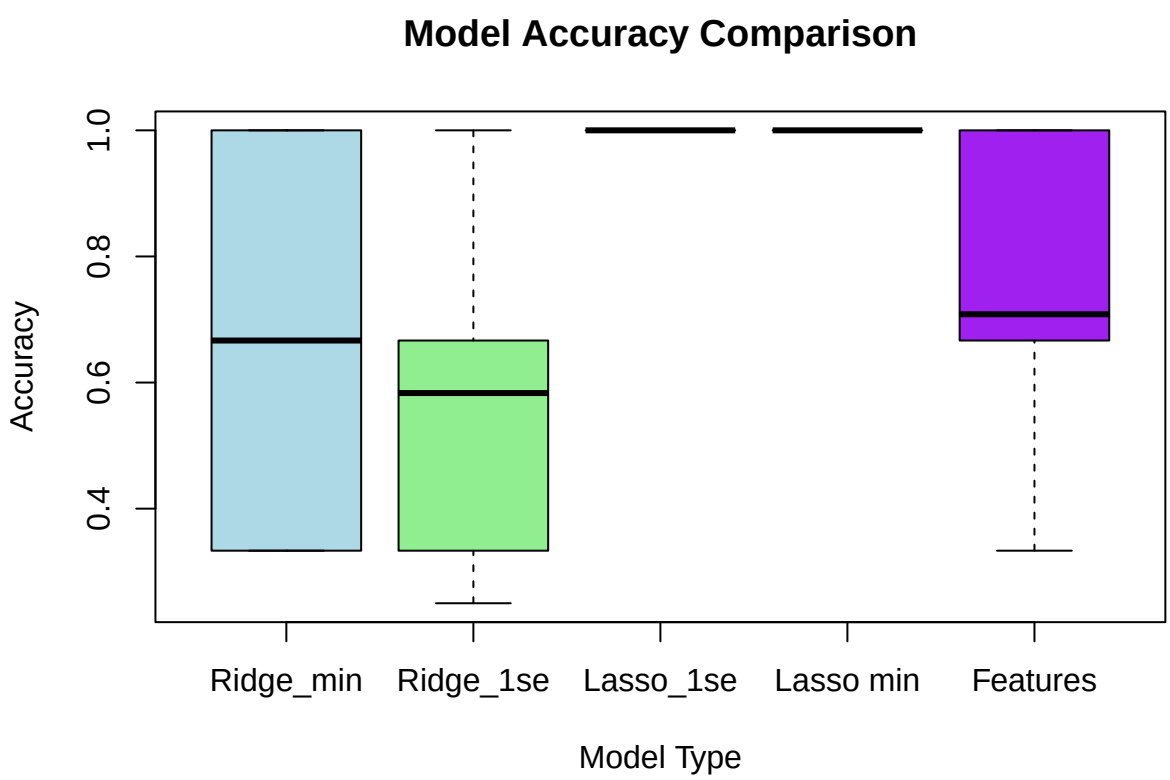
Nous récupérons la liste des précisions pour chaque modèle et proposons de sélectionner un k égal à 10.

```
set.seed(123)
ridge_min_accuracies <- k_fold_cv_glmnet(tab, 10, alpha = 0, lambda = cv_ridge$lambda.min)
ridge_1se_accuracies <- k_fold_cv_glmnet(tab, 10, alpha = 0, lambda = cv_ridge$lambda.1se)
lasso_min_accuracies <- k_fold_cv_glmnet(tab, 10, alpha = 1, lambda = cv_lasso$lambda.min)
lasso_1se_accuracies <- k_fold_cv_glmnet(tab, 10, alpha = 1, lambda = cv_lasso$lambda.1se)
```

Nous allons donc visualiser toutes ces données dans **un graphique de type boxplot**.

```
accuracies_list <- list( Ridge_Min = ridge_min_accuracies,Ridge_1se=ridge_1se_accuracies,Lasso_1se = las

boxplot(accuracies_list,
        main = "Model Accuracy Comparison",
        xlab = "Model Type",
        ylab = "Accuracy",
        col = c("lightblue", "lightgreen", "lightcoral","lightgrey","purple"),
        names = c( "Ridge_min","Ridge_1se","Lasso_1se","Lasso min","Features"))
```



Nous remarquons sur le graphique que les résultats semblent étranges pour “Lasso 1se” et “Lasso min” qui ont une répartition très surprenante. Nous ne savons pas vraiment d’où vient cette potentielle erreur et nous ne les prendrons donc pas en compte dans notre analyse finale.

Ceci dit, **le modèle Features semble être finalement le meilleur parmi les modèles restants**, en raison de sa médiane de précision plus élevée et de sa variabilité relativement faible. Cependant, on voit une potentielle valeur aberrante, ce qui pourrait poser problème dans certains cas.

D’un autre côté, les deux modèles ridge semblent assez distincts, le modèle “Ridge min” a une très grande variabilité (la plus grande de tous les modèles du graphique), mais une médiane très légèrement supérieure à celle de “Ridge 1se”. On note aussi que le modèle “Ridge 1se” possède des valeurs qui semblent aberrantes contrairement à “Ridge min”, ce qui rendrait Ridge min peut-être plus pertinent que 1se. Mais **ces deux derniers ont une médiane et une variabilité de leur précision bien moins intéressante que le modèle Features**.

En conclusion, nous pensons que le modèle le plus adéquat au vu de nos données est **le modèle Features**.

Cependant, si les procédures K-fold des modèles “Lasso min” et “Lasso 1se” sont finalement corrects, cela signifie que **la précision des ces deux modèles est quasiment parfaite**, et qu’il serait alors tout naturel de privilégier ces derniers.