Rapport TP3 - Modèles de Régression Régularisée

Alexis Schneider - Kelvin Wong

09-11-2025

Table des matières

1 II. Electricity Data Set : Approche Comparative

1.1 Introduction et Objectif

La production quotidienne d'électricité à Mexico dépend fortement de facteurs climatiques et socio-économiques. Comprendre ces influences permet non seulement d'anticiper les variations de demande, mais aussi d'optimiser la planification énergétique dans un contexte de transition vers des réseaux plus intelligents.

Ce travail vise à prédire si la production électrique quotidienne sera supérieure ou inférieure à sa médiane historique, à partir de données météorologiques et calendaires. Pour cela, nous comparons des modèles de régression logistique classiques et régularisés (Ridge et Lasso), afin d'évaluer leur performance, leur stabilité et leur capacité à identifier les déterminants majeurs de la production.

1.2 Modèles de Régression Logistique Classique et Sélection de Variables

La première étape consiste à charger les données, à réaliser l'ingénierie des variables et à créer notre variable cible binaire `TotalBin`. Un `1` indiquera une production supérieure à la médiane, et un `0` une production inférieure ou égale.

```
# 1. Chargement des données brutes
data_Mexico <- read.csv("Mexico_data.csv")
df_model <- data_Mexico %>%
  transmute(
    # Variable cible : 1 si la production > médiane, sinon 0
    TotalBin = factor(ifelse(Total > median(Total, na.rm = TRUE), 1, 0)),T2M, T2Mmax, T2Mmin, RH, SSRD,
    Covid, Holidays,DOW = as.factor(DOW),
    sin_TOY = sin(2 * pi * TOY / 365.25),
    cos_TOY = cos(2 * pi * TOY / 365.25)
```

```
  ) %>%
  na.omit()

# Séparation du jeu de données en Entraînement et Test
set.seed(123) # Pour la reproductibilité
train_idx <- createDataPartition(df_model$TotalBin, p = 0.8, list = FALSE)
train_df <- df_model[train_idx, ]
test_df  <- df_model[-train_idx, ]
```

Il est crucial de diviser nos données en un jeu d'entraînement (train_df) et un jeu de test (test_df). Le jeu de test est mis de côté et ne sera utilisé qu'une seule fois, à la toute fin, pour évaluer de manière impartiale la performance de notre modèle final. Toutes les étapes de construction, de sélection de variables (y compris la validation croisée) se feront uniquement sur le jeu d'entraînement. Cette méthodologie stricte est essentielle pour éviter la fuite de données (data leakage), un phénomène où des informations du jeu de test influencent indirectement le choix du modèle, conduisant à une évaluation trop optimiste de sa capacité à généraliser sur de nouvelles données.

1.2.1 Modèle GLM Complet et Sélection Backward

Nous commençons par un modèle de régression logistique (`glm`) incluant tous les prédicteurs. Ensuite, nous utilisons une sélection descendante (`step`) pour obtenir un modèle plus parcimonieux basé sur le critère AIC.

```
# Modèle GLM complet
glm_full <- glm(TotalBin ~ ., data = train_df, family = "binomial")
summary(glm_full)
```

```
##
## Call:
## glm(formula = TotalBin ~ ., family = "binomial", data = train_df)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.739e+01  1.112e+01   1.564 0.117907
## T2M         -9.225e-01  9.175e-01  -1.005 0.314679
## T2Mmax      -4.513e-01  4.425e-01  -1.020 0.307741
## T2Mmin       1.292e+00  4.962e-01   2.604 0.009224 **
## RH           6.821e-03  4.934e-02   0.138 0.890043
## SSRD         1.290e-05  3.722e-06   3.466 0.000529 ***
## STRD        -1.940e-05  1.053e-05  -1.843 0.065279 .
## T2M_sq       1.627e-02  1.448e-02   1.124 0.261156
## Covid       -2.672e-02  5.026e-03  -5.316 1.06e-07 ***
## Holidays    -4.278e+00  8.884e-01  -4.815 1.47e-06 ***
## DOW1         1.209e+00  4.782e-01   2.529 0.011440 *
## DOW2         2.094e+00  4.816e-01   4.347 1.38e-05 ***
## DOW3         2.220e+00  4.913e-01   4.518 6.25e-06 ***
## DOW4         1.758e+00  4.733e-01   3.713 0.000205 ***
## DOW5        -1.038e+00  4.548e-01  -2.282 0.022501 *
## DOW6        -3.747e+00  4.739e-01  -7.907 2.63e-15 ***
## sin_TOY     -2.288e+00  3.350e-01  -6.829 8.57e-12 ***
## cos_TOY     -1.876e+00  9.120e-01  -2.057 0.039678 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1620.58  on 1168  degrees of freedom
## Residual deviance:  474.79  on 1151  degrees of freedom
## AIC: 510.79
##
## Number of Fisher Scoring iterations: 7
```

Le résumé du modèle glm_full révèle plusieurs points importants. On remarque que plusieurs variables comme T2M, T2Mmax ou RH présentent des p-values (Pr(>|z|)) très élevées, suggérant qu'elles ne sont pas statistiquement significatives en présence des autres prédicteurs. Le modèle est donc probablement trop complexe et pourrait être amélioré en retirant les variables qui n'apportent que peu d'information. Pour optimiser ce modèle, nous utilisons une sélection descendante qui va automatiquement retirer les variables les moins pertinentes en se basant sur le critère AIC.

```r
# Modèle GLM avec sélection backward
back_model <- step(glm_full, direction = "backward", trace = 0)
summary(back_model)
```

```
##
## Call:
## glm(formula = TotalBin ~ T2Mmax + T2Mmin + SSRD + STRD + Covid +
##     Holidays + DOW + sin_TOY + cos_TOY, family = "binomial",
##     data = train_df)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.167e+01  9.899e+00   1.179 0.238284
## T2Mmax      -6.046e-01  1.889e-01  -3.200 0.001374 **
## T2Mmin       1.167e+00  2.894e-01   4.031 5.55e-05 ***
## SSRD         1.233e-05  3.566e-06   3.458 0.000543 ***
## STRD        -1.885e-05  8.464e-06  -2.227 0.025921 *
## Covid       -2.633e-02  4.828e-03  -5.453 4.94e-08 ***
## Holidays    -4.232e+00  8.824e-01  -4.797 1.61e-06 ***
## DOW1         1.190e+00  4.737e-01   2.512 0.012001 *
## DOW2         2.115e+00  4.821e-01   4.386 1.15e-05 ***
## DOW3         2.247e+00  4.920e-01   4.567 4.94e-06 ***
## DOW4         1.755e+00  4.712e-01   3.725 0.000195 ***
## DOW5        -1.025e+00  4.454e-01  -2.302 0.021351 *
## DOW6        -3.629e+00  4.533e-01  -8.006 1.19e-15 ***
## sin_TOY     -2.275e+00  3.193e-01  -7.123 1.06e-12 ***
## cos_TOY     -2.052e+00  8.527e-01  -2.406 0.016114 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1620.58  on 1168  degrees of freedom
## Residual deviance:  476.21  on 1154  degrees of freedom
## AIC: 506.21
##
## Number of Fisher Scoring iterations: 7
```

L'analyse des coefficients du modèle back_model nous fournit des informations précieuses sur les moteurs de la production électrique.

Impact des facteurs calendaires : Les coefficients les plus importants en magnitude sont ceux des jours de la semaine (DOW6 pour dimanche) et des jours fériés (Holidays), tous deux fortement négatifs. Cela confirme que le rythme de l'activité économique est un prédicteur dominant de la production électrique. Les termes **sin_TOY** et **cos_TOY**, également très significatifs, modélisent le cycle saisonnier annuel.

Impact de la température : La température minimale (T2Mmin) a un coefficient positif et significatif, suggérant que des nuits plus chaudes (probablement liées à l'usage de la climatisation) augmentent la probabilité d'une production élevée.

L'indice **Covid** a lui aussi, comme attendu, un impact négatif très significatif sur la production.

```
# Comparaison des AIC
aic_diff <- round(AIC(glm_full) - AIC(back_model), 2)
cat("Amélioration de l'AIC grâce à la sélection Backward:", aic_diff, "\n")
```

```
## Amélioration de l'AIC grâce à la sélection Backward: 4.58
```

La comparaison des AIC est sans appel. Le modèle `back_model` possède un AIC de 506.21, ce qui est significativement plus faible que l'AIC du modèle complet 510.79. Cette réduction de 4.58 points confirme que la sélection de variables a réussi à créer un modèle plus parcimonieux tout en conservant l'essentiel de l'information prédictive. Le modèle `back_model` est donc statistiquement préférable au modèle complet.Pour évaluer sa robustesse, nous effectuons une validation croisée K-fold.

1.2.2 Comparaison avec les sélections Forward et Stepwise

Il est interessant de comparer la valeur trouver pour une 'backward selection' avec d'autres méthodes de sélections. Regardons les résultats que l'on obtient avec Forward et Stepwise

```
glm_intercept_only <- glm(TotalBin ~ 1, data = train_df, family = "binomial")

# Lancer la sélection forward à partir de ce modèle simple
step_model_fwd <- step(glm_intercept_only, direction = "forward", scope = formula(glm_full), trace = 0)

# Modèle GLM avec sélection stepwise
step_model_stepwise <- step(glm_full, direction = "both", trace = 0)

# Comparaison des AIC
cat("AIC Backward:", AIC(back_model), "\n")
```

```
## AIC Backward: 506.2072
```

```
cat("AIC Forward:", AIC(step_model_fwd), "\n")
```

```
## AIC Forward: 506.9174
```

```
cat("AIC Stepwise:", AIC(step_model_stepwise), "\n")
```

```
## AIC Stepwise: 506.2072
```

Pour évaluer la performance du modèle, nous allons utiliser ses prédictions pour construire une **matrice de confusion**. Nous utilisons un seuil de décision de 0.5, ce qui correspond au critère **MAP (Maximum A Posteriori)**, où l'on assigne à chaque observation la classe la plus probable.

```
probabilities_train <- predict(back_model, type = "response")

# Classifier les prédictions
predicted_classes_train <- ifelse(probabilities_train > 0.5, 1, 0)

# Créer la matrice de confusion en comparant avec les valeurs du jeu d'entrainement
true_values_train <- factor(train_df$TotalBin, levels = c(0, 1))
predicted_classes_factor_train <- factor(predicted_classes_train, levels = c(0, 1))

conf_matrix_train <- table(Prédit = predicted_classes_factor_train, Réel = true_values_train)

# Affichage et calcul des métriques
cat("Matrice de Confusion pour le modèle Stepwise (sur le jeu d'entraînement):\n")
```

```
## Matrice de Confusion pour le modèle Stepwise (sur le jeu d'entraînement):
```

```
print(conf_matrix_train)
```

```
##       Réel
## Prédit   0   1
##      0 531  50
##      1  54 534
```

```
accuracy_train <- sum(diag(conf_matrix_train)) / sum(conf_matrix_train)
cat("\nPerformance Globale (Accuracy sur le jeu d'entraînement):", round(accuracy_train, 3), "\n")
```

```
##
## Performance Globale (Accuracy sur le jeu d'entraînement): 0.911
```

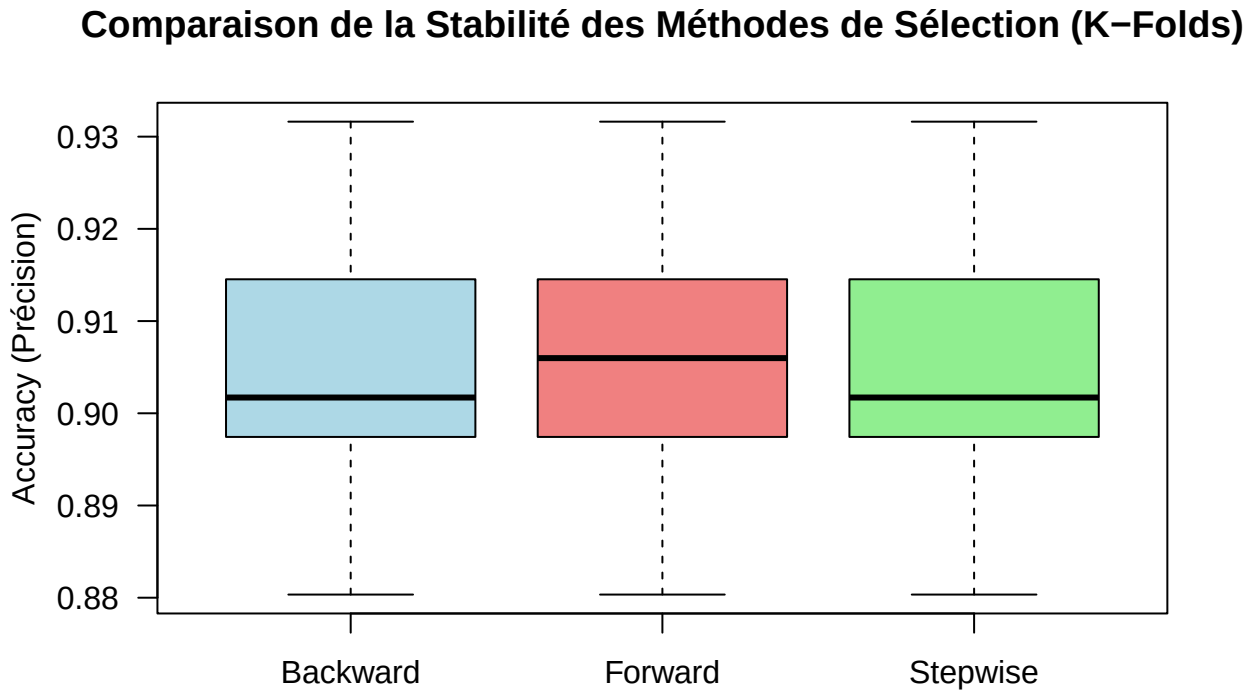
La matrice de confusion révèle une excellente performance globale, avec une Accuracy de 91.1%. Cela signifie que le modèle classe correctement plus de 9 observations sur 10. Cependant, une analyse plus fine des erreurs est nécessaire : Les Faux Négatifs : Le modèle a prédit à tort une production "Basse" pour 50 jours qui étaient en réalité "Hauts". Cela représente un taux de faux négatifs de 8.5%. C'est le type d'erreur le plus critique, car il correspond à une sous-estimation du besoin en électricité. Les Faux Positifs : Le modèle a prédit à tort une production "Haute" pour 54 jours qui étaient en réalité "Bas", soit un taux de faux positifs de 9.2%. Bien que ces taux d'erreur soient faibles, ils ne sont pas nuls. Le modèle est performant, mais pas parfait. La question qui se pose maintenant est de savoir si cette performance est fiable et constante.

Une performance calculée sur train_df peut être optimiste. Pour obtenir une mesure plus réaliste et évaluer la stabilité du modèle, nous effectuons une validation croisée K-fold. Cette méthode consiste à entraîner et tester le modèle 10 fois sur différents sous-échantillons des données, nous donnant ainsi une vision de la variabilité de sa performance.

```
# Validation Croisée Comparative des Méthodes de Sélection (Stepwise, Forward, Backward)
k <- 10
set.seed(123) # Pour la reproductibilité

# Initialisation des vecteurs pour stocker les précisions de chaque méthode
```

```r
accuracies_back <- numeric(k)
accuracies_fwd <- numeric(k)
accuracies_step <- numeric(k)

# Création des indices des plis (folds) sur le jeu d'entraînement
fold_indices <- sample(cut(seq(nrow(train_df)), breaks = k, labels = FALSE))

for(i in 1:k) {
  # Séparation en sous-ensembles d'entraînement et de validation pour ce pli
  test_indices_inner <- which(fold_indices == i)
  train_data_inner <- train_df[-test_indices_inner, ]
  test_data_inner  <- train_df[test_indices_inner, ]

# Modèles de base, entraînés uniquement sur le sous-ensemble d'entraînement du pli
  glm_full_inner <- glm(TotalBin ~ ., data = train_data_inner, family = "binomial")
  glm_intercept_inner <- glm(TotalBin ~ 1, data = train_data_inner, family = "binomial")

  # Exécution des 3 méthodes de sélection sur ce pli
  #    Le trace=0 évite d'afficher les étapes à chaque fois
  model_back <- step(glm_full_inner, direction = "backward", trace = 0)
  model_fwd  <- step(glm_intercept_inner, direction = "forward", scope = formula(glm_full_inner), trace
  model_step <- step(glm_full_inner, direction = "both", trace = 0)

  # Évaluation de chaque modèle sur le sous-ensemble de validation du pli

  # Fonction pour calculer la précision
  calculate_accuracy <- function(model, test_data) {
    predictions <- predict(model, newdata = test_data, type = "response")
    predicted_classes <- ifelse(predictions > 0.5, 1, 0)
    true_labels <- test_data$TotalBin

    # Création de facteurs avec les mêmes niveaux pour éviter les erreurs
    pred_labels <- factor(predicted_classes, levels = c(0, 1))
    true_labels_factor <- factor(true_labels, levels = c(0, 1))

    conf_matrix <- table(pred_labels, true_labels_factor)
    accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
    return(accuracy)
  }

  # 4. Stockage des résultats
  accuracies_back[i] <- calculate_accuracy(model_back, test_data_inner)
  accuracies_fwd[i]  <- calculate_accuracy(model_fwd, test_data_inner)
  accuracies_step[i] <- calculate_accuracy(model_step, test_data_inner)
}

# Visualisation

# Rassembler les résultats dans une liste pour le boxplot
accuracies_list <- list(
  "Backward" = accuracies_back,
  "Forward" = accuracies_fwd,
  "Stepwise" = accuracies_step
```

```
)

boxplot(accuracies_list,
        main = "Comparaison de la Stabilité des Méthodes de Sélection (K-Folds)",
        ylab = "Accuracy (Précision)",
        col = c("lightblue", "lightcoral", "lightgreen"),
        las = 1)
```




```
# Affichage des moyennes et écarts-types pour une analyse plus fine
summary_stats <- data.frame(
  Moyenne = sapply(accuracies_list, mean),
  Ecart_Type = sapply(accuracies_list, sd)
)
print(round(summary_stats, 4))
```

```
##           Moyenne Ecart_Type
## Backward   0.9051     0.0142
## Forward    0.9068     0.0168
## Stepwise   0.9051     0.0142
```

L'analyse de la stabilité des modèles de sélection de variables par validation croisée (K-Folds) est très instructive. L'objectif était de comparer les performances et la robustesse des approches Backward, Forward et Stepwise pour établir une solide performance de référence. Les résultats, présentés dans le boxplot ci-dessus et confirmés par les métriques, montrent que les trois méthodes aboutissent à des **performances quasi**

identiques et très stables. Avec une **précision moyenne de 90.5%** et un **faible écart-type de 0.014**, les méthodes Backward et Stepwise sont non seulement performantes mais aussi remarquablement constantes à travers les différents sous-échantillons de données. Leurs résultats identiques suggèrent qu'elles ont convergé vers les mêmes modèles dans la quasi-totalité des plis. La méthode Forward affiche une performance moyenne très légèrement supérieure, mais au prix d'une variabilité également un peu plus élevée, rendant les trois approches indiscernables en pratique. Contrairement à une attente d'instabilité, ces résultats démontrent que les modèles de sélection classiques sont très robustes sur ce jeu de données. Cela nous fournit un baseline de performance très élevé. La justification de notre passage aux méthodes de régularisation change donc de perspective : il ne s'agit plus de corriger une faiblesse, mais de voir si des approches plus modernes peuvent égaler ou surpasser cette excellente performance, tout en apportant leurs avantages spécifiques.

1.3 Modèles de régression logistique régularisée

Tournons nous maintenant vers les méthodes de régularisation. Celles-ci sont conçues pour construire des modèles plus robustes en pénalisant leur complexité, une approche que nous avions déjà explorée dans le TP2 pour la régression linéaire. Nous allons ici l'adapter à la régression logistique en explorant deux approches : **Ridge (L2)** et **Lasso (L1)**.

1.3.1 Préparation des matrices pour `glmnet`

Comme dans le TP2, les modèles régularisés sont sensibles à l'échelle des variables. Il est donc indispensable de **centrer et réduire** nos prédicteurs. Le package `glmnet` requiert également une matrice de prédicteurs `X` et un vecteur de réponse `y`.

```r
# Création des matrices pour les modèles régularisés (glmnet)

X_train <- model.matrix(TotalBin ~ ., data = train_df)[, -1]
y_train <- train_df$TotalBin

# On applique la même logique au jeu de test
X_test <- model.matrix(TotalBin ~ ., data = test_df)[, -1]
y_test <- test_df$TotalBin


train_means <- colMeans(X_train)
train_sds <- apply(X_train, 2, sd)

# Sécurité pour éviter la division par zéro si une variable a une variance nulle
train_sds[train_sds == 0] <- 1

# Appliquons le centrage et la réduction
X_train_scaled <- scale(X_train, center = train_means, scale = train_sds)
X_test_scaled  <- scale(X_test,  center = train_means, scale = train_sds)
```

1.3.2 Régression Ridge (L2)

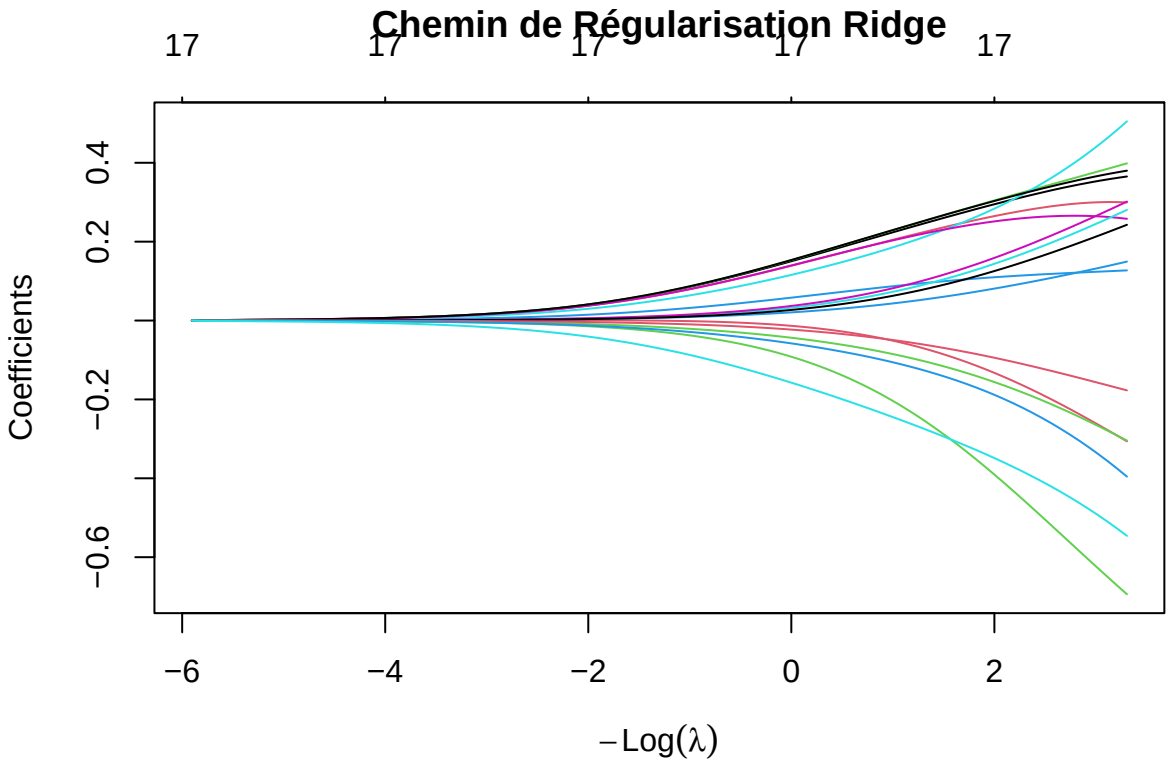
La régression Ridge est particulièrement efficace pour gérer la multicolinéarité. Elle pénalise la somme des carrés des coefficients, ce qui a pour effet de "rétrécir" les coefficients des variables corrélées les uns vers les autres, sans pour autant les annuler. La fonction à minimiser est :

$$\Phi(\beta) = \|Y - X\beta\|_2^2 + \lambda\|\beta\|_2^2$$

1.3.3 Chemin de Régularisation Ridge

Avant de chercher le λ optimal, il est instructif de visualiser le chemin de régularisation. Ce graphique montre l'évolution de chaque coefficient à mesure que la pénalité lambda varie.

```r
ridge_model_path <- glmnet(X_train_scaled, y_train, alpha = 0, family = "binomial")

# On trace le chemin de régularisation
plot(ridge_model_path, xvar = "lambda", label = FALSE, main = "Chemin de Régularisation Ridge")
```



Le graphique montre que lorsque λ augmente (vers la gauche), tous les coefficients convergent vers zéro, mais sans jamais l'atteindre. Cela illustre bien que Ridge conserve toutes les variables, en modulant simplement leur influence.

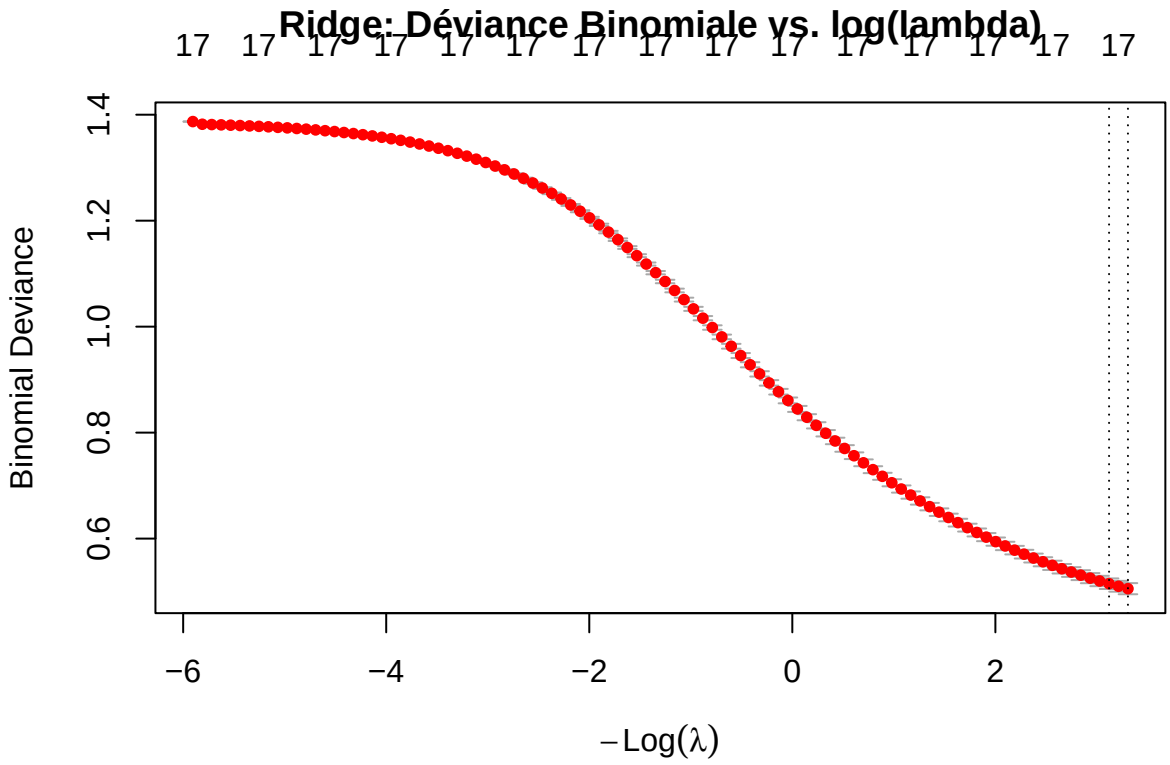
1.3.4 Recherche du λ Optimal (Ridge)

Nous utilisons la validation croisée pour identifier la valeur optimale de λ. L'utilisation de `cv.glmnet` nous donne deux valeurs :

`lambda min` : La valeur qui minimise l'erreur de prédiction moyenne.

`lambda 1se` : La valeur correspondant au modèle le plus parcimonieux (pénalité plus forte) dont la performance se situe à moins d'une erreur standard du minimum (`lambda min`). Ce choix est souvent privilégié pour obtenir un modèle plus simple et potentiellement plus généralisable.

```
cv_ridge <- cv.glmnet(X_train_scaled, y_train,alpha = 0,family = "binomial",nfolds = 10)
plot(cv_ridge, main = "Ridge: Déviance Binomiale vs. log(lambda)")
```




```
cat("Lambda Min (Ridge):", cv_ridge$lambda.min, "\n")
```

```
## Lambda Min (Ridge): 0.03667267
```

```
cat("Lambda 1se (Ridge):", cv_ridge$lambda.1se, "\n")
```

```
## Lambda 1se (Ridge): 0.04417236
```

La validation croisée nous donne deux lambda candidats : lambda.min (0.0367), qui minimise l'erreur de validation croisée, et lambda.1se (0.0442), qui correspond au modèle le plus simple dont la performance reste à moins d'une erreur standard du minimum. Ce dernier est souvent préféré pour sa meilleure capacité de généralisation.

1.3.5 Analyse du modèle :

```
pred_prob_ridge <- predict(cv_ridge, newx = X_test_scaled, s = "lambda.min", type = "response")
pred_ridge <- ifelse(pred_prob_ridge > 0.5, 1, 0)
confusionMatrix(factor(pred_ridge), factor(y_test), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   0   1
##          0 127  13
##          1  19 133
##
##                Accuracy : 0.8904
##                  95% CI : (0.8488, 0.9238)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7808
##
##  Mcnemar's Test P-Value : 0.3768
##
##             Sensitivity : 0.9110
##             Specificity : 0.8699
##          Pos Pred Value : 0.8750
##          Neg Pred Value : 0.9071
##              Prevalence : 0.5000
##          Detection Rate : 0.4555
##    Detection Prevalence : 0.5205
##       Balanced Accuracy : 0.8904
##
##        'Positive' Class : 1
##
```

L'évaluation du modèle Ridge régularisé sur le jeu de test montre une excellente capacité prédictive (Accuracy = 0.890). La sensibilité (0.91) indique que le modèle identifie correctement 91 % des jours à forte production électrique (classe positive), tandis que la spécificité (0.87) reflète sa capacité à reconnaître 87 % des jours à production plus faible. Ces valeurs équilibrées montrent que le modèle ne privilégie pas une classe au détriment de l'autre. L'indice de Kappa (0.781) confirme un accord substantiel entre les prédictions et les observations réelles. Le test de McNemar (p = 0.377) ne met en évidence aucun biais de classification significatif entre les deux classes.

Ces résultats démontrent que la régularisation Ridge permet de stabiliser le modèle logistique tout en conservant un très haut niveau de performance.

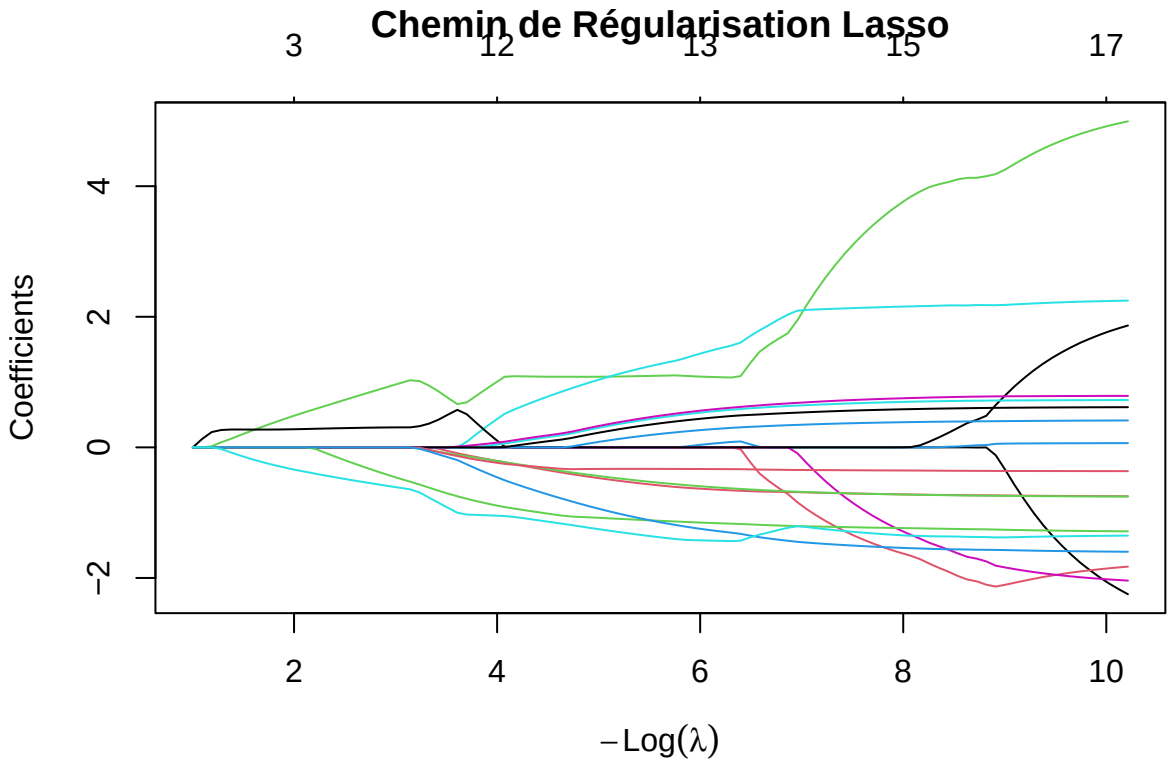
1.3.6 Régression Lasso (L1)

La régression Lasso pénalise la somme des valeurs absolues des coefficients. Sa principale caractéristique est qu'elle peut réduire certains coefficients à exactement zéro, réalisant ainsi une sélection de variables automatique. Elle est très utile pour créer des modèles plus simples (parcimonieux). La fonction à minimiser est :

$$\Phi(\beta) = \|Y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

Avant de chercher le lambda optimal, il est instructif de visualiser le chemin de régularisation. Ce graphique montre l'évolution de chaque coefficient à mesure que la pénalité lambda varie.

```r
lasso_model_path <- glmnet(X_train_scaled, y_train, alpha = 1, family = "binomial")

# On trace le chemin de régularisation
plot(lasso_model_path, xvar = "lambda", label = FALSE, main = "Chemin de Régularisation Lasso")
```

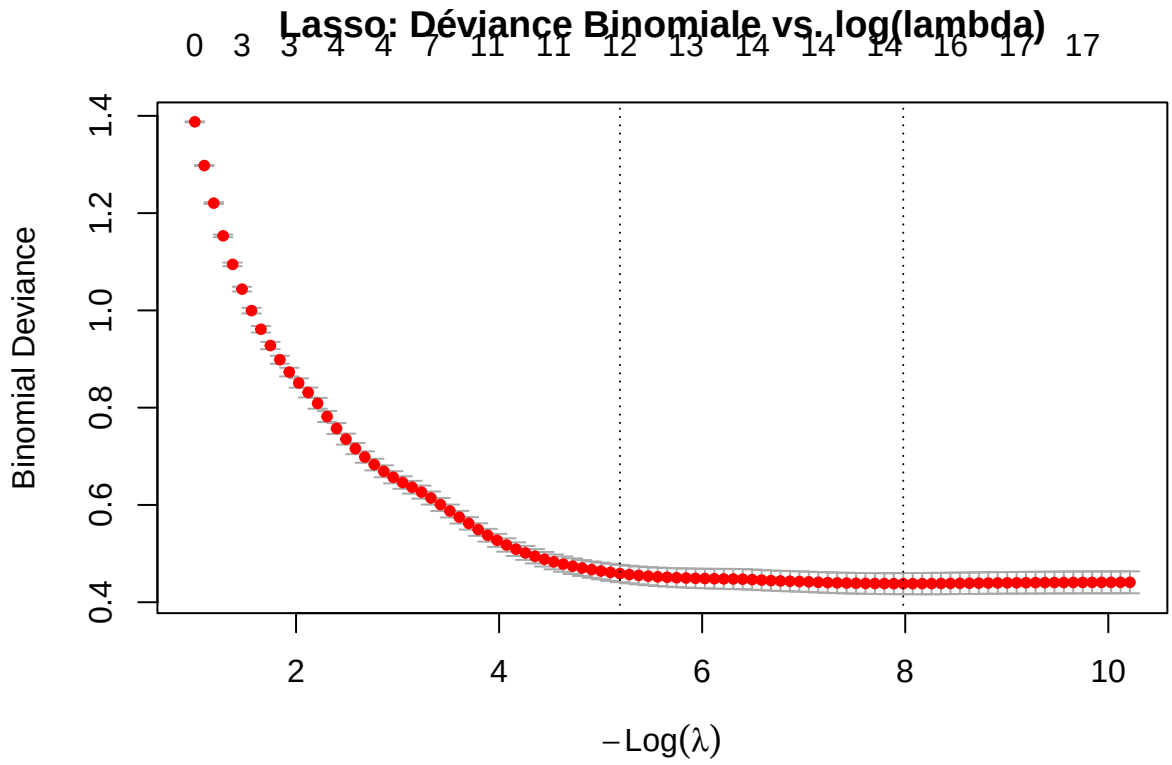


Contrairement à Ridge, on voit ici clairement des coefficients atteindre zéro lorsque la pénalité lambda augmente (vers la gauche). Les variables qui "survivent" le plus longtemps sont considérées comme les plus robustes par le modèle.

1.3.7 Recherche du lambda Optimal (Lasso)

Nous appliquons la même procédure de validation croisée pour trouver le lambda optimal pour le Lasso.

```
cv_lasso <- cv.glmnet(X_train_scaled, y_train,alpha = 1,family = "binomial",nfolds = 10)
plot(cv_lasso, main = "Lasso: Déviance Binomiale vs. log(lambda)")
```




```r
cat("Lambda Min (Lasso):", cv_lasso$lambda.min, "\n")
```

```
## Lambda Min (Lasso): 0.0003420105
```

```r
cat("Lambda 1se (Lasso):", cv_lasso$lambda.1se, "\n")
```

```
## Lambda 1se (Lasso): 0.005573919
```

Nous obtenons ici aussi nos deux lambda candidats, lambda.min (0.00034) et lambda.1se (0.00557), qui seront utilisés dans notre grande comparaison finale.

```r
pred_prob_lasso <- predict(cv_lasso, newx = X_test_scaled, s = "lambda.min", type = "response")
pred_lasso <- ifelse(pred_prob_lasso > 0.5, 1, 0)
confusionMatrix(factor(pred_lasso), factor(y_test),  positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 129  14
##          1  17 132
##
##                Accuracy : 0.8938
##                  95% CI : (0.8527, 0.9267)
```

```
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7877
##
##  Mcnemar's Test P-Value : 0.7194
##
##             Sensitivity : 0.9041
##             Specificity : 0.8836
##          Pos Pred Value : 0.8859
##          Neg Pred Value : 0.9021
##              Prevalence : 0.5000
##          Detection Rate : 0.4521
##    Detection Prevalence : 0.5103
##       Balanced Accuracy : 0.8938
##
##        'Positive' Class : 1
##
```

Le modèle logistique pénalisé par régularisation L1 (Lasso) atteint une précision de 0.894 sur le jeu de test, comparable à celle du modèle Ridge. La sensibilité (0.90) indique que le modèle identifie correctement 90 % des jours à forte production électrique (classe positive), tandis que la spécificité (0.88) reflète sa capacité à reconnaître 88 % des jours à production plus faible. Le test de McNemar ($p = 0.72$) confirme l'absence de biais entre classes, et l'indice Kappa (0.79) reflète un accord substantiel entre les prédictions et les valeurs observées.

1.4 Conclusion Comparative

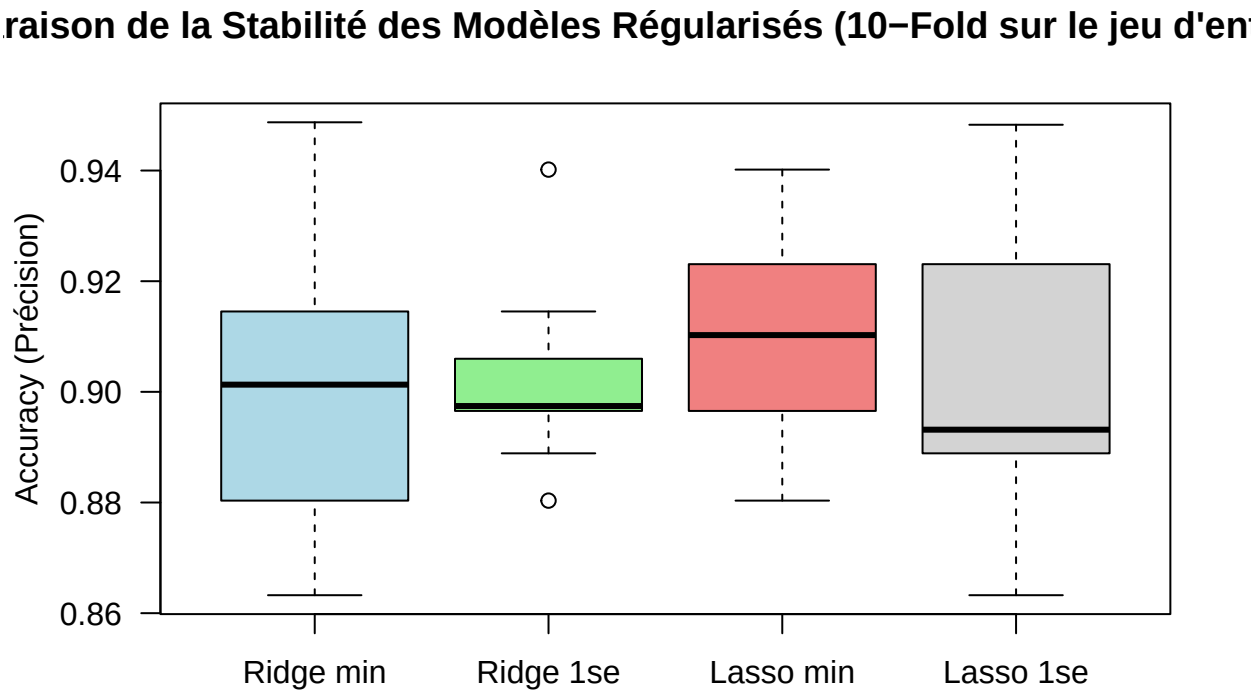
L'objectif de cette analyse était d'identifier le modèle le plus performant et le plus fiable pour prédire la production électrique. Pour ce faire, nous allons adopté une démarche rigoureuse en deux temps : d'abord une validation croisée pour évaluer la robustesse des modèles, puis une évaluation finale sur un jeu de test pour mesurer leur performance prédictive réelle.

1.4.1 Procédure de validation K-Fold

Afin d'évaluer la capacité de généralisation de nos modèles, une procédure de validation croisée K-Fold (avec k=10) a été appliquée sur le jeu d'entraînement.

```
# Stabilité Comparative des Modèles Régularisés par Validation Croisée

# Fonction K-Fold
k_fold_cv_glmnet <- function(X_data, y_data, k = 10, alpha, lambda) {
  fold_indices <- sample(rep(1:k, length.out = nrow(X_data)))
  accuracies <- numeric(k)

  for (i in 1:k) {
    test_indices <- which(fold_indices == i)
    X_train_fold <- X_data[-test_indices, ]
    y_train_fold <- y_data[-test_indices]
    X_test_fold  <- X_data[test_indices, ]
    y_test_fold  <- y_data[test_indices]

    # Entraînement sur (k-1) plis
    model <- glmnet(X_train_fold, y_train_fold, alpha = alpha, lambda = lambda, family = "binomial")

    # Prédiction sur le pli restant. On utilise type="class" pour une prédiction directe.
    pred_class <- predict(model, newx = X_test_fold, s = lambda, type = "class")

    # Calcul de la précision
    accuracies[i] <- mean(pred_class == y_test_fold)
  }
  return(accuracies)
}

# Validation K-Fold sur le jeu d'entraînement complet
set.seed(123) # Pour la reproductibilité
ridge_min_accuracies <- k_fold_cv_glmnet(X_train_scaled, y_train, k = 10, alpha = 0, lambda = cv_ridge$
ridge_1se_accuracies <- k_fold_cv_glmnet(X_train_scaled, y_train, k = 10, alpha = 0, lambda = cv_ridge$
lasso_min_accuracies <- k_fold_cv_glmnet(X_train_scaled, y_train, k = 10, alpha = 1, lambda = cv_lasso$
lasso_1se_accuracies <- k_fold_cv_glmnet(X_train_scaled, y_train, k = 10, alpha = 1, lambda = cv_lasso$

# Rassembler les résultats dans une liste pour le boxplot
accuracies_list_reg <- list(
  "Ridge min" = ridge_min_accuracies,
  "Ridge 1se" = ridge_1se_accuracies,
  "Lasso min" = lasso_min_accuracies,
  "Lasso 1se" = lasso_1se_accuracies
)
```

```r
# Boxplot comparatif
boxplot(
  accuracies_list_reg,
  main = "Comparaison de la Stabilité des Modèles Régularisés (10-Fold sur le jeu d'entraînement)",
  ylab = "Accuracy (Précision)",
  col = c("lightblue", "lightgreen", "lightcoral", "lightgrey"),
  las = 1
)
```




```r
# Moyenne et écart-type pour une analyse plus fine
summary_stats_reg <- round(data.frame(
  Moyenne = sapply(accuracies_list_reg, mean),
  Ecart_Type = sapply(accuracies_list_reg, sd)
), 4)

print(summary_stats_reg)
```

```
##           Moyenne Ecart_Type
## Ridge min  0.9025     0.0246
## Ridge 1se  0.9025     0.0163
## Lasso min  0.9085     0.0181
## Lasso 1se  0.9008     0.0248
```

Les performances issues de la validation croisée K-fold (k = 10) montrent une très bonne capacité de généralisation pour les modèles régularisés. Tous les modèles Ridge et Lasso atteignent une précision moyenne proche de 0.90.

Le modèle Lasso avec λ.min se distingue par sa stabilité remarquable (écart-type = 0.0181), tout en maintenant une performance équivalente aux autres modèles. Ce comportement illustre la capacité du Lasso à sélectionner automatiquement les variables les plus pertinentes, réduisant la complexité sans perte d'efficacité.

En comparaison, les modèles Ridge montrent aussi une précision très compétitive. La régularisation L2 stabilise les coefficients sans simplifier la structure du modèle, contrairement au Lasso qui favorise la parcimonie.

1.4.2 Evaluation Finale :

La confrontation finale a eu lieu sur le jeu de test, où tous les modèles (classiques et régularisés) ont été évalués sur des données qu'ils n'avaient jamais vues.

```
#  Évaluation Finale de Tous les Modèles sur le Jeu de Test

# Préparation des vraies valeurs
true_values <- factor(test_df$TotalBin, levels = c(0, 1))

# 1. Modèle GLM Backward
probabilities_back <- predict(back_model, newdata = test_df, type = "response")
predicted_back <- factor(ifelse(probabilities_back > 0.5, 1, 0), levels = c(0, 1))
cm_back <- confusionMatrix(predicted_back, true_values)

# 2. Modèle GLM Forward
probabilities_fwd <- predict(step_model_fwd, newdata = test_df, type = "response")
predicted_fwd <- factor(ifelse(probabilities_fwd > 0.5, 1, 0), levels = c(0, 1))
cm_fwd <- confusionMatrix(predicted_fwd, true_values)

# 3. Modèle GLM Stepwise
probabilities_step <- predict(step_model_stepwise, newdata = test_df, type = "response")
predicted_step <- factor(ifelse(probabilities_step > 0.5, 1, 0), levels = c(0, 1))
cm_step <- confusionMatrix(predicted_step, true_values)

# 4. Modèle Ridge (lambda.min)
pred_ridge_min <- predict(cv_ridge, newx = X_test_scaled, s = "lambda.min", type = "class")
cm_ridge_min <- confusionMatrix(factor(pred_ridge_min, levels = c(0, 1)), true_values)

# 5. Modèle Ridge (lambda.1se)
pred_ridge_1se <- predict(cv_ridge, newx = X_test_scaled, s = "lambda.1se", type = "class")
cm_ridge_1se <- confusionMatrix(factor(pred_ridge_1se, levels = c(0, 1)), true_values)

# 6. Modèle Lasso (lambda.min)
pred_lasso_min <- predict(cv_lasso, newx = X_test_scaled, s = "lambda.min", type = "class")
cm_lasso_min <- confusionMatrix(factor(pred_lasso_min, levels = c(0, 1)), true_values)

# 7. Modèle Lasso (lambda.1se)
pred_lasso_1se <- predict(cv_lasso, newx = X_test_scaled, s = "lambda.1se", type = "class")
cm_lasso_1se <- confusionMatrix(factor(pred_lasso_1se, levels = c(0, 1)), true_values)

# Création du Tableau Récapitulatif Final

# Extraire les statistiques
stats_back <- cm_back$overall
stats_fwd <- cm_fwd$overall
```

```r
stats_step <- cm_step$overall
stats_ridge_min <- cm_ridge_min$overall
stats_ridge_1se <- cm_ridge_1se$overall
stats_lasso_min <- cm_lasso_min$overall
stats_lasso_1se <- cm_lasso_1se$overall

# Extraire le nombre de variables
get_lasso_vars <- function(cv_model, s_val) {
  coefs <- coef(cv_model, s = s_val)
  return(sum(coefs != 0) - 1)
}

# Créer le data.frame complet
summary_df <- data.frame(
  Modèle = c("GLM Backward", "GLM Forward", "GLM Stepwise", "Ridge (lambda.min)", "Ridge (lambda.1se)",
  Accuracy = c(stats_back['Accuracy'], stats_fwd['Accuracy'], stats_step['Accuracy'], stats_ridge_min['
  Kappa = c(stats_back['Kappa'], stats_fwd['Kappa'], stats_step['Kappa'], stats_ridge_min['Kappa'], sta
  Nbr_Variables = c(
    length(coef(back_model)) - 1,
    length(coef(step_model_fwd)) - 1,
    length(coef(step_model_stepwise)) - 1,
    ncol(X_train),
    ncol(X_train),
    get_lasso_vars(cv_lasso, "lambda.min"),
    get_lasso_vars(cv_lasso, "lambda.1se")
  )
)

# Arrondir pour un affichage propre
summary_df$Accuracy <- round(summary_df$Accuracy, 4)
summary_df$Kappa <- round(summary_df$Kappa, 4)

# Afficher le tableau final
print(summary_df)
```

```
##               Modèle Accuracy  Kappa Nbr_Variables
## 1       GLM Backward   0.8904 0.7808            14
## 2        GLM Forward   0.8973 0.7945            14
## 3       GLM Stepwise   0.8904 0.7808            14
## 4 Ridge (lambda.min)   0.8904 0.7808            17
## 5 Ridge (lambda.1se)   0.8904 0.7808            17
## 6 Lasso (lambda.min)   0.8938 0.7877            15
## 7 Lasso (lambda.1se)   0.9041 0.8082            12
```

Le modèle Lasso avec la pénalité lambda.1se est sans équivoque le modèle le plus performant et le plus équilibré de cette étude. Il offre le meilleur des deux mondes : une performance prédictive supérieure et une interprétabilité accrue grâce à sa simplicité. Sa capacité à éliminer les prédicteurs non essentiels tout en améliorant la généralisation en fait la solution la plus robuste et la plus efficace pour ce problème.

1.4.3 Interprétation des coefficients du modèle Gagnant : Lasso (lambda.1se)

```r
# Extraire les coefficients du meilleur modèle
final_model_coefs <- coef(cv_lasso, s = "lambda.1se")

# Calculer les odd-ratios
odd_ratios <- exp(final_model_coefs)

# Créer un data.frame pour un affichage propre
odd_ratios_df <- data.frame(
  Variable = rownames(odd_ratios),
  Coefficient = final_model_coefs[,1],
  Odd_Ratio = odd_ratios[,1]
)

# Filtrer pour ne garder que les variables sélectionnées par le modèle
final_variables <- odd_ratios_df[odd_ratios_df$Coefficient != 0, ]

# Exclure l'intercept pour l'interprétation des variables
final_variables <- final_variables[-1, ]

# Afficher les résultats
print(final_variables)
```

```
##          Variable Coefficient Odd_Ratio
## T2Mmin     T2Mmin   1.0849977 2.9594331
## SSRD         SSRD   1.1206186 3.0667508
## Covid       Covid  -0.5101139 0.6004272
## Holidays Holidays  -0.4739312 0.6225501
## DOW1         DOW1   0.1271116 1.1355437
## DOW2         DOW2   0.3550517 1.4262544
## DOW3         DOW3   0.3825471 1.4660139
## DOW4         DOW4   0.2751477 1.3167251
## DOW5         DOW5  -0.3291271 0.7195516
## DOW6         DOW6  -1.0934396 0.3350620
## sin_TOY   sin_TOY  -1.0116585 0.3636154
## cos_TOY   cos_TOY  -1.2945738 0.2740146
```

On peut regrouper ces facteurs en trois grandes catégories : météorologiques, contextuels et calendaires. Le modèle Lasso a été ajusté sur des variables centrées et réduites. L'interprétation des odds-ratios qui suit se réfère donc à l'effet d'une augmentation de un écart-type de la variable, et non d'une unité physique (comme 1°C). Cette approche permet de comparer l'influence relative des différents prédicteurs sur une échelle commune.

Facteurs Météorologiques : Les Moteurs Principaux de la Hausse. Les deux variables les plus influentes du modèle sont liées à la météo :

Température Minimale (T2Mmin) : Avec un odd-ratio de **2.96**, cette variable est un prédicteur majeur. Une augmentation de un **écart-type** de la température minimale nocturne (soit une nuit significativement plus chaude que la moyenne) **triple presque** les chances d'une production électrique élevée le lendemain. Cela confirme l'impact important de la chaleur sur l'usage des systèmes de climatisation et de réfrigération.

Rayonnement Solaire (SSRD) : Le rayonnement solaire est le facteur le plus impactant, avec un odd-ratio de **3.07**. Une augmentation de un **écart-type** du rayonnement (soit une journée significativement plus ensoleillée et chaude que la moyenne) **multiplie par plus de trois** les probabilités d'une forte production.

Facteurs Contextuels : Les Freins à la production. Deux variables capturent des événements qui réduisent l'activité économique et sociale :

Jours Fériés (Holidays) : Avec un odd-ratio de **0.62**, les jours fériés **réduisent les chances d'une production élevée de 38%** (1 - 0.622). C'est tout à fait logique, car les activités industrielles et commerciales, grandes consommatrices d'énergie, sont à l'arrêt.

Indice Covid : De même, l'indice de rigueur des mesures Covid a un odd-ratio de **0.60** indique qu'une augmentation de un **écart-type** de l'indice de restriction Covid réduit de **40%**, reflétant l'impact des confinements et des restrictions sur l'économie.

Le Cycle Hebdomadaire : Le Rythme de l'Activité Humaine. Le modèle a parfaitement capturé le rythme de la semaine de travail :

Le cœur de la semaine (Mercredi-Jeudi) : La production atteint son pic en milieu de semaine. Par rapport à un Lundi, les chances d'une production élevée sont **43% à 47% plus importantes** le Mercredi et le Jeudi.

Le Samedi (DOW5) et le Dimanche (DOW6) marquent un net recul, avec des **odds-ratios** de **0.72** et **0.33**, soit respectivement une **réduction de 28% et 67%** des chances de forte production par rapport au Lundi.

Le modèle gagnant ne se contente pas de prédire, il raconte une histoire cohérente et logique.

Synthèse : La production électrique est régi par la météo et le rythme de la société. Elle est fortement poussée à la hausse par la chaleur et le soleil, modulée par le cycle de travail hebdomadaire, et considérablement freinée par les jours de repos et les événements exceptionnels qui mettent l'économie en pause. La capacité du modèle Lasso à isoler ces facteurs pertinents tout en écartant le bruit confirme son statut de meilleur modèle pour cette analyse, fournissant des prédictions précises et des aperçus interprétables.