Projet d'Apprentissage Statistique sur des données issues des protéges-dents des rugbymans

Par Romain Traboul, Marie Aurélie Castel et Marwa Gnaoui

Table of contents

Introduction : Présentation du jeu de données

La base de données que nous avons choisie concerne les informations issues des protège-dents instrumentés durant des matches de rugby de l'équipe de France de moins de 20 ans lors de la compétition des 6 nations 2023. Elle est composée des mesures de vitesse et d'accélération linéaires et angulaires de la tête. Chaque événement induit par le protège-dents est associé à une situation de match détaillée. Chaque situation a été décrite grâce à une grille de codage créée au préalable sur un logiciel d'analyse vidéo "SportCode".

La Fédération française de Rugby a établi différents seuils permettant de déterminer si un événement est extrême ou non. Pour déterminer ces seuils, elle a pris les mesures des vitesses et des accélérations des événements faisant partie du 1% de leur base de données.

Si l'un de ces événements dépasse l'un des quatre seuils, il est alors considéré comme événement extrême, même si, pour l'instant, il n'est pas possible de dire que l'impact qui a induit cet événement met réellement en danger le joueur.

L'objectif de ce travail sera de prédire les situations potentiellement dangereuses à partir des informations disponibles sur notre base de données.

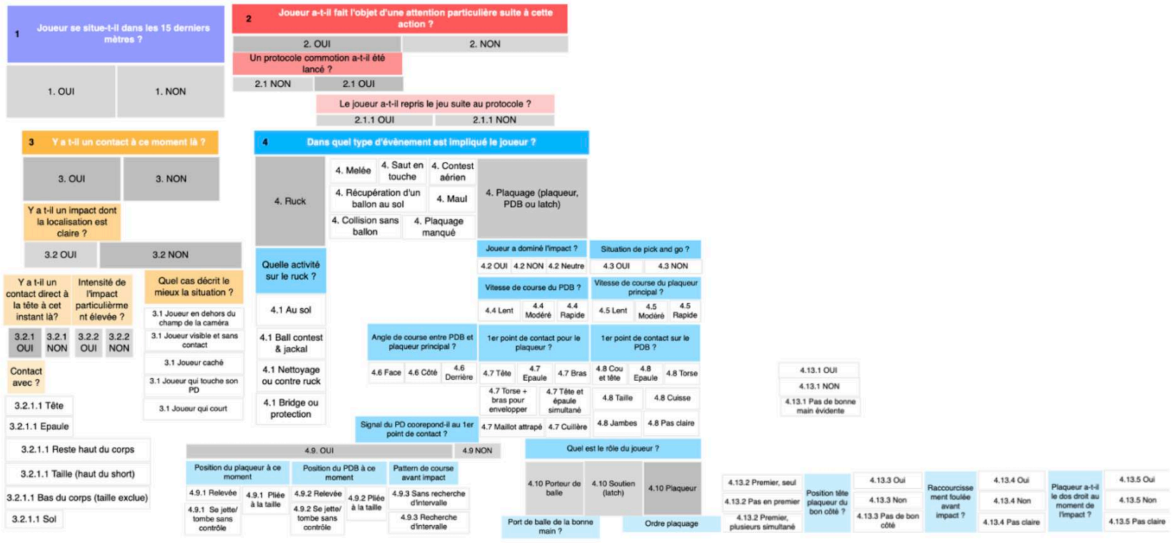


Figure 1: Grille_codage_rugby

Partie 1 : Mise en forme de la base de données

- Chargement des données

```
# Chargement des données initiales contenant uniquement des données numériques
data <- read_delim("BDD/data_pdi.csv", delim = ";")
```

```
New names:
Rows: 226 Columns: 47
-- Column specification
-------------------------------------------------------- Delimiter: ";" chr
(7): Timeline, Row, ID, LocalImpactId, Ungrouped, NAME, Type de situati... dbl
(40): ...1, Start.time, Instance.number, PAA, PAV, PLA, PLV, WLD, TestPL...
i Use `spec()` to retrieve the full column specification for this data. i
Specify the column types or set `show_col_types = FALSE` to quiet this message.
* `` -> `...1`
```

```
#summary(data)
```

Veuillez trouver dans le dossier un pdf récapitulant les correspondance de chaque identifant numériques concernant les variables catégorielles (Dictionnaire_BDD.pdf).

- Transformation du jeu de données (passage en facteurs de certaines variables, 1er tri...

```
# Conversion en forme factor pour les variables de vitesses et
# d'accélérations angulaire de la tête.

data2 <- data |>
  mutate_at(vars(-TestPLA, -TestPAA, -TestPLV, -TestPAV), as.factor)

# 1er tri des données : Suppression de certaines variables
# inutiles pour la suite des opérations

data2 <- data2 |>
  select(!c(0:ID, "WLD", "Crit", "Timeline","Start.time",
            "Row","LocalImpactId", "Ungrouped",))
```

Dans une volonté de ne garder uniquement les variables étant à la fois catégorielles et explicatives, les données quantitatives ont été supprimé du jeu de données. Par la suite, nous avons du mettre en Factor la variable principale “Crit_P” et supprimer les évènements pour lesquels

la colonne Crit_P = 0 (correspondant à l'impact déterminé via l'analyse vidéo mais n'étant pas pour autant détecté par les protèges dents).

```
# Sélection des variables pour le modèle
data_pour_modele <- select(data2, -c(PAA:TestPAV
    , 'Joueur a fait l'objet d'une attention particulière suite à l'action ?'))

# Mise en facteur de la variable Crit_P
data_pour_modele$Crit_P <- factor(data_pour_modele$Crit_P)

# Vérification des niveaux de Crit_P après la première étape
print(levels(data_pour_modele$Crit_P))
```

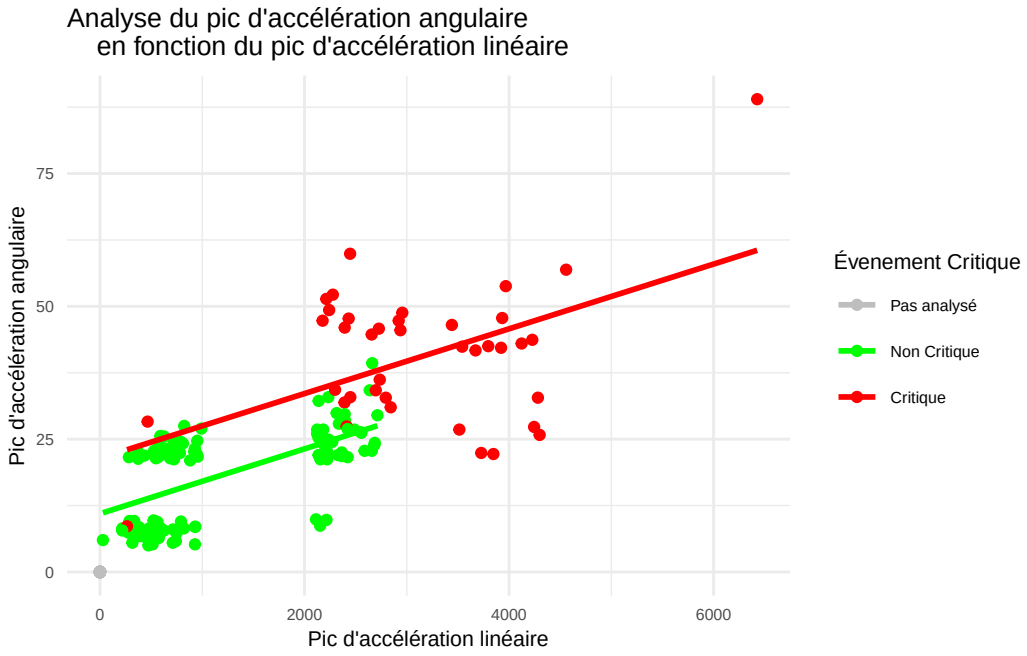
```
[1] "0" "1" "2"
```

```
# Nous remarquons ainsi que la variable Crit_P est composé
# de 3 facteurs, alors qu'il en faut obligatoirement 2.

# Filtrage des faux positifs
data_pour_modele <- data_pour_modele |>
  filter(Crit_P != 0) |>
  mutate(Crit_P = factor(recode(Crit_P, `1` = 0, `2` = 1)))
```

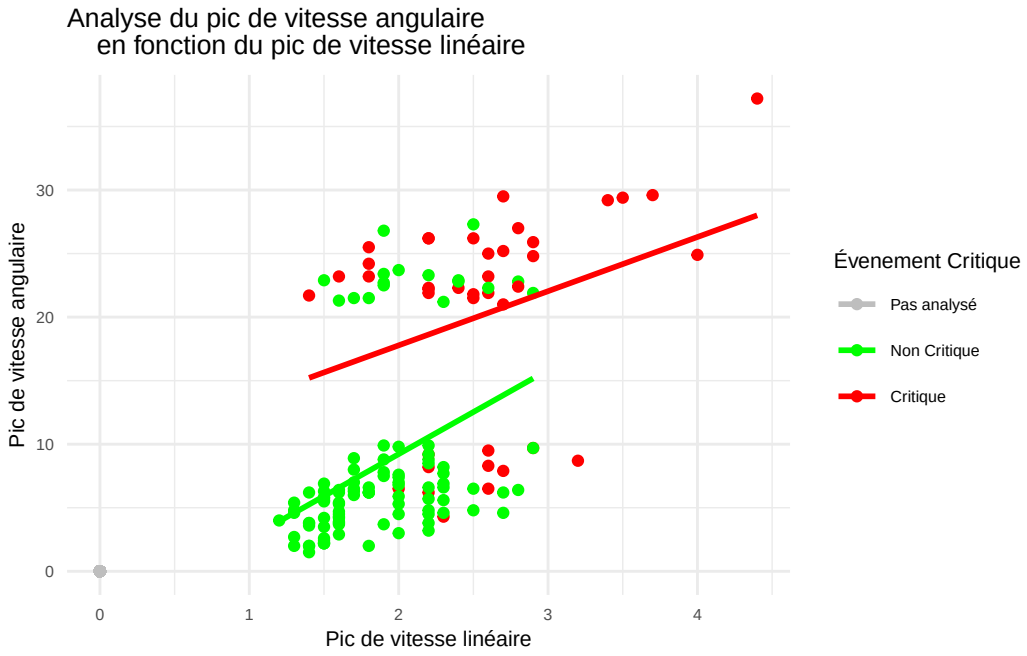
Partie 2 : Quelques graphiques issus des statistiques descriptives

- Nuage de point pour visualisation des événements critiques et non critiques :

```
`geom_smooth()` using formula = 'y ~ x'
```




```
`geom_smooth()` using formula = 'y ~ x'
```



Interprétation : À travers ces deux graphiques, on s'aperçoit que les événements critiques seraient corrélés à des pics d'accélérations et de vitesses plus élevés. Ces données proviennent

d'accéléromètres et du gyroscope, qui mesurent l'accélération et calculent, à partir de cette mesure, la vitesse. Ceci est en accord avec la littérature scientifique sur le sujet.

Partie 3 : Apprentissage supervisé à partir d'1er tri excluant les questions étant trop corrélées entre elles

- 3.1 / Algorithme de prédiction par minimisation de risque empirique

- Utilisation de la régression logistique (GLM : Generalized Linear Model)

```
GLM_model <- glm(Crit_P ~ ., data = data_pour_modele, family = "binomial")
summary(GLM_model)
```

- On selectionne les variables explicatives qui ne sont pas trop corélées avec elles. Pour cela, on exclue donc les variables suivantes que l'on stocke dans le "data_pour_modele2":

```
data_pour_modele2 <- select(
  data_pour_modele,
  -c(
    `Contact avec ?`,
    `Intensité particulièrement élevée ?`,
    `Type de situation de rugby ?`,
    `Pour Ruck : quelle activité au ruck ?`,
    `Pour plaquage : joueur a dominé l'impact ?`,
    `Pour plaquage : situation de pick and go ?`,
    `Pour plaquage : vitesse de course du PDB `,
    `Pour plaquage : vitesse de course du plaqueur `,
    `Pour plaquage : premier point de contact pour le PDB`,
    `Si oui, position du plaqueur à ce moment là`,
    `Si oui, position du PDB à ce moment là`,
    `Si oui, pattern de course avant l'impact`,
    `Pour plaquage : quel est le rôle du joueur ?`,
    `Si PDB : balle du bon côté ?`,
    `Si plaqueur : ordre de plaquage`,
    `Si plaqueur : dos droit ?`,
    `Pour plaquage : premier point de contact pour le plaqueur`
  )
)
```

- Application de GLM pour ce nouveau modèle :

```r
GLM_model <- glm(Crit_P ~ ., data = data_pour_modele2, family = "binomial")
summary(GLM_model)
```

- Application des étapes nécessaires à l'évalution du modèle issu de la régression logistiaue

```r
log_model <- logistic_reg() |>
  set_engine("glm") |>
  set_mode("classification")

# Création du workflow

log_wf <- workflow() |>
  add_model(log_model) |>
  add_formula(Crit_P ~ .)

# Ajustement du modèle au jeu de données

log_fit <- log_wf |>
  fit(data = data_pour_modele2)

# Extraction des coefficients et les p-values avec tidymodels
coefficients_tidy <- log_fit |>
  tidy()

summary_tidy <- log_fit |>
  glance()


# Prédiction de notre modèle
log_class <- log_fit |>
  predict(new_data = data_pour_modele2, type = "class")
```

```
Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading
```

```r
log_prob <- log_fit |>
  predict(new_data = data_pour_modele2,type="prob")
```

```
Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading
```

```r
# Ajout des prédictions aux données d'origine
# avec la fonction `bind_cols()`

data_pour_modele_fit2 <- data_pour_modele2 |>
  bind_cols(log_class,log_prob)

# Préparation de l'évaluation de notre modèle
data_pour_modele_fit2 |> head()

confusion <- yardstick::conf_mat(data_pour_modele_fit2, truth="Crit_P",
                                 estimate=".pred_class",
                                 dnn= c("Prediction", "Truth"))
```

- Évaluation du modèle

```r
# Affichage de la matrice de confusion :
confusion
```

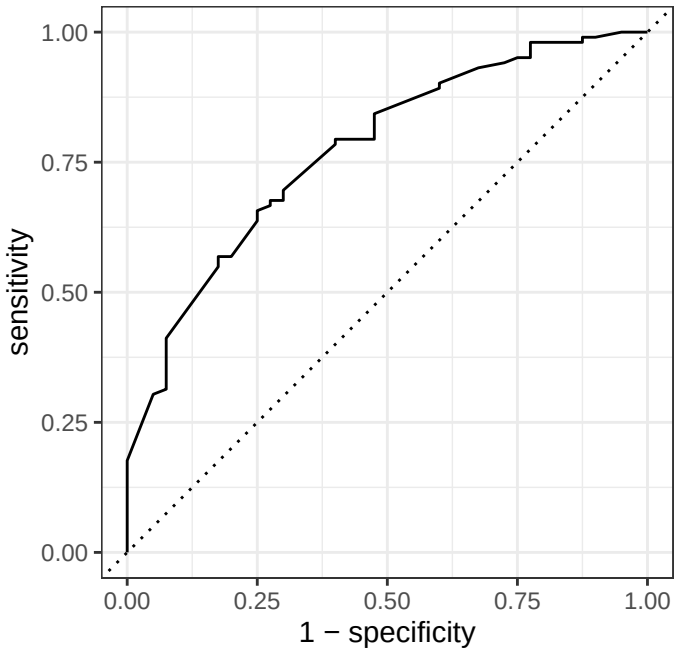
```
          Truth
Prediction  0  1
         0 96 29
         1  6 11
```

```r
summary(confusion)
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.754
 2 kap                  binary         0.262
 3 sens                 binary         0.941
 4 spec                 binary         0.275
 5 ppv                  binary         0.768
 6 npv                  binary         0.647
 7 mcc                  binary         0.300
 8 j_index              binary         0.216
```

```
 9 bal_accuracy         binary         0.608
10 detection_prevalence binary         0.880
11 precision            binary         0.768
12 recall               binary         0.941
13 f_meas               binary         0.846
```

```r
# Affichage de la courbe ROC
data_pour_modele_fit2 |>
  roc_curve(truth = "Crit_P", .pred_0) |>
  autoplot()
```




```r
# Récupération des prédictions de probabilité
predictions_log <- log_prob %>%
  as_tibble() %>%
  select(.pred_0, .pred_1)

truth_values_log <- data_pour_modele_fit2$Crit_P

roc_pred_log <- prediction(predictions_log$.pred_1, truth_values_log)

# Calcule de la performance de la courbe Roc
```

```r
auc_roc_log <- data.frame(Courbe_Roc_log=performance(roc_pred_log, "auc")@y.values[[1]])
auc_roc_log
```

```
  Courbe_Roc_log
1      0.7731618
```

Interprétation : Pour rappel, il s'agit ici des données excluant les questions trop corrélées entre elles (via la fonction glm). À partir de la matrice de confusion, on constate que notre modèle logistique effectue une très bonne prédiction des événements non critiques (94%), tandis qu'il a du mal à discerner les événements critiques (28%).

Ceci est également visible à partir de la courbe ROC, qui montre une spécificité élevée et une sensibilité très faible. On peut penser notamment que cela peut être dû au fait qu'un grand nombre d'événements (critiques ou non) ont été calculés entre 15 et 30 rad/sec. Plus globalement, on obtient une évaluation de la courbe ROC de 0.77. C'est pour cette raison que nous avons décidé d'utiliser d'autres méthodes capables de prédire nos données de façon plus précise.

- Utilisation de la régression logistique Lasso (avec échantillon)

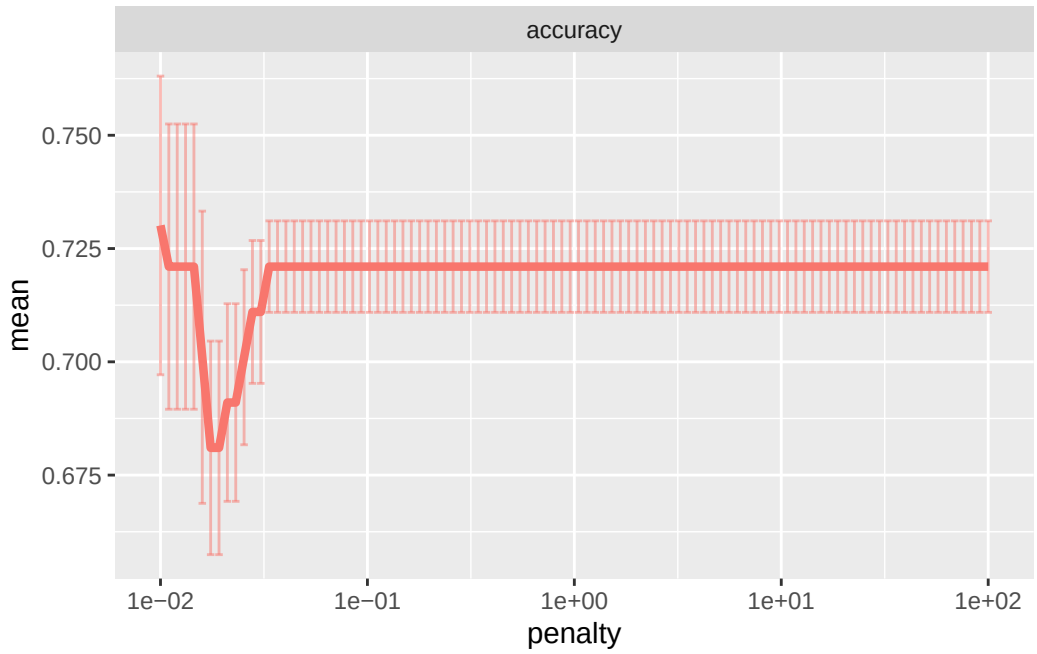
- Réalisation d'un échantillon test et d'un échantillon de validation

```r
# Récupération des indices de notre échantillon de donnée en fonction de Crit_P
nombre_NC <- sum(data_pour_modele2$Crit_P == 0)
nombre_C <- sum(data_pour_modele2$Crit_P == 1)

etiquettes <- ifelse(data_pour_modele2$Crit_P == "1", "1","0")

# Fixation d'une graine aléatoire fixe pour la reproductibilité des résultats
set.seed(123)

index_entrainement <- createDataPartition(etiquettes, p = 0.7, list = FALSE)
donnees_entrainement <- data_pour_modele2[index_entrainement, ]
donnees_validation <- data_pour_modele2[-index_entrainement, ]
```

- Initialisation de la régression Lasso (avec échantillon)

```r
lasso_model <- logistic_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet") |>
  set_mode("classification")
```

```r
# Création d'un workflow pour la régression logistique LASSO
lasso_wf <- workflow() |>
  add_model(lasso_model) |>
  add_formula(Crit_P ~ .)

# Fit du modèle
lasso_fit <- lasso_wf |>
  fit(donnees_entrainement)


# Établissement de la validation croisée avec pour variable d'intêret Crit_P
data_cv <- vfold_cv(donnees_entrainement,v=10, strata=Crit_P)

# On construit une grille afin d'avoir les valeurs de risque estimé
lambda_grid <- grid_regular(penalty(range = c(-2, 2)), levels = 100)

lasso_cv <- lasso_wf |>
  tune_grid(resamples = data_cv, grid = lambda_grid,
            metrics = metric_set(accuracy))

# Estimation du risque

res <- lasso_cv |>
  collect_metrics()

res |>
 ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(
    ymin = mean - std_err,
    ymax = mean + std_err
  ),
  alpha = 0.5
  ) +
  geom_line(linewidth = 1.5) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10() +
  theme(legend.position = "none")
```




```
# Sélectionne le lambda qui minimise le risque estimé par VC
lowest_accuracy <-  lasso_cv |>
  select_best(metric="accuracy")

# On relance (entraine) la régression LASSO avec ce meilleur lambda
# sur le jeu de données de validation pour obtenir
# le "meilleur" prédicteur LASSO :
final_wf <- finalize_workflow(lasso_wf,lowest_accuracy)
print(final_wf)

final_lasso <- final_wf |>
  fit(donnees_validation)

final_lasso |> tidy()


final_lasso |>
  predict(new_data=donnees_validation)

lasso_class <- final_lasso |>
  predict(new_data=donnees_validation,type="class")
```

```r
lasso_prob <- final_lasso |>
  predict(new_data=donnees_validation,type="prob")

predictions <- data.frame(donnees_validation,lasso_class,lasso_prob)

lasso_fit <- donnees_validation |>
  bind_cols(lasso_class, lasso_prob)

lasso_fit |> head()

confusion <- yardstick::conf_mat(data=lasso_fit,truth="Crit_P",
                                 estimate=".pred_class",
                                 dnn = c("Prediction", "Truth"))
```

Interprétation : On remarque que la précision moyenne du modèle serait meilleure lorsque le paramètre de régularisation serait plus faible. Néanmoins, ce dernier aurait, comme on peut le voir ci-dessus, une variance plus élevée, ce qui n'est pas idéal.

- Évaluation du modèle

```r
# Affichage de la matrice de confusion
confusion
```

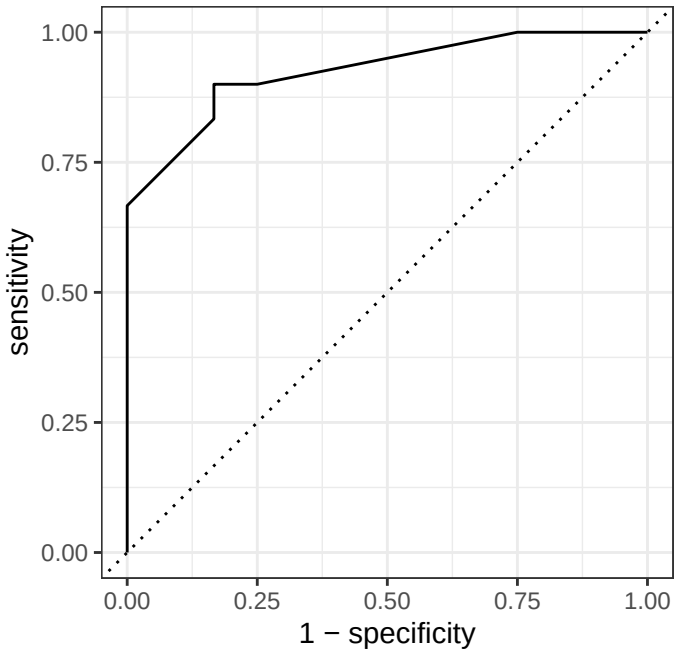
```
          Truth
Prediction  0  1
         0 27  3
         1  3  9
```

```r
summary(confusion)
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.857
 2 kap                  binary         0.65
 3 sens                 binary         0.9
 4 spec                 binary         0.75
 5 ppv                  binary         0.9
 6 npv                  binary         0.75
 7 mcc                  binary         0.65
```

```
 8 j_index              binary         0.65
 9 bal_accuracy         binary         0.825
10 detection_prevalence binary         0.714
11 precision            binary         0.9
12 recall               binary         0.9
13 f_meas               binary         0.9
```

```r
# Construction de la courbe roc
lasso_fit |>
  roc_curve(truth = "Crit_P", .pred_0) |>
  autoplot()
```




```r
# Récupération des prédictions de probabilité
predictions_lasso <- lasso_prob %>%
  as_tibble() %>%
  select(.pred_0, .pred_1)

truth_values_lasso <- lasso_fit$Crit_P

roc_pred_lasso <- prediction(predictions_lasso$.pred_1, truth_values_lasso)
```

```r
# Calcule de la performance de la courbe Roc
auc_roc_lasso <- data.frame(Courbe_Roc_lasso=performance(roc_pred_lasso, "auc")@y.values[[
auc_roc_lasso
```

```
  Courbe_Roc_lasso
1            0.925
```

Interprétation : À partir de la matrice de confusion, on constate que notre modèle logistique effectue une bonne prédiction globale des événements non critiques (90%) et critiques (75%). Ceci est également visible à partir de la courbe ROC, qui montre une spécificité très élevée et une sensibilité élevée. À noter que plus globalement, on obtient même une évaluation de la courbe ROC de 0.93.

Ainsi, l'utilisation de la régression Lasso a fortement contribué à une meilleure sensibilité et à la capacité à discriminer un événement critique d'un non critique.

- 3.2 / Méthode de bagging

- Forêt aléatoire

```r
play_recipe <- recipe(Crit_P ~ ., data = donnees_entrainement)


# Création du modèle et workflow

randomForest_model <- rand_forest(mtry=tune()) |>
  set_engine("ranger") |>
  set_mode("classification")

randomForest_wf <- workflow() |>
  add_recipe(play_recipe) |>
  add_model(randomForest_model)

# Établissement de la méthode de validation croisée
# à partir de la variable d'intêret Crit_P
play_samples_cv <- vfold_cv(donnees_entrainement,v=10,repeats = 1, strata=Crit_P)

# Ajustement du paramètre mtry
par_grid <- tibble(mtry = seq(1,8, by=1)) # Il y a ici 8 variable à analyser

rf_10_fold<- randomForest_wf |>
```

```
  tune_grid(resamples = play_samples_cv,grid = par_grid,
            metrics = metric_set(accuracy))

rf_10_fold |>
  collect_metrics()
```

```
# A tibble: 8 x 7
   mtry .metric  .estimator  mean     n std_err .config
  <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1     1 accuracy binary     0.711    10  0.0158 Preprocessor1_Model1
2     2 accuracy binary     0.701    10  0.0244 Preprocessor1_Model2
3     3 accuracy binary     0.681    10  0.0350 Preprocessor1_Model3
4     4 accuracy binary     0.681    10  0.0350 Preprocessor1_Model4
5     5 accuracy binary     0.681    10  0.0350 Preprocessor1_Model5
6     6 accuracy binary     0.701    10  0.0322 Preprocessor1_Model6
7     7 accuracy binary     0.701    10  0.0193 Preprocessor1_Model7
8     8 accuracy binary     0.701    10  0.0193 Preprocessor1_Model8
```

```
best_mtry <- rf_10_fold |>
  select_best()

final_rf <- randomForest_wf |>
  finalize_workflow(best_mtry) |>
  fit(data = donnees_validation)

# Prédiction et préparation de l'évaluation du modèle
pred <- final_rf |>
  predict(new_data = donnees_validation)

risque_est_rf <- data.frame(Risque_est_rf=mean(donnees_validation$Crit_P!=pred$.pred_class

rf_class <- final_rf |>
  predict(new_data=donnees_validation,type="class")

rf_prob <- final_rf |>
  predict(new_data=donnees_validation,type="prob")


rf_fit <- donnees_validation |>
  bind_cols(rf_class, rf_prob)
```

```r
confusion <- yardstick::conf_mat(data=rf_fit,truth="Crit_P",
                                 estimate=".pred_class",
                                 dnn = c("Prediction", "Truth"))
```

- Évaluation du modèle

```r
# Affichage de la matrice de confusion
confusion
```

```
          Truth
Prediction  0  1
         0 28  4
         1  2  8
```

```r
summary(confusion)
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.857
 2 kap                  binary         0.632
 3 sens                 binary         0.933
 4 spec                 binary         0.667
 5 ppv                  binary         0.875
 6 npv                  binary         0.8
 7 mcc                  binary         0.636
 8 j_index              binary         0.6
 9 bal_accuracy         binary         0.8
10 detection_prevalence binary         0.762
11 precision            binary         0.875
12 recall               binary         0.933
13 f_meas               binary         0.903
```

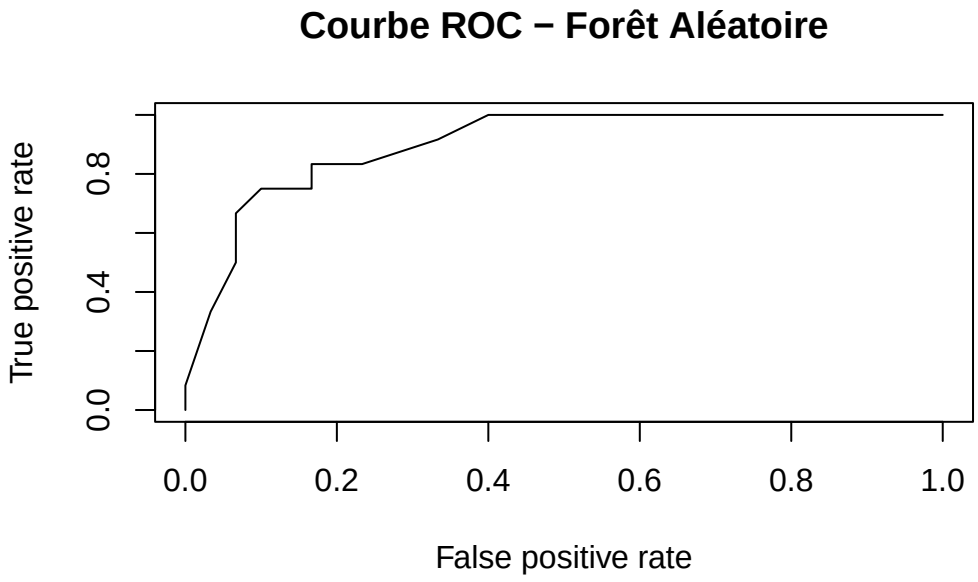
```r
# Affichage du risque estimé
risque_est_rf
```

```
  Risque_est_rf
1     0.1428571
```

```r
# On récupère les prédictions de probabilité
rf_probabilities <- rf_fit %>%
  select(.pred_0, .pred_1)

# On récupère les vraies valeurs
rf_truth_values <- donnees_validation$Crit_P

# On crée un objet ROCR deprediction
rf_roc_pred <- prediction(rf_probabilities$.pred_1, rf_truth_values)

# On calcule l'AUC-ROC
rf_auc_roc <- performance(rf_roc_pred, "auc")@y.values[[1]]

# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", rf_auc_roc))
```

```
[1] "AUC-ROC: 0.901388888888889"
```

```r
# On trace la courbe ROC
rf_roc_perf <- performance(rf_roc_pred, "tpr", "fpr")

plot(rf_roc_perf, main = "Courbe ROC - Forêt Aléatoire")
```



Interprétation : À partir de la matrice de confusion, on constate que notre modèle de forêt aléatoire effectue une très bonne prédiction des événements non critiques (93%) et une prédiction correcte pour ceux étant considérés comme critiques (67%). Au niveau de la courbe Roc, on évalue la performance du modèle de 0.90, ce qui concorde avec l'impression visuel.

Globalement, ce modèle a même obtenu une estimation du risque de seulement 14%, ce qui est tout à fait acceptable compte tenu de nos types de données. Néanmoins, ce dernier reste moins pertinent lorsqu'il s'agit des événements non critiques. Il convient de ce fait de tester d'autres techniques de machine learning.

- Algorithme de Boosting (AdaBoost et XgBoost)

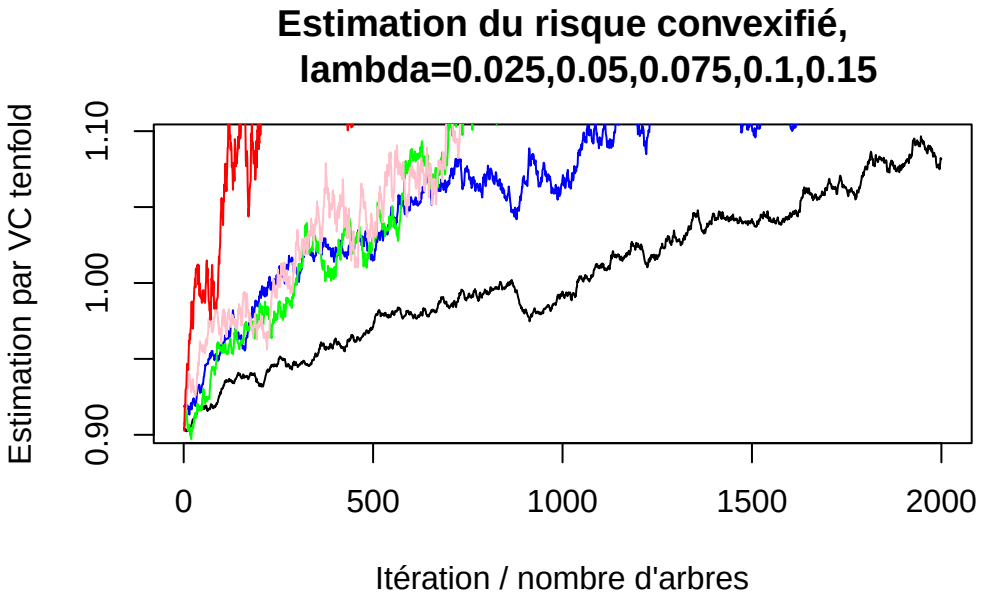
- Mise en forme pour l'algorithme Adaboost:

```
# Transformation de la variable d'intêret en numérique afin de se conformer
# aux caractérisques de la méthode de boosting
dtrain_boost <- donnees_entrainement |>
  mutate(Crit_P=as.numeric(as.character(Crit_P)))

dtest_boost <- donnees_validation |>
  mutate(Crit_P=as.numeric(as.character(Crit_P)))

# Création d'une matrice
Xtrain_boost <- as.matrix(dtrain_boost[,2:length(dtrain_boost)])
Xtest_boost <- as.matrix(dtest_boost[,2:length(dtest_boost)])
Ytrain_boost <- dtrain_boost$Crit_P
```

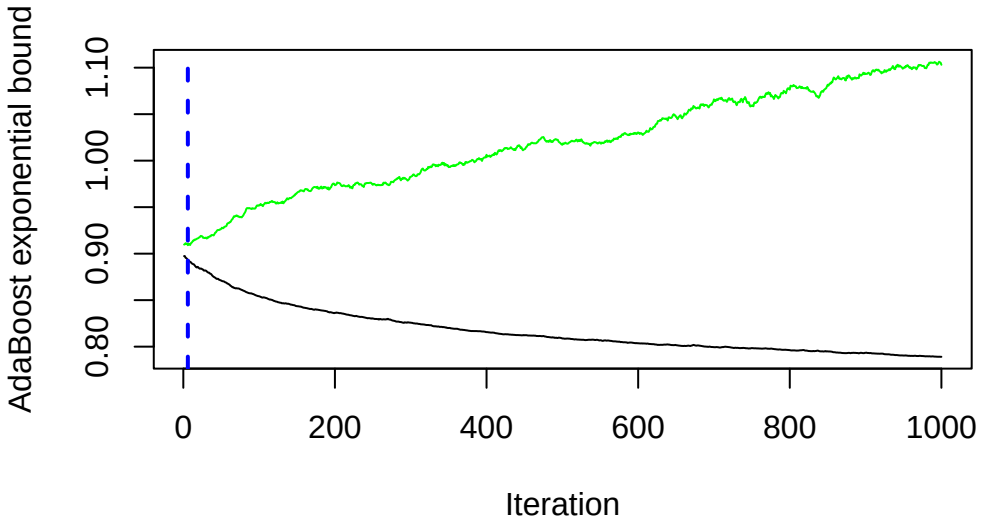
```
Ytest_boost <- dtest_boost$Crit_P
```

- Utilisation de l'algorithme Adaboost

```
### Mise en oeuvre de l'algorithme de gradient
# boosting de type "adaboost" avec le package **gbm**

ada <- gbm::gbm(Crit_P~.,data=dtrain_boost,distribution="adaboost",
                n.trees=2000,interaction.depth=1,shrinkage=0.025,cv.folds=10)

plot(ada$cv.error,type="l",main="Estimation du risque convexifié,
     lambda=0.025,0.05,0.075,0.1,0.15",ylab="Estimation par VC tenfold",
     xlab="Itération / nombre d'arbres")

ada <- gbm::gbm(Crit_P~.,data=dtrain_boost,distribution="adaboost",n.trees=2000,
                interaction.depth=1,shrinkage=0.05,cv.folds=10)
lines(ada$cv.error,col="blue")

ada <- gbm::gbm(Crit_P~.,data=dtrain_boost,distribution="adaboost",n.trees=2000,
                interaction.depth=1,shrinkage=0.075,cv.folds=10)
lines(ada$cv.error,col="green")

ada <- gbm::gbm(Crit_P~.,data=dtrain_boost,distribution="adaboost",n.trees=2000,
                interaction.depth=1,shrinkage=0.1,cv.folds=10)
lines(ada$cv.error,col="pink")

ada <- gbm::gbm(Crit_P~.,data=dtrain_boost,distribution="adaboost",n.trees=2000,
                interaction.depth=1,shrinkage=0.15,cv.folds=10)
lines(ada$cv.error,col="red")
```




```
# Pour obtenir un graphe des estimateurs empirique (en noir)
# et par VC (en vert) du risque convexifié pour lambda=0.025 (par exemple) :

ada <- gbm::gbm(Crit_P~.,data=dtrain_boost,distribution="adaboost",n.trees=1000,
                interaction.depth=1,shrinkage=0.025,cv.folds=10)

Boptada <- gbm::gbm.perf(ada,method="cv")
```



Interprétation : Concernant l'estimation du risque convexifié, on distingue rapidemment que plus on entraine notre modèle, plus l'estimation de risque grandit. Or, dans l'idéal, on aurait du obtenir une courbe décroissante, se rapprochant à terme de 0. Cela signifierait ainsi que cet algorithme ne serait pas le plus adapté à notre type de données

Par ailleurs, on peut également appuyer ce propos à travers le deuxième graphique pour lequel l'estimation du risque convexifié augmente avec la hausse du nombre d'itérations, ce qui atteste d'un apprentissage relativement mauvais de ces données, et ceux qu'importe le lambda choisi.

- Évaluation du modèle

```
# Évaluation des performances de l'algorithme pour lambda=0.025

pred <- predict(ada,newdata=dtest_boost,n.trees=Boptada)

risque.est_ada <- data.frame(Adaboost_gbm_0.025=
                             ModelMetrics::ce(Ytest_boost,as.numeric(pred>0)))

risque.est_ada
```

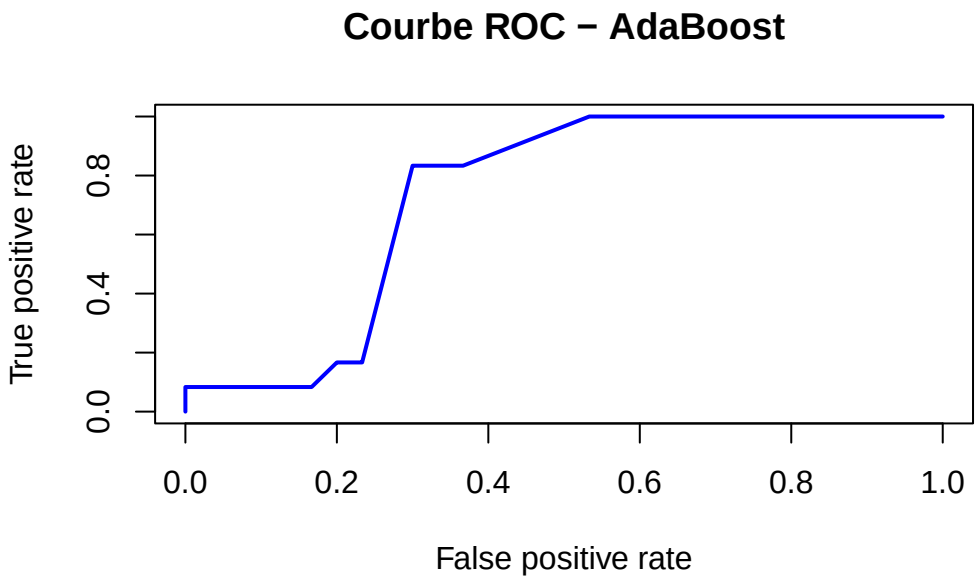
```
  Adaboost_gbm_0.025
1          0.2857143
```

```
# On convertit les prédictions en classe binaire (0 ou 1)
pred_class <- as.numeric(pred > 0)

# On crée un objet de prédiction ROC
roc_pred_ada <- prediction(pred, dtest_boost$Crit_P)

# On calcule l'AUC-ROC
ada_auc_roc <- performance(roc_pred_ada, "auc")@y.values[[1]]

print(paste("AUC-ROC:", ada_auc_roc))
```

```
[1] "AUC-ROC: 0.731944444444445"
```

```
# On calculer la courbe ROC
roc_perf_ada <- performance(roc_pred_ada, "tpr", "fpr")

# On traçe la courbe
```

```r
plot(roc_perf_ada, main = "Courbe ROC - AdaBoost", col = "blue", lwd = 2)
```



Interprétation : De plus, le risque estimé reste nettement plus élevé que celui de la forêt aléatoire (Adaboost : 28%, Forêt Aléatoire : 14%). Ceci est également perceptible à travers la courbe Roc, pour lequel la spécificité et sensibilité semblent faibles. Nous allons donc explorer une méthode sensiblement similaire avec XgBoost.

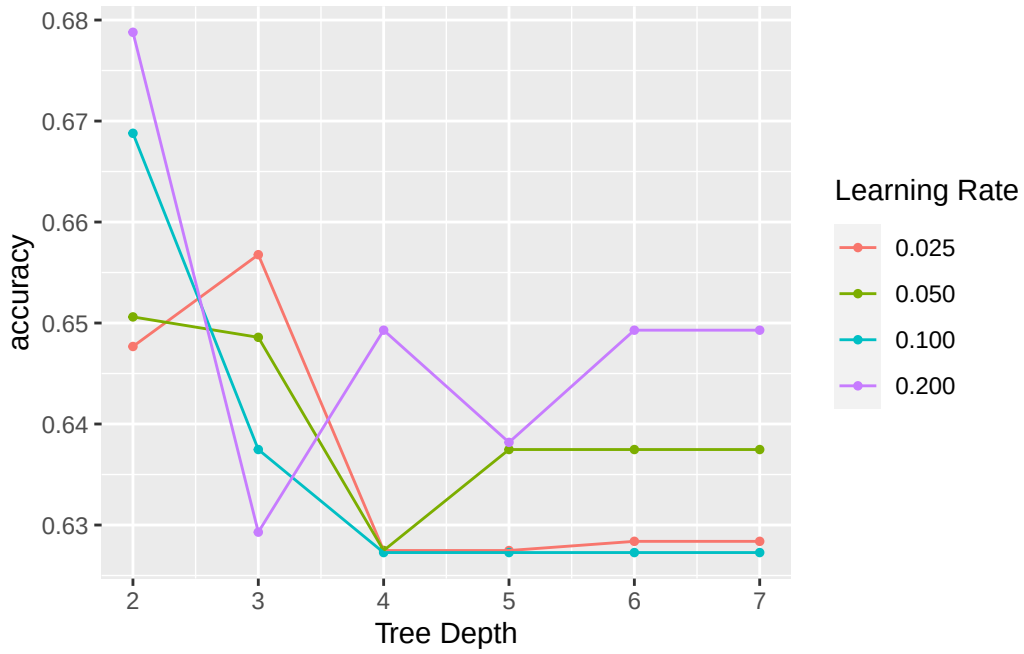
- Mise en forme pour l'algorithme XgBoost

```r
# Convertion de toutes les colonnes en numérique pour donnees_entrainement
donnees_entrainement_num <- donnees_entrainement %>%
  mutate_all(~as.numeric(as.factor(.))) %>%
  mutate(Crit_P = as.factor(Crit_P))

# Convertion de toutes les colonnes en numérique pour donnees_validation
donnees_validation_num <- donnees_validation %>%
  mutate_all(~as.numeric(as.factor(.))) %>%
  mutate(Crit_P = as.factor(Crit_P))
```

- Utilisation de l'algorithme XgBoost

```r
# Ajustement des hyperparamètres de **xgboost**
# à l'aide du package **tidymodels**
```

```r
data_pour_modele_2_recipe <-
  recipe(Crit_P ~ ., data = donnees_entrainement_num)

# Création du modèle et du workflow
xgb_model <-
  boost_tree(
    trees = 500,
    stop_iter=20,
    min_n = 1,
    tree_depth = tune(),
    learn_rate = tune()
  ) |>
  set_mode("classification") |>
  set_engine("xgboost")

xgb_wf <- workflow() |>
  add_recipe(data_pour_modele_2_recipe) |>
  add_model(xgb_model)

# Établissement de la méthode de la validation croisée
# à partir de la variable d'intêret Crit_P
play_samples_cv <- vfold_cv(donnees_entrainement_num, v = 10,
                            repeats = 1, strata = "Crit_P")

xgb_grid <- expand.grid(tree_depth = c(2,3,4,5,6,7),
                        learn_rate = c(0.025,0.05,0.1,0.2))

xgb_10fold <- xgb_wf |>
  tune_grid(resamples = play_samples_cv, grid = xgb_grid,
            metrics = metric_set(accuracy))

xgb_10fold |> autoplot()
```




```
best_par <- xgb_10fold |>
  select_best()

final_xgb <- xgb_wf |>
  finalize_workflow(best_par) |>
  fit(data = donnees_entrainement_num)

# Prédiction du modèle
pred_xgb <- final_xgb |>
  predict(new_data = donnees_validation_num)

xgb_class <- final_xgb |>
  predict(new_data=donnees_validation_num,type="class")

xgb_prob <- final_xgb |>
  predict(new_data=donnees_validation_num,type="prob")


final_xgb <- donnees_validation_num |>
  bind_cols(xgb_class, xgb_prob)
```

```r
confusion_xgb <- yardstick::conf_mat(data=final_xgb,truth="Crit_P",
                                     estimate=".pred_class",
                                     dnn = c("Prediction", "Truth"))
```

Interprétation : Concernant le graphique de la précision du modèle XgBoost en fonction de la profondeur de l'arbre, on constate que la courbe ayant un taux d'apprentissage de 0.2 garantirait globalement une meilleure précision, quelle que soit la profondeur des arbres.

- Évaluation du modèle

```r
# Estimation du risque
risque.est_xgb <- data.frame(Risque_est_xbg=mean(
  donnees_validation_num$Crit_P != pred_xgb$.pred_class))

print(risque.est_xgb)
```

```
  Risque_est_xbg
1      0.3571429
```

```r
# Affichage de la matrice de confusion
confusion_xgb
```

```
          Truth
Prediction  1  2
         1 25 10
         2  5  2
```

```r
summary(confusion_xgb)
```

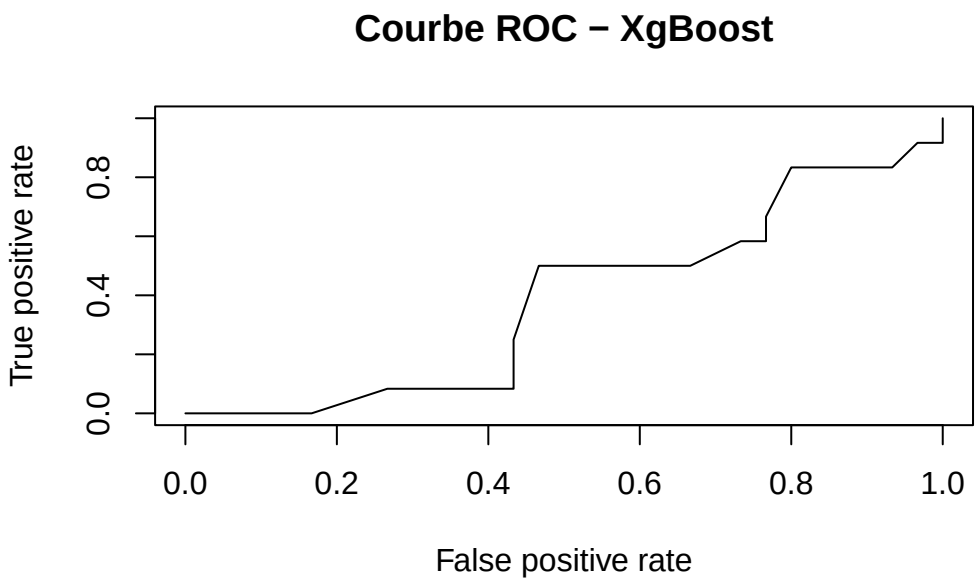
```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.643
 2 kap                  binary         0
 3 sens                 binary         0.833
 4 spec                 binary         0.167
 5 ppv                  binary         0.714
 6 npv                  binary         0.286
```

```
 7 mcc                  binary         0
 8 j_index              binary         0
 9 bal_accuracy         binary         0.5
10 detection_prevalence binary         0.833
11 precision            binary         0.714
12 recall               binary         0.833
13 f_meas               binary         0.769
```

```r
# On récupère les prédictions de probabilité
xgb_probabilities <- final_xgb %>%
  select(.pred_1, .pred_2)

# Récupération des vraies valeurs
xgb_truth_values <- donnees_validation_num$Crit_P

# Prédiction du modèle
xgb_roc_pred <- prediction(xgb_probabilities$.pred_1, xgb_truth_values)

xgb_auc_roc <- performance(xgb_roc_pred, "auc")@y.values[[1]]

# On affiches la valeur de l'AUC-ROC
print(paste("AUC-ROC:", xgb_auc_roc))
```

```
[1] "AUC-ROC: 0.381944444444444"
```

```r
xgb_roc_perf <- performance(xgb_roc_pred, "tpr", "fpr")

# On trace la courbe ROC
plot(xgb_roc_perf, main = "Courbe ROC - XgBoost")
```



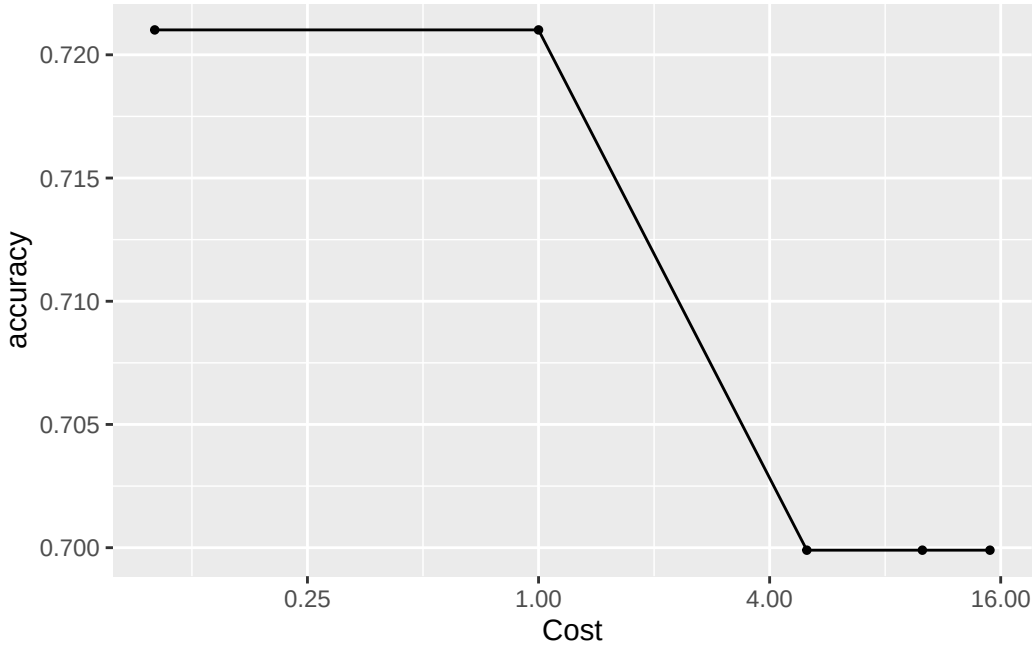
Interprétation : À partir de la matrice de confusion, on constate que notre modèle XgBoost effectue une bonne prédiction des événements non critiques (83%) et une mauvaise prédiction pour ceux étant considérés comme critiques (17%). Globalement, ce modèle a obtenu une estimation du risque de seulement 36%, ce qui est tout à fait faible en comparaison avec les autres algorithmes utilisés. Concernant la courbe Roc, on obtient une évaluation de performance du modèle de 0.29, ce qui concorde avec ceux qui a été observé précédemment.

- 3.3 / Bonus : Algorithme de prédiction par minimisation de risque empirique pénalisé : SVM

- Utilisation de SVM linéaire

```
# Création du modèle et du workflow
svml_model <-
     svm_linear(cost = tune()) |>
     set_mode("classification") |>
     set_engine("kernlab")
svml_wf <- workflow() |>
     add_recipe(play_recipe) |>
     add_model(svml_model)

# Ajustement de la constante de tolérance C par validation croisée
par_grid <- tibble(cost=c(0.1,1,5,10,15))
```

```
svml_cv <- svml_wf |>
    tune_grid(resamples=play_samples_cv,grid=par_grid,
                metrics=metric_set(accuracy))
svml_cv |>
  autoplot()
```




```
svml_cv |>
    collect_metrics()
```

```
# A tibble: 5 x 7
   cost .metric  .estimator  mean     n std_err .config
  <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1   0.1 accuracy binary     0.721    10  0.0101 Preprocessor1_Model1
2   1   accuracy binary     0.721    10  0.0101 Preprocessor1_Model2
3   5   accuracy binary     0.700    10  0.0144 Preprocessor1_Model3
4  10   accuracy binary     0.700    10  0.0144 Preprocessor1_Model4
5  15   accuracy binary     0.700    10  0.0144 Preprocessor1_Model5
```

```
# Évaluation de la performance du modèle
best_C <- svml_cv |>
```

```
  select_best()
final_svml <- svml_wf |>
  finalize_workflow(best_C) |>
  fit(data = donnees_entrainement)
```

```
 Setting default kernel parameters
```

```
Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svml_class <- final_svml |>
    predict(new_data=donnees_validation,type="class")
svml_prob <- final_svml |>
    predict(new_data=donnees_validation,type="prob")
svml_fit <- donnees_validation |>
  bind_cols(svml_class, svml_prob)

accuracy_result <- yardstick::accuracy(
  data = svml_fit,
  truth = "Crit_P",
  estimate = ".pred_class",
  dnn = c("Prediction", "Truth")
)


pred_svml <- final_svml |>
     predict(new_data=donnees_validation)
```

Interprétation : On observe à travers ce graphique que la plus grande précision pour notre modèle est établie à partir d'un coût de 0.25, tandis qu'un coût supérieur semble dégrader sa performance.

- Évaluation du modèle

```
# Estimation du risque
risque.est_svml <- data.frame(Risque_est_svml=mean(
  donnees_validation$Crit_P!=pred_svml$.pred_class))

print(risque.est_svml)
```

```
  Risque_est_svml
1       0.2857143
```

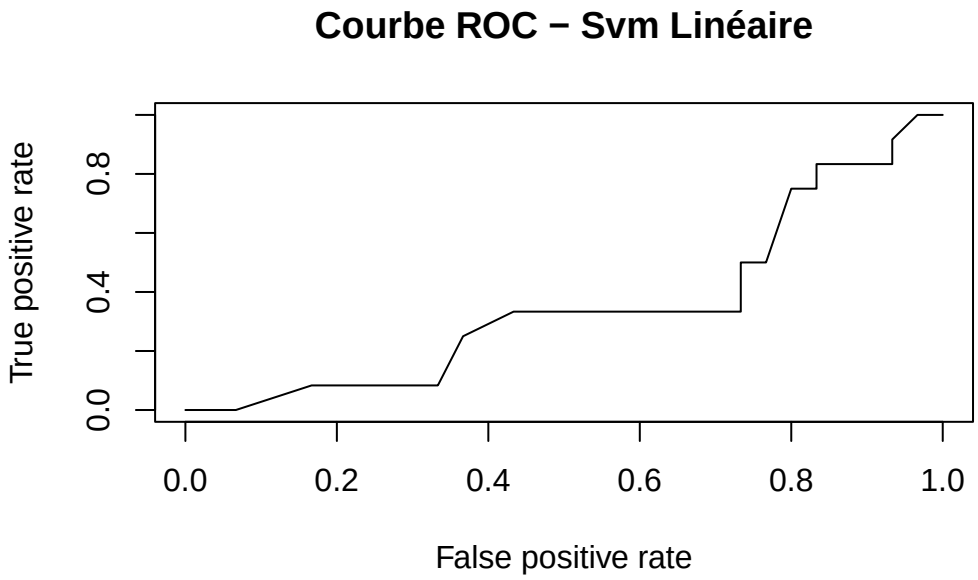
```r
# Accuracy du modèle
print(accuracy_result)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.714
```

```r
# On récupère les prédictions de probabilité
svml_probabilities <- svml_fit %>%
  select(.pred_0, .pred_1)

# On récupère les vraies valeurs
svml_truth_values <- donnees_validation$Crit_P

# On crée un objet ROCR prediction
svml_roc_pred <- prediction(svml_probabilities$.pred_1, svml_truth_values)

# On calcule l'AUC-ROC
svml_auc_roc <- performance(svml_roc_pred, "auc")@y.values[[1]]

# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", svml_auc_roc))
```

```
[1] "AUC-ROC: 0.354166666666667"
```

```r
# On trace la courbe ROC
svml_roc_perf <- performance(svml_roc_pred, "tpr", "fpr")

plot(svml_roc_perf, main = "Courbe ROC - Svm Linéaire")
```



Interprétation :

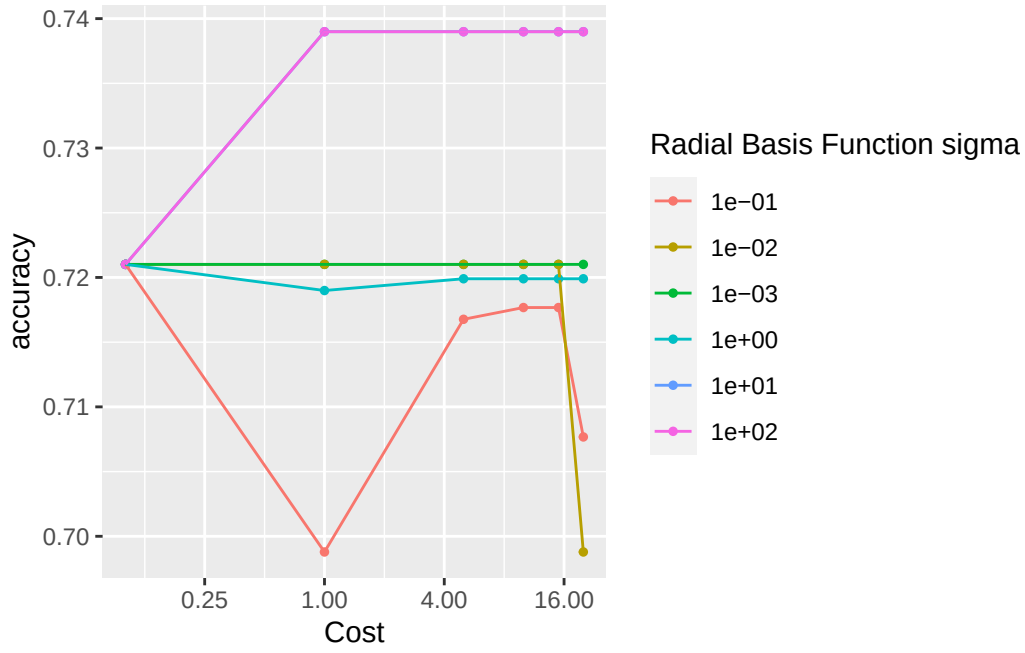
Globalement, ce modèle a obtenu une estimation du risque de 28%, ce qui est relativement faible compte tenu de nos types de données et des résultats des autres algorithmes.

Ceci est également visible à partir de la courbe ROC où l'on observe que la courbe ne s'approche pas de l'angle supérieur gauche du graphique, ce qui traduirait une spécificité et une sensibilité faibles. De ce fait, ce modèle ne serait donc pas idéal (0.35 de performance) en vue de prédire ce type de données.

- Utilisation de SVM non linéaire à noyau gaussien (ou radial)

```
# Création du modèle et du workflow
svmr_model <-
     svm_rbf(cost = tune(),rbf_sigma=tune()) |>
     set_mode("classification") |>
     set_engine("kernlab")

svmr_wf <- workflow() |>
     add_recipe(play_recipe) |>
     add_model(svmr_model)

# Ajustement de la constante de tolérance C et
# de la fenêtre (variance) du noyau gaussien par validation croisée

par_grid <- expand.grid(cost=c(0.1,1,5,10,15,20),rbf_sigma=10^(-3:2))
```

```
svmr_cv <- svmr_wf |>
     tune_grid(resamples=play_samples_cv,grid=par_grid,
               metrics=metric_set(accuracy))

svmr_cv|> autoplot()
```




```
svmr_cv |>
    collect_metrics()
```

```
# A tibble: 36 x 8
    cost rbf_sigma .metric  .estimator  mean     n std_err .config
   <dbl>     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1   0.1     0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model01
 2   1       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model02
 3   5       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model03
 4  10       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model04
 5  15       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model05
 6  20       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model06
 7   0.1     0.01  accuracy binary     0.721    10  0.0101 Preprocessor1_Model07
 8   1       0.01  accuracy binary     0.721    10  0.0101 Preprocessor1_Model08
 9   5       0.01  accuracy binary     0.721    10  0.0101 Preprocessor1_Model09
```

```
10  10       0.01  accuracy binary     0.721    10  0.0101 Preprocessor1_Model10
# i 26 more rows
```

```
# Évaluation de la performance du modèle
best_par<- svmr_cv |>
    select_best()

final_svmr <- svmr_wf |>
    finalize_workflow(best_par) |>
    fit(data = donnees_entrainement)
```

```
Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svmr_class <- final_svmr |>
    predict(new_data=donnees_validation,type="class")

svmr_prob <- final_svmr |>
    predict(new_data=donnees_validation,type="prob")

svmr_fit <- donnees_validation |>
    bind_cols(svmr_class,svmr_prob)


pred_svmr <- final_svmr |>
     predict(new_data=donnees_validation)
```

Interprétation : Le graphique traitant de la précision du modèle SVM à noyau gaussien (via divers sigma) en fonction du coût nous montre qu'un noyau égal à 1e+02 nous fournirait une meilleure performance du modèle, quel que soit son coût. Cela permettrait ainsi de faire un compromis entre le sous-apprentissage et le sur-apprentissage du modèle.

- Évaluation du modèle

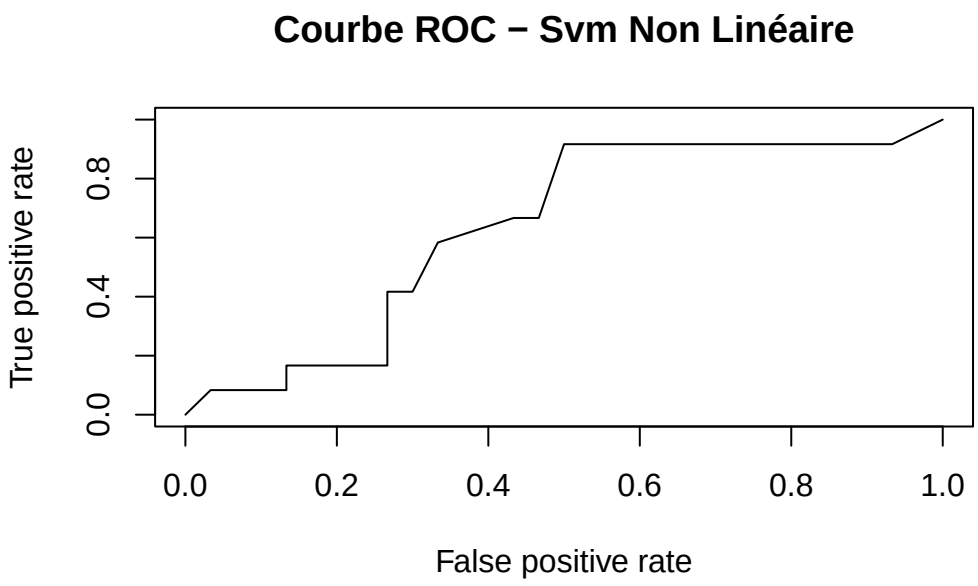
```
# Estimation du risque
risque.est_svmr <- data.frame(Risque_est_svmr=mean(donnees_validation$Crit_P!=pred_svmr$.p
print(risque.est_svmr)
```

```
  Risque_est_svmr
1       0.3571429
```

```r
# On récupère les prédictions de probabilité
svmr_probabilities <- svmr_fit %>%
  select(.pred_0, .pred_1)

# On récupère les vraies valeurs
svmr_truth_values <- donnees_validation$Crit_P

# On crée un objet ROCR prediction
svmr_roc_pred <- prediction(svmr_probabilities$.pred_1, svmr_truth_values)

# On calcule l'AUC-ROC
svmr_auc_roc <- performance(svmr_roc_pred, "auc")@y.values[[1]]

# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", svmr_auc_roc))
```

```
[1] "AUC-ROC: 0.634722222222222"
```

```r
# On trace la courbe ROC
svmr_roc_perf <- performance(svmr_roc_pred, "tpr", "fpr")

plot(svmr_roc_perf, main = "Courbe ROC - Svm Non Linéaire")
```



Interprétation : Globalement, ce modèle a obtenu une estimation du risque de 37%, ce qui est relativement élevé compte tenu de nos types de données et des résultats des autres algorithmes.

Ceci est également visible à partir de la courbe ROC, qui montre une spécificité faible avec tout de même une sensibilité correcte (0.63 de performance). Ainsi, l'utilisation de l'algorithme SVM non linéaire à noyau gaussien serait acceptable afin de discriminer un événement critique d'un non critique, mais semble plus limitée pour le reste.

Partie 4 : Apprentissage supervisé des variables ayant déjà été validé grâce à une analyse statistique descriptive réalisée en amont du devoir

- 4.1 / Algorithme de prédiction par minimisation de risque empirique

- Utilisation de la régression logistique (GLM : Generalized Linear Model)

- On sélectionne par la suite les évènements qui représente les situations critiques ayant une incidence sur l'apparition des évènements critiques lors d'un match (**`data_pour_modele4`**). Ces situations ont été validé grace à une analyse statistique descriptive réalisée sur le logiciel JAMOVI avec les tests CHI 2 et V de Cramer.

```
data_pour_modele4 <- select(
  data_pour_modele,
  c(
    Crit_P,
    `Intensité particulièrement élevée ?`,
    `Pour plaquage : angle de course entre PDB et plaqueur`,
    `Si oui, position du plaqueur à ce moment là`,
    `Si plaqueur : ordre de plaquage`,
    `Pour plaquage : quel est le rôle du joueur ?`
  )
)
```

- Application de GLM pour ce modèle :

```
GLM_model_4 <- glm(Crit_P ~ ., data = data_pour_modele4, family = "binomial")
```

- Utilisation de la régression logistique Lasso (avec échantillon)

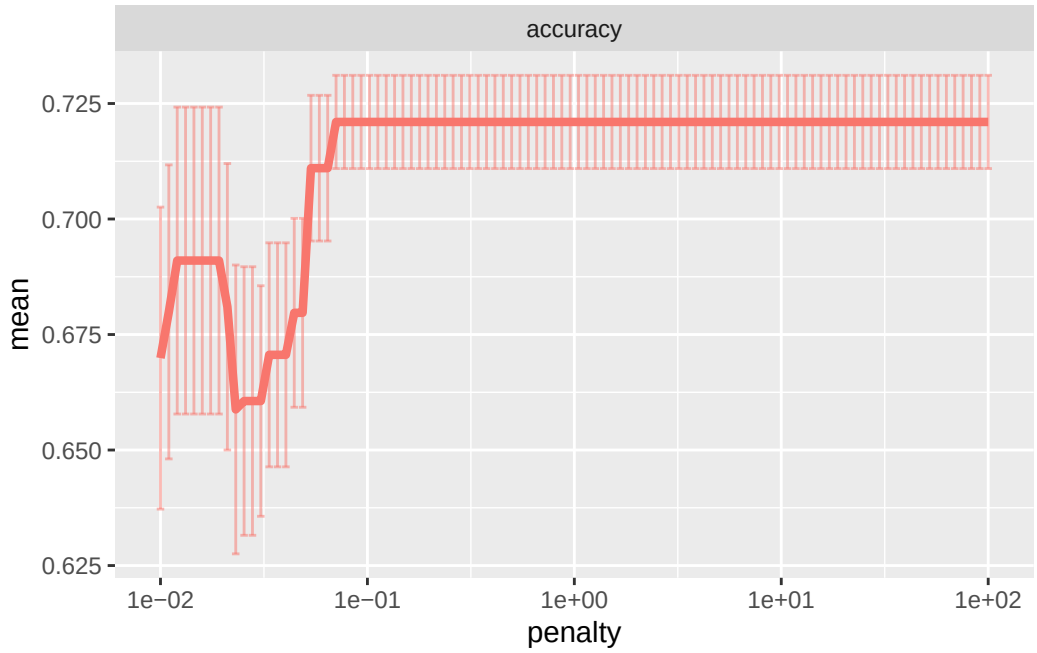
- Création d'un échantillon d'entraînement et d'un échantillon de validation :

```
nombre_NC_2 <- sum(data_pour_modele4$Crit_P == 0)
nombre_C_2 <- sum(data_pour_modele4$Crit_P == 1)


etiquettes <- ifelse(data_pour_modele4$Crit_P == "1", "1","0")
set.seed(123) # Pour la reproductibilité des résultats
index_entrainement_2 <- createDataPartition(etiquettes, p = 0.7, list = FALSE)
donnees_entrainement_2 <- data_pour_modele4[index_entrainement, ]
donnees_validation_2 <- data_pour_modele4[-index_entrainement, ]
```

- Régression logistique Lasso :

```
lasso_model_2 <- logistic_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet") |>
  set_mode("classification")

# Création d'un workflow pour la régression logistique LASSO
lasso_wf_2 <- workflow() |>
  add_model(lasso_model_2) |>
  add_formula(Crit_P ~ .)

# Fit du modèle
lasso_fit_2 <- lasso_wf_2 |>
  fit(donnees_entrainement_2)

# Établissement de la méthode de validation croisée
data_cv_2 <- vfold_cv(donnees_entrainement_2,v=10, strata=Crit_P)

# CPour avoir les valeurs de risque estimé
lambda_grid_2 <- grid_regular(penalty(range = c(-2, 2)), levels = 100)

lasso_cv_2 <- lasso_wf_2 |>
  tune_grid(resamples = data_cv_2, grid = lambda_grid_2,
            metrics = metric_set(accuracy))

res_2 <- lasso_cv_2 |>
  collect_metrics()

res_2 |>
 ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(
```

```
    ymin = mean - std_err,
    ymax = mean + std_err
  ),
  alpha = 0.5
  ) +
  geom_line(linewidth = 1.5) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10() +
  theme(legend.position = "none")
```




```
# Sélectionne le lambda qui minimise le risque estimé par VC
lowest_accuracy_2 <-  lasso_cv_2 |>
  select_best(metric="accuracy")

# On relance (entraine) la régression LASSO avec ce meilleur lambda
# sur le jeu de données de validation pour obtenir
# le "meilleur" prédicteur LASSO :
final_wf_2 <- finalize_workflow(lasso_wf_2,lowest_accuracy_2)
print(final_wf_2)

final_lasso_2 <- final_wf_2 |>
  fit(donnees_validation_2)
```

```r
final_lasso_2 |> tidy()

# Prédiction du modèle

final_lasso_2 |>
  predict(donnees_validation_2)

lasso_class_2 <- final_lasso_2 |>
  predict(donnees_validation_2,type="class")

lasso_prob_2 <- final_lasso_2 |>
  predict(donnees_validation_2,type="prob")

predictions_2 <- data.frame(donnees_validation_2,lasso_class_2,lasso_prob_2)

lasso_fit_2 <- donnees_validation_2 |>
  bind_cols(lasso_class_2, lasso_prob_2)

lasso_fit_2 |> head()

confusion_2 <- yardstick::conf_mat(data=lasso_fit_2,truth="Crit_P",
                                   estimate=".pred_class",
                                   dnn = c("Prediction", "Truth"))
```

Interprétation : On remarque que la précision moyenne du modèle serait meilleure lorsque le paramètre de régularisation serait supérieur à 0.1, avec une variance plutôt faible, ce qui est acceptable compte tenu du caractère de nos données.

- Évaluation du modèle

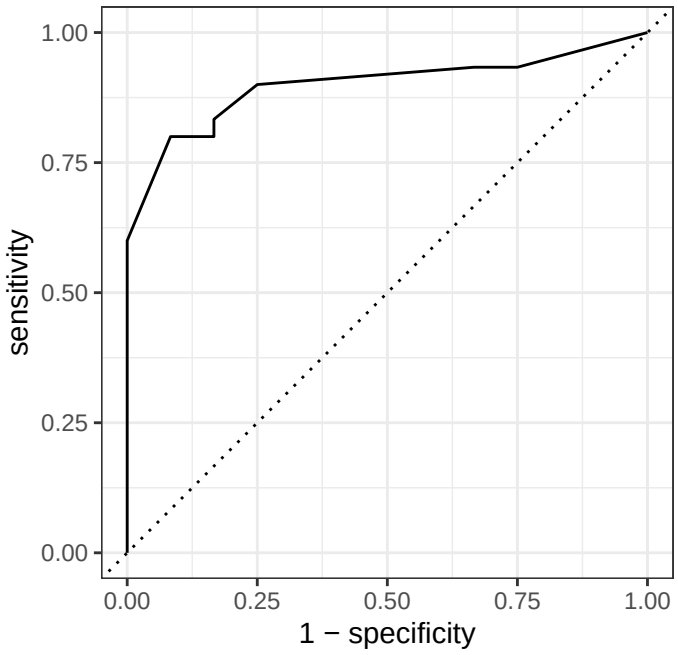
```r
# Affichage de la matrice de confusion :
confusion_2
```

```
          Truth
Prediction  0  1
         0 28  8
         1  2  4
```

```r
summary(confusion_2)
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.762
 2 kap                  binary         0.314
 3 sens                 binary         0.933
 4 spec                 binary         0.333
 5 ppv                  binary         0.778
 6 npv                  binary         0.667
 7 mcc                  binary         0.344
 8 j_index              binary         0.267
 9 bal_accuracy         binary         0.633
10 detection_prevalence binary         0.857
11 precision            binary         0.778
12 recall               binary         0.933
13 f_meas               binary         0.848
```

```r
# Construction de la courbe roc
lasso_fit_2 |>
  roc_curve(truth = "Crit_P", .pred_0) |>
  autoplot()
```



```r
# Récupération des prédictions de probabilité
predictions_lasso_2 <- lasso_prob_2 %>%
  as_tibble() %>%
  select(.pred_0, .pred_1)

truth_values_lasso_2 <- lasso_fit_2$Crit_P

roc_pred_lasso_2 <- prediction(predictions_lasso_2$.pred_1, truth_values_lasso_2)

# Calcule de la performance de la courbe Roc
auc_roc_lasso_2 <- data.frame(Courbe_Roc_lasso=performance(
  roc_pred_lasso_2, "auc")@y.values[[1]])
auc_roc_lasso_2
```

```
  Courbe_Roc_lasso
1        0.8986111
```

Interprétation : À partir de la matrice de confusion, on constate que notre modèle logistique effectue une très bonne prédiction des événements non critiques (93%) et très faible pour ceux critiques (33%). Ceci est également visible à partir de la courbe ROC, pour laquelle la courbe est orientée vers l'angle gauche, montrant ainsi une spécificité très élevée et une sensibilité faible. Ainsi, l'utilisation de la régression Lasso a encore une fois fortement contribué à une meilleure sensibilité (performance de 0.90), qui est la capacité à discriminer un événement critique d'un non critique.

- 4.2 / Méthode de bagging

- Forêt aléatoire

```r
play_recipe_2 <- recipe(Crit_P ~ ., data = donnees_entrainement_2)


# Création du modèle et du workflow
randomForest_model_2 <- rand_forest(mtry=tune()) |>
  set_engine("ranger") |>
  set_mode("classification")

randomForest_wf_2 <- workflow() |>
  add_recipe(play_recipe_2) |>
  add_model(randomForest_model_2)
```

```r
# Établissement de la méthode de validation croisée
# à partir de la variable d'intêret Crit_P
play_samples_cv_2 <- vfold_cv(donnees_entrainement_2,
                              v=10,repeats = 1, strata=Crit_P)

# Ajustement du paramètre mtry
par_grid_2 <- tibble(mtry = seq(1,5, by=1)) # Il y a ici 5 variables à analyser

rf_10_fold_2<- randomForest_wf_2 |>
  tune_grid(resamples = play_samples_cv_2,grid = par_grid_2,
            metrics = metric_set(accuracy))

rf_10_fold_2 |>
  collect_metrics()
```

```
# A tibble: 5 x 7
   mtry .metric  .estimator  mean     n std_err .config
  <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1     1 accuracy binary     0.702    10  0.0228 Preprocessor1_Model1
2     2 accuracy binary     0.670    10  0.0338 Preprocessor1_Model2
3     3 accuracy binary     0.668    10  0.0321 Preprocessor1_Model3
4     4 accuracy binary     0.687    10  0.0401 Preprocessor1_Model4
5     5 accuracy binary     0.677    10  0.0352 Preprocessor1_Model5
```

```r
best_mtry_2 <- rf_10_fold_2 |>
  select_best()

final_rf_2 <- randomForest_wf_2 |>
  finalize_workflow(best_mtry_2) |>
  fit(data = donnees_validation_2)

# Prédiction puis préparation à l'évaluation du modèle
pred_2 <- final_rf_2 |>
  predict(new_data = donnees_validation_2)

risque_est_2 <- data.frame(Risque_est_Rf=mean(
  donnees_validation_2$Crit_P!=pred_2$.pred_class))

rf_class_2 <- final_rf_2 |>
  predict(new_data=donnees_validation_2,type="class")
```

```r
rf_prob_2 <- final_rf_2 |>
  predict(new_data=donnees_validation_2,type="prob")


rf_fit_2 <- donnees_validation_2 |>
  bind_cols(rf_class_2, rf_prob_2)


confusion <- yardstick::conf_mat(data=rf_fit_2,truth="Crit_P",
                                 estimate=".pred_class",
                                 dnn = c("Prediction", "Truth"))
```

- Évaluation du modèle

```r
# Affichage du risque estimé
risque_est_2
```

```
  Risque_est_Rf
1     0.1190476
```

```r
# Affichage de la matrice de confusion
confusion
```

```
          Truth
Prediction  0  1
         0 28  3
         1  2  9
```

```r
summary(confusion)
```

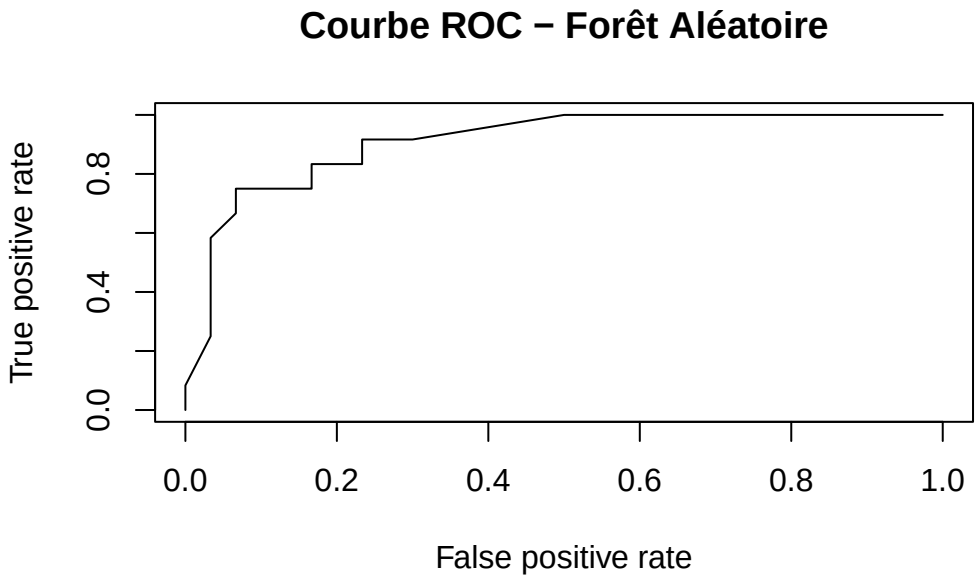
```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.881
 2 kap                  binary         0.701
 3 sens                 binary         0.933
 4 spec                 binary         0.75
```

```
 5 ppv                  binary         0.903
 6 npv                  binary         0.818
 7 mcc                  binary         0.702
 8 j_index              binary         0.683
 9 bal_accuracy         binary         0.842
10 detection_prevalence binary         0.738
11 precision            binary         0.903
12 recall               binary         0.933
13 f_meas               binary         0.918
```

```r
# On récupère les prédictions de probabilité
rf_probabilities_2 <- rf_fit_2 %>%
  select(.pred_0, .pred_1)

# On récupère les vraies valeurs
rf_truth_values_2 <- donnees_validation_2$Crit_P

# On crée un objet ROCR deprediction
rf_roc_pred_2 <- prediction(rf_probabilities_2$.pred_1, rf_truth_values_2)

# On calcule l'AUC-ROC
rf_auc_roc_2 <- performance(rf_roc_pred_2, "auc")@y.values[[1]]

# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", rf_auc_roc_2))
```

```
[1] "AUC-ROC: 0.909722222222222"
```

```r
# On trace la courbe ROC
rf_roc_perf_2 <- performance(rf_roc_pred_2, "tpr", "fpr")

plot(rf_roc_perf_2, main = "Courbe ROC - Forêt Aléatoire")
```



Interprétation : La matrice de confusion obtenue indique que notre modèle de forêt aléatoire prédit très bien les événements non critiques (93%) et offre des prédictions correctes pour ceux considérés comme critiques (66%). En examinant la courbe Roc, on évalue la performance du modèle de 0.909, ce qui concorde avec notre évaluation visuel.

Dans l'ensemble, ce modèle affiche une estimation du risque de seulement 11%, ce qui est tout à fait acceptable compte tenu de notre type de données. Cependant, il montre une moindre pertinence lorsqu'il s'agit des événements non critiques. Ainsi, il pourrait être judicieux d'explorer d'autres techniques de machine learning.

- Algorithme de Boosting (AdaBoost et XgBoost)

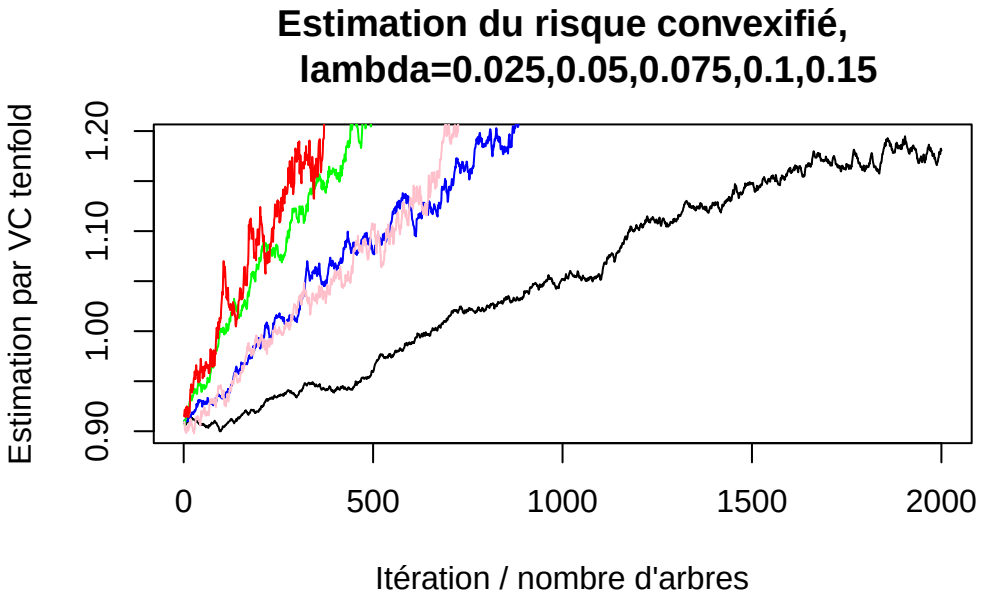
- Mise en forme pour l'algorithme Adaboost:

```
# Transformation de la variable d'intêret en numérique afin de
# conformer aux caractéritiques propre du boosting
dtrain_boost_2 <- donnees_entrainement_2 |>
  mutate(Crit_P=as.numeric(as.character(Crit_P)))

dtest_boost_2 <- donnees_validation_2 |>
  mutate(Crit_P=as.numeric(as.character(Crit_P)))

# Création d'une matrice
Xtrain_boost_2 <- as.matrix(dtrain_boost_2[,2:length(dtrain_boost_2)])
Xtest_boost_2 <- as.matrix(dtest_boost_2[,2:length(dtest_boost_2)])
Ytrain_boost_2 <- dtrain_boost_2$Crit_P
```

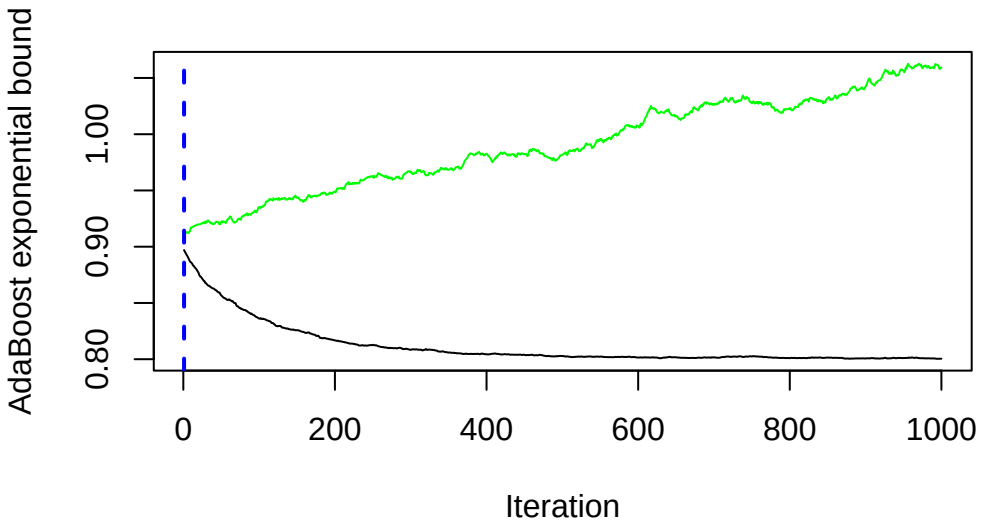
```
Ytest_boost_2 <- dtest_boost_2$Crit_P
```

- Utilisation de l'algorithme Adaboost

```
### Mise en oeuvre de l'algorithme de gradient boosting de type "adaboost" avec le package

ada_2 <- gbm::gbm(Crit_P~.,data=dtrain_boost_2,distribution="adaboost",
                  n.trees=2000,interaction.depth=1,shrinkage=0.025,cv.folds=10)

plot(ada_2$cv.error,type="l",main="Estimation du risque convexifié,
     lambda=0.025,0.05,0.075,0.1,0.15",ylab="Estimation par VC tenfold",
     xlab="Itération / nombre d'arbres")

ada_2 <- gbm::gbm(Crit_P~.,data=dtrain_boost_2,distribution="adaboost",
                  n.trees=2000,interaction.depth=1,shrinkage=0.05,cv.folds=10)
lines(ada_2$cv.error,col="blue")

ada_2 <- gbm::gbm(Crit_P~.,data=dtrain_boost_2,distribution="adaboost",
                  n.trees=2000,interaction.depth=1,shrinkage=0.075,cv.folds=10)
lines(ada_2$cv.error,col="green")

ada_2 <- gbm::gbm(Crit_P~.,data=dtrain_boost_2,distribution="adaboost",
                  n.trees=2000,interaction.depth=1,shrinkage=0.1,cv.folds=10)
lines(ada_2$cv.error,col="pink")

ada_2 <- gbm::gbm(Crit_P~.,data=dtrain_boost_2,distribution="adaboost",
                  n.trees=2000,interaction.depth=1,shrinkage=0.15,cv.folds=10)
lines(ada_2$cv.error,col="red")
```




```
# Pour obtenir un graphe des estimateurs empirique (en noir)
# et par VC (en vert) du risque convexifié pour lambda=0.025 (par exemple) :

ada_2 <- gbm::gbm(Crit_P~.,data=dtrain_boost_2,distribution="adaboost",
                  n.trees=1000,interaction.depth=1,shrinkage=0.025,cv.folds=10)

Boptada_2 <- gbm::gbm.perf(ada_2,method="cv")
```



Interprétation : En ce qui concerne l'estimation du risque convexifié, on observe rapidement que plus notre modèle est entraîné, plus l'estimation du risque augmente. Pourtant, idéalement, nous aurions dû obtenir une courbe décroissante, se rapprochant progressivement de 0 à terme, ce qui est en réalité le contraire dans notre modèle.

Cela suggère que cet algorithme pourrait ne pas être le mieux adapté à notre type de données. On peut également appuyer ce propos à travers le deuxième graphique, où la courbe d'estimation du risque convexifié augmente avec le nombre croissant d'itérations, indiquant un apprentissage relativement médiocre de ces données. Et ceux que n'importe quelle lambda choisit.

- Évaluation du modèle

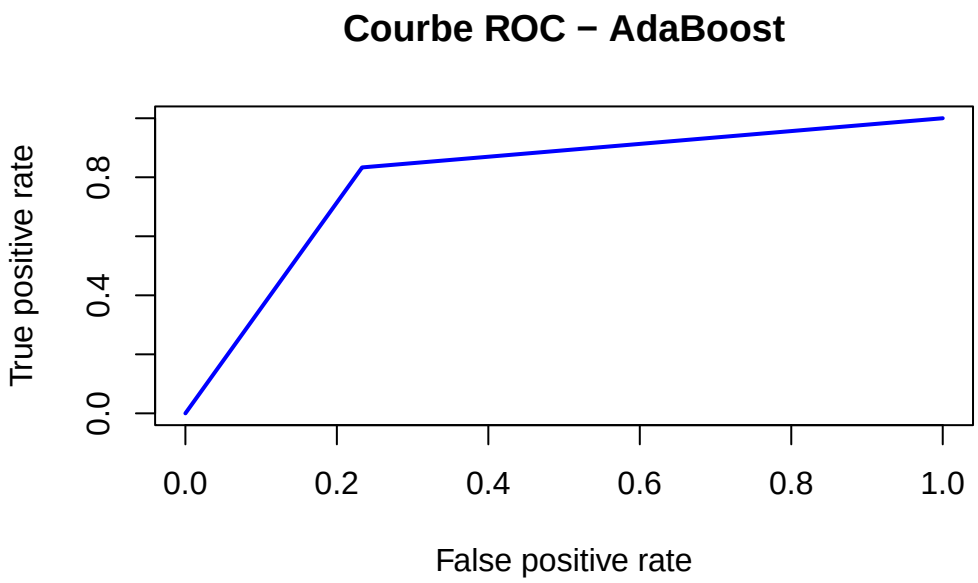
```
# Évaluation des performances de l'algorithme pour lambda=0.025
pred_2 <- predict(ada_2,newdata=dtest_boost_2,n.trees=Boptada_2)
risque.est_ada_2 <- data.frame(Adaboost_gbm_0.025=
                            ModelMetrics::ce(Ytest_boost_2,as.numeric(pred_2>0)))
risque.est_ada_2
```

```
  Adaboost_gbm_0.025
1          0.2857143
```

```
# On convertit les prédictions en classe binaire (0 ou 1)
pred_class_2 <- as.numeric(pred_2 > 0)

# On crée un objet de prédiction ROC
roc_pred_ada_2 <- prediction(pred_2, dtest_boost_2$Crit_P)

# On calcule l'AUC-ROC
ada_auc_roc_2 <- performance(roc_pred_ada_2, "auc")@y.values[[1]]

print(paste("AUC-ROC:", ada_auc_roc_2))
```

```
[1] "AUC-ROC: 0.8"
```

```
# On calculer la courbe ROC
roc_perf_ada_2 <- performance(roc_pred_ada_2, "tpr", "fpr")

# On traçe la courbe
plot(roc_perf_ada_2, main = "Courbe ROC - AdaBoost", col = "blue", lwd = 2)
```



Interprétation : Notons également que le risque estimé reste significativement plus élevé que celui de la forêt aléatoire (Adaboost : 28%, Forêt Aléatoire : 11%). Ceci est également observable à travers la courbe Roc, où la spécificité et la sensibilité semblent satisfaisantes, comme en témoigne la valeur de l'AUC se rapprochant de 1 (0.8). De plus, la représentation graphique indique que plus la courbe ROC s'éloigne de la ligne diagonale, meilleure est la performance du modèle. Dans cette perspective, nous allons explorer une méthode sensiblement similaire avec XgBoost.

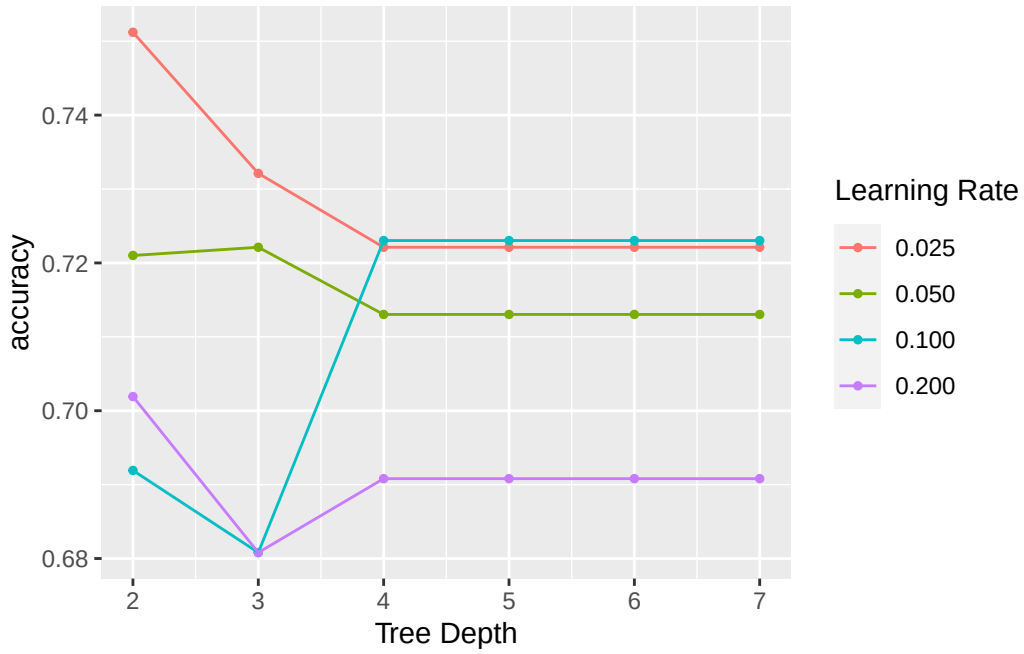
- Mise en forme pour l'algorithme XgBoost

```
# Convertion de toutes les colonnes en numérique pour donnees_entrainement
donnees_entrainement_num_2 <- donnees_entrainement_2 %>%
  mutate_all(~as.numeric(as.factor(.))) %>%
  mutate(Crit_P = as.factor(Crit_P))

# Convertion de toutes les colonnes en numérique pour donnees_validation
donnees_validation_num_2 <- donnees_validation_2 %>%
  mutate_all(~as.numeric(as.factor(.))) %>%
  mutate(Crit_P = as.factor(Crit_P))
```

- Utilisation de l'algorithme XgBoost

```
# Ajustement des hyperparamètres de **xgboost**
# à l'aide du package **tidymodels**
```

```r
data_pour_modele_4_recipe <-
  recipe(Crit_P ~ ., data = donnees_entrainement_num_2)

# Création du modèle et du workflow
xgb_model_2 <-
  boost_tree(
    trees = 500,
    stop_iter=20,
    min_n = 1,
    tree_depth = tune(),
    learn_rate = tune()
  ) |>
  set_mode("classification") |>
  set_engine("xgboost")

xgb_wf_2 <- workflow() |>
  add_recipe(data_pour_modele_4_recipe) |>
  add_model(xgb_model_2)

# Établissement de la méthode de validation croisée à partir de Crit_P
play_samples_cv_2 <- vfold_cv(donnees_entrainement_num_2, v = 10,
                              repeats = 1, strata = "Crit_P")

xgb_grid_2 <- expand.grid(tree_depth = c(2,3,4,5,6,7),
                          learn_rate = c(0.025,0.05,0.1,0.2))

xgb_10fold_2 <- xgb_wf_2 |>
  tune_grid(resamples = play_samples_cv_2, grid = xgb_grid_2,
            metrics = metric_set(accuracy))

xgb_10fold_2 |> autoplot()
```




```
best_par_2 <- xgb_10fold_2 |>
  select_best()

final_xgb_2 <- xgb_wf_2 |>
  finalize_workflow(best_par_2) |>
  fit(data = donnees_entrainement_num_2)

# Prédiction du modèle
pred_xgb_2 <- final_xgb_2 |>
  predict(new_data = donnees_validation_num_2)

xgb_class_2 <- final_xgb_2 |>
  predict(new_data=donnees_validation_num_2,type="class")

xgb_prob_2 <- final_xgb_2 |>
  predict(new_data=donnees_validation_num_2,type="prob")


final_xgb_2 <- donnees_validation_num_2 |>
  bind_cols(xgb_class_2, xgb_prob_2)
```

```
confusion_xgb_2 <- yardstick::conf_mat(data=final_xgb_2,truth="Crit_P",
                                       estimate=".pred_class",
                                       dnn = c("Prediction", "Truth"))
```

Interprétation : En analysant le graphique de la précision du modèle XgBoost en fonction de la profondeur de l'arbre, il est évident que la courbe associée à un taux d'apprentissage de 0.025 (courbe rouge) semble assurer une meilleure précision de manière générale, quelle que soit la profondeur des arbres.

- Évaluation du modèle

```
# Estimation du risque
risque.est_xgb_2 <- data.frame(Risque_est_xgb=mean(
  donnees_validation_num_2$Crit_P != pred_xgb_2$.pred_class))

print(risque.est_xgb_2)
```

```
  Risque_est_xgb
1      0.3095238
```

```
# Affichage de la matrice de confusion
confusion_xgb_2
```

```
          Truth
Prediction  1  2
         1 26  9
         2  4  3
```

```
summary(confusion_xgb_2)
```

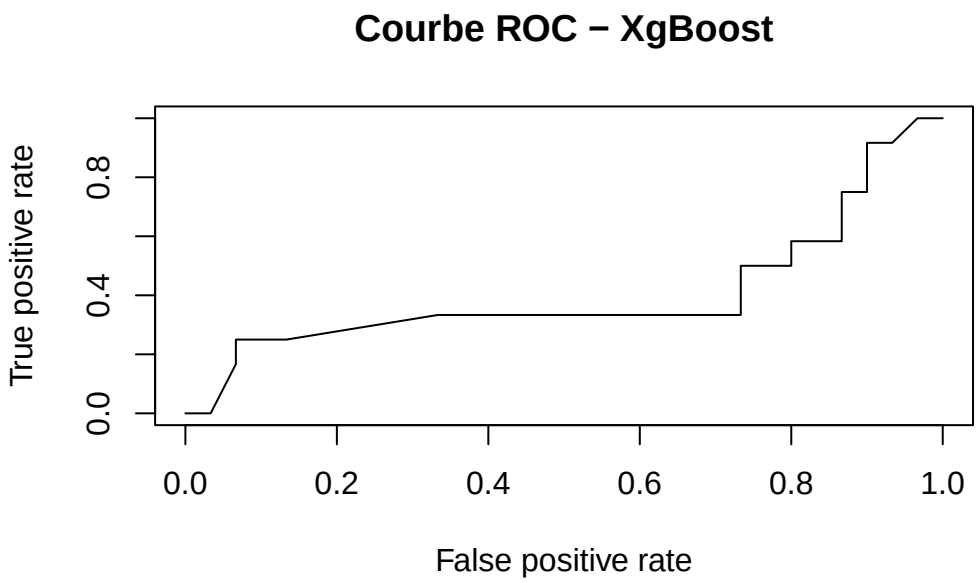
```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.690
 2 kap                  binary         0.133
 3 sens                 binary         0.867
 4 spec                 binary         0.25
 5 ppv                  binary         0.743
```

```
 6 npv                  binary         0.429
 7 mcc                  binary         0.141
 8 j_index              binary         0.117
 9 bal_accuracy         binary         0.558
10 detection_prevalence binary         0.833
11 precision            binary         0.743
12 recall               binary         0.867
13 f_meas               binary         0.8
```

```r
# On récupère les prédictions de probabilité
xgb_probabilities_2 <- final_xgb_2 %>%
  select(.pred_1, .pred_2)

# Récupération des vraies valeurs
xgb_truth_values_2 <- donnees_validation_num_2$Crit_P

# Prédiction du modèle
xgb_roc_pred_2 <- prediction(xgb_probabilities_2$.pred_1, xgb_truth_values_2)

xgb_auc_roc_2 <- performance(xgb_roc_pred_2, "auc")@y.values[[1]]

# On affiches la valeur de l'AUC-ROC
print(paste("AUC-ROC:", xgb_auc_roc_2))
```

```
[1] "AUC-ROC: 0.404166666666667"
```

```r
xgb_roc_perf_2 <- performance(xgb_roc_pred_2, "tpr", "fpr")

# On trace la courbe ROC
plot(xgb_roc_perf_2, main = "Courbe ROC - XgBoost")
```



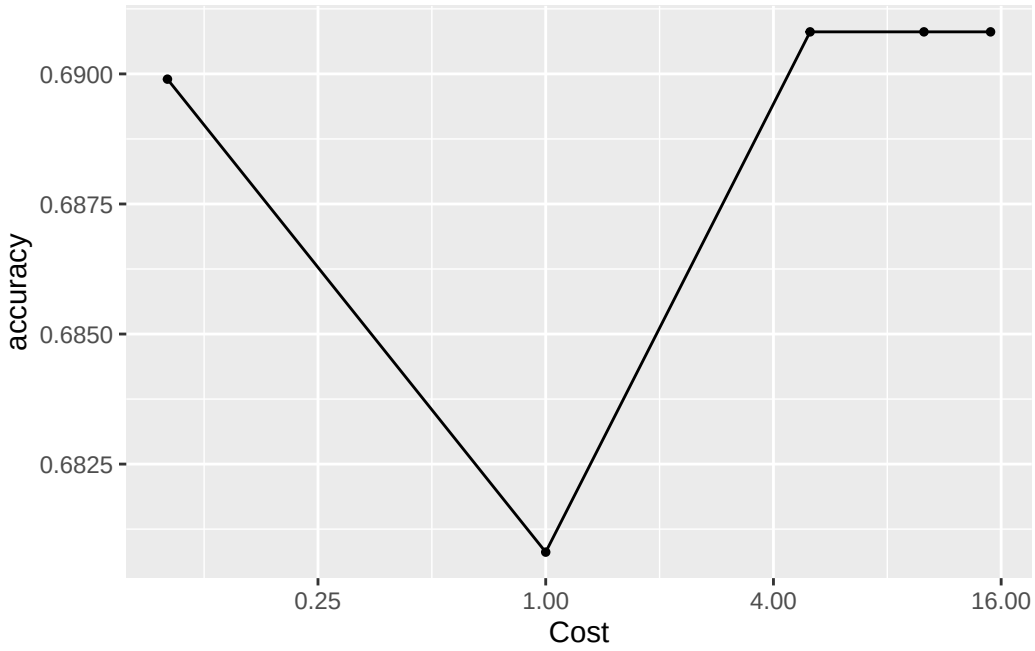
Interprétation : En se basant sur la matrice de confusion, on observe que notre modèle parvient à bien prédire les événements non critiques (86%), mais présente des prédictions moins précises pour ceux considérés comme critiques (17%). Dans l'ensemble, ce modèle affiche une estimation du risque relativement basse, atteignant seulement 30%, ce qui le place en dessous des autres algorithmes utilisés en termes de performance. En ce qui concerne la courbe ROC, l'évaluation de la performance du modèle s'élève à 0.40, ce qui est cohérent avec les observations précédentes.

- 4.3 / Bonus : Algorithme de prédiction par minimisation de risque empirique pénalisé : SVM

- Utilisation de SVM linéaire

```
# Création du modèle et du workflow
svml_model_2 <-
     svm_linear(cost = tune()) |>
     set_mode("classification") |>
     set_engine("kernlab")
svml_wf_2 <- workflow() |>
     add_recipe(play_recipe_2) |>
     add_model(svml_model_2)

# Ajustement de la constante de tolérance C par validation croisée
par_grid_2 <- tibble(cost=c(0.1,1,5,10,15))
```

```
svml_cv_2 <- svml_wf_2 |>
    tune_grid(resamples=play_samples_cv_2,grid=par_grid_2,
              metrics=metric_set(accuracy))
svml_cv_2 |>
  autoplot()
```




```
svml_cv_2 |>
    collect_metrics()
```

```
# A tibble: 5 x 7
   cost .metric  .estimator  mean     n std_err .config
  <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1   0.1 accuracy binary     0.690    10  0.0230 Preprocessor1_Model1
2   1   accuracy binary     0.681    10  0.0329 Preprocessor1_Model2
3   5   accuracy binary     0.691    10  0.0350 Preprocessor1_Model3
4  10   accuracy binary     0.691    10  0.0350 Preprocessor1_Model4
5  15   accuracy binary     0.691    10  0.0350 Preprocessor1_Model5
```

```
# Évaluation de la performance du modèle
best_C_2 <- svml_cv_2 |>
```

```
  select_best()
final_svml_2 <- svml_wf_2 |>
  finalize_workflow(best_C_2) |>
  fit(data = donnees_entrainement_2)
```

```
 Setting default kernel parameters
```

```
Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svml_class_2 <- final_svml_2 |>
    predict(new_data=donnees_validation_2,type="class")
svml_prob_2 <- final_svml_2 |>
    predict(new_data=donnees_validation_2,type="prob")
svml_fit_2 <- donnees_validation_2 |>
  bind_cols(svml_class_2, svml_prob_2)

pred_svml_2 <- final_svml_2 |>
     predict(new_data=donnees_validation_2)
```

Interprétation : On observe à travers ce graphique que la plus grande précision pour notre modèle est établie à partir d'un coût infèrieur 0.2, tandis qu'un coût supérieur semble dégrader sa performance (à partir de 4).

- Évaluation du modèle

```
# Estimation du risque
risque.est_svml_2 <- data.frame(Risque_est_svml=mean(
  donnees_validation_2$Crit_P!=pred_svml_2$.pred_class))

print(risque.est_svml_2)
```

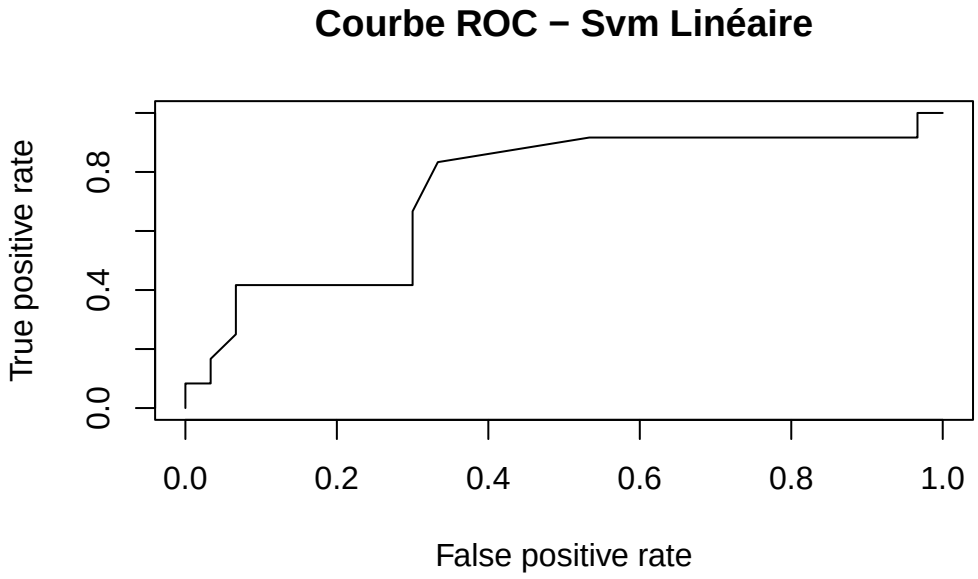
```
  Risque_est_svml
1       0.2619048
```

```
# On récupère les prédictions de probabilité
svml_probabilities_2 <- svml_fit_2 %>%
  select(.pred_0, .pred_1)
```

```r
# On récupère les vraies valeurs
svml_truth_values_2 <- donnees_validation_2$Crit_P

# On crée un objet ROCR prediction
svml_roc_pred_2 <- prediction(svml_probabilities_2$.pred_1, svml_truth_values_2)

# On calcule l'AUC-ROC
svml_auc_roc_2 <- performance(svml_roc_pred_2, "auc")@y.values[[1]]

# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", svml_auc_roc_2))
```

```
[1] "AUC-ROC: 0.7375"
```

```r
# On trace la courbe ROC
svml_roc_perf_2 <- performance(svml_roc_pred_2, "tpr", "fpr")

plot(svml_roc_perf_2, main = "Courbe ROC - Svm Linéaire")
```

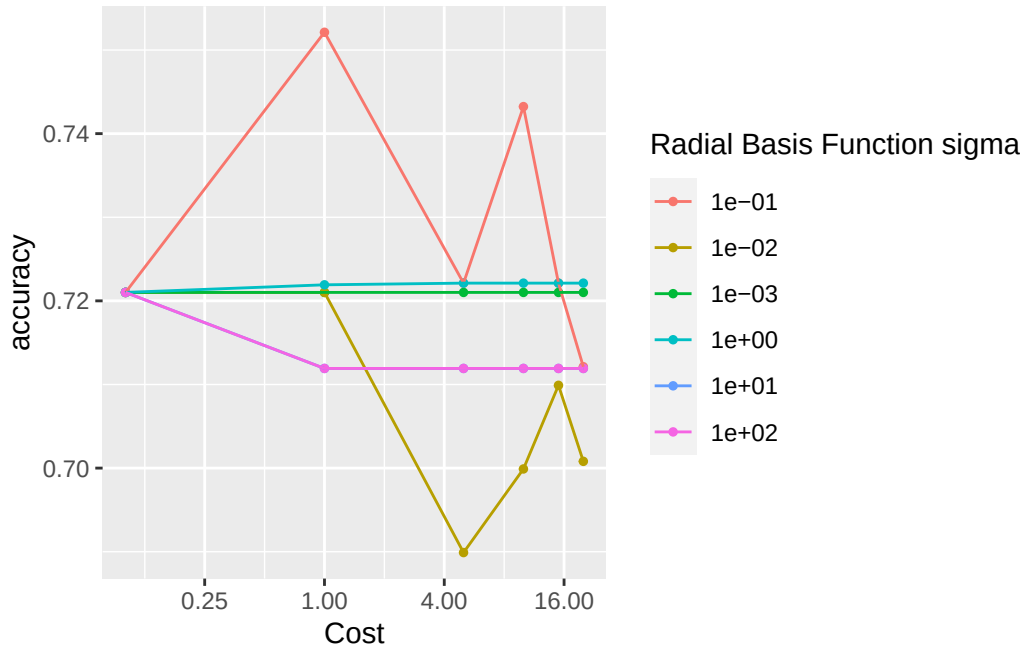


Interprétation : Dans l'ensemble, l'estimation du risque obtenue dans ce modèle est de 26%, un chiffre relativement bas au regard de la nature de nos données et des résultats obtenus par d'autres algorithmes. Cette observation est en addéquat avec l'analyse de la courbe ROC, laquelle s'approche de l'angle supérieur gauche du graphique, indiquant une spécificité et une

sensibilité acceptables. En conséquence, ce modèle pourrait être considéré comme relativement idéal, avec une performance de 0.73, pour la prédiction de ce type de données.

- Utilisation de SVM non linéaire à noyau gaussien (ou radial)

```
# Création du modèle et du workflow
svmr_model_2 <-
     svm_rbf(cost = tune(),rbf_sigma=tune()) |>
     set_mode("classification") |>
     set_engine("kernlab")

svmr_wf_2 <- workflow() |>
     add_recipe(play_recipe_2) |>
     add_model(svmr_model_2)

# Ajustement de la constante de tolérance C
# et de la fenêtre (variance) du noyau gaussien par validation croisée

par_grid_2 <- expand.grid(cost=c(0.1,1,5,10,15,20),rbf_sigma=10^(-3:2))

svmr_cv_2 <- svmr_wf_2 |>
     tune_grid(resamples=play_samples_cv_2,grid=par_grid_2,
               metrics=metric_set(accuracy))

svmr_cv_2|> autoplot()
```




```
svmr_cv_2 |>
    collect_metrics()
```

```
# A tibble: 36 x 8
    cost rbf_sigma .metric  .estimator  mean     n std_err .config
   <dbl>     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1   0.1     0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model01
 2   1       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model02
 3   5       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model03
 4  10       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model04
 5  15       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model05
 6  20       0.001 accuracy binary     0.721    10  0.0101 Preprocessor1_Model06
 7   0.1     0.01  accuracy binary     0.721    10  0.0101 Preprocessor1_Model07
 8   1       0.01  accuracy binary     0.721    10  0.0101 Preprocessor1_Model08
 9   5       0.01  accuracy binary     0.690    10  0.0230 Preprocessor1_Model09
10  10       0.01  accuracy binary     0.700    10  0.0255 Preprocessor1_Model10
# i 26 more rows
```

```
# Évaluation de la performance du modèle

best_par_2 <- svmr_cv_2 |>
```

```
    select_best()

final_svmr_2 <- svmr_wf_2 |>
    finalize_workflow(best_par_2) |>
    fit(data = donnees_entrainement_2)
```

```
Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svmr_class_2 <- final_svmr_2 |>
    predict(new_data=donnees_validation_2,type="class")

svmr_prob_2 <- final_svmr_2 |>
    predict(new_data=donnees_validation_2,type="prob")

svmr_fit_2 <- donnees_validation_2 |>
    bind_cols(svmr_class_2,svmr_prob_2)


pred_svmr_2 <- final_svmr_2 |>
     predict(new_data=donnees_validation_2)
```

Interprétation :

Lorsqu'on examine le graphique portant sur la précision du modèle SVM avec noyau gaussien (en variant le paramètre sigma) en fonction du coût, on observe que l'utilisation d'un noyau égal à 1e-01 nous fournit une performance optimale. Cela nous permettrait de trouver un compromis entre le sous-apprentissage et le sur-apprentissage du modèle.

- Évaluation du modèle

```
# Affichage du risque estimé
risque.est_svmr_2 <- data.frame(Risque_est_svmr=mean(donnees_validation_2$Crit_P!=pred_svm
print(risque.est_svmr_2)
```

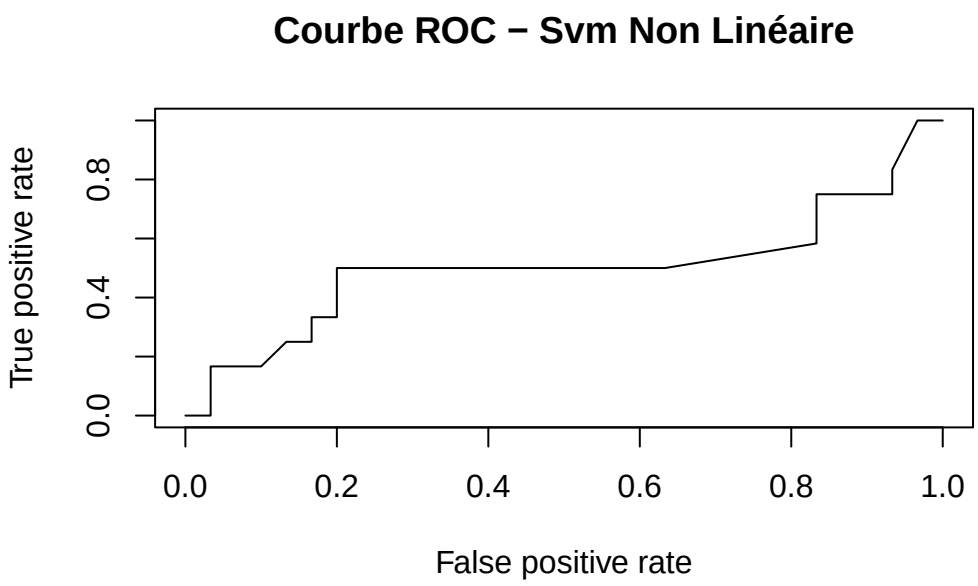
```
  Risque_est_svmr
1       0.2857143
```

```
# On récupère les prédictions de probabilité
svmr_probabilities_2 <- svmr_fit_2 %>%
```

```r
  select(.pred_0, .pred_1)

# On récupère les vraies valeurs
svmr_truth_values_2 <- donnees_validation_2$Crit_P

# On crée un objet ROCR prediction
svmr_roc_pred_2 <- prediction(svmr_probabilities_2$.pred_1, svmr_truth_values_2)

# On calcule l'AUC-ROC
svmr_auc_roc_2 <- performance(svmr_roc_pred_2, "auc")@y.values[[1]]

# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", svmr_auc_roc_2))
```

```
[1] "AUC-ROC: 0.501388888888889"
```

```r
# On trace la courbe ROC
svmr_roc_perf_2 <- performance(svmr_roc_pred_2, "tpr", "fpr")

plot(svmr_roc_perf_2, main = "Courbe ROC - Svm Non Linéaire")
```



Interprétation : Dans l'ensemble, ce modèle a généré une estimation du risque de 28%, un chiffre plus au moins élevé en comparaison avec la nature de nos données et les performances

d'autres algorithmes. Cette observation est également remarquée sur la courbe ROC, qui révèle une faible spécificité mais tout de même une sensibilité correcte (0.50 de performance). Par conséquent, l'utilisation de l'algorithme SVM non linéaire à noyau gaussien serait acceptable pour discerner un événement critique d'un événement non critique, mais semble plus limitée dans d'autres contextes.

Partie 5 : Apprentissage supervisé des questions relatives à une situation de plaquage

- 5.1 / Algorithme de prédiction par minimisation de risque empirique

- Utilisation de la régression logistique (GLM : Generalized Linear Model)

- On sélectionne maintenant uniquement les évènements qui ont été induit par une situation de plaquage (**`data_pour_modele_situation_plaquage`**) :

```
data_pour_modele_situation_plaquage <- select(
  data_pour_modele,
  c(Crit_P,
    `Type de situation de rugby ?`,
    `Pour plaquage : joueur a dominé l'impact ?`,
    `Pour plaquage : situation de pick and go ?`,
    `Pour plaquage : vitesse de course du PDB `,
    `Pour plaquage : vitesse de course du plaqueur `,
    `Pour plaquage : angle de course entre PDB et plaqueur`,
    `Pour plaquage : quel est le rôle du joueur ?`,
    `Intensité particulièrement élevée ?`,
    `Pour plaquage : premier point de contact pour le plaqueur`,
    `Pour plaquage : premier point de contact pour le PDB`,
    `Pour plaquage : signal du point de contact correspond au premier point de contact ?`
    )
  )

data_pour_modele_situation_plaquage <- subset(
  data_pour_modele_situation_plaquage, `Type de situation de rugby ?` == 10)

data_pour_modele_situation_plaquage <- select(
  data_pour_modele_situation_plaquage,
  -c(`Type de situation de rugby ?`))
```

- Application de GLM pour ce modèle :

```
GLM_model_5 <- glm(Crit_P ~ .,
                   data = data_pour_modele_situation_plaquage, family = "binomial")
```

- Utilisation de la régression logistique Lasso (avec échantillon)

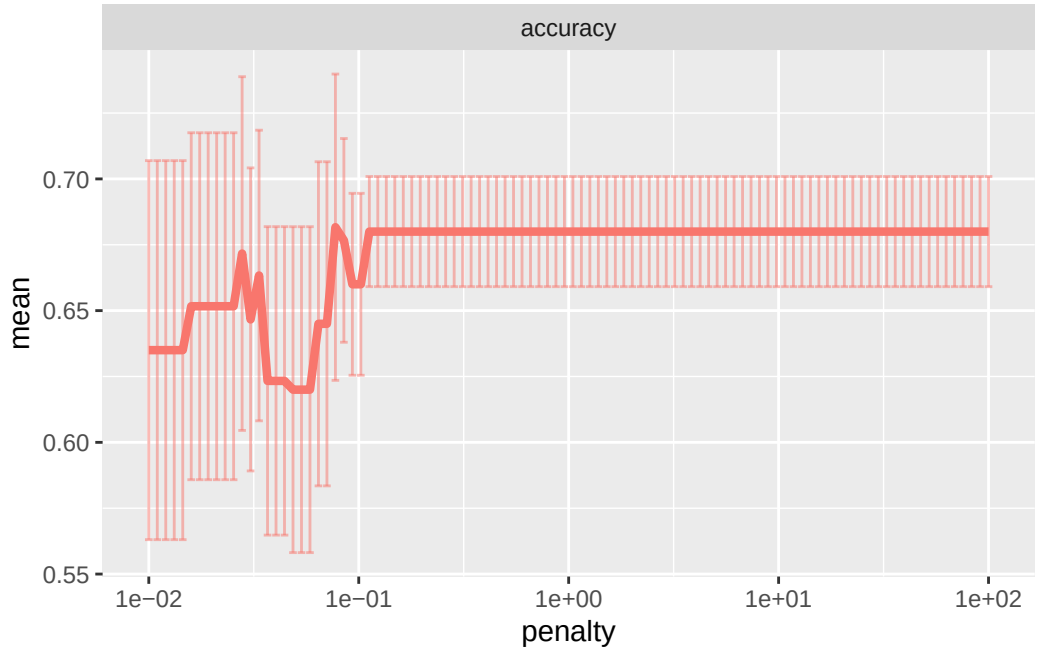
- Création d'un échantillon d'entraînement et d'un échantillon de validation :

```
nombre_NC_3 <- sum(data_pour_modele_situation_plaquage$Crit_P == 0)
nombre_C_3 <- sum(data_pour_modele_situation_plaquage$Crit_P == 1)

etiquettes <- ifelse(data_pour_modele_situation_plaquage$Crit_P == "1", "1","0")
set.seed(123) # Pour la reproductibilité des résultats
index_entrainement_3 <- createDataPartition(etiquettes, p = 0.7, list = FALSE)
donnees_entrainement_3 <- data_pour_modele_situation_plaquage[index_entrainement, ]
donnees_validation_3 <- data_pour_modele_situation_plaquage[-index_entrainement, ]

# Pour enlever les lignes composées uniquement de NA
donnees_entrainement_3 <- head(donnees_entrainement_3, 49)
```

- Régression logistique Lasso :

```
lasso_model_3 <- logistic_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet") |>
  set_mode("classification")

# Création d'un workflow pour la régression logistique LASSO
lasso_wf_3 <- workflow() |>
  add_model(lasso_model_3) |>
  add_formula(Crit_P ~ .)

# Fit du modèle
lasso_fit_3 <- lasso_wf_3 |>
  fit(donnees_entrainement_3)

# Validation croisée
data_cv_3 <- vfold_cv(donnees_entrainement_3,v=10, strata=Crit_P)

# Création d'une grille Pour avoir les valeurs de risque estimé
lambda_grid_3 <- grid_regular(penalty(range = c(-2, 2)), levels = 100)

lasso_cv_3 <- lasso_wf_3 |>
```

```
  tune_grid(resamples = data_cv_3, grid = lambda_grid_3,
            metrics = metric_set(accuracy))

res_3 <- lasso_cv_3 |>
  collect_metrics()

res_3 |>
 ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(
    ymin = mean - std_err,
    ymax = mean + std_err
  ),
  alpha = 0.5
  ) +
  geom_line(linewidth = 1.5) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10() +
  theme(legend.position = "none")
```




```
# Sélectionne le lambda qui minimise le risque estimé par VC
lowest_accuracy_3 <-  lasso_cv_3 |>
```

```
  select_best(metric="accuracy")

# On relance (entraine) la régression LASSO avec ce meilleur lambda
# sur le jeu de données de validation pour obtenir
# le "meilleur" prédicteur LASSO :
final_wf_3 <- finalize_workflow(lasso_wf_3,lowest_accuracy_3)
print(final_wf_3)

final_lasso_3 <- final_wf_3 |>
  fit(donnees_validation_3)

final_lasso_3 |> tidy()

# Prédiction du modèle

final_lasso_3 |>
  predict(donnees_validation_3)

lasso_class_3 <- final_lasso_3 |>
  predict(donnees_validation_3,type="class")

lasso_prob_3 <- final_lasso_3 |>
  predict(donnees_validation_3,type="prob")

predictions_3 <- data.frame(donnees_validation_3,lasso_class_3,lasso_prob_3)

lasso_fit_3 <- donnees_validation_3 |>
  bind_cols(lasso_class_3, lasso_prob_3)

lasso_fit_3 |> head()

confusion_3 <- yardstick::conf_mat(data=lasso_fit_3,truth="Crit_P",
                                   estimate=".pred_class",
                                   dnn = c("Prediction", "Truth"))
```

Interprétation : On remarque que la précision moyenne du modèle serait meilleure lorsque le paramètre de régularisation serait plus faible. De plus, ce dernier aurait, comme on peut le voir ci-dessus, une variance relativement faible, ce qui n'est pas négligeable.

- Évaluation du modèle

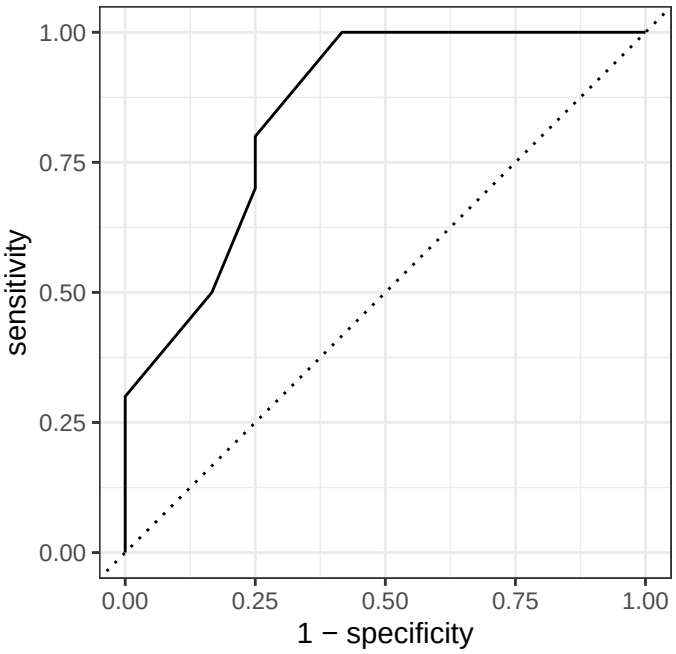
```r
# Affichage de la matrice de confusion :
confusion_3
```

```
          Truth
Prediction 0 1
         0 8 3
         1 2 9
```

```r
summary(confusion_3)
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.773
 2 kap                  binary         0.545
 3 sens                 binary         0.8
 4 spec                 binary         0.75
 5 ppv                  binary         0.727
 6 npv                  binary         0.818
 7 mcc                  binary         0.548
 8 j_index              binary         0.55
 9 bal_accuracy         binary         0.775
10 detection_prevalence binary         0.5
11 precision            binary         0.727
12 recall               binary         0.8
13 f_meas               binary         0.762
```

```r
# Création de la courbe roc
lasso_fit_3 |>
  roc_curve(truth = "Crit_P", .pred_0) |>
  autoplot()
```




```
# Récupération des prédictions de probabilité
predictions_lasso_3 <- lasso_prob_3 %>%
  as_tibble() %>%
  select(.pred_0, .pred_1)

truth_values_lasso_3 <- lasso_fit_3$Crit_P

roc_pred_lasso_3 <- prediction(predictions_lasso_3$.pred_1, truth_values_lasso_3)

# Calcule de la performance de la courbe Roc
auc_roc_lasso_3 <- data.frame(Courbe_Roc_lasso=performance(
  roc_pred_lasso_3, "auc")@y.values[[1]])
auc_roc_lasso_3
```

```
  Courbe_Roc_lasso
1             0.85
```

Interprétation : À partir de la matrice de confusion, on constate que notre modèle logistique effectue une bonne prédiction globale des événements non critiques (80%) et critiques (75%). Ceci est également visible à partir de la courbe ROC, qui montre une spécificité et une sensibilité correcte (performance de 0.85). Ainsi, on peut penser que l'utilisation de la régression Lasso serait viable pour utiliser ce type de données. Il convient néanmoins de tester

d'autres algorithmes, notamment de bagging qui ont montré de bonnes capacités de prédictions auparavant.

- 5.2 / Méthode de bagging

- Forêt aléatoire

```
play_recipe_3 <- recipe(Crit_P ~ ., data = donnees_entrainement_3)


# Création du modèle et du workflow
randomForest_model_3 <- rand_forest(mtry=tune()) |>
  set_engine("ranger") |>
  set_mode("classification")

randomForest_wf_3 <- workflow() |>
  add_recipe(play_recipe_3) |>
  add_model(randomForest_model_3)

# Utilisation de la méthode de validation croisée à partir de Crit_P
play_samples_cv_3 <- vfold_cv(donnees_entrainement_3,v=10,
                              repeats = 1, strata=Crit_P)

# Ajustement de mtry
par_grid_3 <- tibble(mtry = seq(1,8, by=1)) # Il y a ici 5 variables à analyser

rf_10_fold_3<- randomForest_wf_3 |>
  tune_grid(resamples = play_samples_cv_3,grid = par_grid_3,
            metrics = metric_set(accuracy))

rf_10_fold_3 |>
  collect_metrics()
```

```
# A tibble: 8 x 7
   mtry .metric  .estimator  mean     n std_err .config
  <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1     1 accuracy binary     0.68     10  0.0209 Preprocessor1_Model1
2     2 accuracy binary     0.647    10  0.0406 Preprocessor1_Model2
3     3 accuracy binary     0.642    10  0.0447 Preprocessor1_Model3
4     4 accuracy binary     0.642    10  0.0447 Preprocessor1_Model4
5     5 accuracy binary     0.633    10  0.0668 Preprocessor1_Model5
```

```
6     6 accuracy binary     0.592    10  0.0529 Preprocessor1_Model6
7     7 accuracy binary     0.595    10  0.0674 Preprocessor1_Model7
8     8 accuracy binary     0.612    10  0.0592 Preprocessor1_Model8
```

```
best_mtry_3 <- rf_10_fold_3 |>
  select_best()

final_rf_3 <- randomForest_wf_3 |>
  finalize_workflow(best_mtry_3) |>
  fit(data = donnees_validation_3)

# Prédiction du modèle et préparation à son évaluation
pred_3 <- final_rf_3 |>
  predict(new_data = donnees_validation_3)

risque_est_3 <- data.frame(Risque_est_Rf=mean(
  donnees_validation_3$Crit_P!=pred_3$.pred_class))

rf_class_3 <- final_rf_3 |>
  predict(new_data=donnees_validation_3,type="class")

rf_prob_3 <- final_rf_3 |>
  predict(new_data=donnees_validation_3,type="prob")


rf_fit_3 <- donnees_validation_3 |>
  bind_cols(rf_class_3, rf_prob_3)


confusion <- yardstick::conf_mat(data=rf_fit_3,truth="Crit_P",
                                 estimate=".pred_class",
                                 dnn = c("Prediction", "Truth"))
```

- Évaluation du modèle

```
# Affichage du risque estimé
risque_est_3
```

```
  Risque_est_Rf
1     0.1818182
```

```r
# Affichage de la matrice de confusion
confusion
```

```
          Truth
Prediction  0  1
         0  7  1
         1  3 11
```

```r
summary(confusion)
```

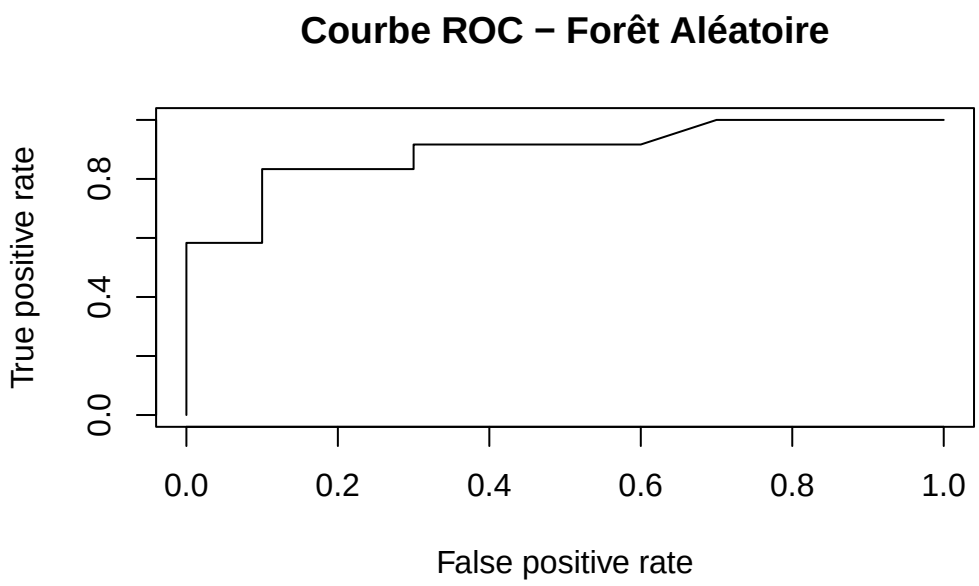
```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.818
 2 kap                  binary         0.627
 3 sens                 binary         0.7
 4 spec                 binary         0.917
 5 ppv                  binary         0.875
 6 npv                  binary         0.786
 7 mcc                  binary         0.638
 8 j_index              binary         0.617
 9 bal_accuracy         binary         0.808
10 detection_prevalence binary         0.364
11 precision            binary         0.875
12 recall               binary         0.7
13 f_meas               binary         0.778
```

```r
# On récupère les prédictions de probabilité
rf_probabilities_3 <- rf_fit_3 %>%
  select(.pred_0, .pred_1)

# On récupère les vraies valeurs
rf_truth_values_3 <- donnees_validation_3$Crit_P

# On crée un objet ROCR deprediction
rf_roc_pred_3 <- prediction(rf_probabilities_3$.pred_1, rf_truth_values_3)

# On calcule l'AUC-ROC
rf_auc_roc_3 <- performance(rf_roc_pred_3, "auc")@y.values[[1]]
```

```r
# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", rf_auc_roc_3))
```

```
[1] "AUC-ROC: 0.895833333333333"
```

```r
# On trace la courbe ROC
rf_roc_perf_3 <- performance(rf_roc_pred_3, "tpr", "fpr")

plot(rf_roc_perf_3, main = "Courbe ROC - Forêt Aléatoire")
```



Interprétation : À partir de la matrice de confusion, on constate que notre modèle de forêt aléatoire effectue une bonne prédiction des événements non critiques (70%) et une prédiction très satisfaisante pour ceux étant considérés comme critiques (92%). Au niveau de la courbe Roc, on évalue la performance du modèle de 0.90, ce qui concorde avec l'impression visuel. Globalement, ce modèle a même obtenu une estimation du risque de seulement 18%, ce qui est tout à fait acceptable compte tenu de nos types de données (bien que plus faible comparé aux autres parties du projet).

À noter que ce dernier aurait une meilleur capacité à distinguer les événements critiques que ceux non critiques, ce qui est assez surprenant compte des observations effectuées précedemment. Il convient de ce fait de tester d'autres techniques de machine learning afin d'attester de cette éventuelle tendance. En outre, il est important de souligner tout de même le nombre

relativement faible d'observations de tests pour ce type de données, ce qui nous inciterait à ne pas faire forcément de conclusions hâtives sur le sujet.

- Algorithme de Boosting (AdaBoost et XgBoost)

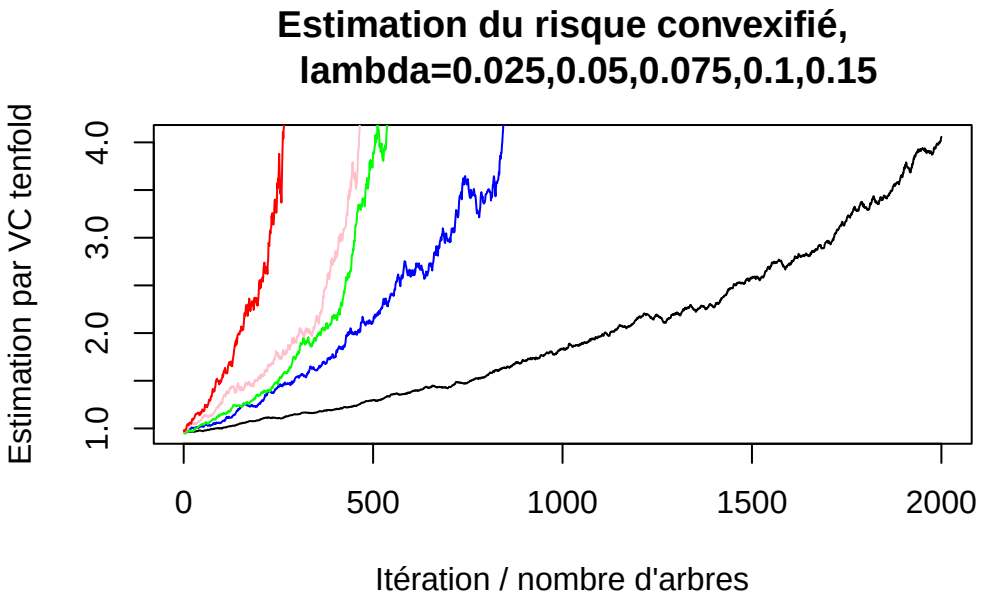
- Mise en forme pour l'algorithme Adaboost:

```
# Transformation de la variable d'intêret en numérique par rapport
# aux caractèristiques de l'algorithme de boosting
dtrain_boost_3 <- donnees_entrainement_3 |>
  mutate(Crit_P=as.numeric(as.character(Crit_P)))

dtest_boost_3 <- donnees_validation_3 |>
  mutate(Crit_P=as.numeric(as.character(Crit_P)))

# Création d'une matrice
Xtrain_boost_3 <- as.matrix(dtrain_boost_3[,2:length(dtrain_boost_3)])
Xtest_boost_3 <- as.matrix(dtest_boost_3[,2:length(dtest_boost_3)])
Ytrain_boost_3 <- dtrain_boost_3$Crit_P
Ytest_boost_3 <- dtest_boost_3$Crit_P
```

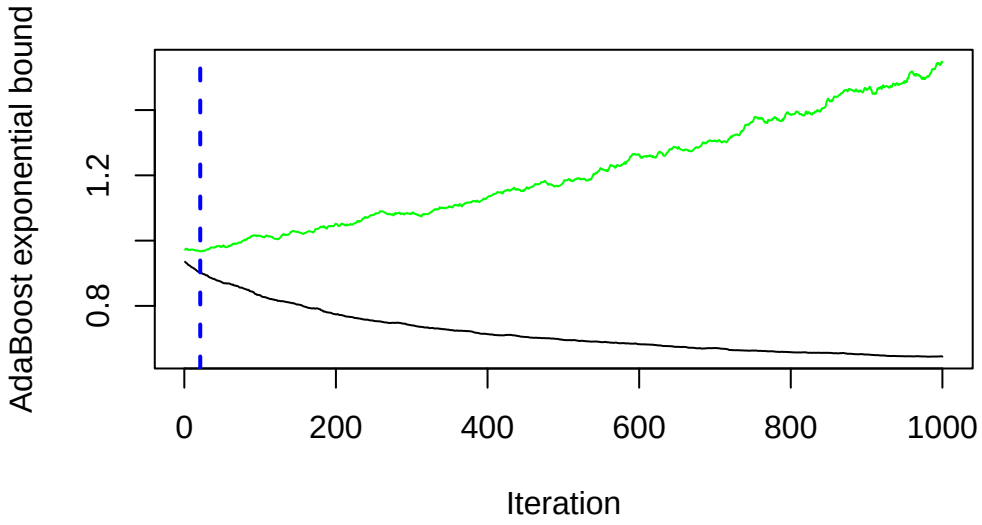
- Utilisation de l'algorithme Adaboost

```
### Mise en oeuvre de l'algorithme de gradient boosting
# de type "adaboost" avec le package **gbm**

ada_3 <- gbm::gbm(Crit_P~.,data=dtrain_boost_3,distribution="adaboost",
                  n.trees=2000,interaction.depth=1,shrinkage=0.025,cv.folds=10)

plot(ada_3$cv.error,type="l",main="Estimation du risque convexifié,
     lambda=0.025,0.05,0.075,0.1,0.15",ylab="Estimation par VC tenfold",
     xlab="Itération / nombre d'arbres")

ada_3 <- gbm::gbm(Crit_P~.,data=dtrain_boost_3,distribution="adaboost",
                  n.trees=2000,interaction.depth=1,shrinkage=0.05,cv.folds=10)
lines(ada_3$cv.error,col="blue")

ada_3 <- gbm::gbm(Crit_P~.,data=dtrain_boost_3,distribution="adaboost",
                  n.trees=2000,interaction.depth=1,shrinkage=0.075,cv.folds=10)
lines(ada_3$cv.error,col="green")

ada_3 <- gbm::gbm(Crit_P~.,data=dtrain_boost_3,distribution="adaboost",
```

```
                    n.trees=2000,interaction.depth=1,shrinkage=0.1,cv.folds=10)
lines(ada_3$cv.error,col="pink")

ada_3 <- gbm::gbm(Crit_P~.,data=dtrain_boost_3,distribution="adaboost",
                    n.trees=2000,interaction.depth=1,shrinkage=0.15,cv.folds=10)
lines(ada_3$cv.error,col="red")
```




```
# Pour obtenir un graphe des estimateurs empirique (en noir)
# et par VC (en vert) du risque convexifié pour lambda=0.025 (par exemple) :

ada_3 <- gbm::gbm(Crit_P~.,data=dtrain_boost_3,distribution="adaboost",
                    n.trees=1000,interaction.depth=1,shrinkage=0.025,cv.folds=10)

Boptada_3 <- gbm::gbm.perf(ada_3,method="cv")
```



Interprétation : Concernant l'estimation du risque convexifié et comme pour les parties précédentes, on distingue rapidemment que plus on entraine notre modèle, plus l'estimation de risque grandit. Or, dans l'idéal, on aurait du obtenir une courbe décroissante, se rapprochant à terme de 0.

Cela signifierait ainsi que cet algorithme ne serait pas forcément le plus adapté à notre type de données. Par ailleurs, on peut également appuyer ce propos à travers le deuxième graphique pour lequel l'estimation du risque convexifié augmente avec la hausse du nombre d'itérations, ce qui atteste d'un apprentissage relativement mauvais de ces données, et ceux qu'importe le lambda choisi.

- Évaluation du modèle

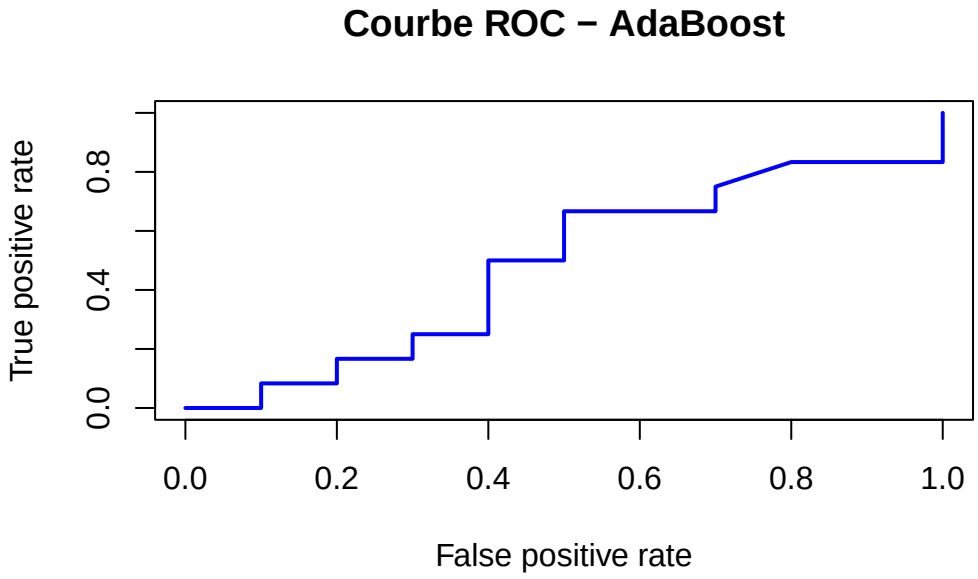
```
# Évaluation des performances de l'algorithme pour lambda=0.025

pred_3 <- predict(ada_3,newdata=dtest_boost_3,n.trees=Boptada_3)

risque.est_ada_3 <- data.frame(
  Adaboost_gbm_0.025 = ModelMetrics::ce(Ytest_boost_3, as.numeric(pred_3 > 0)))

risque.est_ada_3
```

```
  Adaboost_gbm_0.025
1          0.5454545
```

```r
# On convertit les prédictions en classe binaire (0 ou 1)
pred_class_3 <- as.numeric(pred_3 > 0)

# On crée un objet de prédiction ROC
roc_pred_ada_3 <- prediction(pred_3, dtest_boost_3$Crit_P)

# On calcule l'AUC-ROC
ada_auc_roc_3 <- performance(roc_pred_ada_3, "auc")@y.values[[1]]

print(paste("AUC-ROC:", ada_auc_roc_3))
```

```
[1] "AUC-ROC: 0.479166666666667"
```

```r
# On calculer la courbe ROC
roc_perf_ada_3 <- performance(roc_pred_ada_3, "tpr", "fpr")

# On traçe la courbe
plot(roc_perf_ada_3, main = "Courbe ROC - AdaBoost", col = "blue", lwd = 2)
```



Interprétation : De plus, le risque estimé reste largement plus élevé que celui de la forêt aléatoire (Adaboost : 54%, Forêt Aléatoire : 18%). Ceci est également perceptible à travers la courbe Roc, où l'on observe qu'elle tarde à atteindre le haut du graphique. Cela signiferait que la spécifité et la sensibilité du modèle serait très faibles, surtout en comparaison avec les

autres parties. Nous allons donc explorer une méthode sensiblement similaire avec XgBoost afin d'attester de cela.

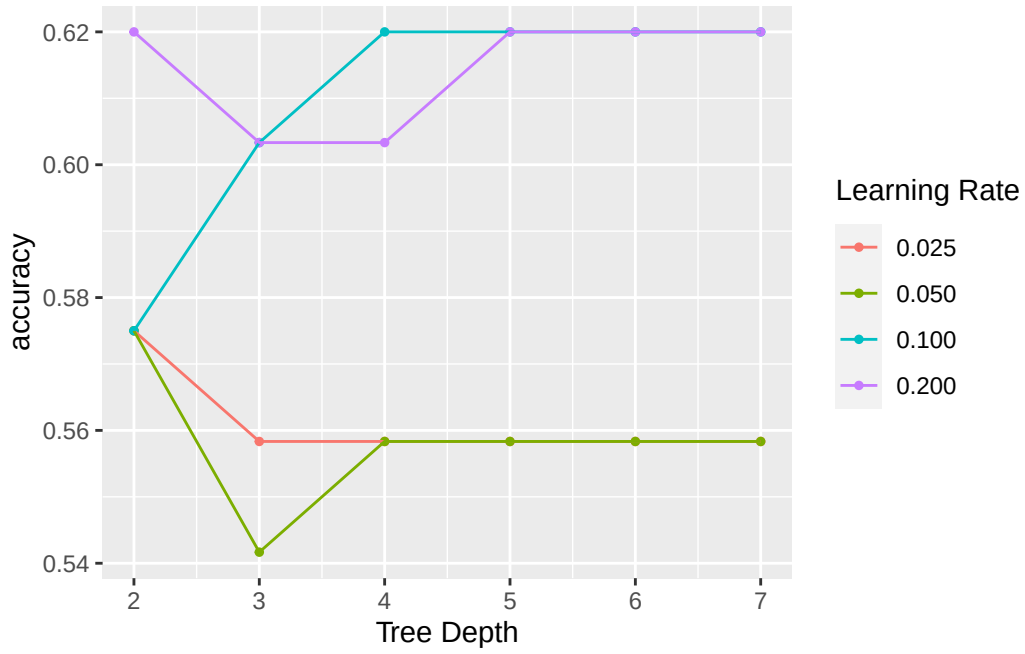
- Mise en forme pour l'algorithme XgBoost

```
# Convertion de toutes les colonnes en numérique pour donnees_entrainement
donnees_entrainement_num_3 <- donnees_entrainement_3 %>%
  mutate_all(~as.numeric(as.factor(.))) %>%
  mutate(Crit_P = as.factor(Crit_P))

# Convertion de toutes les colonnes en numérique pour donnees_validation
donnees_validation_num_3 <- donnees_validation_3 %>%
  mutate_all(~as.numeric(as.factor(.))) %>%
  mutate(Crit_P = as.factor(Crit_P))
```

- Utilisation de l'algorithme XgBoost

```
# Ajustement des hyperparamètres de **xgboost**
# à l'aide du package **tidymodels**

data_pour_modele_situation_plaquage_recipe <-
  recipe(Crit_P ~ ., data = donnees_entrainement_num_3)

# Création du modèle et du workflow
xgb_model_3 <-
  boost_tree(
    trees = 500,
    stop_iter=20,
    min_n = 1,
    tree_depth = tune(),
    learn_rate = tune()
  ) |>
  set_mode("classification") |>
  set_engine("xgboost")

xgb_wf_3 <- workflow() |>
  add_recipe(data_pour_modele_situation_plaquage_recipe) |>
  add_model(xgb_model_3)

# Établissement de la methode de validation croisée à partir de Crit_P
play_samples_cv_3 <- vfold_cv(donnees_entrainement_num_3, v = 10,
                              repeats = 1, strata = "Crit_P")
```

```r
xgb_grid_3 <- expand.grid(tree_depth = c(2,3,4,5,6,7),
                          learn_rate = c(0.025,0.05,0.1,0.2))

xgb_10fold_3 <- xgb_wf_3 |>
  tune_grid(resamples = play_samples_cv_3, grid = xgb_grid_3,
            metrics = metric_set(accuracy))

xgb_10fold_3 |> autoplot()
```




```r
best_par_3 <- xgb_10fold_3 |>
  select_best()

final_xgb_3 <- xgb_wf_3 |>
  finalize_workflow(best_par_3) |>
  fit(data = donnees_entrainement_num_3)

# Prédiction du modèle
pred_xgb_3 <- final_xgb_3 |>
  predict(new_data = donnees_validation_num_3)

xgb_class_3 <- final_xgb_3 |>
```

```
  predict(new_data=donnees_validation_num_3,type="class")

xgb_prob_3 <- final_xgb_3 |>
  predict(new_data=donnees_validation_num_3,type="prob")


final_xgb_3 <- donnees_validation_num_3 |>
  bind_cols(xgb_class_3, xgb_prob_3)


confusion_xgb_3 <- yardstick::conf_mat(data=final_xgb_3,truth="Crit_P",
                                       estimate=".pred_class",
                                       dnn = c("Prediction", "Truth"))
```

Interprétation : Concernant le graphique de la précision du modèle XgBoost en fonction de la profondeur de l'arbre, on constate que les courbes ayant un taux d'apprentissage de 0.2 puis de 0.1 (à partir d'une profondeur d'arbre de 4) garantiraient une meilleure précision plus la profondeur des arbres s'élève.

- Evaluation du modèle

```
# Estimation du risque
risque.est_xgb_3 <- data.frame(Risque_est_xgb=mean(
  donnees_validation_num_3$Crit_P != pred_xgb_3$.pred_class))

print(risque.est_xgb_3)
```

```
  Risque_est_xgb
1            0.5
```

```
# Affichage de la matrice de confusion
confusion_xgb_3
```

```
          Truth
Prediction 1 2
         1 5 6
         2 5 6
```

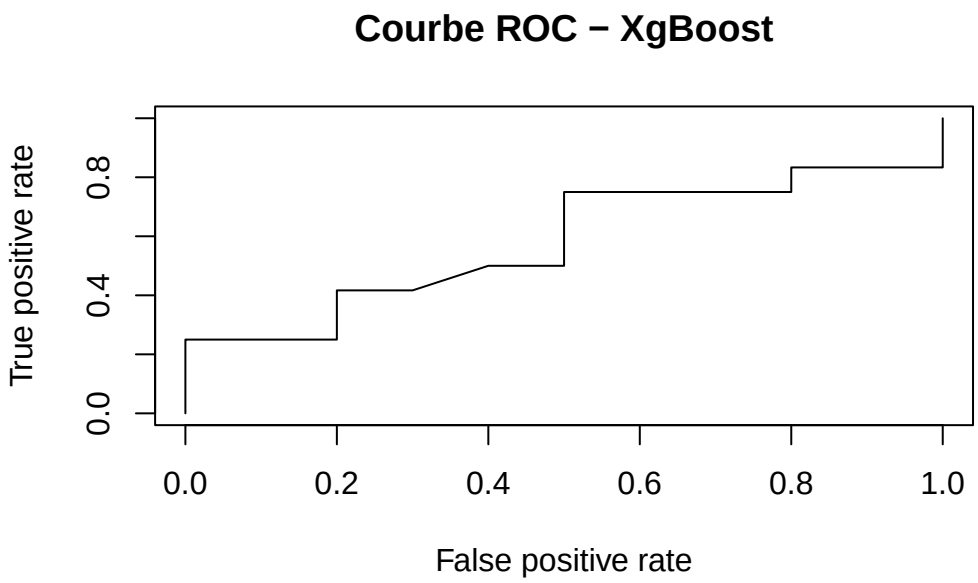
```
summary(confusion_xgb_3)
```

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.5
 2 kap                  binary         0
 3 sens                 binary         0.5
 4 spec                 binary         0.5
 5 ppv                  binary         0.455
 6 npv                  binary         0.545
 7 mcc                  binary         0
 8 j_index              binary         0
 9 bal_accuracy         binary         0.5
10 detection_prevalence binary         0.5
11 precision            binary         0.455
12 recall               binary         0.5
13 f_meas               binary         0.476
```

```r
# On récupère les prédictions de probabilité
xgb_probabilities_3 <- final_xgb_3 %>%
  select(.pred_1, .pred_2)

# Récupération des vraies valeurs
xgb_truth_values_3 <- donnees_validation_num_3$Crit_P

# Prédiction du modèle
xgb_roc_pred_3 <- prediction(xgb_probabilities_3$.pred_1, xgb_truth_values_3)

xgb_auc_roc_3 <- performance(xgb_roc_pred_3, "auc")@y.values[[1]]

# On affiches la valeur de l'AUC-ROC
print(paste("AUC-ROC:", xgb_auc_roc_3))
```

```
[1] "AUC-ROC: 0.579166666666667"
```

```r
xgb_roc_perf_3 <- performance(xgb_roc_pred_3, "tpr", "fpr")

# On trace la courbe ROC
plot(xgb_roc_perf_3, main = "Courbe ROC - XgBoost")
```



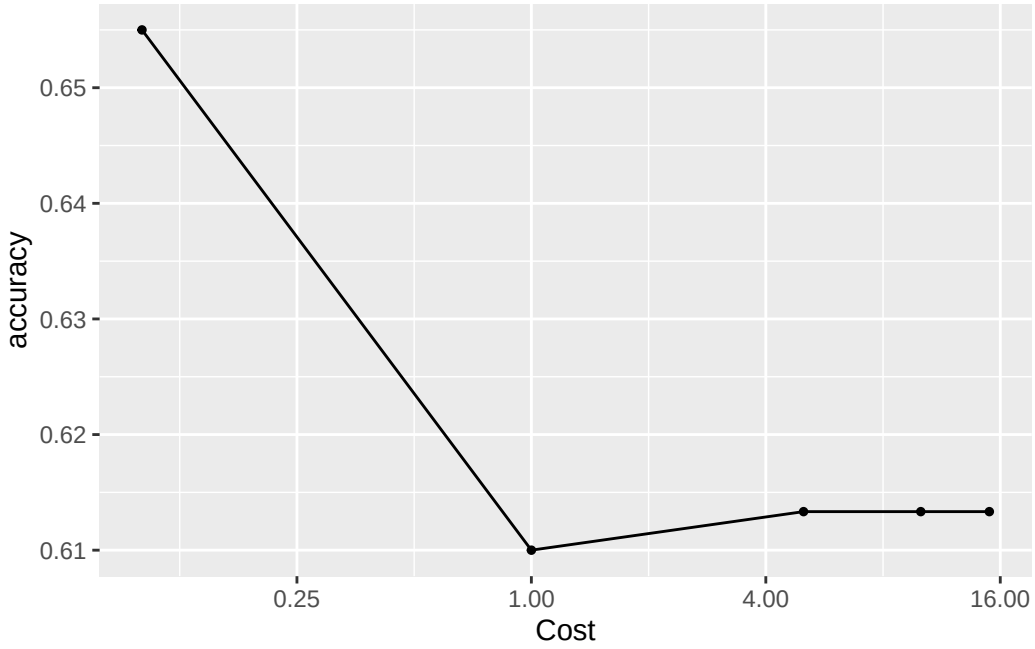
Interprétation : À partir de la matrice de confusion, on constate que notre modèle XgBoost effectue une prédiction plutôt moyenne des événements critiques et non critiques (50%). Logiquement, ce modèle a obtenu une estimation du risque d'également 50%, ce qui est faible en comparaison avec les autres algorithmes utilisés et les autres parties. Concernant la courbe Roc, on obtient une évaluation de performance du modèle de 0.55, ce qui concorde avec ceux qui a été observé précédemment.

- 5.3 / Bonus : Algorithme de prédiction par minimisation de risque empirique pénalisé : SVM

- Utilisation de SVM linéaire

```
# Création du modèle et du workflow
svml_model_3 <-
     svm_linear(cost = tune()) |>
     set_mode("classification") |>
     set_engine("kernlab")
svml_wf_3 <- workflow() |>
     add_recipe(play_recipe_3) |>
     add_model(svml_model_3)

# Ajustement de la constante de tolérance C par validation croisée
par_grid_3 <- tibble(cost=c(0.1,1,5,10,15))
```

```
svml_cv_3 <- svml_wf_3 |>
    tune_grid(resamples=play_samples_cv_3,grid=par_grid_3,
                metrics=metric_set(accuracy))
svml_cv_3 |>
  autoplot()
```




```
svml_cv_3 |>
    collect_metrics()
```

```
# A tibble: 5 x 7
   cost .metric  .estimator  mean     n std_err .config
  <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1   0.1 accuracy binary     0.655    10  0.0259 Preprocessor1_Model1
2   1   accuracy binary     0.61     10  0.0562 Preprocessor1_Model2
3   5   accuracy binary     0.613    10  0.0582 Preprocessor1_Model3
4  10   accuracy binary     0.613    10  0.0582 Preprocessor1_Model4
5  15   accuracy binary     0.613    10  0.0582 Preprocessor1_Model5
```

```
# Évaluation de la performance du modèle
best_C_3 <- svml_cv_3 |>
```

```
    select_best()
  final_svml_3 <- svml_wf_3 |>
    finalize_workflow(best_C_3) |>
    fit(data = donnees_entrainement_3)
```

```
 Setting default kernel parameters
```

```
Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
  svml_class_3 <- final_svml_3 |>
      predict(new_data=donnees_validation_3,type="class")
  svml_prob_3 <- final_svml_3 |>
      predict(new_data=donnees_validation_3,type="prob")
  svml_fit_3 <- donnees_validation_3 |>
    bind_cols(svml_class_3, svml_prob_3)

  pred_svml_3 <- final_svml_3 |>
       predict(new_data=donnees_validation_3)
```

Interprétation : On observe à travers ce graphique que la plus grande précision pour notre modèle est établie à partir d'un coût infèrieur 0.2, tandis qu'un coût supérieur semble dégrader sa performance (à partir de 4).

- Évaluation du modèle

```
  # Estimation du risque
  risque.est_svml_3 <- data.frame(Risque_est_svml=mean(
    donnees_validation_3$Crit_P!=pred_svml_3$.pred_class))

  print(risque.est_svml_3)
```

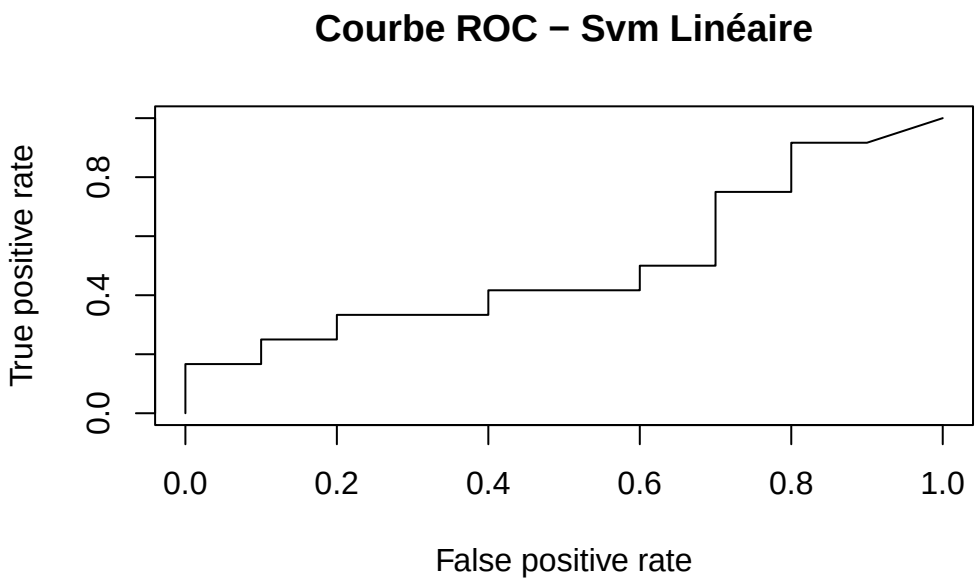
```
  Risque_est_svml
1       0.5454545
```

```
  # On récupère les prédictions de probabilité
  svml_probabilities_3 <- svml_fit_3 %>%
    select(.pred_0, .pred_1)
```

```r
# On récupère les vraies valeurs
svml_truth_values_3 <- donnees_validation_3$Crit_P

# On crée un objet ROCR prediction
svml_roc_pred_3 <- prediction(svml_probabilities_3$.pred_1, svml_truth_values_3)

# On calcule l'AUC-ROC
svml_auc_roc_3 <- performance(svml_roc_pred_3, "auc")@y.values[[1]]

# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", svml_auc_roc_3))
```

```
[1] "AUC-ROC: 0.504166666666667"
```

```r
# On trace la courbe ROC
svml_roc_perf_3 <- performance(svml_roc_pred_3, "tpr", "fpr")

plot(svml_roc_perf_3, main = "Courbe ROC - Svm Linéaire")
```



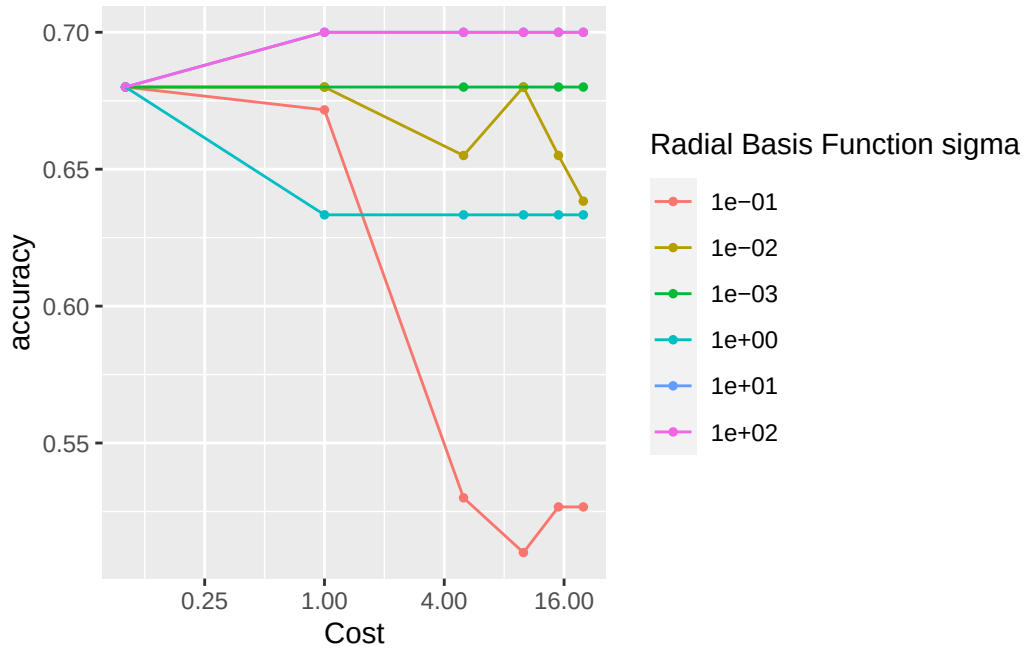
Interprétation : Globalement, ce modèle a obtenu une estimation du risque de 54%, ce qui est moyen compte tenu de nos types de données et des résultats des autres algorithmes.

Ceci est également visible à partir de la courbe ROC où l'on observe que la courbe ne s'approche pas de l'angle supérieur gauche du graphique, et tarde à atteindre le haut du graphique, ce qui

traduirait une spécificité et une sensibilité faibles. De ce fait, ce modèle ne serait définitivement pas idéal (0.50 de performance) en vue de prédire ce type de données de plaquage.

- Utilisation de SVM non linéaire à noyau gaussien (ou radial)

```
# Création du modèle et du workflow
svmr_model_3 <-
     svm_rbf(cost = tune(),rbf_sigma=tune()) |>
     set_mode("classification") |>
     set_engine("kernlab")

svmr_wf_3 <- workflow() |>
     add_recipe(play_recipe_3) |>
     add_model(svmr_model_3)

# Ajustement de la constante de tolérance C
# et de la fenêtre (variance) du noyau gaussien par validation croisée

par_grid_3 <- expand.grid(cost=c(0.1,1,5,10,15,20),rbf_sigma=10^(-3:2))

svmr_cv_3 <- svmr_wf_3 |>
     tune_grid(resamples=play_samples_cv_3,grid=par_grid_3,
               metrics=metric_set(accuracy))

svmr_cv_3 |> autoplot()
```




```
svmr_cv_3 |>
    collect_metrics()
```

```
# A tibble: 36 x 8
    cost rbf_sigma .metric  .estimator  mean     n std_err .config
   <dbl>     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1   0.1     0.001 accuracy binary     0.68     10  0.0209 Preprocessor1_Model01
 2   1       0.001 accuracy binary     0.68     10  0.0209 Preprocessor1_Model02
 3   5       0.001 accuracy binary     0.68     10  0.0209 Preprocessor1_Model03
 4  10       0.001 accuracy binary     0.68     10  0.0209 Preprocessor1_Model04
 5  15       0.001 accuracy binary     0.68     10  0.0209 Preprocessor1_Model05
 6  20       0.001 accuracy binary     0.68     10  0.0209 Preprocessor1_Model06
 7   0.1     0.01  accuracy binary     0.68     10  0.0209 Preprocessor1_Model07
 8   1       0.01  accuracy binary     0.68     10  0.0209 Preprocessor1_Model08
 9   5       0.01  accuracy binary     0.655    10  0.0259 Preprocessor1_Model09
10  10       0.01  accuracy binary     0.68     10  0.0427 Preprocessor1_Model10
# i 26 more rows
```

```
# Évaluation de la performance du modèle

best_par_3 <- svmr_cv_3 |>
```

```
    select_best()

final_svmr_3 <- svmr_wf_3 |>
    finalize_workflow(best_par_3) |>
    fit(data = donnees_entrainement_3)
```

```
Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svmr_class_3 <- final_svmr_3 |>
    predict(new_data=donnees_validation_3,type="class")

svmr_prob_3 <- final_svmr_3 |>
    predict(new_data=donnees_validation_3,type="prob")

svmr_fit_3 <- donnees_validation_3 |>
    bind_cols(svmr_class_3,svmr_prob_3)

# Prédiction du modèle

pred_svmr_3 <- final_svmr_3 |>
     predict(new_data=donnees_validation_3)
```

Interprétation : Le graphique traitant de la précision du modèle SVM à noyau gaussien (via divers sigma) en fonction du coût nous montre qu'un noyau égal à 1e+02 nous fournirait une meilleure performance du modèle (s'approchant de 0.70 en accuracy), quel que soit son coût. Cela permettrait ainsi de faire un compromis entre le sous-apprentissage et le sur-apprentissage du modèle.

- Évaluation du modèle

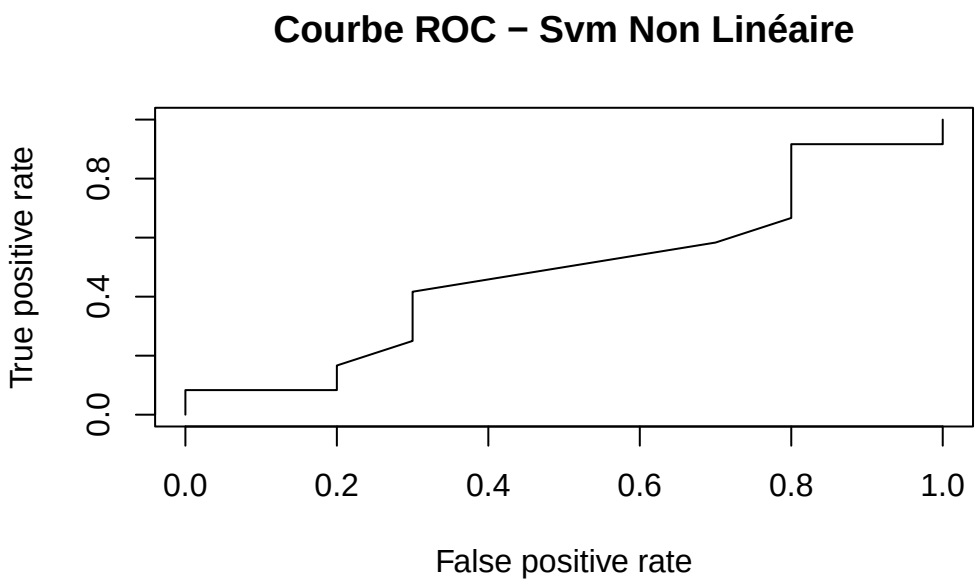
```
# Affichage du risque estimé
risque.est_svmr_3 <- data.frame(Risque_est_svmr=mean(
  donnees_validation_3$Crit_P!=pred_svmr_3$.pred_class))
print(risque.est_svmr_3)
```

```
  Risque_est_svmr
1       0.5454545
```

```r
# On récupère les prédictions de probabilité
svmr_probabilities_3 <- svmr_fit_3 %>%
  select(.pred_0, .pred_1)

# On récupère les vraies valeurs
svmr_truth_values_3 <- donnees_validation_3$Crit_P

# On crée un objet ROCR prediction
svmr_roc_pred_3 <- prediction(svmr_probabilities_3$.pred_1, svmr_truth_values_3)

# On calcule l'AUC-ROC
svmr_auc_roc_3 <- performance(svmr_roc_pred_3, "auc")@y.values[[1]]

# On affiche la valeur de l'AUC-ROC
print(paste("AUC-ROC:", svmr_auc_roc_3))
```

```
[1] "AUC-ROC: 0.483333333333333"
```

```r
# On trace la courbe ROC
svmr_roc_perf_3 <- performance(svmr_roc_pred_3, "tpr", "fpr")

plot(svmr_roc_perf_3, main = "Courbe ROC - Svm Non Linéaire")
```



Interprétation : Globalement, ce modèle a obtenu une estimation du risque de 54%, ce qui est élevé compte tenu de nos types de données et des résultats des autres algorithmes et des autres types de données. Ceci est également visible à partir de la courbe ROC, qui montre une spécificité et sensibilité faible, associé à une performance en dessous de 0.5 (0.48 de performance). Ainsi, on peut penser que l'utilisation de l'algorithme SVM non linéaire à noyau gaussien serait discutable pour ce type de données.

Récapitulatif et Conclusion des prédictions effectuées

- Tableau récapitulatif des valeurs obtenues pour le risque estimé et l'évaluation de performance de la courbe Roc à partir de plusieurs algorithmes de prédictions

Algorithme	Jeu de données	Risque estimé	Courbe ROC
Forêt aléatoire	Analyse glm	0.143	0.901
Adaboost	Analyse glm	0.286	0.732
XgBoost	Analyse glm	0.357	0.382
Svm linéaire	Analyse glm	0.286	0.354
Svm non linéaire	Analyse glm	0.357	0.635
Forêt aléatoire	Analyse stat m1	0.119	0.910
Adaboost	Analyse stat m1	0.286	0.800
XgBoost	Analyse stat m1	0.310	0.404
Svm linéaire	Analyse stat m1	0.262	0.738
Svm non linéaire	Analyse stat m1	0.286	0.501
Forêt aléatoire	Plaquage	0.182	0.900
Adaboost	Plaquage	0.545	0.479
XgBoost	Plaquage	0.500	0.579
Svm linéaire	Plaquage	0.545	0.504
Svm non linéaire	Plaquage	0.545	0.483

Concernant les capacités de prédictions de ces algorithmes à distinguer un évènement critique d'un non-critique, on constate que celui des forêt aléatoires se détache nettement du reste, et ceux qu'importe les données utilisées (avec des valeurs de risque s'approchant des 10% et une évaluation de la courbe Roc aux alentours de 0.90). À noter que les autres algorithmes sont relativement similaires entre eux, bien que assez peu précis en réalité, surtout pour prédire les évènements non critiques comme on a pu le voir précedemment.

Si on compare cette fois-ci les valeurs obtenus selon les types de données utilisées, il est intéressant de souligner que nos modèles ont réalisé de meilleurs prédictions à partir des variables selectionnés à partir du glm, ainsi qu'en amont du devoir. Néanmoins, concernant les données exclusivement de plaquage, nos modèles se sont en réalité trompé 1 fois sur 2, ce qui n'est pas

satisfaisant.

Il convient tout de même de tempérer nos conclusions effectuées puisque la quantité de données disponibles n'est pas suffisament grande pour attester de l'efficicacité réelle d'un modèle en termes de prédictions (surtout pour celles de plaquages pour lequel on avait seulement 22 données de tests par exemple). Un jeu de données plus complet nous aurait ainsi permis d'avoir plus de certitudes sur cela.

À noter que nous avons obtenu nos meilleurs résultats en termes de prédictions via l'algorithme de forêt aléatoire associé aux jeu de données issues des variables sélectionné à partir d'une analyse statistique réalisé en amont du devoir. De ce fait, il serait judicieux de creuser cette piste afin de founir un travail plus affiné sur le sujet, et de comprendre les raisons de cela.