Rendu Travaux Pratiques 3

Règles de rendu

- Chaque TP donne lieu à un bref compte-rendu portant sur certaines questions posées dans l'énoncé du TPs.
- Le compte-rendu doit être complété à partir du texte de l'énoncé. Les codes R doivent être inclus dans le texte du compte-rendu (menu **Insert**) et commentés avec précision. **Les commentaires compteront pour une part importante dans la note**.
- Le compte-rendu doit être déposé **sur TEIDE à la fin de la séance de TP**. Les rendus en retard seront fortement pénalisés.
- Le compte-rendu doit être déposé **sur TEIDE au format HTML uniquement**. Utiliser la fonction **Preview** ou **knitr** du menu de rstudio pour obtenir le document au format souhaité. **Les fichiers "source" (Rmd) ne seront pas acceptés par les correcteurs**.

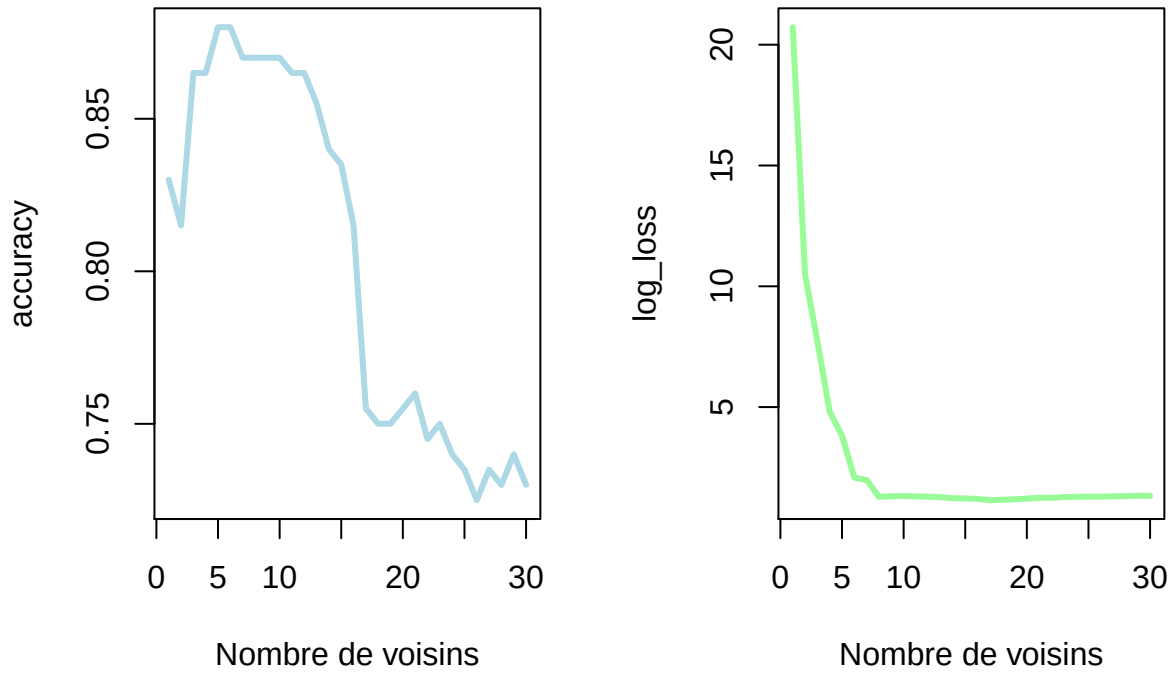
```
#install.packages("devtools")
#devtools::install_github("bcm-uga/isd")
library(isd)
```

Exercice 1 : Données simulées (knn)

- Méthode `knn` : Pour $k = 1, \ldots, 30$, calculer l'erreur de classification et la perte log loss à partir de l'ensemble test (on modifiera 0 ou 1 les probabilités pour éviter les valeurs "Inf"). Représenter ces résultats sous la forme de graphes "accuracy" et "logloss" en fonction de k.

```
x <- isd::rhastib(n_train = 200,
                  n_test = 200,
                  n_subclass = 10,
                  sigma2 = 0.05)

accuracy <- NULL
log_loss <-  NULL

for (k in 1:30){

  # Méthode k-plus proches voisins de l'ensemble test récupérant 10 voisins entre l'ensemble test et l'
  mod_knn <- class::knn(train = x$train,
                        test = x$test,
                        cl = x$class_test,
                        k = k,
                        prob = TRUE)

  # On considère la classification des points comme notre classification de prédiction (on récupère en
  class_pred <- mod_knn
  prob_class <- attr(mod_knn,"prob")

  # les probabilités sont modifiées pour éviter Inf
  prob_class[prob_class == 1] <- 1 - 1e-09
  prob_class[prob_class == 0] <- 1e-09
```

```
  # On calcule la précision pour chaque k
  accuracy[k] <- mean(class_pred == x$class_test)

  boo <- (class_pred == x$class_test)
  log_loss[k] <- - mean(log(prob_class[boo])) - mean(log(1 - prob_class[!boo]))
}

par(mfrow = c(1, 2)) # divise la fenetre en 1 ligne 2 colonnes
plot(accuracy, col = "lightblue", type = "l", lwd = 3, xlab = "Nombre de voisins")
plot(log_loss, col = "palegreen", type = "l", lwd = 3, xlab = "Nombre de voisins")
```




```
# k meilleure précision
k_max_accuracy = match(max(accuracy),accuracy)
print(k_max_accuracy)
```

```
## [1] 5
```

```
print(c("accuracy: ", accuracy[k_max_accuracy]))
```

```
## [1] "accuracy: " "0.88"
```

```
print(c("logloss: ", log_loss[k_max_accuracy]))
```

```
## [1] "logloss: "        "3.81402504331137"
```

```
# k plus faible perte
k_min_log_loss = match(min(log_loss),log_loss)
print(k_min_log_loss)
```

```
## [1] 17
```

```
print(c("accuracy: ", accuracy[k_min_log_loss]))
```

```
## [1] "accuracy: " "0.755"
```

```
print(c("logloss: ", log_loss[k_min_log_loss]))
```

```
## [1] "logloss: "        "1.14982548058221"
```

- Quel choix de k vous parait le plus pertinent pour la simulation effectuée ?

On sait que la valeur de k dépend de l'échantillon et de la simulation. Mais d'après moi, pour $k \in [0, 13]$, on trouve qu'il y a une meilleure précision sur la prédiction dans l'ensemble test ce qui parait le plus pertinent, même si pour $k \in]11, 30]$, la perte y est minimale. De plus, le logloss de k ayant la meilleure précision ne présente pas une grande différence avec le k ayant le logloss minimum.

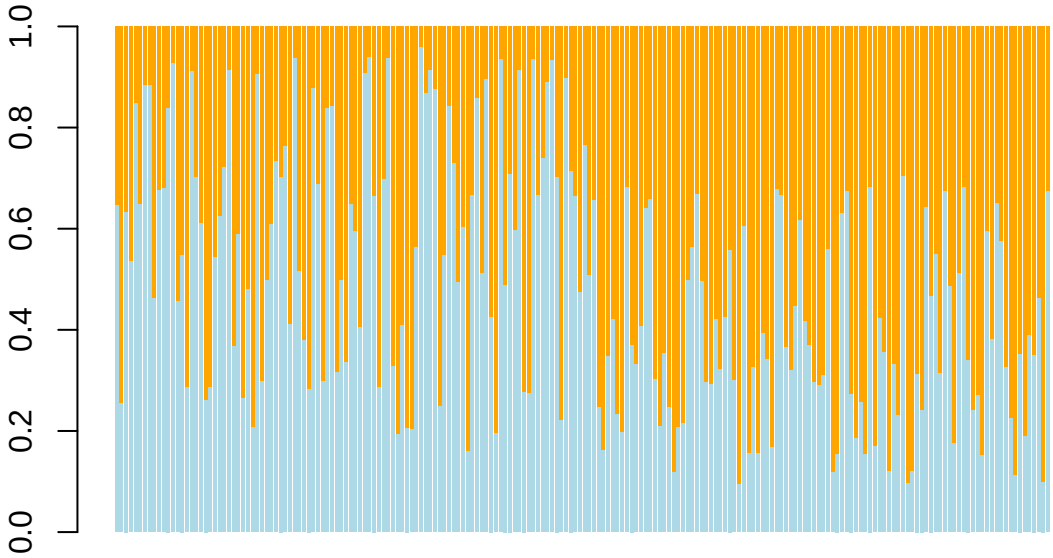
Exercice 2 : Données simulées (lda)

- Méthode `lda` : Calculer le taux de bonne classification et la perte log loss sur l'ensemble test.

```
require(MASS)
```

```
## Loading required package: MASS
```

```
help(lda)

#
    mod_lda <- MASS::lda(x = x$test,
                         grouping = x$class_test)

help(predict.lda)

#
    pred <- predict(mod_lda, newdata = x$test)

# Les  probabilités correspondent aux classes des variables testées
    prob_class <- pred$posterior

# On génère un diagramme en barres montrant les répartitions des probabilités entre les points bleus et
    barplot(t(prob_class), col = c("lightblue", "orange"),
            border = NA,
            xlab = "Test set")
```



Test set

```
    head(prob_class)
```

```
##      lightblue    orange
## [1,] 0.6461318 0.3538682
## [2,] 0.2555792 0.7444208
## [3,] 0.6331156 0.3668844
## [4,] 0.5369947 0.4630053
## [5,] 0.8481167 0.1518833
## [6,] 0.6480960 0.3519040
```

```
# On détermine le taux de bonne classification (précision)
  accuracy <- mean(x$class_test == pred$class)
  cat("Accuracy = ", accuracy, "\n")
```

```
## Accuracy =  0.685
```

```
#
  log_loss <- -mean( (x$class_test == "lightblue")*log(1-prob_class) + (x$class_test == "orange")*log(p
  cat("Logloss = ", log_loss, "\n")
```

```
## Logloss =  0.8553465
```

Exercice 3 : Données simulées (nnet)

- Méthode `nnet` : Pour `decay`$ = 0, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1$, calculer le taux de bonne classification et la perte log loss sur l'ensemble test. Représenter ces résultats sous forme de tableau (accuracy/logloss en fonction de `decay`).

```r
  accuracy <- NULL
  log_loss <-  NULL

  decay <- c(0, 10^(-(6:0)))

  require(nnet)
```

```
## Loading required package: nnet
```

```r
  is_it_orange <- (x$class_train == "orange")
# On ajuste des modeles nnet pour 7 valeurs du paramètre decay
# decay est un paramètre de régularisation

  for (lambda in decay){

    # neural net
    mod_nnet <- nnet( x = x$train,
                      y = is_it_orange,
                      size = 30,
                      decay = lambda,
                      maxit = 500,
                      entropy = TRUE,
                      trace = FALSE)

    # probabilité de prédiction en orange
    prob_class <- predict(mod_nnet, newdata = data.frame(x$test))

    prob_class[prob_class > 1 - 1e-08]  <- 1 - 1e-08
    prob_class[prob_class < 1e-08] <- 1e-08

    # calcul du taux de bonne précision
    accuracy <- c(accuracy,
                  mean(is_it_orange == (prob_class > 0.5)))

    # calul de la perte log loss
    log_loss <- c( log_loss,
                - mean((x$class_test == "lightblue")*log(1 - prob_class)) - mean((is_it_orange)*log(prob
  }

  library(magrittr)

  names(accuracy) <- as.character(decay)
  names(log_loss) <- as.character(decay)

  data.frame(accuracy, log_loss)  %>% knitr::kable(digit = 2)
```

	accuracy	log_loss
0	0.82	3.00
1e-06	0.82	2.29
1e-05	0.82	1.94
1e-04	0.82	1.55
0.001	0.84	0.88
0.01	0.86	0.31
0.1	0.70	0.51

	accuracy	log_loss
1	0.68	0.57

Exercice 4 : "Wisconsin Breast Cancer Database"

```
library(mlbench)
data(BreastCancer)
boo_na <- !apply(BreastCancer, 1, anyNA)
breast_cancer <- BreastCancer[boo_na,-1]
```

- À l'aide de l'ensemble test, évaluer les taux de classification et de perte log loss pour les méthodes `lda`, `nnet` et `knn`. Pour `knn` et `nnet`, utiliser, dans un premier temps, les paramètres $k = 15$ et `decay`$= 0.01$.

```
# On retranche les données en ensembles de tests (cancer_test) et d'apprentissage (cancer_train)

cancer_train <- breast_cancer[(0:546),]
cancer_test <- breast_cancer[-(0:546),]
```

Méthode analyse discriminante linéaire:

```
  # lda de l'ensemble d'apprentissage
    mod_lda <- MASS::lda(cancer_train$Class ~ ., data = cancer_train[,-10])

  # probabilité à priori de l'ensemble d'apprentissage
    pred_lda <- predict(mod_lda, newdata = cancer_test[,-10])$class

  # précision entre les 2 ensemble (apprentissage et test)
    accuracy_lda <- mean(pred_lda == cancer_test$Class)

  # probabilité de l'ensemble testé
    prob_lda <- predict(mod_lda, newdata = cancer_test[,-10])$posterior

  # logloss
    log_loss_lda <- -mean((cancer_test$Class == "benign")*log(1 - prob_lda) + (cancer_test$Class == "
    accuracy_lda
```

```
## [1] 0.9708029
```

```
    log_loss_lda
```

```
## [1] 9.121967
```

Méthode du réseau de neuronnes:

```
  # nnet
require(nnet)
y <-  as.numeric(cancer_train$Class == "malignant")

mod_nnet <- nnet::nnet(x = cancer_train[,-10],
                       y = y,
                       size = 30,
                       entropy = TRUE,
                       decay = 0.01,
                       trace = FALSE)

prob_nnet <- predict(mod_nnet, cancer_test[,-10])
```

```
  prob_nnet[prob_nnet == 0] <- 1e-08
  prob_nnet[prob_nnet == 1] <- 1 - 1e-08

  accuracy_nnet <- mean((prob_nnet > 0.5) == (cancer_test$Class == "malignant"))

  log_loss_nnet <- -mean((cancer_test$Class == "benign")*log(1 - prob_nnet) + (cancer_test$Class == "ma

  accuracy_nnet
```

```
## [1] 0.9708029
```

```
  log_loss_nnet
```

```
## [1] 0.066871
```

Méthode des k-plus proches voisins:

```
    # knn
     mod_knn <- class::knn(cancer_train[,-10],
                           cancer_test[,-10],
                           cancer_train$Class,
                           k = 15,
                           prob = TRUE)

      accuracy_knn <- mean(mod_knn == cancer_test$Class)
      accuracy_knn
```

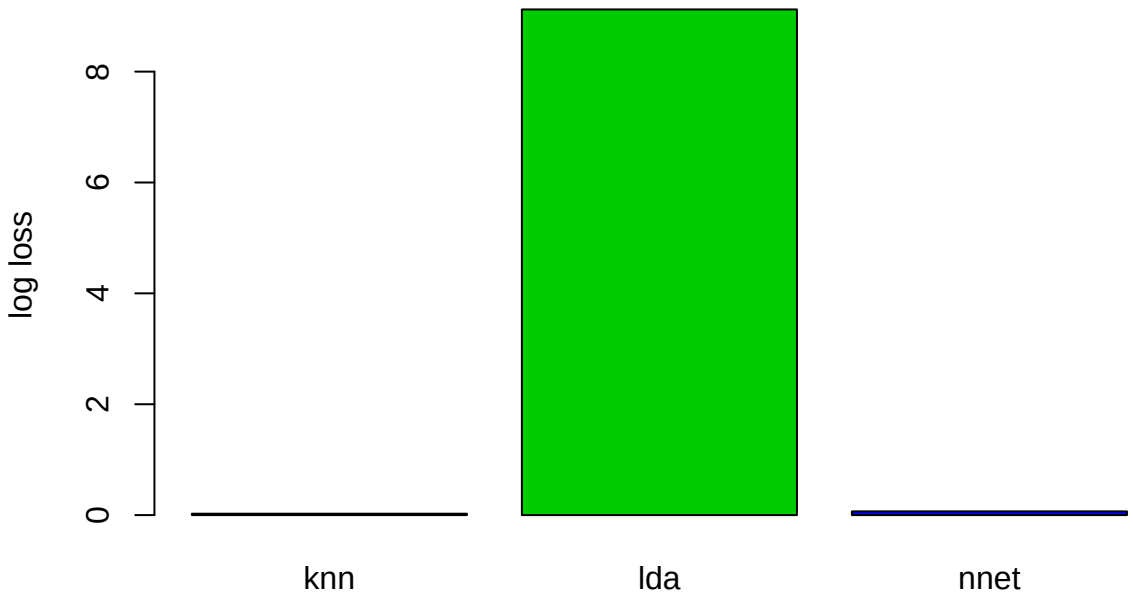
```
## [1] 1
```

```
      prob_class <- attr(mod_knn, "prob")
      prob_class[prob_class == 1] <-  1 - 1e-08

      log_loss_knn <- - mean( (mod_knn == cancer_test$Class)*log(prob_class) + (mod_knn !=     cancer_t
      log_loss_knn
```

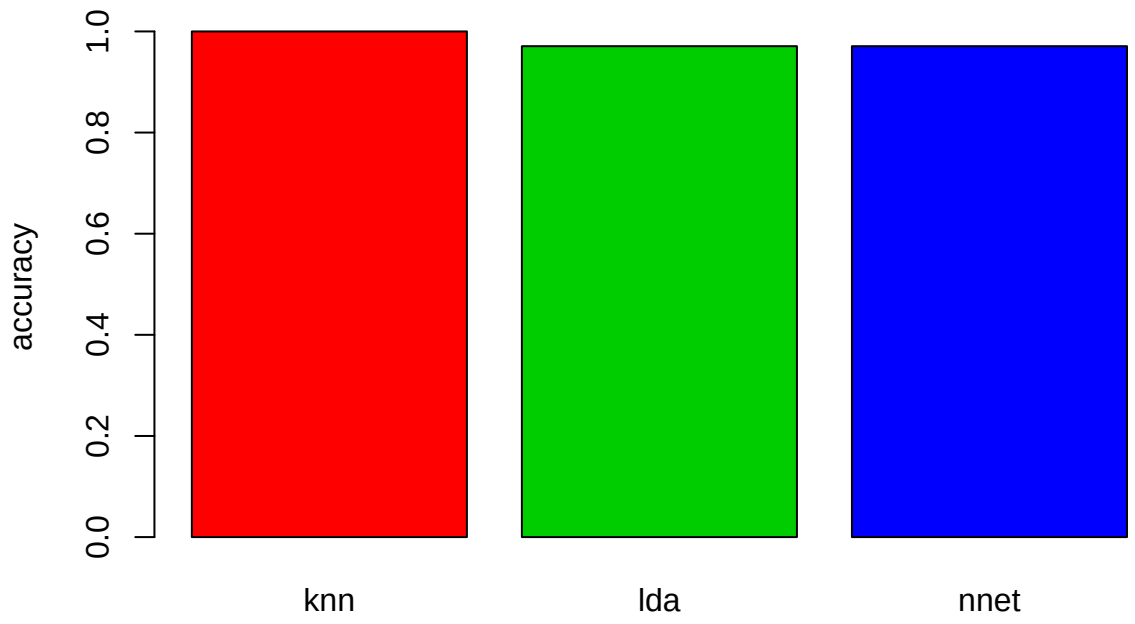
```
## [1] 0.02489309
```

- Reporter sous forme de tableau avec des valeurs arrondies les valeurs des taux de classification et de perte log loss obtenues pour les méthodes `knn`, `lda`, `nnet` (dans cet ordre). Quel choix de prédicteur vous parait être le meilleur ? Justifier.

```
  log_loss <-  c(log_loss_knn, log_loss_lda, log_loss_nnet)

  names(log_loss) <-  c("knn", "lda", "nnet")

  barplot(log_loss, col = 2:4, ylab = "log loss")
```

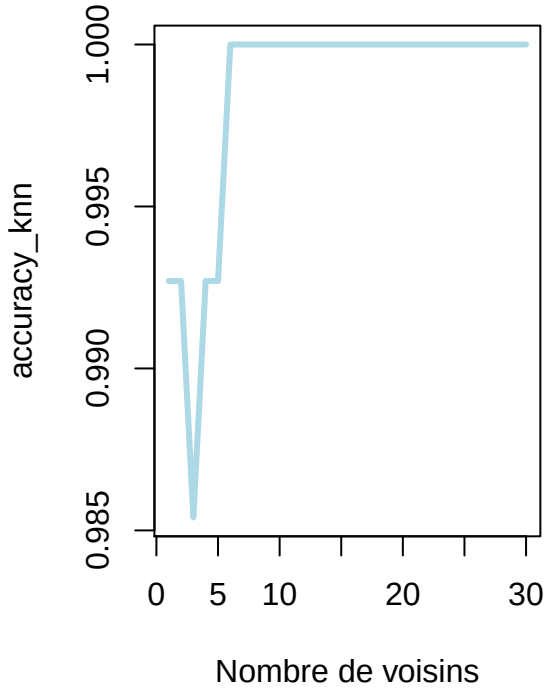


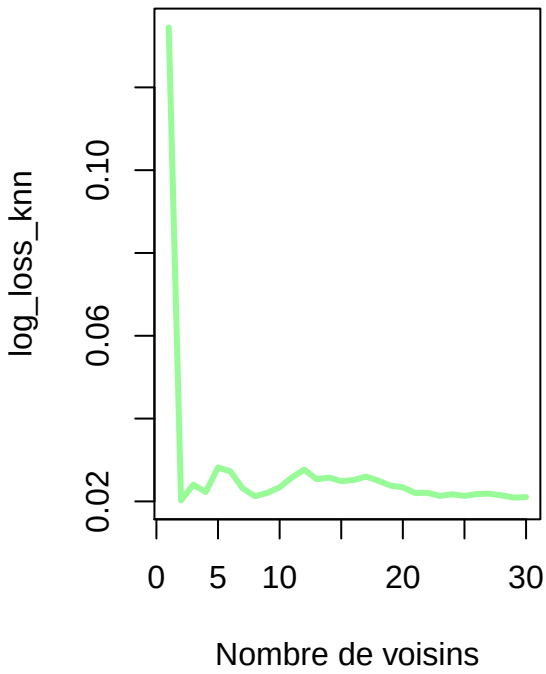

```
accuracy <- c(accuracy_knn, accuracy_lda, accuracy_nnet)

names(accuracy) <-  c("knn", "lda", "nnet")

barplot(accuracy, col = 2:4, ylab = "accuracy")
```



- Pour `knn` et `nnet`, explorer les paramètres de "complexité" (k et `decay`) conduisant aux meilleures performances. Reporter les performances des modèles correspondant dans un tableau.

```
  # knn
accuracy_knn = NULL
log_loss_knn = NULL
n = c(1:30)
for (k in n){
  mod_knn <- class::knn(cancer_train[,-10],
                        cancer_test[,-10],
                        cancer_train$Class,
                        k = k,
                        prob = TRUE)

  accuracy_knn[k] <- mean(mod_knn == cancer_test$Class)
  accuracy_knn[k]

  prob_class <- attr(mod_knn, "prob")
  prob_class[prob_class == 1] <-  1 - 1e-08

  log_loss_knn[k] <- - mean( (mod_knn == cancer_test$Class)*log(prob_class) + (mod_knn !=     cancer_
  log_loss_knn[k]
}

par(mfrow = c(1, 2)) # divise la fenetre en 1 ligne 2 colonnes
plot(accuracy_knn, col = "lightblue", type = "l", lwd = 3, xlab = "Nombre de voisins")
plot(log_loss_knn, col = "palegreen", type = "l", lwd = 3, xlab = "Nombre de voisins")
```






```
library(magrittr)
data.frame(n, accuracy_knn, log_loss_knn)  %>% knitr::kable(digit = 2)
```

n	accuracy_knn	log_loss_knn
1	0.99	0.13
2	0.99	0.02
3	0.99	0.02
4	0.99	0.02
5	0.99	0.03
6	1.00	0.03
7	1.00	0.02
8	1.00	0.02
9	1.00	0.02
10	1.00	0.02
11	1.00	0.03
12	1.00	0.03
13	1.00	0.03
14	1.00	0.03
15	1.00	0.02
16	1.00	0.03
17	1.00	0.03
18	1.00	0.03
19	1.00	0.02
20	1.00	0.02

n	accuracy_knn	log_loss_knn
21	1.00	0.02
22	1.00	0.02
23	1.00	0.02
24	1.00	0.02
25	1.00	0.02
26	1.00	0.02
27	1.00	0.02
28	1.00	0.02
29	1.00	0.02
30	1.00	0.02

```
  # nnet
accuracy_nnet <- NULL
log_loss_nnet <-  NULL
decay <- c(0, 10^(-(6:0)))
for (lambda in decay){
    y <-  as.numeric(cancer_train$Class == "malignant")

    mod_nnet <- nnet::nnet(x = cancer_train[,-10],
                           y = y,
                           size = 30,
                           entropy = TRUE,
                           decay = lambda,
                           trace = FALSE)

    prob_nnet <- predict(mod_nnet, cancer_test[,-10])

    prob_nnet[prob_nnet == 0] <- 1e-08
    prob_nnet[prob_nnet == 1] <- 1 - 1e-08

    accuracy_nnet <- c(accuracy_nnet, mean((prob_nnet > 0.5) == (cancer_test$Class == "malignant")))

    log_loss_nnet <- c(log_loss_nnet, -mean((cancer_test$Class == "benign")*log(1 - prob_nnet) + (can

}
library(magrittr)

names(accuracy_nnet) <- as.character(decay)
names(log_loss_nnet) <- as.character(decay)

data.frame(accuracy_nnet, log_loss_nnet)  %>% knitr::kable(digit = 2)
```

	accuracy_nnet	log_loss_nnet
0	0.97	0.16
1e-06	0.95	0.53
1e-05	0.98	0.19
1e-04	0.94	0.58
0.001	0.98	0.11
0.01	0.97	0.04
0.1	0.97	0.07
1	1.00	0.04

Défi "Wisconsin Breast Cancer Database"

- Pour les méthodes `knn`, `lda`, `nnet`, calculer les probabilités de la classe "malignant" pour chaque élément de la l'ensemble test. On choisira les paramètres `k` et `decay` donnant les meilleures performances possibles.
- Pour les méthodes `knn`, `lda`, `nnet`, appliquer la fonction `eval_cancer` et présenter les résultat sous forme de tableau (`data.frame`).