Web page Phishing Detection

Thomas Fernandes, Yassine Ouerghi, Vanessa Kenniche, Mario Miron Ramos

2023-11-09

L'étude vise à prédire la légitimité des sites web en utilisant diverses techniques de machine learning. Le phénomène du phishing consiste en des tentatives de fraude en ligne par le biais de sites web frauduleux imitant des sites légitimes.

La variable que nous cherchons à prédire est "status", qui indique si un site web est légitime ou potentiellement frauduleux (phishing). Pour ce faire, nous disposons d'un ensemble de données équilibré de 87 variables explicatives différentes, chacune fournissant des informations sur divers aspects de 11430 sites web différents. Ces données incluent 56 variables basées sur la structure, 24 extraites du contenu des pages web correspondantes, 7 obtenues par des requêtes auprès de services externes.

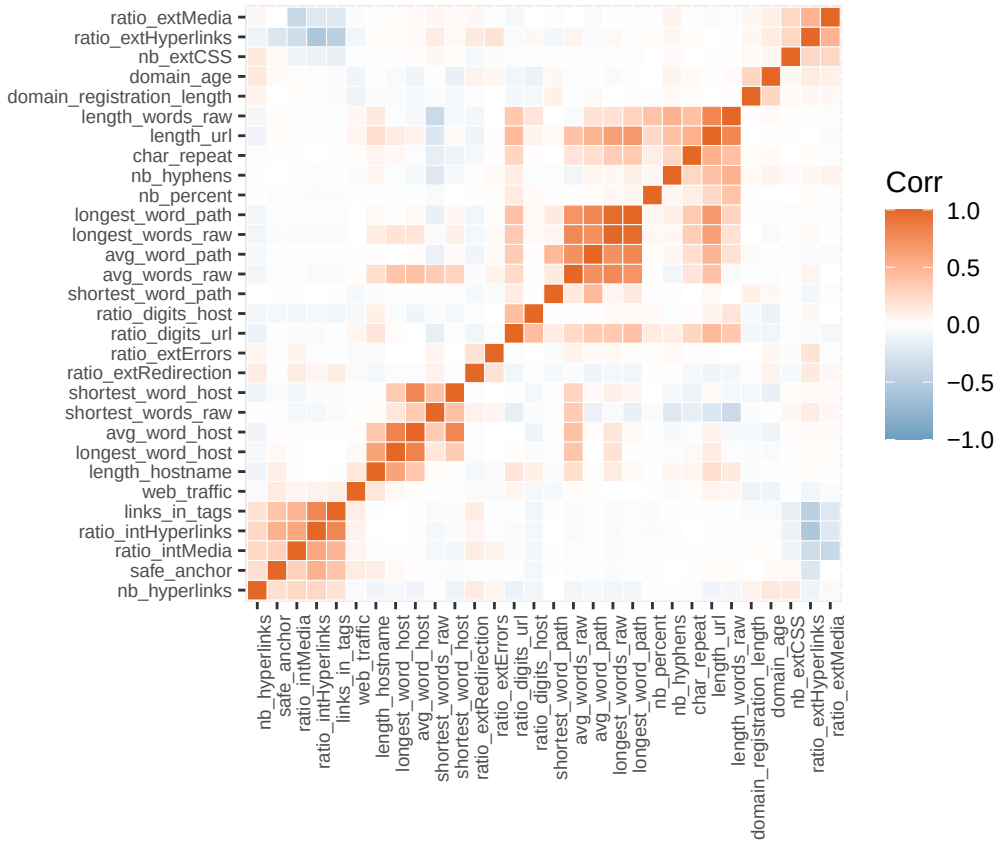
1. Présentation des données

Avant de commencer les différentes modélisations, nous allons regarder comment se structurent nos données.

1.1. Corrélation entre les variables quantitatives

```
df_present <- df

#Extraire les variables qualitatives
v_quali <- vector("logical", length = ncol(df_present) - 1)
for (i in 2:ncol(df_present)) {
  v_quali[[i]] <- (length(unique(df_present[[i]])) / sum(!is.na(df_present[[i]]))) < 0.002
}

num_cols <- character()
cat_cols <- character()

for (i in 1:length(v_quali)) {
  if (!v_quali[[i]]) {
    num_cols <- c(num_cols, names(df_present)[i])
  } else {
    cat_cols <- c(cat_cols, names(df_present)[i])
  }
}

corr <- cor(df_present[num_cols])

ggcorrplot(
  corr,
  hc.order = TRUE,
  type = "full",
  outline.color = "white",
```

```
  ggtheme = ggplot2::theme_gray,
  colors = c("#6D9EC1", "white", "#E46726"),
  show.diag = TRUE,
  tl.cex = 7,
  tl.srt = 90
)
```

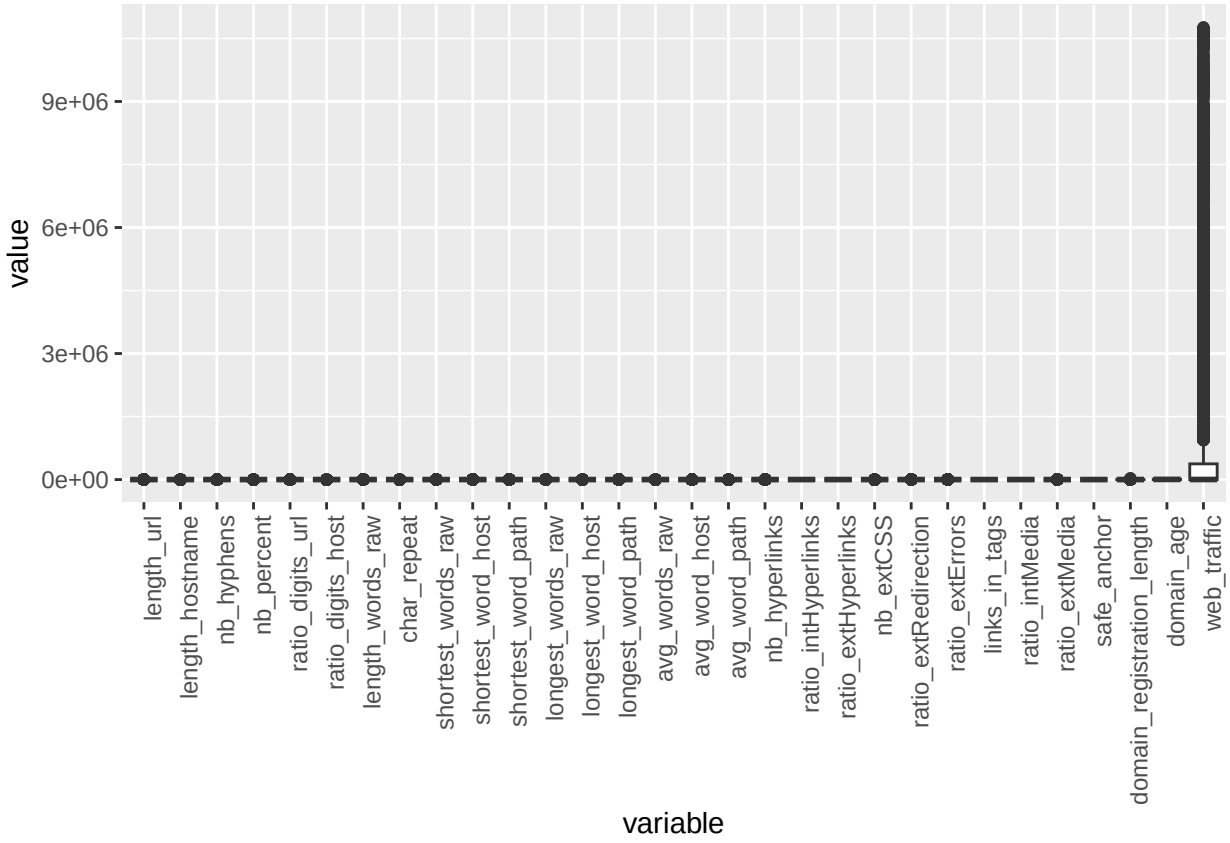


Comme on s'y attendait, on remarque que de nombreuses variables sont corrélées entre elles. C'est le cas par exemple de longest_word_path et de avg_word_path.

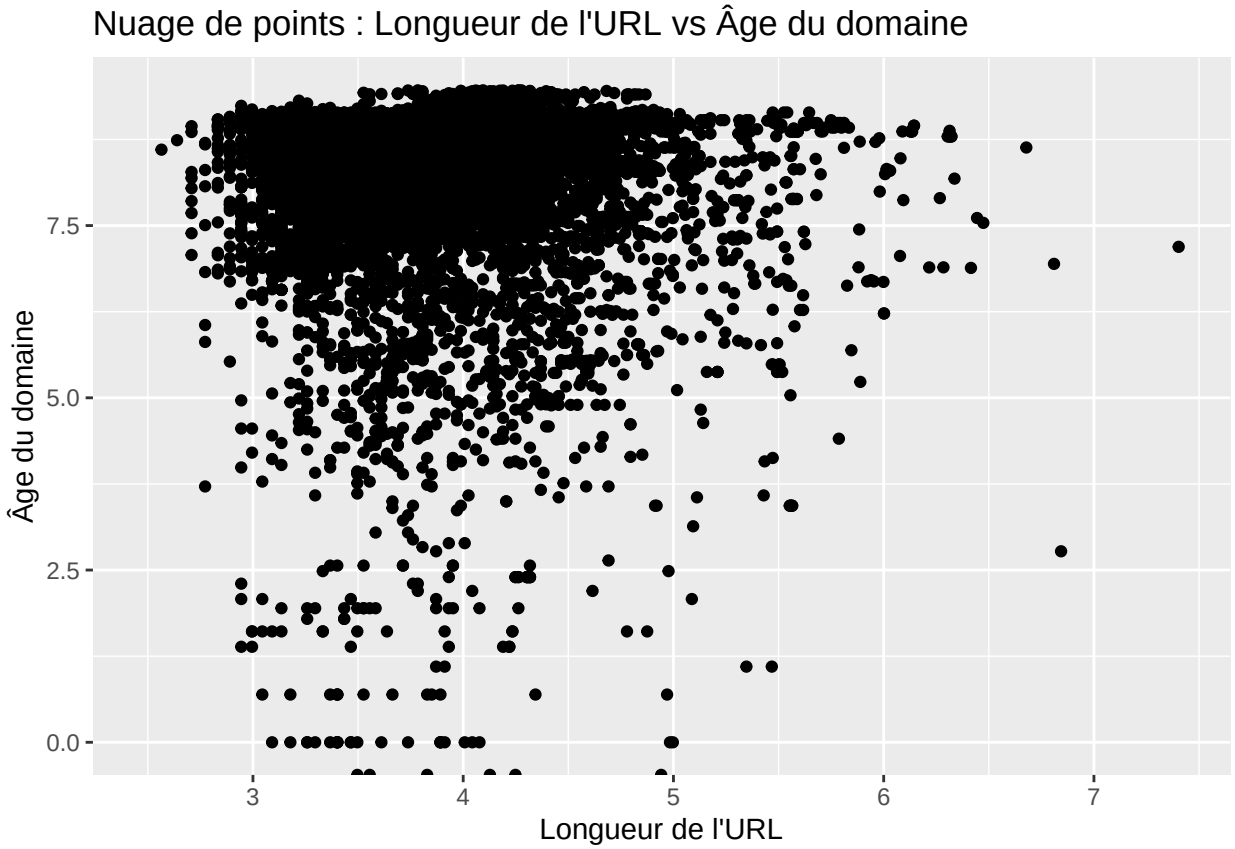
On fait un boxplot de toutes les variables

```
ggplot(data = melt(df_present[, num_cols]), aes(x = variable, y = value)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

```
## No id variables; using all as measure variables
```




```r
attach(df)

ggplot(df, aes(x = log(length_url), y = log(domain_age))) +
  geom_point() +
  labs(x = "Longueur de l'URL", y = "Âge du domaine") +
  ggtitle("Nuage de points : Longueur de l'URL vs Âge du domaine")
```



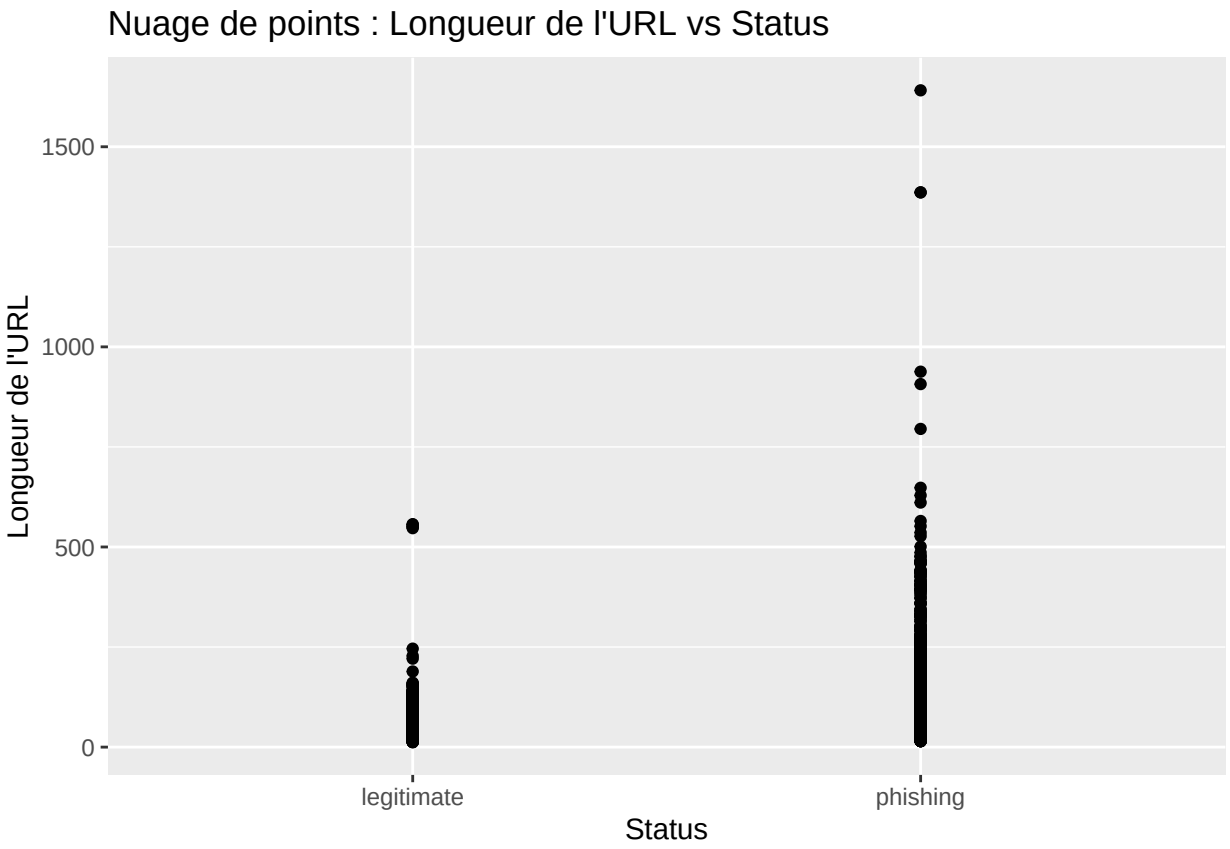

```r
max(domain_age)
```

```
## [1] 12874
```

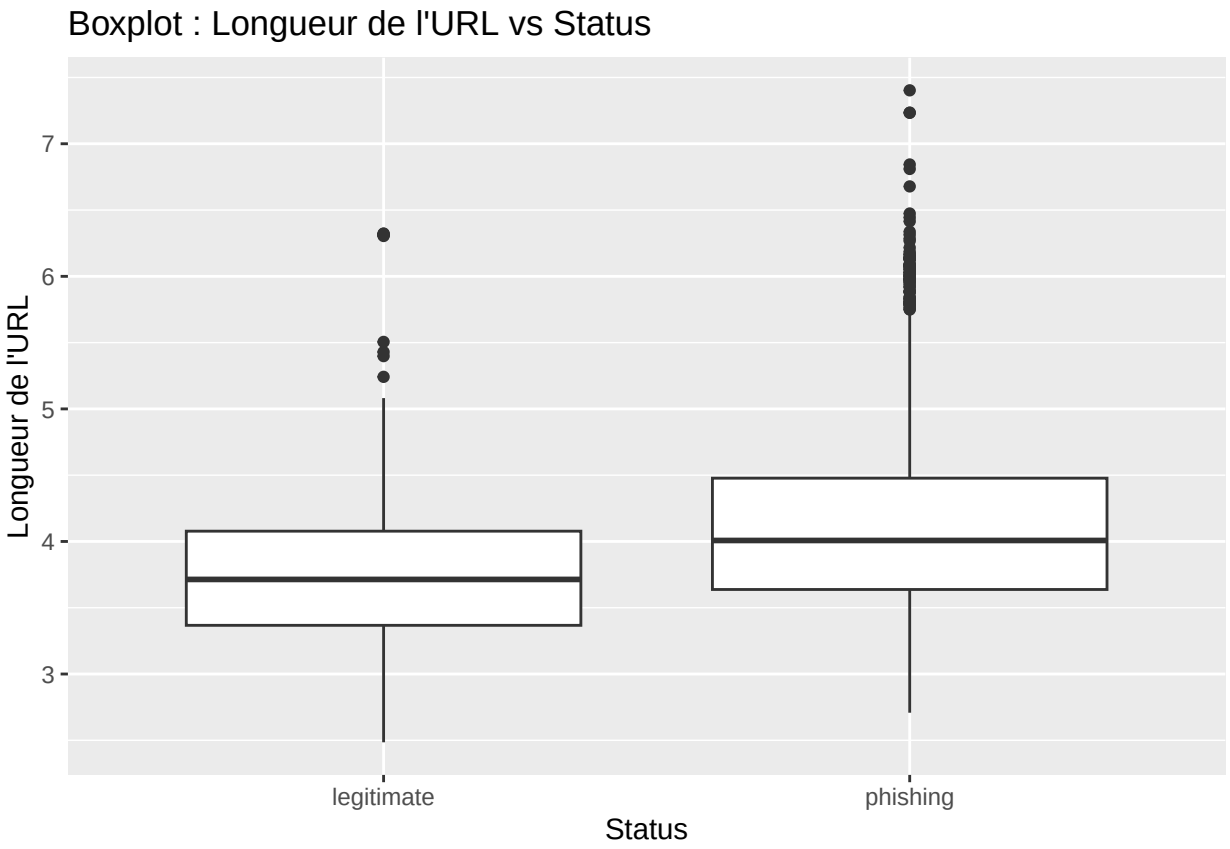
```r
min(domain_age)
```

```
## [1] -12
```

```r
# Nuage de point y = longueur url, x = status
ggplot(df, aes(x = status, y = length_url)) +
  geom_point() +
  labs(x = "Status", y = "Longueur de l'URL") +
  ggtitle("Nuage de points : Longueur de l'URL vs Status")
```

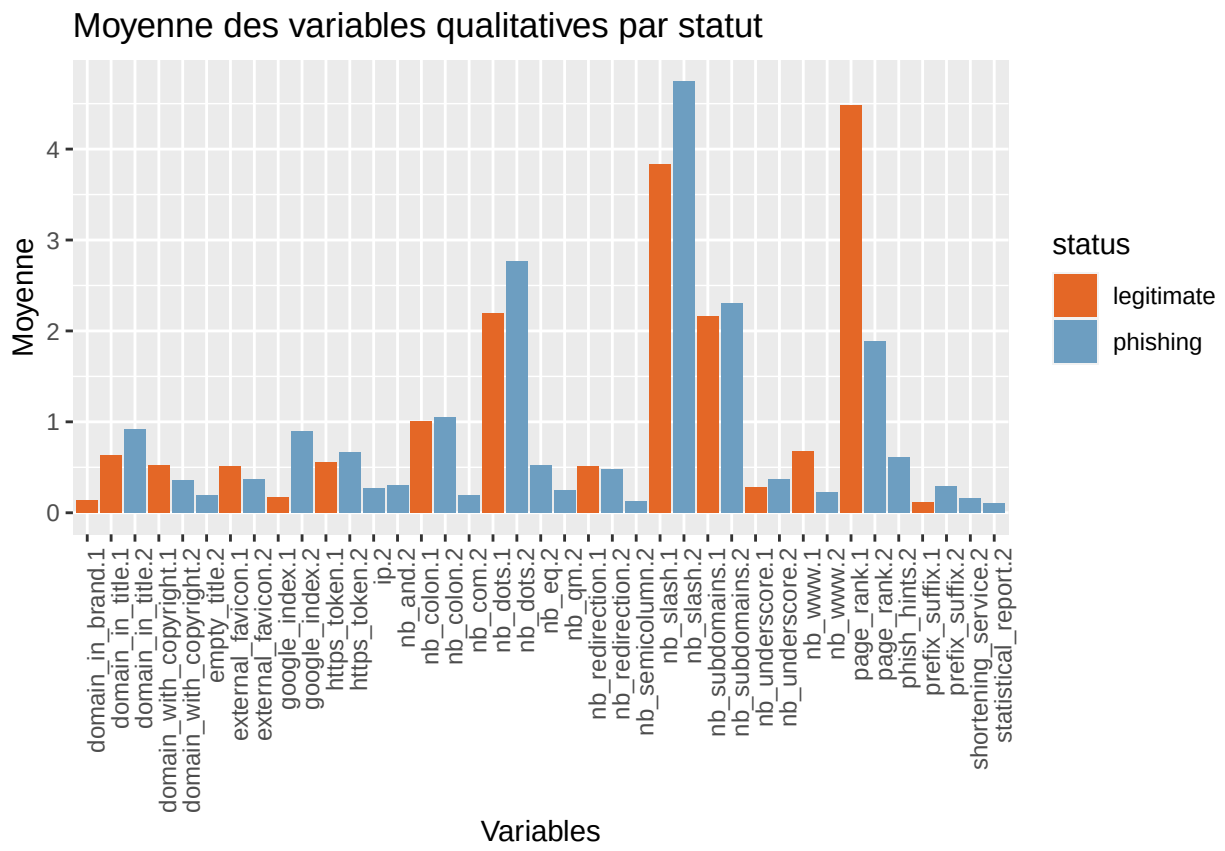



```
# Boxplot
ggplot(df, aes(x = status, y = log(length_url))) +
  geom_boxplot() +
  labs(x = "Status", y = "Longueur de l'URL") +
  ggtitle("Boxplot : Longueur de l'URL vs Status")
```



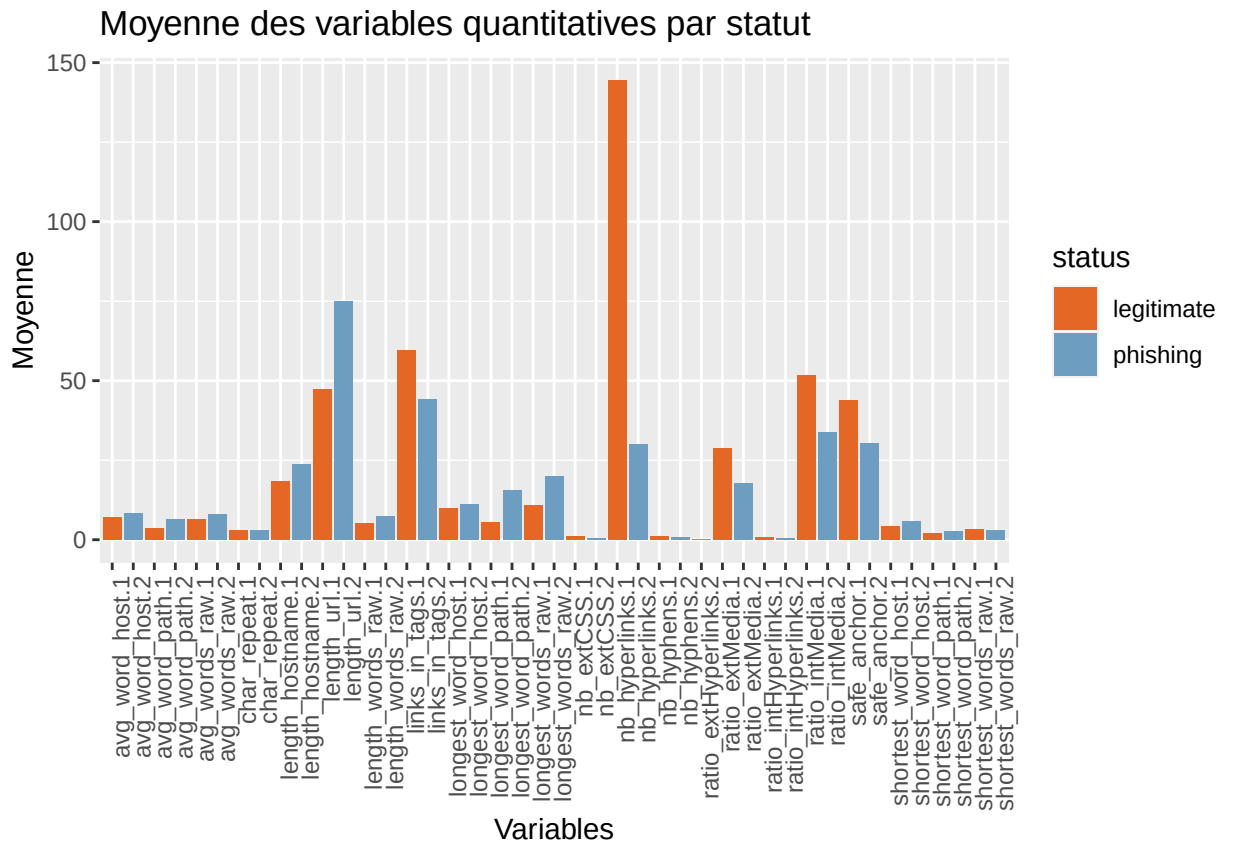
1.2. Moyenne par statut

```r
mean_by_status <- function(df, col_name) {
  df %>%
    group_by(status) %>%
    summarise(mean_value = mean(.data[[col_name]], na.rm = TRUE))
}
mean_values_list_cat <- list()

for (col in cat_cols) {
  mean_values_list_cat[[col]] <- mean_by_status(df_present, col)
}

mean_values_df_cat <- do.call(rbind, mean_values_list_cat)
mean_values_df_cat$col_names <- rownames(mean_values_df_cat)

mean_values_df_cat <- mean_values_df_cat[mean_values_df_cat$mean_value > 0.1 | mean_values_df_cat$mean_v
mean_values_df_cat <- mean_values_df_cat[!is.na(mean_values_df_cat$mean_value), ]

ggplot(mean_values_df_cat, aes(x = col_names, y = mean_value, fill = status)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Variables", y = "Moyenne", title = "Moyenne des variables qualitatives par statut") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_fill_manual(values = c("#E46726", "#6D9EC1"))
```



Le rang de la page semble être la variable qualitative qui influe le plus. C'est la variable pour laquelle on voit la plus grande différence entre (en % de l'autre) la moyenne du groupe 1 et celle du 2.

```
mean_values_list_num <- list()

for (col in num_cols) {
  if (col != "web_traffic" && col != "domain_age" && col != "domain_registration_length") {
    mean_values_list_num[[col]] <- mean_by_status(df_present, col)
  }
}

mean_values_df_num <- do.call(rbind, mean_values_list_num)
mean_values_df_num$col_names <- rownames(mean_values_df_num)

mean_values_df_num <- mean_values_df_num[mean_values_df_num$mean_value > 0.3 | mean_values_df_num$mean_v
mean_values_df_num <- mean_values_df_num[!is.na(mean_values_df_num$mean_value), ]

ggplot(mean_values_df_num, aes(x = col_names, y = mean_value, fill = status)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Variables", y = "Moyenne", title = "Moyenne des variables quantitatives par statut") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_fill_manual(values = c("#E46726", "#6D9EC1"))
```



Le nombre d'hyperlien semble être la variable quantitative qui influe le plus.

2. K-NN

2.1. Préparation des données

On sépare les données en deux sous-ensembles : un pour la phase d'entraînement et l'autre pour la phase de test.

```
set.seed(123)
indxTrain <- createDataPartition(df$status, p = 0.75, list = FALSE)
DTrain <- df[indxTrain, ]
DTest <- df[-indxTrain, ]

cat("Nombre d'observations dans l'ensemble d'entraînement:", nrow(DTrain), "\n")
```

```
## Nombre d'observations dans l'ensemble d'entraînement: 8574
```

```
cat("Nombre d'observations dans l'ensemble de test:", nrow(DTest), "\n")
```

```
## Nombre d'observations dans l'ensemble de test: 2856
```

2.2. Prédictions

Pour commencer, nous choisissons arbitrairement un k = 5.

```
set.seed(123)
ctrl <- trainControl(method = "none")
fit.knn <- train(status ~ .,
```

```
                  data = DTrain,
                  method = "knn",
                  tuneGrid = data.frame(k = 5),
                  trControl = ctrl,
                  na.action = na.omit)
predictions <- predict(fit.knn, newdata = DTest)
```

2.3. Évaluation du modèle

Dans cette partie de l'analyse, nous évaluons la performance du modèle des k-plus proches voisins (K-NN) que nous avons entraîné pour la détection de sites de phishing. L'utilisation d'une matrice de confusion nous permet de comparer les prédictions du modèle par rapport aux valeurs réelles.

```
confusionMatrix <- confusionMatrix(predictions, DTest$status)

print(confusionMatrix$table)
```

```
##             Reference
## Prediction   legitimate phishing
##   legitimate       1164      218
##   phishing          264     1210
```

```
print(confusionMatrix$overall['Accuracy'])
```

```
##  Accuracy
## 0.8312325
```

```
errorRate <- 1 - confusionMatrix$overall['Accuracy']
print(errorRate)
```

```
##  Accuracy
## 0.1687675
```

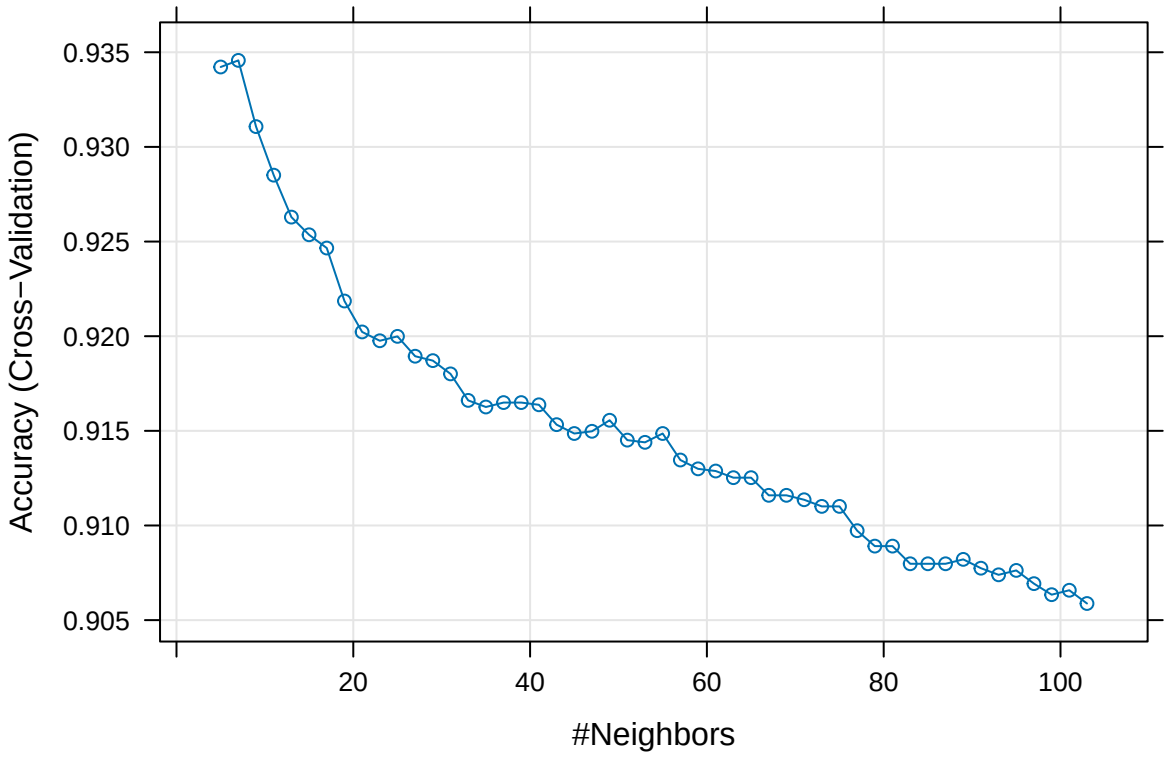
D'après les résultats obtenus, le modèle a une précision d'environ 84.17%, ce qui signifie qu'il a correctement prédit 84.17% des URL comme étant légitimes ou de phishing. La matrice montre également les répartitions spécifiques des vrais positifs, vrais négatifs, faux positifs et faux négatifs. Plus précisément, le modèle a correctement identifié 1164 URL légitimes (Vrais positifs) et 1210 URL de phishing (vrais négatifs), tandis qu'il a incorrectement classé 264 URL légitimes comme phishing (Faux négatifs) et 218 URL de phishing comme légitimes (faux positifs). Le taux d'erreur de 15.82% reflète la proportion de prédictions incorrectes par rapport au total des prédictions.

2.4. Choix du K : Cross-Validation

Pour choisir le nombre optimal de voisins k pour notre modèle, nous allons utiliser la validation-croisée sous 10 sous-ensembles. Cela implique de diviser l'ensemble de données d'entraînement en 10 parties, d'utiliser 9 d'entre elles pour l'entraînement et une pour la validation, et de répéter ce processus 10 fois avec des parties différentes à chaque fois pour la validation.

```
set.seed(123)

ctrl <- trainControl(method = "cv", number = 10)

fit.knn.cv <- train(status ~ .,
                    data = DTrain,
                    method = "knn",
                    trControl = ctrl,
                    tuneLength = 50,
```

```
                preProcess = c("center", "scale"),
                na.action = na.omit)

plot(fit.knn.cv)
```




```
print(fit.knn.cv$results)
```

```
##       k  Accuracy     Kappa  AccuracySD    KappaSD
## 1     5 0.9342209 0.8684417 0.008370824 0.01674103
## 2     7 0.9345713 0.8691428 0.009644583 0.01928876
## 3     9 0.9310721 0.8621439 0.009614671 0.01922987
## 4    11 0.9285066 0.8570132 0.009587755 0.01917508
## 5    13 0.9262907 0.8525812 0.009455714 0.01891267
## 6    15 0.9253577 0.8507149 0.010461351 0.02092441
## 7    17 0.9246574 0.8493140 0.006943111 0.01388807
## 8    19 0.9218587 0.8437166 0.008319502 0.01664135
## 9    21 0.9202248 0.8404490 0.009280188 0.01856158
## 10   23 0.9197580 0.8395150 0.011876311 0.02375448
## 11   25 0.9199909 0.8399806 0.013048270 0.02609886
## 12   27 0.9189415 0.8378820 0.013422811 0.02684721
## 13   29 0.9187085 0.8374159 0.012065937 0.02413337
## 14   31 0.9180091 0.8360175 0.011731717 0.02346416
## 15   33 0.9166093 0.8332179 0.011395919 0.02279199
## 16   35 0.9162598 0.8325187 0.011131604 0.02226274
## 17   37 0.9164925 0.8329837 0.011607167 0.02321474
## 18   39 0.9164925 0.8329835 0.012090737 0.02418229
```

```
## 19  41 0.9163756 0.8327499 0.011811625 0.02362469
## 20  43 0.9153260 0.8306509 0.010977518 0.02195528
## 21  45 0.9148594 0.8297179 0.010752665 0.02150622
## 22  47 0.9149762 0.8299515 0.011479436 0.02296041
## 23  49 0.9155595 0.8311177 0.011592110 0.02318564
## 24  51 0.9145096 0.8290182 0.010907760 0.02181670
## 25  53 0.9143931 0.8287849 0.011316431 0.02263483
## 26  55 0.9148593 0.8297176 0.012215760 0.02443348
## 27  57 0.9134593 0.8269175 0.012154709 0.02431130
## 28  59 0.9129923 0.8259834 0.012385104 0.02477271
## 29  61 0.9128761 0.8257509 0.012598073 0.02519819
## 30  63 0.9125259 0.8250508 0.013447659 0.02689756
## 31  65 0.9125263 0.8250514 0.013490180 0.02698387
## 32  67 0.9115931 0.8231852 0.013689740 0.02738362
## 33  69 0.9115930 0.8231847 0.013877721 0.02776055
## 34  71 0.9113595 0.8227178 0.013268686 0.02654256
## 35  73 0.9110094 0.8220177 0.013270191 0.02654544
## 36  75 0.9110094 0.8220178 0.012771729 0.02554805
## 37  77 0.9097260 0.8194511 0.012477456 0.02495929
## 38  79 0.9089096 0.8178184 0.012780004 0.02556443
## 39  81 0.9089096 0.8178186 0.011973699 0.02395142
## 40  83 0.9079767 0.8159527 0.012955424 0.02591492
## 41  85 0.9079765 0.8159523 0.011795633 0.02359590
## 42  87 0.9079767 0.8159529 0.011148220 0.02229980
## 43  89 0.9082096 0.8164188 0.011146570 0.02229655
## 44  91 0.9077434 0.8154862 0.011997763 0.02399831
## 45  93 0.9073935 0.8147865 0.011373490 0.02274928
## 46  95 0.9076268 0.8152532 0.012265069 0.02453303
## 47  97 0.9069272 0.8138543 0.011708682 0.02342061
## 48  99 0.9063448 0.8126896 0.011442765 0.02288853
## 49 101 0.9065779 0.8131556 0.011723113 0.02344992
## 50 103 0.9058781 0.8117558 0.011948427 0.02390042
```

```r
print(fit.knn.cv$bestTune)
```

```
##   k
## 2 7
```

```r
bestK <- fit.knn.cv$bestTune$k
predictionsBestK <- predict(fit.knn.cv, newdata = DTest)
confusionMatrixBestK <- confusionMatrix(predictionsBestK, DTest$status)
errorRateBestK <- 1 - confusionMatrixBestK$overall['Accuracy']
print(errorRateBestK)
```

```
##   Accuracy
## 0.06127451
```

Le graphique généré illustre comment la précision de la validation croisée varie en fonction du nombre de voisins k. D'après la visualisation, il semble que la précision augmente lorsque le nombre de voisins est faible et diminue après avoir atteint un pic. Cela suggère qu'un nombre plus réduit de voisins aide le modèle à mieux capturer les nuances des données sans tomber dans le surajustement, où le modèle est trop spécifique aux données d'entraînement et ne généralise pas bien aux nouvelles données.

En analysant les données, j'observe que la précision la plus élevée est obtenue avec k=5. Cela suggère que le modèle classifie les données avec le plus de précision lorsqu'il considère les 5 voisins les plus proches. Ensuite, bien que la précision diminue légèrement avec k=7, elle reste relativement élevée pour k allant jusqu'à 13,

après quoi la précision commence à diminuer de manière plus significative. Cela peut indiquer que des valeurs de k plus faibles sont préférables pour ce dataset spécifique, mais qu'il existe une marge avant que l'augmentation de k n'entraîne un sous-ajustement notable.

Il est également important de noter les écarts-types des précisions (AccuracySD) et des scores Kappa (KappaSD), qui fournissent une indication de la variabilité de la performance du modèle à travers les différentes itérations de la validation croisée. Des écarts-types plus faibles sont préférables car ils impliquent une performance plus constante du modèle.

3. Régression logistique

Avec les mêmes partitions utilisées pour les KNN, on effectue une régression logistique.

```
ctrl <- trainControl(method = "none")
fit.lr <- train(status ~ .,
                data = DTrain,
                method = "glm",
                trControl = ctrl,
                na.action = na.omit)

class.lr <- predict(fit.lr, newdata = DTest)
(confusionMatrix(class.lr, DTest$status))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   legitimate phishing
##   legitimate       1351       76
##   phishing           77     1352
##
##                Accuracy : 0.9464
##                  95% CI : (0.9375, 0.9544)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8929
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9461
##             Specificity : 0.9468
##          Pos Pred Value : 0.9467
##          Neg Pred Value : 0.9461
##              Prevalence : 0.5000
##          Detection Rate : 0.4730
##    Detection Prevalence : 0.4996
##       Balanced Accuracy : 0.9464
##
##        'Positive' Class : legitimate
##
```

3.1. Importance et sélection des variables

A l'aide d'un test de Student, on regarde quels sont les variables les plus importantes pour notre analyse. Plus la p.value est petite, plus elle est significative et plus sons "Overall" sera haut.

```
print(varImp(fit.lr))
```

```
## glm variable importance
##
##   only 20 most important variables shown (out of 81)
##
##                              Overall
## google_index                  100.00
## page_rank                      81.32
## nb_www                         56.29
## phish_hints                    52.56
## domain_age                     34.17
## nb_hyperlinks                  32.68
## shortening_service             32.05
## longest_words_raw              29.62
## ratio_digits_host              26.20
## length_hostname                25.31
## nb_hyphens                     24.42
## domain_in_title                23.75
## nb_underscore                  23.70
## https_token                    22.65
## domain_registration_length     22.34
## ratio_extHyperlinks            21.37
## ratio_extRedirection           20.95
## ratio_extMedia                 19.75
## avg_words_raw                  19.56
## nb_space                       19.21
```

La variable la plus importante est `google_index`. On voit que l'importance décroit très vite alors qu'on est seulement sur 20 de nos 87 variables explicatives. Faire une sélection des variables selon le critère de l'AIC peut être pertinent.

```
ctrl <- trainControl(method = "none")
#fit.lr.aic <- train(status ~ ., data = DTrain, method = "glmStepAIC", trControl = ctrl, na.action = na
load("fit.lr.aic.RDATA")
```

3.2. Prédiction

```
score.lr.aic <- predict(fit.lr.aic, newdata = DTest, type = "prob")
#Distribution des classes prédites
#table(score.lr.aic)
```

3.3. Scoring

Pour évaluer la performance de notre modèle, on utilisera analysera d'abord la matrice de confusion. Ensuite, on calculera l'aire sous la courbe ROC (AUC) qui correspond à la probabilité que le modèle classe un exemple positif au hasard plus haut qu'un exemple négatif au hasard. Plus l'AUC est proche de 1, plus le modèle est performant.

3.3.1. Matrice de confusion

```
class.lr.aic <- predict(fit.lr.aic, newdata = DTest)
(confusionMatrix(class.lr.aic, DTest$status))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   legitimate phishing
##   legitimate       1349       76
##   phishing           79     1352
##
##                Accuracy : 0.9457
##                  95% CI : (0.9368, 0.9538)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8915
##
##  Mcnemar's Test P-Value : 0.8724
##
##             Sensitivity : 0.9447
##             Specificity : 0.9468
##          Pos Pred Value : 0.9467
##          Neg Pred Value : 0.9448
##              Prevalence : 0.5000
##          Detection Rate : 0.4723
##    Detection Prevalence : 0.4989
##       Balanced Accuracy : 0.9457
##
##        'Positive' Class : legitimate
##
```

Le taux de vrais positifs (TVR, sensibilité) est de :

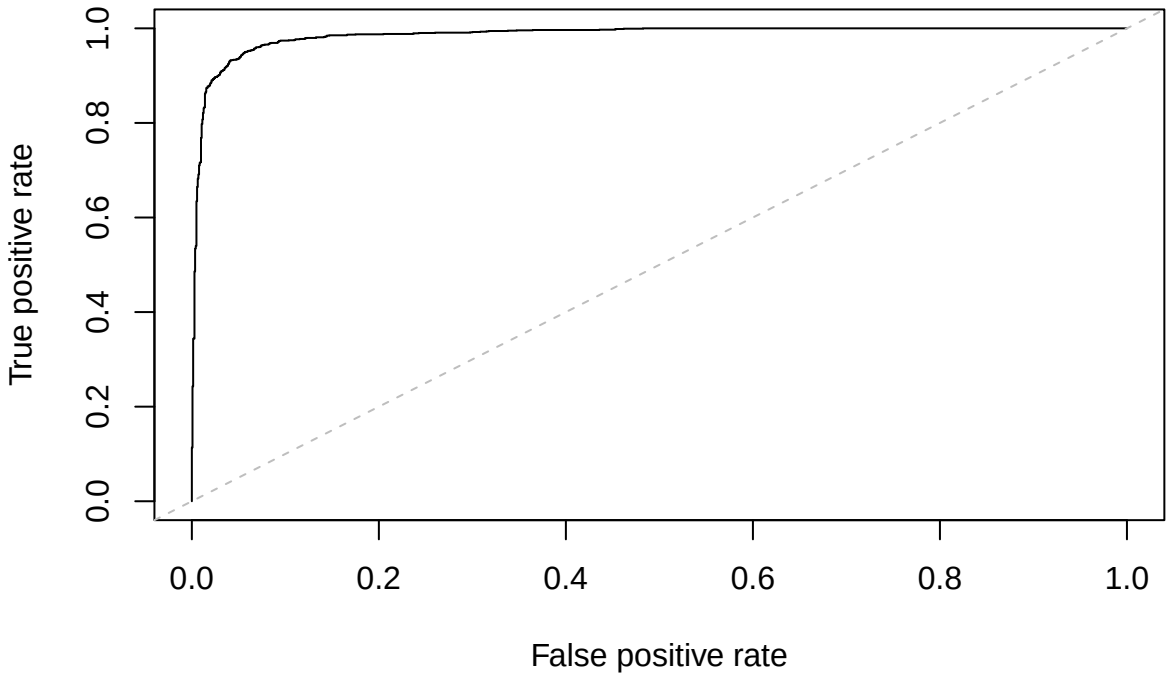
$$\text{TVR} = \text{VP} / (\text{VP} + \text{FN}) = 1352 / (1352 + 79) = 0.944$$

Le taux de vrais négatifs (TVN, spécificité) est de :

$$\text{TVN} = \text{VN} / (\text{VN} + \text{FP}) = 1349 / (1349 + 76) = 0.947$$

3.3.2. Courbe ROC

```r
pred <- prediction(score.lr.aic[,2], DTest$status)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = FALSE)
abline(a = 0, b = 1, lty = 2, col = "gray")
```



3.4. Avec cross-validation

```
ctrl = trainControl(method = "cv", classProbs = TRUE,
                    summaryFunction = twoClassSummary,
                    savePredictions = "all")

fit.lr.cv <- train(status ~ .,
                   data = DTrain,
                   method = "glm",
                   trControl = ctrl,
                   na.action = na.omit)

print(fit.lr.cv)
```

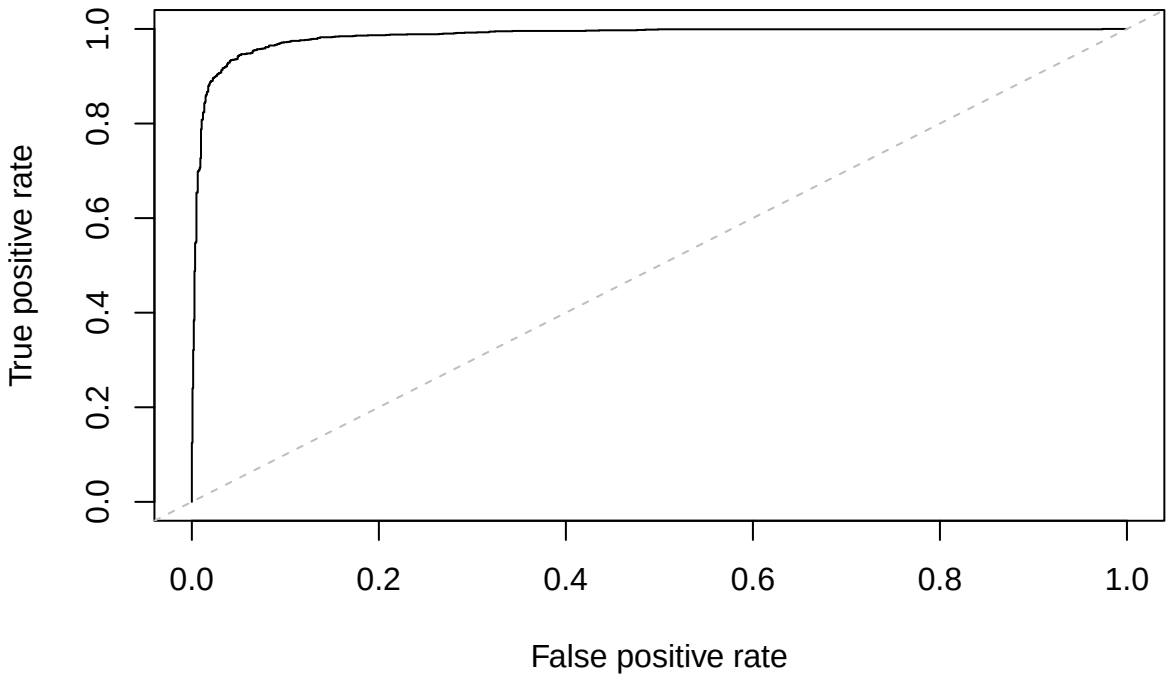
```
## Generalized Linear Model
##
## 8574 samples
##   87 predictor
##    2 classes: 'legitimate', 'phishing'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 7718, 7718, 7716, 7716, 7717, 7716, ...
## Resampling results:
##
##   ROC        Sens       Spec
```

```
##   0.9847499  0.9498508  0.9402844
```

```
scoreCV.lr <- fit.lr.cv$pred
print(head(scoreCV.lr))
```

```
##         pred        obs   legitimate   phishing rowIndex parameter Resample
## 1   phishing legitimate 3.828796e-01 0.61712040        1      none   Fold01
## 2 legitimate legitimate 8.897632e-01 0.11023683       16      none   Fold01
## 3   phishing   phishing 1.191998e-09 1.00000000       24      none   Fold01
## 4 legitimate   phishing 6.967122e-01 0.30328781       31      none   Fold01
## 5   phishing   phishing 1.488138e-05 0.99998512       33      none   Fold01
## 6 legitimate legitimate 9.342777e-01 0.06572226       61      none   Fold01
```

```
predictions <- predict(fit.lr.cv, newdata = DTest, type = "prob")
actuals <- DTest$status
pred <- prediction(predictions[,2], actuals)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = FALSE)
abline(a = 0, b = 1, lty = 2, col = "gray")
```

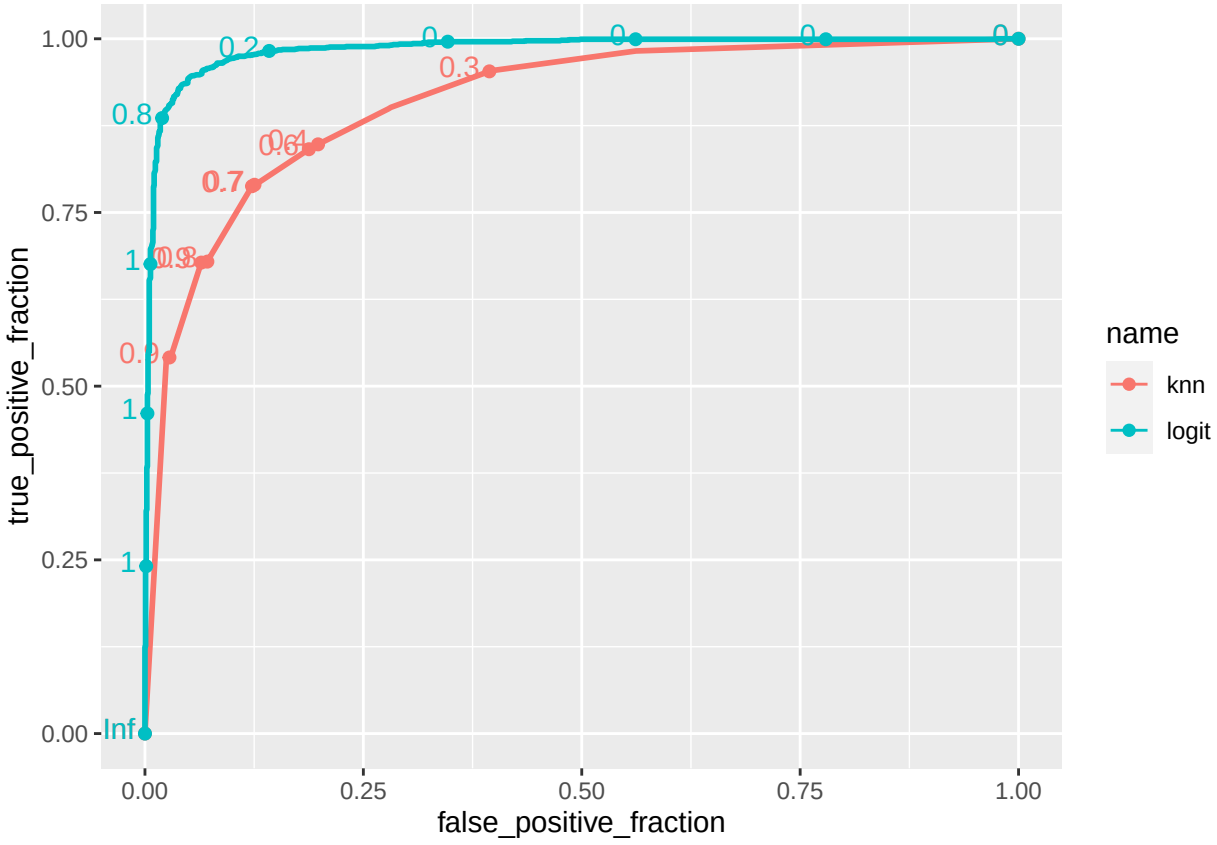


4. Comparaison des modèles

```
ctrl <- trainControl(method = "none")
fit.knn <- train(status ~ .,
                 data = DTrain,
                 method = "knn",
```

```
                trControl = ctrl,
                tuneGrid = data.frame(k = 7),
                na.action = na.omit)

fit.lr = train(status ~ .,
               data = DTrain,
               method = "glm",
               trControl = ctrl,
               na.action = na.omit)

score.knn = predict(fit.knn ,newdata = DTest, type="prob")
score.lr = predict(fit.lr ,newdata = DTest, type="prob")
```

```
score.data = cbind(DTest$status,score.knn["phishing"],score.lr["phishing"])
colnames(score.data) = c("type.test","knn","logit")
score.data <- melt_roc(score.data,"type.test",c("knn","logit"))
g=ggplot(score.data, aes(m = M,d = D,color = name)) + geom_roc()
g
```




```
print(calc_auc(g)$AUC)
```

```
## [1] 0.9086890 0.9852792
```