Introduction à la reconnaissance de contexte

- Auteur : Sébastien Mosser
- Version : 06.2020

Cette étude de cas est fortement inspiré des chapitres 3, 5 et 7 du livre *Machine Learning with R* de Brent Lenz (éditions PACKT Publishing). Elle a bénéficié des conseils des professeur • e • s Marie-Jean Meurs (informatique) et Arthur Charpentier (mathématiques), qui donnent le cours d'Initiation à la science des données et à l'intelligence artificielle (INF7100).

Analyse du jeux de données

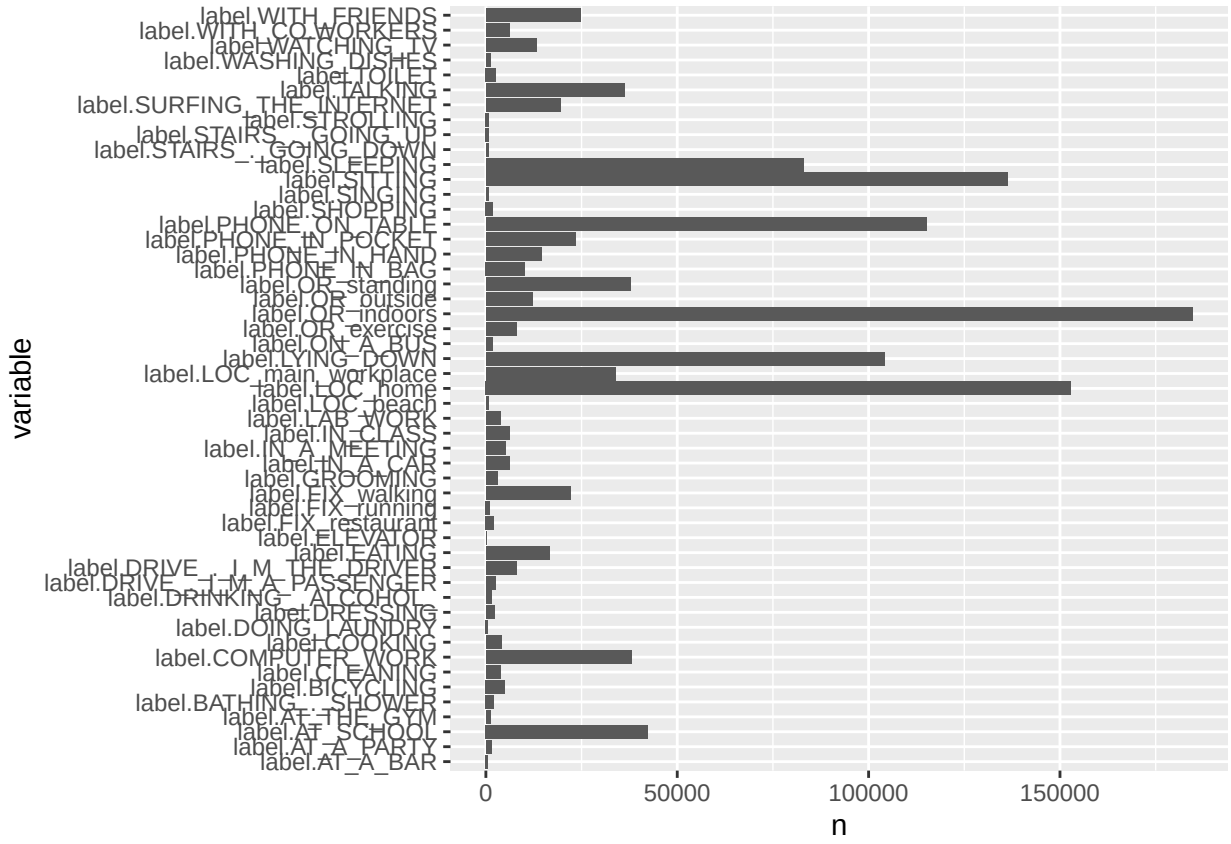
```
dataset <- load_complete_dataset()
```

- Le jeux de données `dataset` contient 377,346 observations sur 278 variables.

```
labels <- dataset %>% select(starts_with("label:"))
```

- Parmi ces variables, 51 sont des étiquettes de reconnaissance de contexte

```
plot_labels_count <- function (data) {
  transposed <- rownames_to_column(data.frame(t(data)))
  names(transposed) <- c("variable", "n")
  ggplot(data=transposed, mapping=aes(x=variable, y=n)) +
    geom_bar(stat = "identity") +
    coord_flip()
}
plot_labels_count(data.frame(as.list(colSums(labels, na.rm = TRUE))))
```

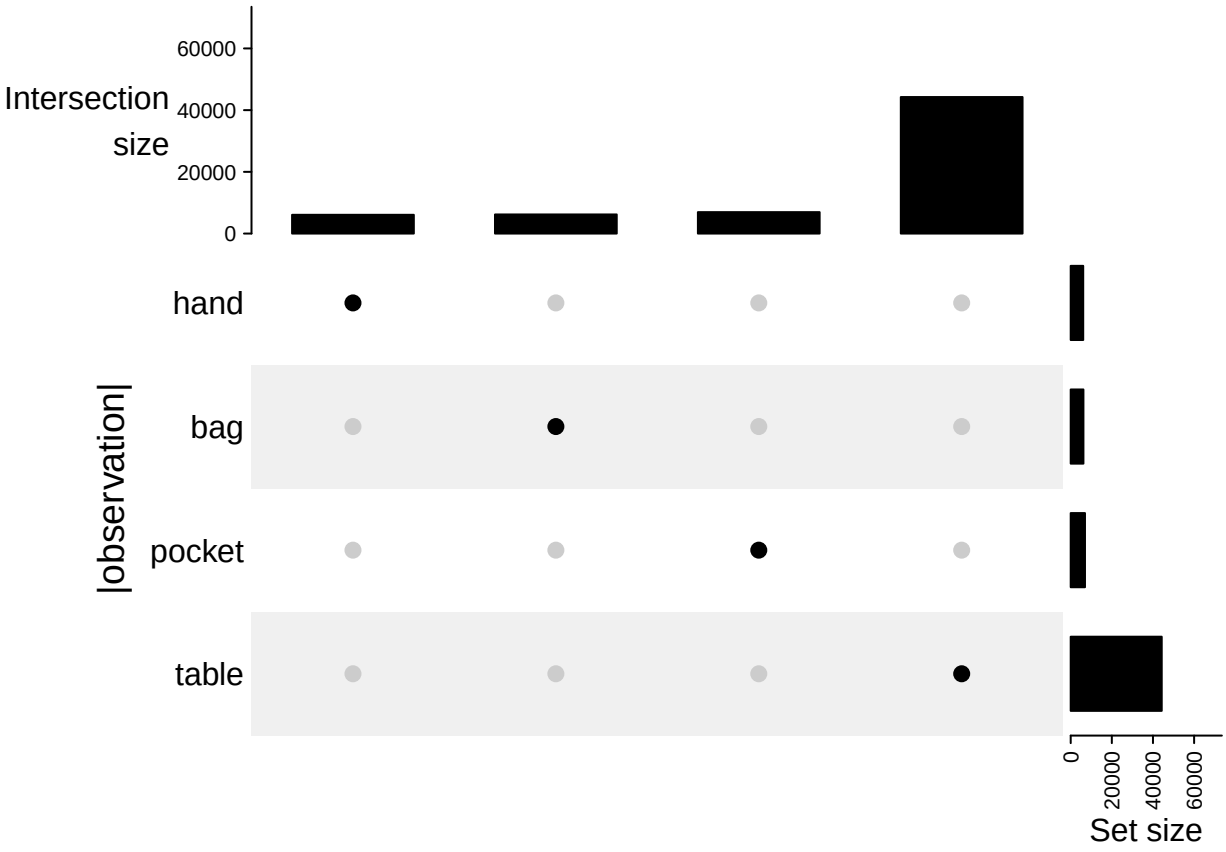


Détection du Contexte d'utilisation du téléphone

```
phone_labels <- labels %>% select(starts_with("label:PHONE_"))
phone_labels <- phone_labels[complete.cases(phone_labels),]
```

On s'interesse ici uniquement aux observations de la localisation du téléphone de l'utilisateur. On dispose de 63,449 observations. Dans un premier temps, on vérifie que toutes les observations sont bien disjointes.

```
phone_labels_cases <- list(
  "bag"    = rownames(phone_labels[phone_labels$'label:PHONE_IN_BAG' == 1,]),
  "hand"   = rownames(phone_labels[phone_labels$'label:PHONE_IN_HAND' == 1,]),
  "pocket" = rownames(phone_labels[phone_labels$'label:PHONE_IN_POCKET' == 1,]),
  "table"  = rownames(phone_labels[phone_labels$'label:PHONE_ON_TABLE' == 1,])
)
phone_labels_matrix = make_comb_mat(phone_labels_cases)
```



Fabrication du jeu de données initial

Récupération des données d'intêret

On va maintenant fabriquer le jeu de données qui va nous permettre de faire l'experience de classification. On va réduire le jeu de données initial aux valeurs pour l'accéléromètre, le gyroscope et le magnétomètre.

```
phone_observations <- dataset %>% select(starts_with("raw_acc"),
                                         starts_with("proc_gyro"),
                                         starts_with("raw_magnet"),
                                         starts_with("label:PHONE_"))
phone_observations <- phone_observations[complete.cases(phone_observations),]
```

En restreignant uniquement aux cas où nous disposons de toutes les valeurs pour chaque observations, cela donne 59882 observations.

Plutot que 4 colonnes indiquant 0 ou 1 pour chaque classe, les algorithmes d'apprentissage attendent une colonne contenant directement la classe qui nous interesse (sous la forme d'un *facteur*).

```
phone_observations$STATUS <- "?"
phone_observations[phone_observations$`label:PHONE_IN_BAG` == 1,]$STATUS    <- "B"
phone_observations[phone_observations$`label:PHONE_IN_POCKET` == 1,]$STATUS <- "P"
phone_observations[phone_observations$`label:PHONE_IN_HAND` == 1,]$STATUS   <- "H"
phone_observations[phone_observations$`label:PHONE_ON_TABLE` == 1,]$STATUS  <- "T"
phone_observations$STATUS <- factor(phone_observations$STATUS,
                                    levels = c("B", "P", "H", "T"),
```

```
                                    labels = c("bag", "pocket", "hand", "table"))
phone_observations <- phone_observations %>% select(-starts_with("label:PHONE_"))
```

Normalisation des données

Les donnés de chaque capteurs sont des des intervales très différents les uns des autres.

```
summary(phone_observations[c("raw_acc:3d:mean_x", "raw_acc:3d:mean_y", "raw_acc:3d:mean_z")])
```

```
##  raw_acc:3d:mean_x   raw_acc:3d:mean_y   raw_acc:3d:mean_z
##  Min.   :-1.134850   Min.   :-1.022699   Min.   :-1.08164
##  1st Qu.:-0.020165   1st Qu.:-0.028909   1st Qu.:-0.99016
##  Median : 0.012589   Median : 0.004741   Median :-0.11415
##  Mean   : 0.005651   Mean   : 0.028688   Mean   :-0.05196
##  3rd Qu.: 0.064781   3rd Qu.: 0.082794   3rd Qu.: 0.96098
##  Max.   : 1.289129   Max.   : 1.290554   Max.   : 1.05319
```

On va donc normaliser les données, pour ramener tous les calculs sur des intervalles dans `[0,1]`. On utilise une formule classique de normalisation :

```
norm_variable <- function (v) { return ((v - min(v)) / (max(v) - min(v))) }
```

Pour fabriquer un jeux de donnée normalisée, il suffit d'appliquer cette fonction de normalisation à toutes les variables, sauf la dernière (qui est le contexte a reconnaitre).

```
normalize <- function(dataset) {
  ds_n <- as.data.frame(lapply(dataset[1:ncol(dataset)-1],
                               norm_variable))
  ds_n$status <- dataset$STATUS
  return(ds_n)
}
```

On peut maintenant fabriquer le jeux de donnée normalisée :

```
with(obs_n <- normalize(phone_observations),
     summary(obs_n[c("raw_acc.3d.mean_x", "raw_acc.3d.mean_y", "raw_acc.3d.mean_z")]))
```

```
##  raw_acc.3d.mean_x raw_acc.3d.mean_y raw_acc.3d.mean_z
##  Min.   :0.0000    Min.   :0.0000    Min.   :0.00000
##  1st Qu.:0.4599    1st Qu.:0.4296    1st Qu.:0.04285
##  Median :0.4734    Median :0.4442    Median :0.45319
##  Mean   :0.4705    Mean   :0.4545    Mean   :0.48233
##  3rd Qu.:0.4949    3rd Qu.:0.4779    3rd Qu.:0.95680
##  Max.   :1.0000    Max.   :1.0000    Max.   :1.00000
```

Fabrication des jeux de données d'entrainement et de test

On va entrainer notre classifieur sur 80% des données disponibles. Et on utilisera les 20% restantes pour vérifier les résultats et mesurer à quel point notre classifieur est pertinent.

```
prepare_datasets <- function(complete, random.seed=42) {
  set.seed(random.seed) # Pour rendre les experiences reproductibles
  training <- complete %>% sample_n(size = 0.8*nrow(complete))
  test     <- setdiff(complete, training)
  list(train = training, test = test)
}
```

Fabrication d'une première prédiction et évaluation des résultats

```
library(class)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
run_knn <- function(training, test, k.value=42) {
  # Preparing the datasets for training
  training_data <- training %>% select(-status)
  training_classes <- training %>% select(status)
  # Same, for testing
  test_data <- test %>% select(-status)
  test_classes <- test %>% select(status)
  # Running the classifier
  predictions <- knn(train = training_data,
                     test  = test_data,
                     cl    = training_classes[,1],
                     k     = k.value)
  # Building the confusion matrix
  cmat <- confusionMatrix(reference = test_classes[,1],
                          data = predictions)
  return(cmat)
}
```

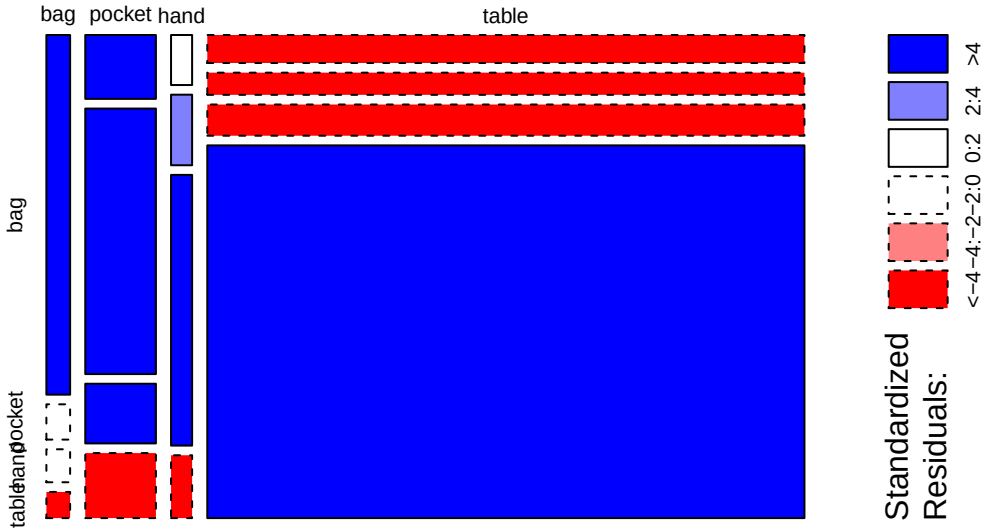
Application des *n-plus proches voisins* à notre jeux de données

```
cmat <- with(ds <- prepare_datasets(normalize(phone_observations)),
             run_knn(ds$train, ds$test))
```

Conclusion : En appliquant naivement la méthode des plus proches voisins à notre jeux de données normalisé, on obtient une *accuracy* de 78.98% ! **C'est vraiment bien pratique l'apprentissage automatique !**

Regardons un peu plus en détails les résultats

```
library(graphics)
plot_confusion_matrix <- function(mat, title) {
  mosaicplot(mat,
             xlab = "", ylab = "",
             main = title,
             shade = TRUE)
}
  plot_confusion_matrix(cmat$table, "kNN, version 1")
```



On va regarder un peu plus dans le détails pour chaque classe ce qu'il en est, en s'interessant à la précision, au rappel, et à la F-mesure.

- Précision : nombre de contexte reconnus rapporté sur le nombre total de contextes;
- Rappel : nombre de contextes reconnus qui sont pertinents;
- F-Mesure : Moyenne harmonique de la prévision et du rappel.

```
print_stats <- function(mat) {
  values <- data.frame(mat$byClass)
  kable(values %>%
          select(contains("Accuracy"), Precision, Recall, F1),
        digits = 2)
}
print_stats(cmat)
```

	Balanced.Accuracy	Precision	Recall	
Class: bag	0.63	0.79	0.28	
Class: pocket	0.75	0.58	0.55	
Class: hand	0.59	0.60	0.19	
Class: table	0.73	0.82	0.97	
Un predicteur id	éal aurait une préci	sion (_preci	sion_) et	un rappel (*recall*) valant 1 (et donc idem pour sa F-mes

Dans notre cas, on a une forte *accuracy* (~80%), mais une précision complètement déraisonnable. On voit aussi une sur-représentation des contextes `table`.

Effet de la sur-représentation d'une classe

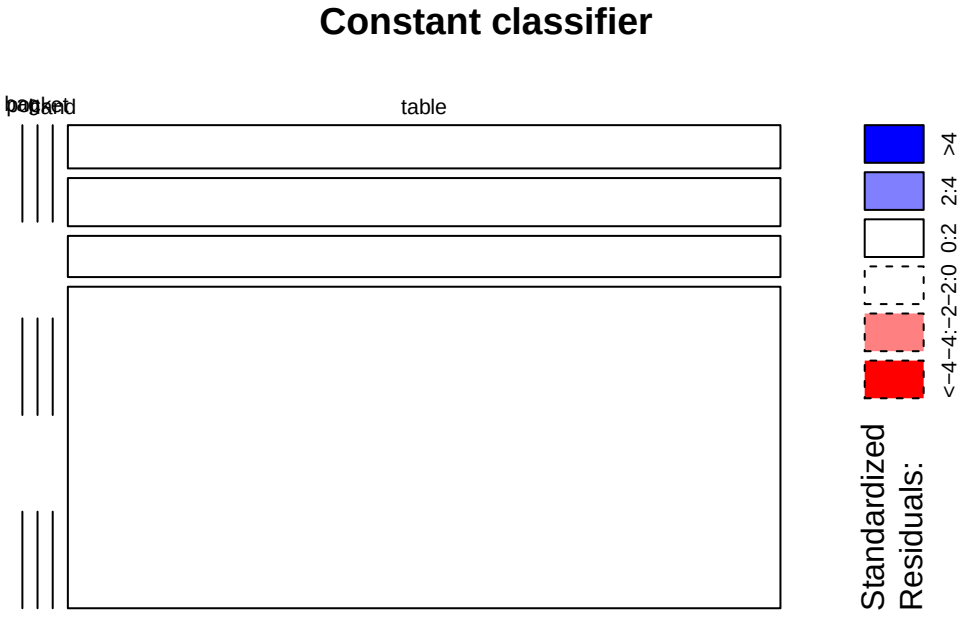
On va fabriquer un prédicteur encore plus naif : il répond toujours `table`.

```
constant_table <- function(training, test) {
  test_classes <- test %>% select(status)
  # On répond toujours table ... facile !
  predictions <- as.factor(rep('table',nrow(test)))
  cmat <- confusionMatrix(reference = test_classes[,1],
```

```
                          data = predictions)
  return(cmat)
}
cmat_constant <- with(ds <- prepare_datasets(normalize(phone_observations)),
                      constant_table(ds$train, ds$test))
```

On obtient avec cette prédiction une *accuracy* de 70.80% !! Il est important de garder à l'esprit que, prise seule, l'*accuracy* ne veut pas dire grand chose.

```
plot_confusion_matrix(cmat_constant$table, "Constant classifier")
```




```
print_stats(cmat_constant)
```

	Balanced.Accuracy	Precision	Recall	F1
Class: bag	0.5	NA	0	NA
Class: pocket	0.5	NA	0	NA
Class: hand	0.5	NA	0	NA
Class: table	0.5	0.71	1	0.83

Nettoyage des données disponibles

Équilibrage du jeux de données

On commence par regarder dans quel état est notre sur-représentation de la class `table`.

```
summary(phone_observations$STATUS)
```

```
##     bag pocket   hand  table
##    5668   6533   5465  42216
```

On a environ 6, 000 enregistrements pour les trois autres classes, mais 42, 000 pour la classe `table`.

On va fabriquer un jeu de données équilibré en retenant uniquement 6, 000 observations pour la classe `table`.

```r
balance <- function(dataset, random.seed=42) {
  set.seed(random.seed)
  res <- dataset[dataset$STATUS == 'bag',]
  res <- rbind(res, dataset[dataset$STATUS == 'pocket',])
  res <- rbind(res, dataset[dataset$STATUS == 'hand',])
  res <- rbind(res, sample_n(dataset[dataset$STATUS == 'table',], 6000))
  return(res)
}
summary(balance(phone_observations)$STATUS)
```

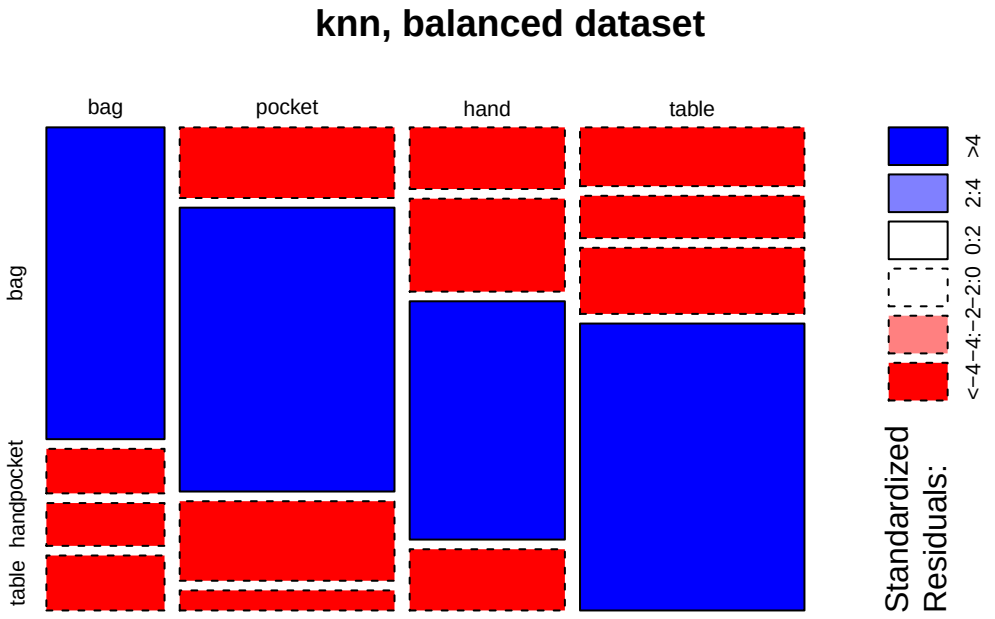
```
##    bag pocket   hand  table
##   5668   6533   5465   6000
```

Fabrication d'une prédiction sur le modèle équilibré

```r
cmat_balanced <-
  with(ds <- prepare_datasets(normalize(balance(phone_observations))),
                       run_knn(ds$train, ds$test))
```

On obtient avec cette prédiction une *accuracy* de 61.55%, C'est moins bien qu'avant ! (vraiment ?)

```r
plot_confusion_matrix(cmat_balanced$table, "knn, balanced dataset")
```




```r
print_stats(cmat_balanced)
```

	Balanced.Accuracy	Precision	Recall	F1
Class: bag	0.71	0.69	0.49	0.57
Class: pocket	0.76	0.63	0.68	0.65
Class: hand	0.68	0.52	0.50	0.51
Class: table	0.80	0.63	0.76	0.69

Les résultats ne sont pas fantastique, mais par contre on commence a faire remonter le rappel,et a avoir des

detection a peu près équivalente en fonction des differentes classes.

Une autre approche de la normalisation (Z-score)

La normalisation dans [0,1] avec notre formule "naive" a pour effet de bord d'écraser des valeurs extrêmes, ce qui les a rapprochées trop naivement de leurs voisines. On peut utiliser une méthode alternative (*z-score*), qui n'a ni minimum ni maximum prédéfinis.

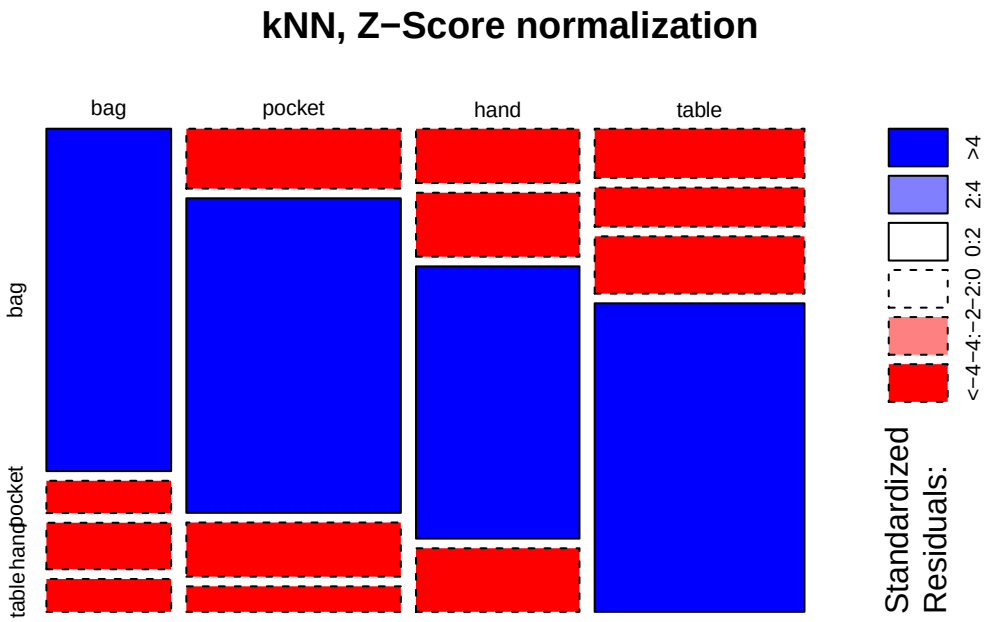
```
z_normalize <- function(dataset) {
  ds_n <- as.data.frame(scale(dataset[1:ncol(dataset)-1]))
  ds_n$status <- dataset$STATUS
  return(ds_n)
}
```

On peut maintenant relancer une prédiction en utilisant cette normalisation plutôt que notre version naive initiale.

```
cmat_z <-
  with(ds <- prepare_datasets(z_normalize(balance(phone_observations))),
                       run_knn(ds$train, ds$test))
```

On obtient avec cette prédiction une *accuracy* de 67.81%. On remonte. Mais est-ce vraiment mieux ?

```
plot_confusion_matrix(cmat_z$table, "kNN, Z-Score normalization")
```




```
print_stats(cmat_z)
```

	Balanced.Accuracy	Precision	Recall	F1
Class: bag	0.76	0.75	0.57	0.65
Class: pocket	0.81	0.69	0.75	0.72
Class: hand	0.74	0.60	0.60	0.60
Class: table	0.82	0.68	0.76	0.72

On s'améliore, la précision et le rappel (et donc la F-mesure) augmentent.

Taille de l'espace des données

Un des principes de l'apprentissage machine est d'apprendre sur des données d'entraînement, et on ne s'est jamais vraiment intéressé aux données que l'on manipule jusqu'à présent !

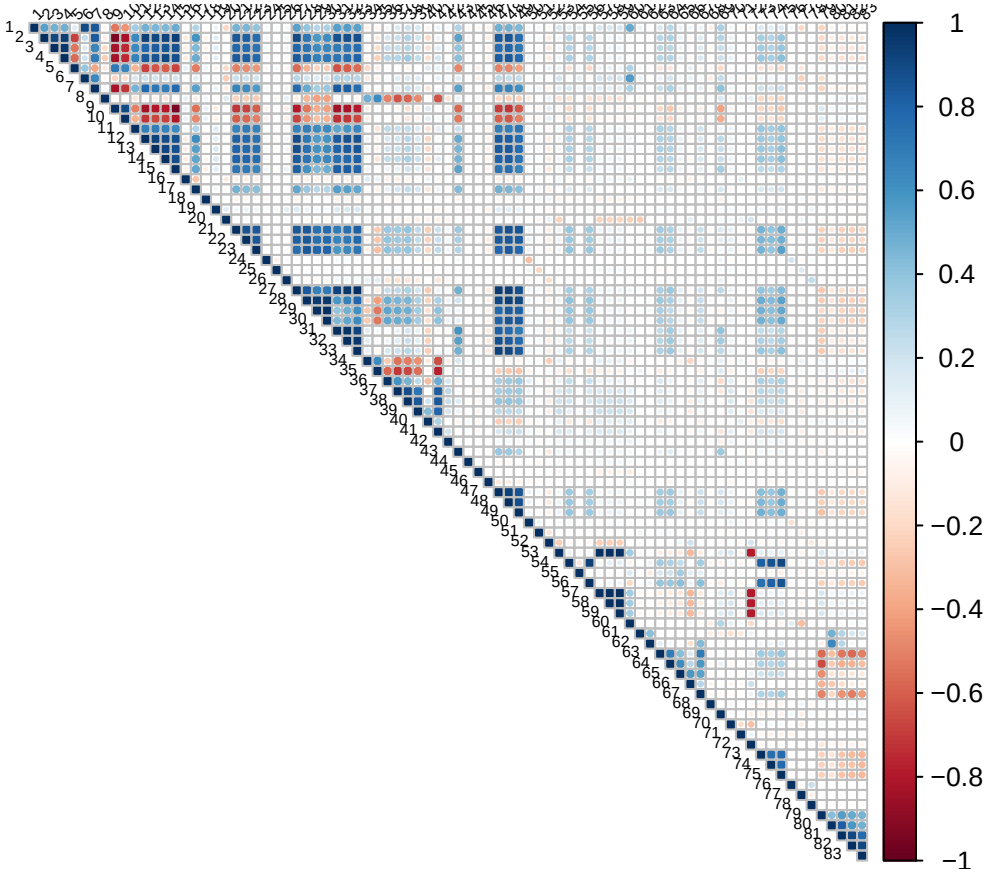
Notre jeux de données équilibré contient 23666, pour `ncol(phone_observations)-1` variables. Vu la taille de l'espace, la méthode des k-plus proches voisins pose problème : l'espace est tellement grand qu'il est très facile d'être le voisin de quelqu'un !

État des lieux de la corrélation entre variables

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

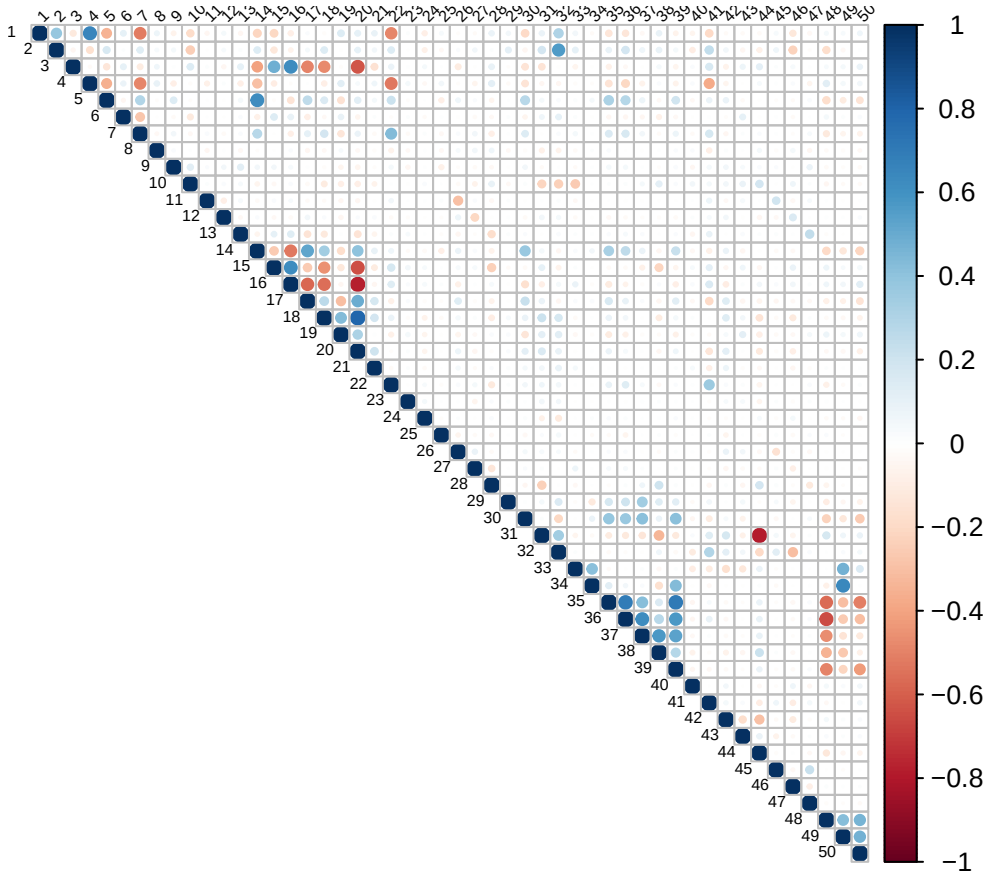
```
print_corplot <- function(dataset) {
  corr_obs <- dataset %>% select(-STATUS)
  colnames(corr_obs) <- 1:ncol(corr_obs)
  corr_matrix <- cor(corr_obs)
  corrplot(corr_matrix, type = "upper",
           tl.col = "black", tl.srt = 45, tl.cex=0.5)
}
print_corplot(phone_observations)
```



En affichant cette matrice de corrélation, on se rend compte que beaucoup de variables sont corélées entre elles.

On va maintenant netoyer nos observations, pour garder uniquement les variables avec un seuil de corélation inférieur à 80%.

```
slice_relevant <- function(dataset, threshold = 0.8) {
  corr_obs <- dataset %>% select(-STATUS)
  corr_matrix <- cor(corr_obs)
  high <- findCorrelation(corr_matrix, cutoff = threshold)
  res <- phone_observations[,-c(high)]
  res$STATUS <- phone_observations$STATUS
  return(res)
}
print_corplot(slice_relevant(phone_observations))
```



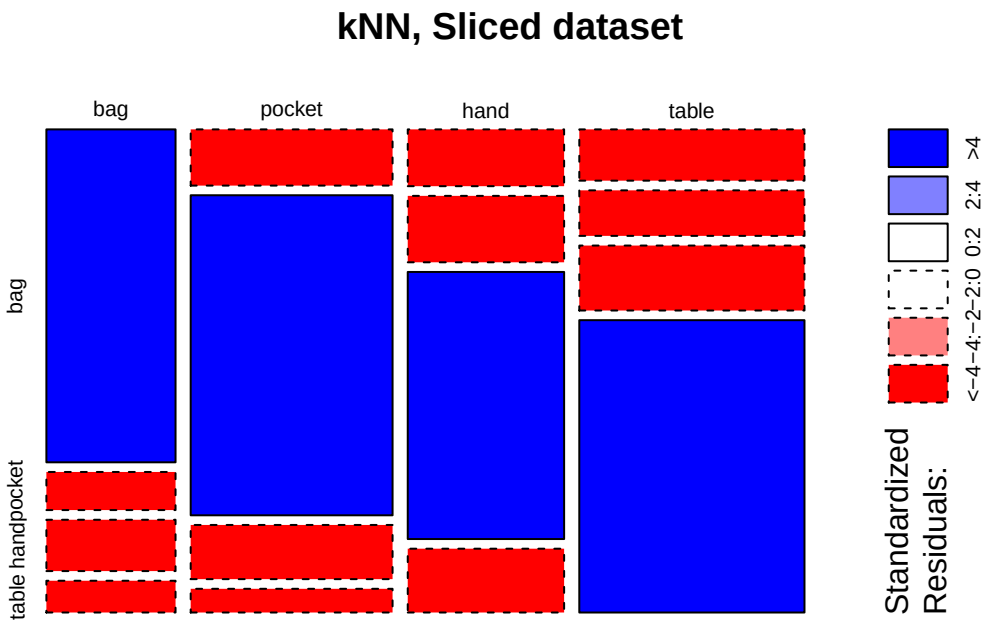
Avec cette approche, on réduit l'espace des données de 33 dimensions. Cela va accélerer drastiquement le temps d'entrainement de notre prédicteur.

Prédiction sur les données épurées

```
cmat_sliced <-
  with(ds <- prepare_datasets(z_normalize(balance(slice_relevant(phone_observations)))),
                       run_knn(ds$train, ds$test))
```

On obtient avec cette prédiction une *accuracy* de 66.50%. On est équivalent a la prédiction précédente, mais on va *beaucoup* plus vite pour l'entrainement. Qu'en est-il des autres dimensions ?

```
plot_confusion_matrix(cmat_sliced$table, "kNN, Sliced dataset")
```




```
print_stats(cmat_sliced)
```

	Balanced.Accuracy	Precision	Recall	F1
Class: bag	0.76	0.73	0.57	0.64
Class: pocket	0.80	0.70	0.72	0.71
Class: hand	0.72	0.59	0.56	0.58
Class: table	0.81	0.64	0.78	0.70
Ceci étant dit,	on commence a sentir	les limites	de la na	iveté de notre approche ...

Utilisation d'un arbre de décision

La méthode des plus proche voisins est simple, mais a pour inconvénient d'être très fragile aux variables bruitées et à la taille des jeux de données. De plus, elle demande enormément de puissance de calcul.

On va s'interesser ici à la mise en place d'un arbre de décision, en utilisant l'algorithme `C5.0`.

```
library(C50)

run_dt <- function(training, test, nb.trials=1) {
  training_data <- training %>% select(-status)
  training_classes <- training %>% select(status)
  test_data <- test %>% select(-status)
  test_classes <- test %>% select(status)
  model <- C5.0(training_data,
                training_classes[,1],
                trials = nb.trials)
  predictions <- predict(model, test_data, type = "class")
  cmat <- confusionMatrix(reference = test_classes[,1],
                          data = predictions)
  return(list(cmat=cmat, tree=model))
```

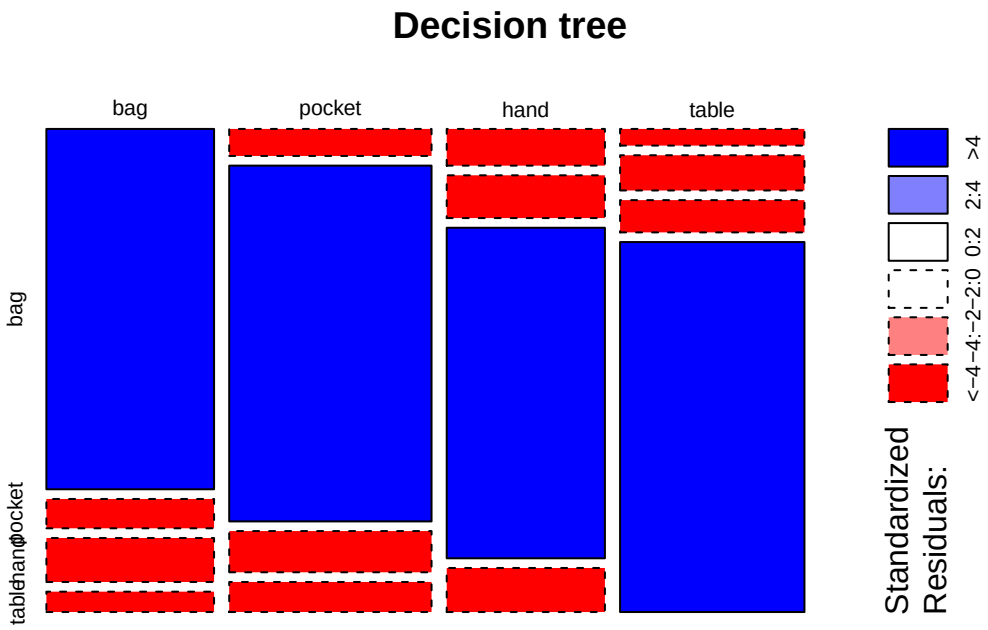
```
}
```

Fabrication d'une prediction avec arbre de décision

```
dt_result <-
  with(ds <- prepare_datasets(z_normalize(balance(slice_relevant(phone_observations)))),
                       run_dt(ds$train, ds$test))
```

On obtient avec cette prédiction une *accuracy* de 78.16%.

```
plot_confusion_matrix(dt_result$cmat$table, "Decision tree")
```




```
print_stats(dt_result$cmat)
```

	Balanced.Accuracy	Precision	Recall	F1
Class: bag	0.87	0.79	0.81	0.80
Class: pocket	0.86	0.78	0.80	0.79
Class: hand	0.81	0.73	0.71	0.72
Class: table	0.87	0.81	0.81	0.81
On devient bien	meilleur !			

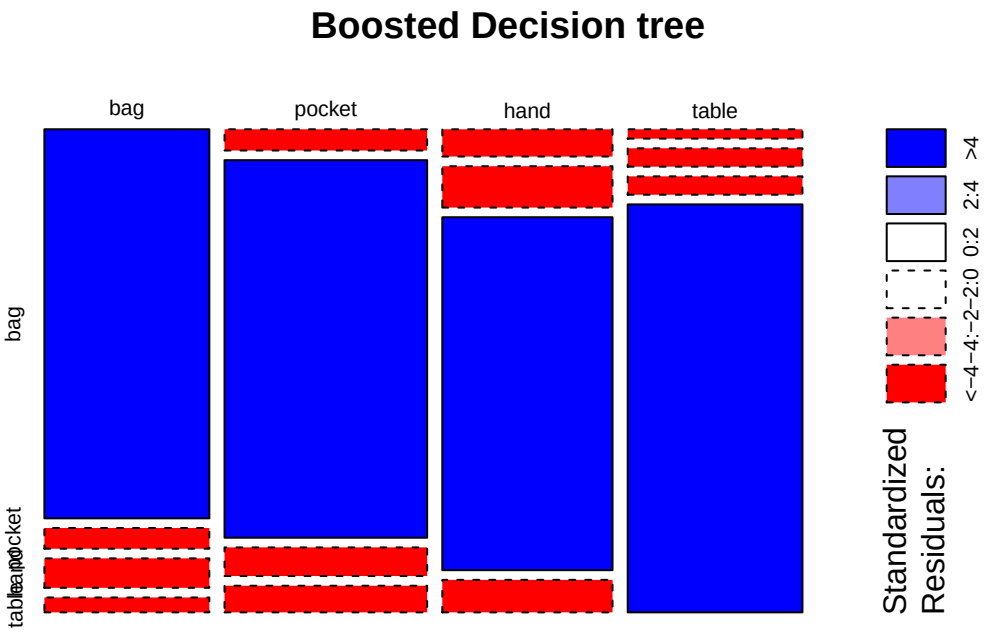
Utilisation de l'*adaptive boosting* pour améliorer le modèle

La technique d'*adaptive boosting* est un *méta*-algorithme, c.à.d un algorithme qui fonctionne au niveau d'autres algorithmes. Le principe sous-jacent à cette technique est de fabriquer non pas un seula rbre de décision, mais d'en fabriquer n. Pour chaque exemple, chaque arbre va faire une proposition de classification, et les arbres éliront la réponse qui semble la meilleure par un vote en fonction de leurs prédictions.

```
boost_result <-
  with(ds <- prepare_datasets(z_normalize(balance(slice_relevant(phone_observations)))),
                       run_dt(ds$train, ds$test, nb.trials=10))
```

On obtient avec cette prédiction une *accuracy* de 84.07%.

```
plot_confusion_matrix(boost_result$cmat$table, "Boosted Decision tree")
```




```
print_stats(boost_result$cmat)
```

	Balanced.Accuracy	Precision	Recall	F1
Class: bag	0.91	0.86	0.86	0.86
Class: pocket	0.89	0.83	0.85	0.84
Class: hand	0.87	0.78	0.81	0.79
Class: table	0.90	0.90	0.84	0.87

Méta-analyse

On regarde ici l'évolution des performances de nos prédicteurs.

Evolution globale de l'*accuracy*

Évolution des différentes métriques par classe

On commence par rassembler toutes les données d'interêt (les matrices de confusion des différentes étapes) dans un même jeux de données.

```
build_frame <- function(mat, stepName) {
  values <- rownames_to_column(data.frame(mat$byClass),"class") %>%
    select(contains("Accuracy"), Precision, Recall, F1, contains("Class"))
  values[values$class == 'Class: bag',]$class <- 'bag'
  values[values$class == 'Class: pocket',]$class <- 'pocket'
  values[values$class == 'Class: hand',]$class <- 'hand'
  values[values$class == 'Class: table',]$class <- 'table'
  values$step <- stepName
  return(values)
```

```
}

build_global_frames <- function() {
  tmp <-
  rbind.fill(build_frame(cmat, "1_kNN"),
             build_frame(cmat_balanced, "2_bal"),
             build_frame(cmat_z, "3_zsc"),
             build_frame(cmat_sliced, "4_sli"),
             build_frame(dt_result$cmat, "5_dtr"),
             build_frame(boost_result$cmat, "6_boo")
             )
  tmp$class <- as.factor(tmp$class)
  tmp$step  <- as.factor(tmp$step)
  return(tmp)
}
cmats <-  build_global_frames()
kable(cmats, digits=2)
```

Balanced.Accuracy	Precision	Recall	F1	class	step
0.63	0.79	0.28	0.41	bag	1_kNN
0.75	0.58	0.55	0.56	pocket	1_kNN
0.59	0.60	0.19	0.29	hand	1_kNN
0.73	0.82	0.97	0.89	table	1_kNN
0.71	0.69	0.49	0.57	bag	2_bal
0.76	0.63	0.68	0.65	pocket	2_bal
0.68	0.52	0.50	0.51	hand	2_bal
0.80	0.63	0.76	0.69	table	2_bal
0.76	0.75	0.57	0.65	bag	3_zsc
0.81	0.69	0.75	0.72	pocket	3_zsc
0.74	0.60	0.60	0.60	hand	3_zsc
0.82	0.68	0.76	0.72	table	3_zsc
0.76	0.73	0.57	0.64	bag	4_sli
0.80	0.70	0.72	0.71	pocket	4_sli
0.72	0.59	0.56	0.58	hand	4_sli
0.81	0.64	0.78	0.70	table	4_sli
0.87	0.79	0.81	0.80	bag	5_dtr
0.86	0.78	0.80	0.79	pocket	5_dtr
0.81	0.73	0.71	0.72	hand	5_dtr
0.87	0.81	0.81	0.81	table	5_dtr
0.91	0.86	0.86	0.86	bag	6_boo
0.89	0.83	0.85	0.84	pocket	6_boo
0.87	0.78	0.81	0.79	hand	6_boo
0.90	0.90	0.84	0.87	table	6_boo

Conclusion

Version courte : l'apprentissage machine et l'intelligence artificielle, c'est pour les grandes personnes. Il existe des cours au département (p.ex, INF4230, INF5081, INF7100) si vous voulez allez plus loin sur le sujet. Il est aussi possible de faire un projet de recherche crédité avec un prof (INF6200).

Version plus longue: S'il est très facile d'attraper un algorithme sur étagère et de l'appliquer à un jeu de données (quelques lignes de R), le faire intelligemment est beaucoup plus difficile. Cela demande une bonne

compréhension de la provenance des données, des méthodes sous-jacentes aux algorithmes, ...

Si vous souhaitez explorer la dimension “prédiction” plus avant, vous pourrez choisir cette spécialisation pour la dernière séquence de développement.