MNIST et Fashion MNIST (Illustration dans R)

Kanlanfeyi Kabirou, Hounsinou Jordy

Sommaire

1. Introduction

De nos jours, les technologies de Machine Learning et de Deep Learning s'imposent dans tous les secteurs d'activités du fait de la numérisation des données. Ainsi, elles permettent à ces secteurs de réaliser des progrès spectaculaires grâce à l'exploitation de grands volumes de données.

Le but de ce document est de fournir une présentation succinte des bases de données MNIST et Fashion MNIST qui sont entre autres des bases de l'apprentissage de ces technologies novatrices. Il s'en suivra une application de certains algorithmes de Machine Learning afin de montrer leur utilité dans la prédiction de chiffres pour le cas de la base de données MNIST ou encore d'articles de mode en ce qui concerne la base de données Fashion.

2. Présentation des bases de données

L'acronyme MNIST (Modified ou Mixed National Institute of Standards and Technology), est une base de données de chiffres écrits à la main. C'est un jeu de données très utilisé en apprentissage automatique. Créée dans un premier temps pour répondre à un problème de reconnaissance de l'écriture manuscrite, la base MNIST est devenue un test standard grâce à son efficacité pour les algorithmes d'apprentissage. Elle regroupe 60,000 images d'apprentissage et 10,000 images de test, issues d'une base de données antérieure, appelée simplement NIST. Ce sont des images en noir et blanc, normalisées centrées de 28 pixels de côté. (source Wikipédia)

La Fashion MNIST est un jeu de données de qui contient elle aussi 70 000 images en niveaux de gris répartie sur 1 des 10 catégories. Dans ce cas, les images montrent des vêtements, d'articles de Zalando , en basse résolution (28 x 28 pixels). La répartition des données "Apprentissage-Test" étant similaire, cette base de données vise à remplacer le jeu de données MNIST (de chiffres écrit à la main) plus assez complexe dans une logique d'apprentissage automatique.

Le processus de reconnaissance des chiffres et des articles de mode est constitué de plusieurs étapes spécifiques et utilise différents algorithmes de Machine Learning ou de Deep Learning. Nous vous proposons dans les prochaines lignes de monnayer cette procédure et de vous en expliquer les étapes.

3.Algorithmes choisis

Ici, nous exposons sur les algorithmes principaux utilisés dans notre application, notamment Naive Bayes, Random Forest.

Random Forest (les forêts d'arbres décisionnels) est un algorithme d'apprentissage automatique qui combine les concepts de sous-espaces aléatoires et de bagging. Le concept de cet algorithme est basé sur l'apprentissage de multiples arbres de décision entraînés sur des sous-ensembles de données légèrement différents.

Naive Bayes quant à lui est un algorithme de classification basé sur le théorème de Bayes, lui même basé sur les probabilités conditionnelles.

Figure 1: MNIST.

4.Exemple de code R

Dans cette partie nous allons utiliser des algorithmes de Machine Learning pour entrainer nos données afin de construire des modèles prédictifs qui seront à leurs tours utilisés pour des tests de nouvelles données. Une comparaison de nos modèles sera alors dans notre cas sur la base de leur précision en ce qui concerne la prédicition sur de nouvelles images. Cette partie du documet est subdivisée comme suit:

Importation des librairies et lecture des données

```
library(readr)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(caret)
```

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

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

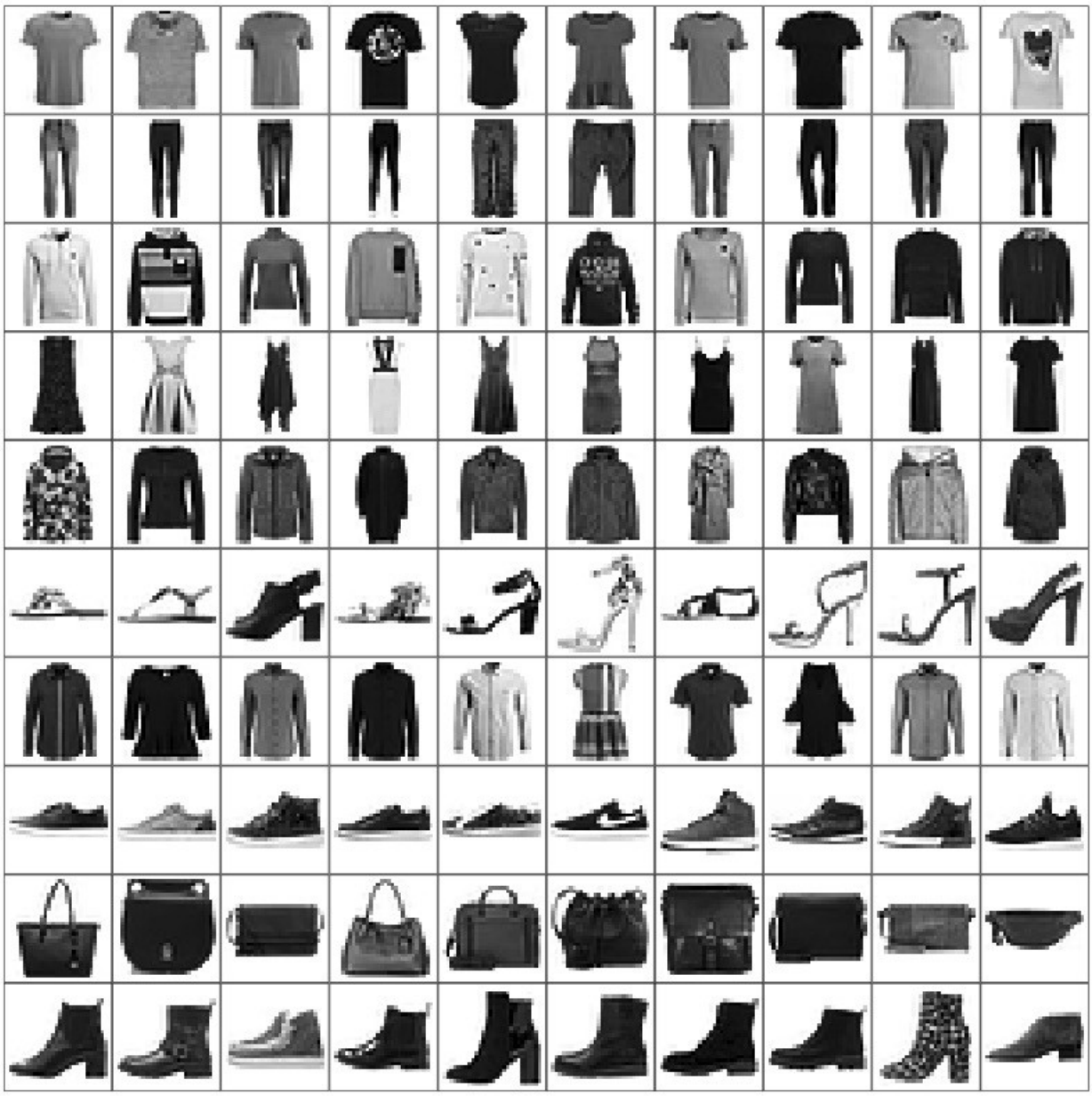

Figure 2: Fashion MNIST.

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
library(naivebayes)
```

```
## naivebayes 0.9.7 loaded
```

```
library(class)

#Pour fractionner les données
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:randomForest':
##
##     combine

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#Librairies Installée
#caret pour la matrice de confusion

#Lire les deux données: MNIST et fashion MNIST
mnist <- read_csv("train.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
fashion <- read_csv("fashion.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double()
## )
## i Use `spec()` for the full column specifications.
```

Encodage de la colonne label sous forme de catégorie avec la fonction *factor*

```
mnist$label = factor(mnist$label)
fashion$label = factor(fashion$label)
```

Normalisation des données

Afin de normaliser les données, nous divisons nos valeurs par 255 afin d'éviter par exemple les valeurs abérantes. La valeur maximale étant 255 pour chacune des colonnes, nous faisons alors une division par 255 afin d'avoir des valeurs comprises entre 0 et 1

```
mnist[,2:785] = mnist[,2:785]/255
fashion[,2:785] = fashion[,2:785]/255
```

Dans cette partie nous allons afficher les contenues des deux données avec la fonction head

Nous pouvons également remarquer que certaines valeurs sont comprise entre 0 et 1 dû à la normalisation des données

```
#Visualisation de la structure des données en affichant les premières lignes
head(mnist[1:10,1:10])
```

```
## # A tibble: 6 x 10
##   label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8
##   <fct>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 1          0      0      0      0      0      0      0      0      0
## 2 0          0      0      0      0      0      0      0      0      0
## 3 1          0      0      0      0      0      0      0      0      0
## 4 4          0      0      0      0      0      0      0      0      0
## 5 0          0      0      0      0      0      0      0      0      0
## 6 0          0      0      0      0      0      0      0      0      0
```

```
head(fashion[1:10,1:10])
```

```
## # A tibble: 6 x 10
##   label pixel1 pixel2 pixel3  pixel4  pixel5 pixel6 pixel7 pixel8 pixel9
##   <fct>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 2          0      0      0 0       0       0      0      0      0
## 2 9          0      0      0 0       0       0      0      0      0
## 3 6          0      0      0 0       0       0      0      0.0196 0
## 4 0          0      0      0 0.00392 0.00784 0      0      0      0
## 5 3          0      0      0 0       0       0      0      0      0
## 6 4          0      0      0 0.0196  0.0157  0.0196 0.0196 0.0118 0.0196
```

Nous allons alors scinder nos données en 2 parties: -une pour l'entrainement des données afin de construire des modèles prédictifs avec des algorithmes de machine Learning -une autre partie pour tester notre modèle construit

Nous allons utiliser la fonction sample!frac qui est fournie par la librairie *dplyr* Dans notre cas nous allons choisir un ratio de 80% pour l'entrainement et le reste pour le test

```
train_mnist <-  sample_frac(mnist, 0.8)
test_mnist <- anti_join(mnist, train_mnist)
```

```
## Joining, by = c("label", "pixel0", "pixel1", "pixel2", "pixel3", "pixel4", "pixel5", "pixel6", "pixe
```

```
train_fashion <-  sample_frac(fashion, 0.8)
test_fashion <- anti_join(fashion, train_fashion)
```

```
## Joining, by = c("label", "pixel1", "pixel2", "pixel3", "pixel4", "pixel5", "pixel6", "pixel7", "pixel
```

Construction de nos modèles:

Notons que nous utiliserons deux fois le même algorithme pour les deux jeux de données différents

Nous aimerons ajouter que pour la construction de nos modèles, nous avons minimisé l'ajout de paramètres au strict nécéssaire pour faciliter la durée de traitement.

Random Forest

```
rf_MNIST <- randomForest(label ~ ., data = train_mnist, ntree = 10)
pred_MNIST1 <- predict(rf_MNIST, test_mnist)

rf_FASH <- randomForest(label ~ ., data = train_fashion, ntree = 10)
pred_FASH1 <- predict(rf_FASH, test_fashion)
```

Naive Bayes

```
bayes_MNIST <- randomForest(label ~ ., data = train_mnist)
pred_MNIST2 <- predict(bayes_MNIST, test_mnist)

bayes_FASH <- randomForest(label ~ ., data = train_fashion)
pred_FASH2 <- predict(bayes_FASH, test_fashion)
```

Utilisation de la Matrice de confusion pour évaluer nos modelès construit

```
cm_rf1 <- confusionMatrix(pred_MNIST1, test_mnist$label)
cm_rf2 <- confusionMatrix(pred_FASH1, test_fashion$label)
cm_nb1 <- confusionMatrix(pred_MNIST2, test_mnist$label)
cm_nb2 <- confusionMatrix(pred_FASH2, test_fashion$label)
```

Affichage des résultats

Après les test de nos modèles sur les deux bases de données, nous nous retrouvons avec les résultats suivants

```
#Nous créons une matrice 2x2
valeurs <- matrix(c(cm_nb1$overall["Accuracy"],cm_nb2$overall["Accuracy"],cm_rf1$overall["Accuracy"],cm_
colnames(valeurs)<- c("Naive Bayes", "Random Forest")
rownames(valeurs)<- c("MNIST", "Fashion MNIST")
tableau <- as.table(valeurs)
print(tableau)
```

```
##               Naive Bayes Random Forest
## MNIST           0.9678571     0.9383333
## Fashion MNIST   0.8815779     0.8584772
```

Visualisation d'une matrice de confusion

```
cm_rf1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 796   0   5   5   0   8   6   2   3   4
##          1   0 948   2   4   2   3   0   3   5   2
##          2   3   7 835  23   4   0   2  13   5   2
##          3   2   2   5 817   0  36   0   7  15   4
##          4   0   3   4   3 780   4   3   5   5  33
##          5   4   0   4  26   0 667  11   0  12   8
##          6   3   1   4   0   6   8 770   0   4   3
##          7   1   2  10   6   1   0   0 782   3  11
##          8   7   2  10  14   9   9   6   1 758   7
##          9   1   1   2   7  32  13   2  18   5 729
##
## Overall Statistics
##
##                Accuracy : 0.9383
##                  95% CI : (0.933, 0.9434)
##     No Information Rate : 0.115
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9314
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.97430   0.9814   0.9478  0.90276  0.93525  0.89171
## Specificity           0.99565   0.9972   0.9922  0.99053  0.99207  0.99151
## Pos Pred Value        0.96019   0.9783   0.9340  0.92005  0.92857  0.91120
## Neg Pred Value        0.99723   0.9976   0.9939  0.98829  0.99286  0.98944
## Prevalence            0.09726   0.1150   0.1049  0.10774  0.09929  0.08905
## Detection Rate        0.09476   0.1129   0.0994  0.09726  0.09286  0.07940
## Detection Prevalence  0.09869   0.1154   0.1064  0.10571  0.10000  0.08714
## Balanced Accuracy     0.98497   0.9893   0.9700  0.94664  0.96366  0.94161
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.96250  0.94103  0.93006  0.90785
## Specificity           0.99618  0.99551  0.99143  0.98934
## Pos Pred Value        0.96370  0.95833  0.92102  0.90000
## Neg Pred Value        0.99605  0.99354  0.99248  0.99025
## Prevalence            0.09524  0.09893  0.09702  0.09560
## Detection Rate        0.09167  0.09310  0.09024  0.08679
## Detection Prevalence  0.09512  0.09714  0.09798  0.09643
## Balanced Accuracy     0.97934  0.96827  0.96075  0.94859
```

Interprétation des résultats

Les résultats obtenus nous montrent que les deux algorithmes sont relativement adéquats en ce qui concerne la prédiction de nouvelles images avec des précisions de plus de 80% pour les deux algorithmes dans notre cas

d'étude. Cette précision pourrait toutefois être optimisée en utilisant par exemple plus d'arbres de décisions dans le cas de l'algorithme Random Forest.

Afin de pallier au problème du bon choix des paramètres et d'éviter un travail manuel, certaines méthodes sont très utilisées telles que l'Hyperparameters Tuning.

Références

https://www.kaggle.com/c/digit-recognizer
https://mrmint.fr/naive-bayes-classifier
https://www.kaggle.com/zalando-research/fashionmnist
https://fr.wikipedia.org/wiki/For%C3%AAt_d%27arbres_d%C3%A9cisionnels#Algorithme
https://www.kaggle.com/arathee2/random-forest-vs-xgboost-vs-deep-neural-network
https://thinkr.fr/premiers-pas-en-machine-learning-avec-r-volume-4-random-forest
https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/factor
https://www.rdocumentation.org/packages/e1071/versions/1.7-3/topics/naiveBayes
https://fr.wikipedia.org/wiki/Matrice_de_confusion
https://stackoverflow.com/questions/24677642/centering-image-and-text-in-r-markdown-for-a-pdf-report