Rapport DM Data Mining

Mor SAMB

14 Mars 2017

On commence par charger les données :

```
install.packages("ElemStatLearn")
library(ElemStatLearn)
datazip.train <- as.data.frame(zip.train)
datazip.test <- as.data.frame(zip.test)
# On converti V1 en factor dans les deux bases :
datazip.train$V1 <- as.factor(datazip.train$V1)
datazip.test$V1 <- as.factor(datazip.test$V1)
```

1. Analyse Exploratoire :

   - Pour résumer nos données on applique la commande str() qui donne :

     ```
     # Base 1
      str(datazip.train)
     'data.frame': 7291 obs. of  257 variables:
      $ V1  : Factor w/ 10 levels "0","1","2","3",..:
      $ V2  : num  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
      $ V3  : num  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
      $ V4  : num  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
      $ V5  : num  -1 -0.813 -1 -1 -1 -1 -0.83 -1 -1 -1 ...
      $ V6  : num  -1 -0.671 -1 -1 -1 -1 0.442 -1 -1 -1 ...
      $ V7  : num  -1 -0.809 -1 -0.273 -0.928 -0.397 1 ...
      $ V8  : num  -1 -0.887 -1 0.684 -0.204 0.983 1 ...
      $ V9  : num  -0.631 -0.671 -1 0.96 0.751 -0.535 ...
     ...

     str(datazip.test)
     'data.frame': 2007 obs. of  257 variables:
      $ V1  : Factor w/ 10 levels "0","1","2","3",..:
      $ V2  : num  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
      $ V3  : num  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
      $ V4  : num  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
      $ V5  : num  -1 -1 -0.593 -1 -1 -1 -1 -1 -1 -1 ...
      $ V6  : num  -1 -1 0.7 -1 -1 -1 -1 -1 -1 -1 ...
     ....
     ```

     Donc on a : 7291 observations pour les données d'apprentissage et 2007 observations pour les données de test.

   - La distribution des classes :

     ```
     # Pour les Données d'apprentissage :
     table(datazip.train$V1)
     ```

```
   0    1    2    3    4    5    6    7    8    9
1194 1005  731  658  652  556  664  645  542  644
# Proportion :
round(prop.table(table(datazip.train$V1)), digits = 4)

     0      1      2      3      4      5      6
0.1638 0.1378 0.1003 0.0902 0.0894 0.0763 0.0911
     7      8      9
0.0885 0.0743 0.0883

# Pour les Données de test :
table(datazip.test$V1)

  0   1   2   3   4   5   6   7   8   9
359 264 198 166 200 160 170 147 166 177

# Proportion
round(prop.table(table(datazip.test$V1)), digits = 4)

     0      1      2      3      4      5      6
0.1789 0.1315 0.0987 0.0827 0.0997 0.0797 0.0847
     7      8      9
0.0732 0.0827 0.0882
```

- On voit bien que les classes ne sont pas bien equilibrées.

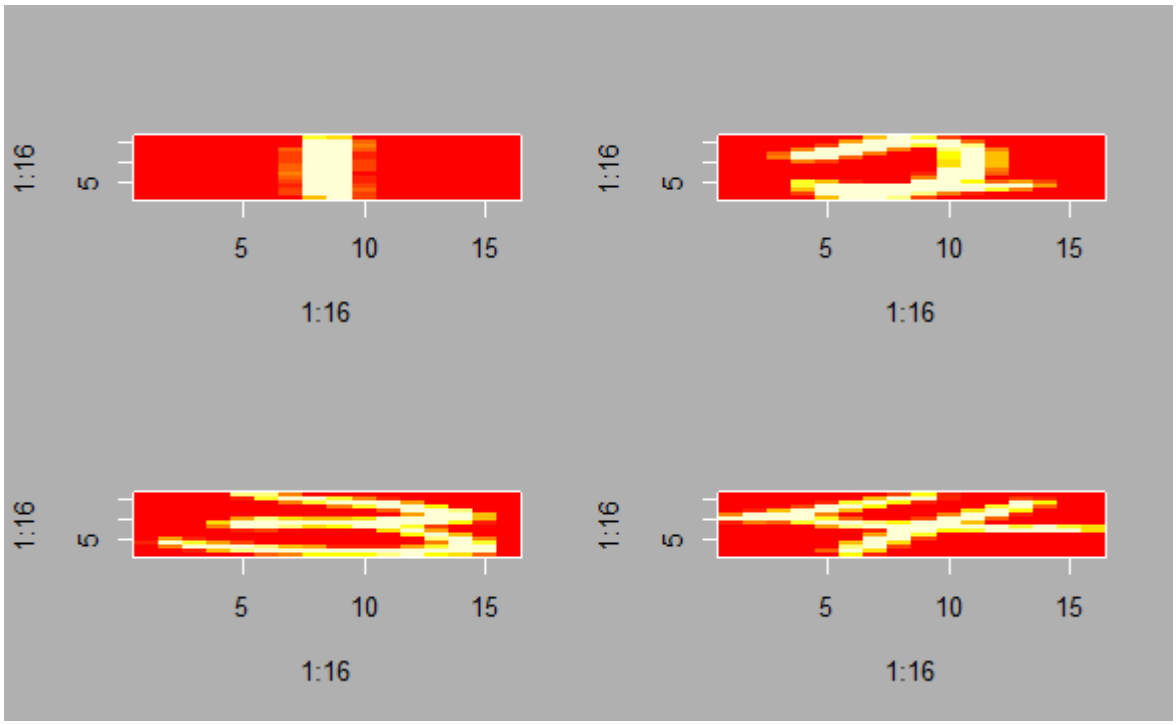
2. On crée une fonction image.Class(), qui prend comme paramètre le numéro de la classe :

```
1 image.Class <- function(data, k)
2 {
3   zz <- subset(data, data$V1 == k)
4
5   z <- array(as.numeric(zz[34,-1]), dim = c(16,16))
6   z <- z[,16:1]
7   im <- image(1:16, 1:16, z)
8
9 }
```

Ainsi, pour on peut faire par exemple :

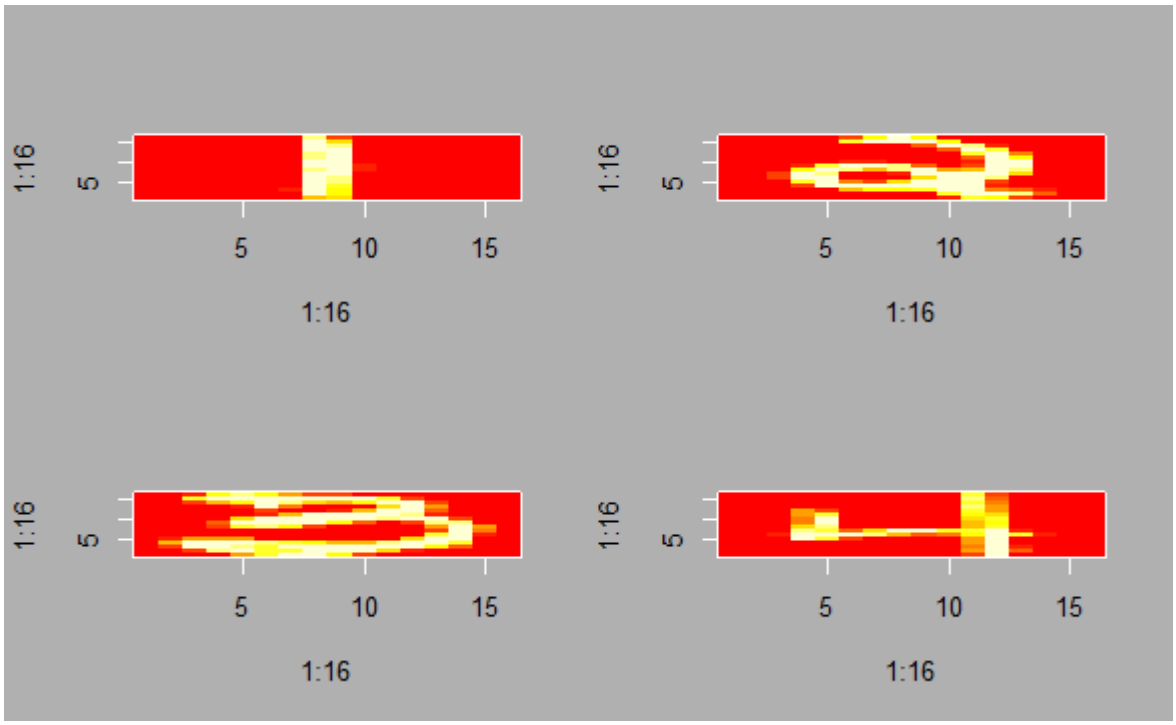
- Pour les données d'apprentissage :

```
par(mfrow = c(2,2))
image.Class(datazip.train, 1)
image.Class(datazip.train, 2)
image.Class(datazip.train, 3)
image.Class(datazip.train, 4)
```



- Pour les données de test :

```
image.Class(datazip.test, 1)
image.Class(datazip.test, 2)
image.Class(datazip.test, 3)
image.Class(datazip.test, 4)
```



3. Classification :
   Nous avons créé une fonction qui affiche la matrice de confusion et l' Error rate comme suit :

```
error.rate.CM <- function(data.test,pred)
  {

    # Matrice de confusion
    mc <- table(data.test$V1,pred)
    print(mc)
    # Error Rate
    err.rate <- 1-sum(diag(mc))/sum(mc)
    return(c(err.rate)
  }
```

   Ainsi on l'applique sur les différentes algorithmes de classification :

   (a) Avec LDA :

```
library(MASS)
prior <- c(0.1638, 0.1378, 0.1003, 0.0902, 0.0894,
           0.0763, 0.0911, 0.0885, 0.0743, 0.0883)
lda.app1 <- lda(datazip.train$V1 ~., data = datazip.train,
                prior = prior)
lda_pred <- predict(lda.app1, datazip.test)
print(error.rate.CM(datazip.test, lda_pred$class))
pred
      0   1   2   3   4   5   6   7   8   9
  0 342   0   0   4   3   1   5   0   3   1
```

```
  1   0 251   0   2   5   0   3   0   1   2
  2   7   2 157   4  12   2   1   1  12   0
  3   3   0   3 142   3   9   0   1   4   1
  4   1   4   6   0 174   0   2   2   1  10
  5   6   0   0  16   3 125   0   0   5   5
  6   1   0   3   0   3   3 157   0   3   0
  7   0   1   0   2   7   0   0 129   1   7
  8   5   0   2  11   7   4   0   0 135   2
  9   0   0   0   0   4   0   0   5   3 165
[1] 0.1145989
```

Avec la validation croisée :

```
d  <- datazip.train[100:1000,]
lda.prior <- lda(d$V1~., data = d, CV=TRUE)
print(error.rate.CM(d, lda.prior$class))
  pred
      0   1   2   3   4   5   6   7   8   9
  0 194   0   1   3   1   0   1   0   0   0
  1   0 108   0   0   0   0   0   0   0   1
  2   2   1 107   3   3   0   5   1   6   1
  3   1   0   0  52   0   2   0   2   3   1
  4   0   2   2   0  42   0   0   1   1   5
  5   3   0   2   7   3  25   2   1   2   0
  6   1   2   2   0   1   0  90   0   0   0
  7   0   1   1   0   2   0   1  46   1   7
  8   4   1   2   4   0   4   3   0  67   0
  9   1   0   0   0   1   0   0   2   1  59
[1] 0.1231964
```

On obtient un taux d'erreur de 12.32% qui est très élevé.

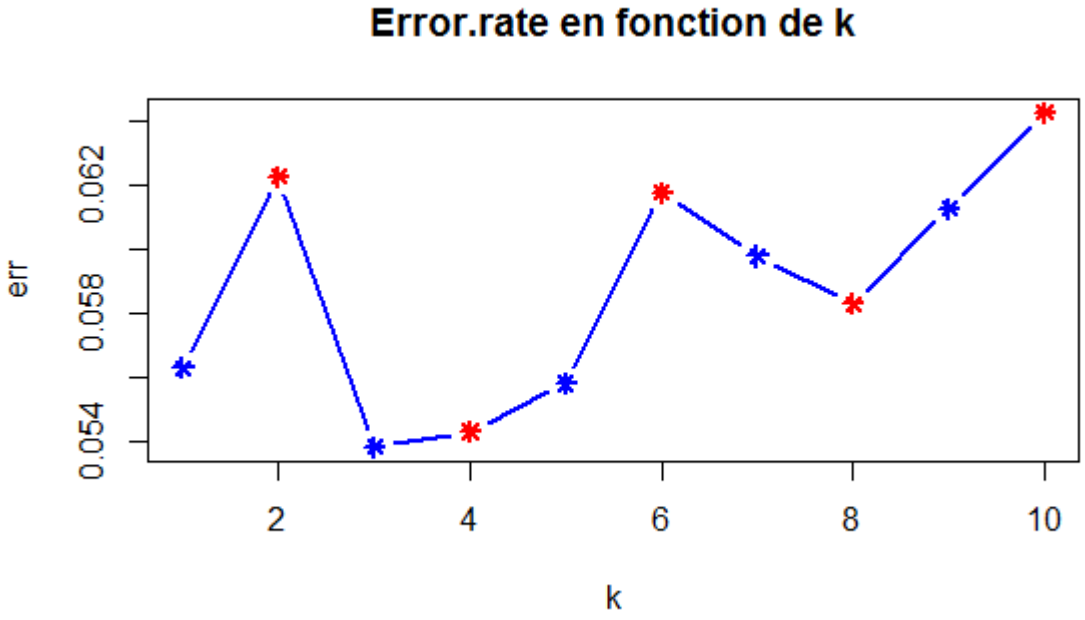
(b) Avec K nearest neighbors :

```
library(class)
data.knn <- knn(datazip.train[,-1], datazip.test[,-1],
                cl = datazip.train$V1, k=1, prob = FALSE)
print(error.rate.CM(datazip.test, data.knn))
   pred
      0   1   2   3   4   5   6   7   8   9
  0 355   0   2   0   0   0   0   1   0   1
  1   0 255   0   0   6   0   2   1   0   0
  2   6   1 183   2   1   0   0   2   3   0
```

```
  3   3   0   2 154   0   5   0   0   0   2
  4   0   3   1   0 182   1   2   2   1   8
  5   2   1   2   4   0 145   2   0   3   1
  6   0   0   1   0   2   3 164   0   0   0
  7   0   1   1   1   4   0   0 139   0   1
  8   5   0   1   6   1   1   0   1 148   3
  9   0   0   1   0   2   0   0   4   1 169
[1] 0.05630294
```

Nous obtenons un taux d'erruer de 5,63% en test, qui est meilleur que la précédente.
Le graphe en fonction de k

```
err<- rep(0,10)
for (i in 1:10){
  data.knn <- knn(datazip.train[,-1],datazip.test[,-1],
                  cl=datazip.train$V1, k=i, prob=TRUE)
  mc <-table(datazip.test$V1,data.knn)
  err[i] <- 1-sum(diag(mc))/sum(mc)
}
plot(err, xlab = "k",main ="Error.rate en fonction de k",
     type="b", col= c("blue","red"), pch=8, lwd = 2)
```



Ainsi on obtient une meilleur classifacation pour k=3 avec un taux d'erreur de 5.38%

(c) Avec SVM

- SVM avec noyeau linear :

```
library(e1071)

model1 <- svm(datazip.train$V1 ~., data = datazip.train,
              kernel = "linear")
## Prediction
pred.model1 <- predict(model1, newdata = datazip.test,
                       type="class")
print(error.rate.CM(datazip.test, pred.model1))
   pred
      0   1   2   3   4   5   6   7   8   9
  0 348   0   3   0   3   0   4   0   1   0
  1   0 256   0   0   5   0   3   0   0   0
  2   1   0 178   6   3   4   2   1   3   0
  3   2   0   2 148   0   9   0   1   4   0
  4   1   2   5   0 180   2   4   2   0   4
  5   5   0   0   7   2 141   0   1   3   1
  6   0   0   1   0   4   2 162   0   1   0
  7   0   0   1   0   6   0   0 135   1   4
  8   5   0   1   7   0   4   1   0 147   1
  9   0   0   0   0   2   1   0   3   0 171
[1] 0.07025411
```

- SVM avec noyeau radial :

```
model2 <- svm(datazip.train$V1 ~., data = datazip.train,
              kernel = "radial")
## Prediction
pred.model2 <- predict(model2, newdata = datazip.test,
                       type="class")
print(error.rate.CM(datazip.test, pred.model2))
   pred
      0   1   2   3   4   5   6   7   8   9
  0 351   0   6   0   1   0   0   0   1   0
  1   0 253   1   0   5   1   3   1   0   0
  2   2   0 183   4   3   0   1   1   4   0
  3   0   0   5 146   0  11   0   1   3   0
  4   0   1   3   0 186   1   2   3   1   3
  5   3   0   2   3   1 147   0   0   1   3
  6   4   0   4   0   2   1 158   0   1   0
  7   0   0   2   0   5   0   0 138   0   2
  8   4   0   2   3   0   2   1   0 151   3
```

```
  9   0   0   0   0   4   1   0   0   2 170
[1] 0.06178376
```

- SVM avec noyeau Polynomial :

```
model3 <- svm(datazip.train$V1 ~., data = datazip.train,
              kernel = "polynomial")
## Prediction
pred.model3 <- predict(model3, newdata = datazip.test,
                       type="class")
print(error.rate.CM(datazip.test, pred.model3))
   pred
      0   1   2   3   4   5   6   7   8   9
  0 352   0   3   0   2   0   1   0   1   0
  1   0 251   1   0   6   1   4   0   0   1
  2   0   0 179   3   3   2   0   1  10   0
  3   0   0   2 151   0  10   0   1   2   0
  4   0   1   2   0 190   0   2   1   0   4
  5   2   0   0   3   1 146   0   0   6   2
  6   3   1   3   0   2   1 159   0   1   0
  7   0   0   0   1   5   1   0 135   2   3
  8   0   0   0   3   0   2   0   0 158   3
  9   0   0   0   0   4   0   0   0   3 170
[1] 0.05779771
```

Ainsi On trouve un meilleur taux d'erreur (5,78%) avec le noyeau polynomial.

(d) Regression logistique

```
library(MASS)
model_logic <- polr(V1~., data = datazip.train)
model_logic
model.pred <- predict(model_logic, datazip.test, type = "class")
print(error.rate.CM(datazip.test, model.pred))
   pred
      0   1   2   3   4   5   6   7   8   9
  0 274  66   7   6   1   0   4   1   0   0
  1  16 207  24   5   8   0   1   2   0   1
  2  20  60  39  29  14   0  23  10   0   3
  3   7  41  28  32  16   0  23  17   0   2
  4   2   7  20  26   9   0  39  37   0  60
  5   8  24  27  30   5   0  21  34   0  11
  6   2  16  31  25   9   0  52  32   0   3
```

```
  7   0   5   1  15   3   0  22  58   0  43
  8   2   8  14  16   3   0  32  49   0  42
  9   0   0   7   5   0   0  21  24   0 120
[1] 0.6058794
```

(e) Regression Tree :

```
library(rpart)
arb1 <- rpart(V1 ~ .,method="class", data=datazip.train)
pred1<-predict(arb1,datazip.test, type="class")
print(error.rate.CM(datazip.test, pred))
   pred
      0   1   2   3   4   5   6   7   8   9
  0 294   3  31  12   1   2   1   0  15   0
  1   1 243   2   0   3   1   0   1   5   8
  2   9   9 111  10  13   2  11   5  24   4
  3  10   4   2 105   3  22   0   3  16   1
  4   1  26  11   0 138   1   0   5   5  13
  5  17  11   6  15   2  89  10   2   7   1
  6  11   8   4   1   2  10 104   3  27   0
  7   0   9   6   1  10   1   0 113   3   4
  8   1   9  11  12   2   3   1   2 120   5
  9   0  16   2   5   1   0   0   8   7 138
[1] 0.2750374
```

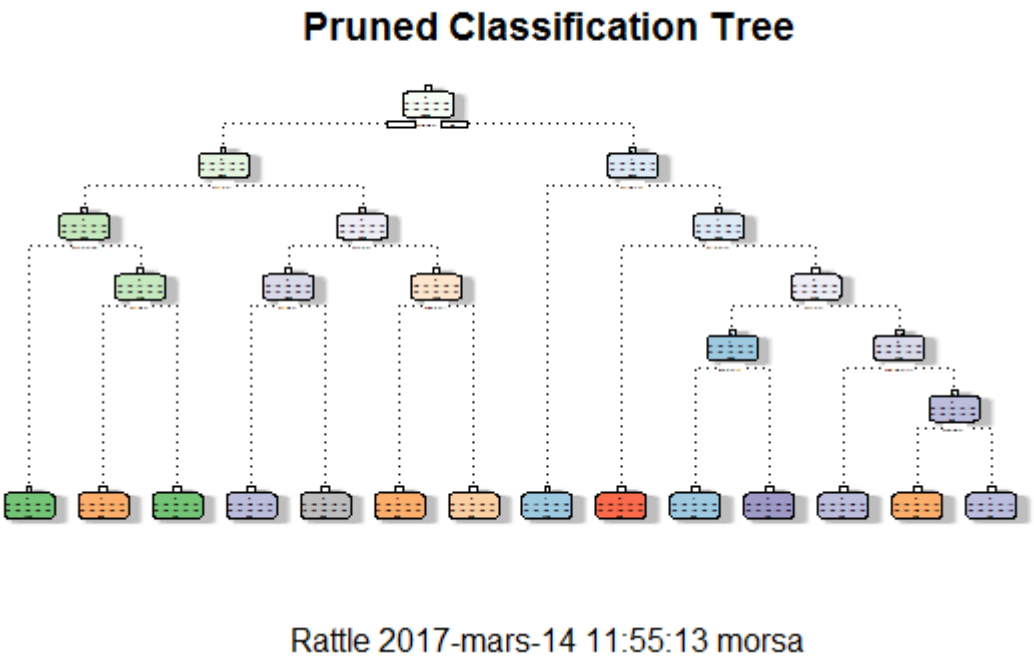
Arbre Elagé :

```
arb2 <- rpart(V1 ~ .,method="class", data=datazip.train,
              control=sctrl)
pred2 <- predict(arb2, datazip.test, type= "class")
print(error.rate.CM(datazip.test, pred2))
   pred
      0   1   2   3   4   5   6   7   8   9
  0 326   0   7   9   6   2   4   0   4   1
  1   0 247   0   2   5   0   5   1   4   0
  2   7   0 154   6   8   5   3   4   9   2
  3   7   0   6 128   1  17   2   0   3   2
  4   2   4   8   2 160   3   7   3   4   7
  5   5   0   5  15   5 113   7   1   5   4
  6   4   2   4   4   6   2 141   2   4   1
  7   0   1   1   4   3   2   3 128   1   4
  8   5   2  13   4   6   8   4   1 114   9
  9   1   3   3   4   5   6   1   4   5 145
```

```
[1] 0.1748879
```

Pruned Tree :

```
arb3 <- prune(arb1,
cp = arb1$cptable[which.min(arb1$cptable[,"xerror"]),"CP"])
pred3 <- predict(arb3, datazip.test, type = "class")
print(error.rate.CM(datazip.test, pred3))
  pred
      0   1   2   3   4   5   6   7   8   9
  0 294   3  31  12   1   2   1   0  15   0
  1   1 243   2   0   3   1   0   1   5   8
  2   9   9 111  10  13   2  11   5  24   4
  3  10   4   2 105   3  22   0   3  16   1
  4   1  26  11   0 138   1   0   5   5  13
  5  17  11   6  15   2  89  10   2   7   1
  6  11   8   4   1   2  10 104   3  27   0
  7   0   9   6   1  10   1   0 113   3   4
  8   1   9  11  12   2   3   1   2 120   5
  9   0  16   2   5   1   0   0   8   7 138
[1] 0.2750374
library(rattle)
library(rpart.plot)
library(RColorBrewer)
fancyRpartPlot(arb3, uniform=TRUE,
               main="Pruned Classification Tree")
```



4. Comparaison des modèles :

Méthode	Taux d'erreur(%)	Temps d'exécution
LDA	11.46	5.69
1-nearest	5.63	18.74
3-nearest	5.38	16.38
SVM Linear	7.03	16.99
SVM Radial	6.18	33.38
SVM Polynomial	5.78	39.19
Regression Logistique	60.58	10.62
Default Tree	27.5	5.66
Maximal Tree	17.49	11.34
Optimal Tree	27.5	1.06

On obtient de meilleurs résultats avec le k-nearest et le SVM polynomial avec des taux d'erreurs respectifs de 5.63% et 5.78% avec temps d'exécution assez élevé de 18.74 et 39.19 secondes.
Donc la meilleur est la méthode du k-nearest.

5.