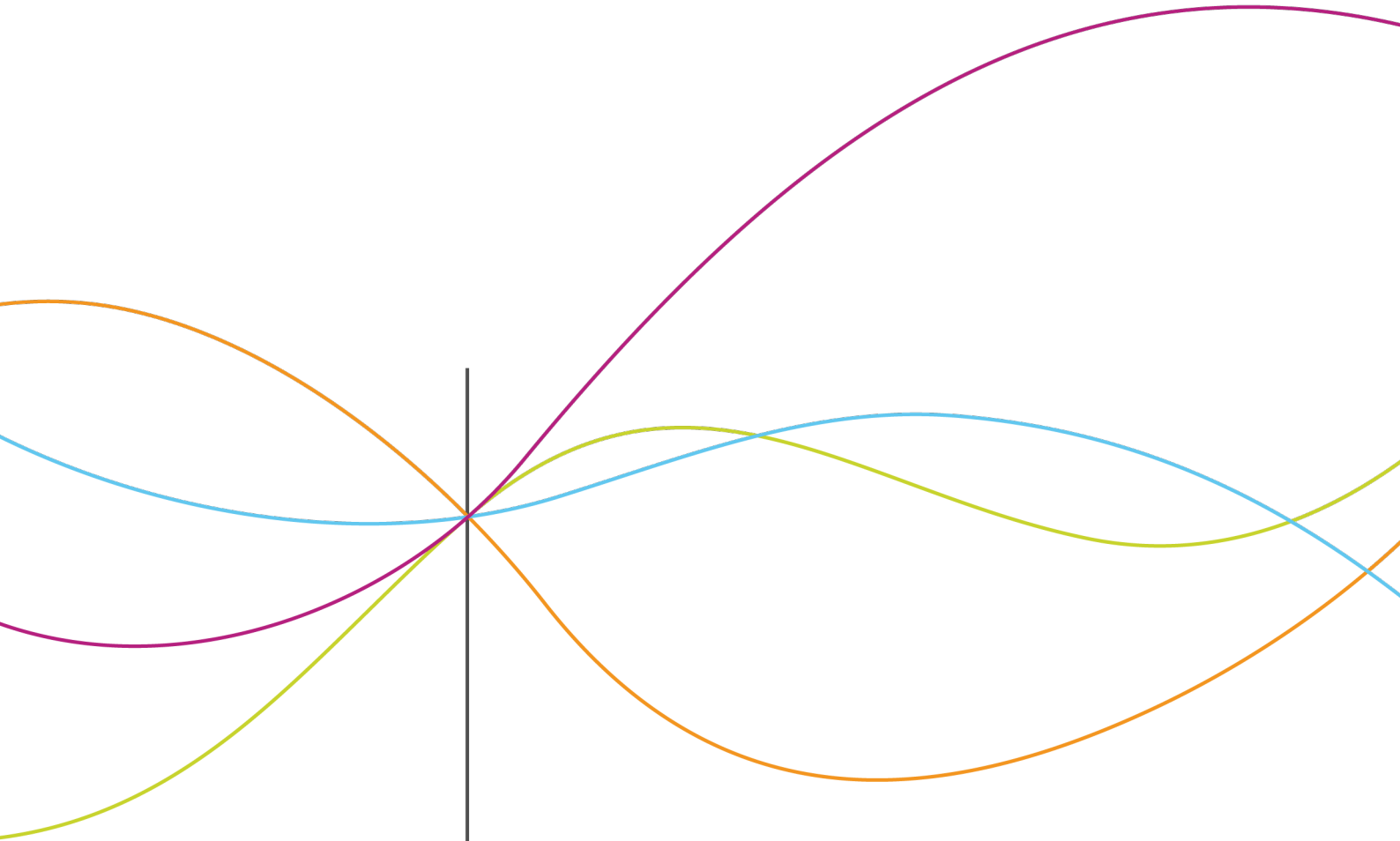
École nationale
de la statistique
et de l'administration
économique
ENSAE ParisTech
université PARIS-SACLAY



TP2 : KNN, DECISION TREES AND STOCK MARKET RETURNS

2017/2018

Gouysse Margaux
Grelety Antoine
Watrigant Timothée

Question 1

La commande $apply(cvpred, 2, function(x)sum(class!x))$ permet de calculer l'erreur de classification sur les prédictions (cvpred) en faisant la somme de toutes les mauvaises prédictions (celles n'ayant pas données la classe attendue) de chaque colonne de cvpred. On obtient alors une erreur pour chaque valeur de k possible.

Question 2

Lorsqu'on relance plusieurs fois les commandes

```
> # 5-fold cross-validation to select k
> # from the set {1,...,10}
> fold = sample(rep(1:5,each=18))
> cvpred = matrix(NA,nrow=90,ncol=10)
> for (k in 1:10)
+   for (v in 1:5)
+   {
+     sample1 = train[which(fold!=v),1:4]
+     sample2 = train[which(fold==v),1:4]
+     # creation des groupes B_v
+     # initialisation de la matrice
+     # des prédicteurs
+     class1 = train[which(fold!=v),5]
+     cvpred[which(fold==v),k] = knn(sample1,sample2,class1,k=k)
+   }
> class = as.numeric(train[,5])
> # display misclassification rates for k=1:10
> apply(cvpred,2,function(x) sum(class!=x)) # calcule l'erreur de classif.
```

nous n'obtenons pas le même résultat car on ne s'entraîne pas sur le même échantillon. En effet, la variable fold est aléatoire à chaque relance des commandes R ci-dessus.
Si l'on veut choisir k en combinant les résultats obtenus après 100 itérations de ce code, il serait possible de calculer la moyenne des erreurs obtenus pour chaque k et ensuite de choisir la valeur la plus petite. La moyenne permettrait de limiter le côté aléatoire du tirage fait pour la cross validation.

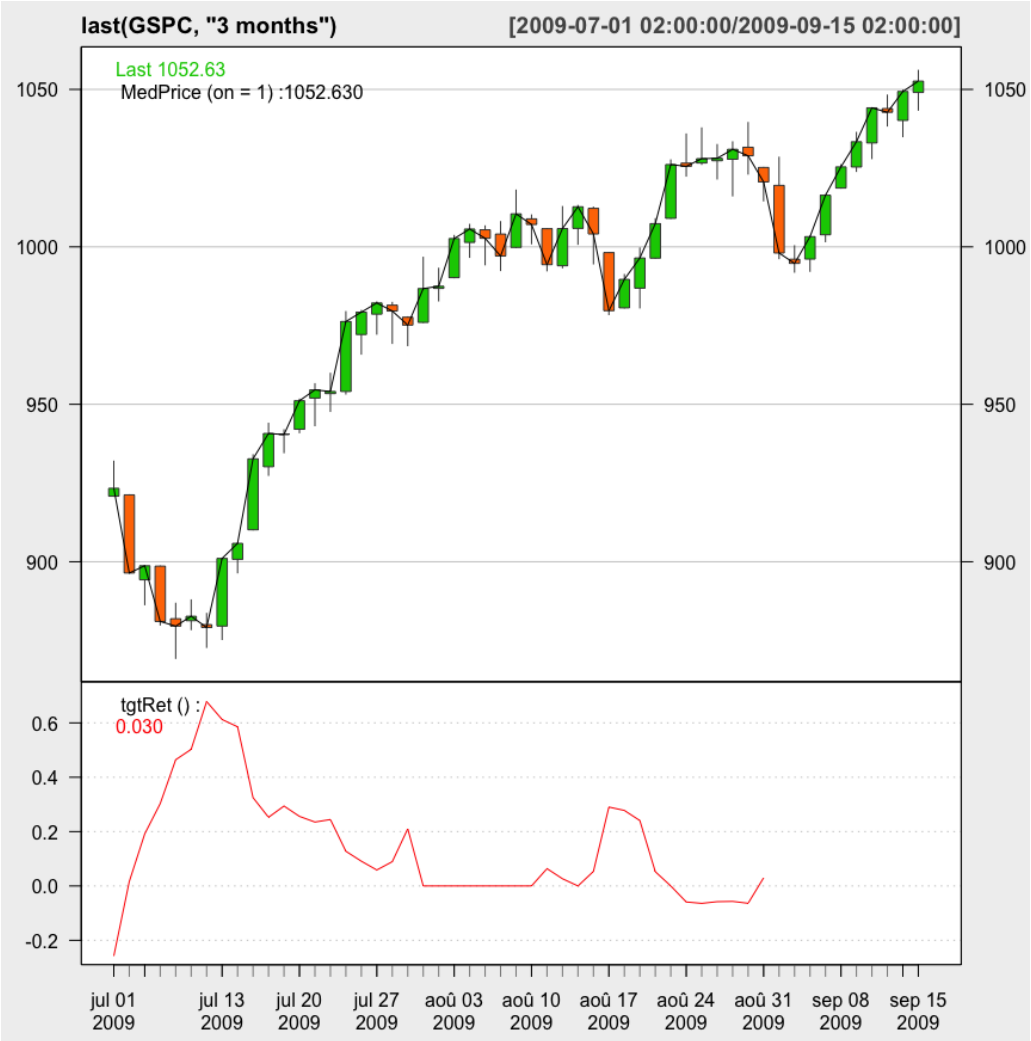
Question 3

Question 3.1

Code permettant d'ajouter aux chandeliers japonais la courbe des valeurs médianes de (C_i, H_i, L_i) :

```
> candleChart(last(GSPC, "3 months"), theme = "white", TA = NULL)
> medPrice = function(p) apply(HLC(p), 1, median)
> addMedPrice = newTA(FUN = medPrice, col = 1, legend = "MedPrice",cex=0.3)
> addT.ind = newTA(FUN = T.ind, col = "red", legend = "tgtRet",cex=0.3)
> get.current.chob<-function(){quantmod:::get.current.chob()}
> candleChart(last(GSPC, "3 months"), theme = "white", TA = "addMedPrice(on=1)",cex=0.3)
> candleChart(last(GSPC, "3 months"), theme = "white", TA = "addT.ind();addMedPrice(on=1)",cex=0.3)
```

Si on retire $on = 1$ dans la fonction $addAvgPrice$, on obtient une courbe séparée pour la moyenne des prix et non plus superposée sur la première courbe.


last(GSPC, "3 months")
[2009-07-01 02:00:00/2009-09-15 02:00:00]
Last 1052.63
MedPrice (on = 1) :1052.630
1050
1000
950
900
tgtRet () :
0.030
0.6
0.4
0.2
0.0
-0.2
jul 01 2009
jul 13 2009
jul 20 2009
jul 27 2009
aoû 03 2009
aoû 10 2009
aoû 17 2009
aoû 24 2009
aoû 31 2009
sep 08 2009
sep 15 2009

Question 3.2

L'option $training.per$ de la fonction $buildModel$ correspond à l'intervalle de temps sur lequel le modèle va s'entraîner. Donc dans notre cas, de la première date au 31/12/1999 qui correpond à la première date + 30 ans. L'option $importance = True$ permet d'obtenir un vecteur qui calcule l'importance de chaque variable (myATR, myBB, etc).

Question 3.3

Les 8 variables les plus pertinentes sont les 8 variables pour lesquelles le pourcentage d'augmentation de l'erreur quadratique sont les plus importants, car cela signifie que lorsqu'on la supprime l'erreur augmente. On va donc retenir les 8 variables suivantes :

- mySAR,
- myADX,
- myMACD,
- myVolat,
- myATR,
- mySMI,
- myMFI,
- myCLV.

Question 3.4

```
> data.model = specifyModel(T.ind(GSPC) ~
+                            myATR(GSPC) + mySMI(GSPC) + myADX(GSPC) +
+                           myCLV(GSPC) +
+                            myVolat(GSPC) + myMACD(GSPC) + myMFI(GSPC)  +
+                            mySAR(GSPC))
```

Question 3.5

La fonction $na.omit$ retire les lignes pour lesquelles il y a des valeurs "NA". Son utilisation est plus importante dans la définition de l'échantillon test car on ne peut pas faire de prédictions sur une valeur "NA" alors que dans l'échantillon d'entraînement, si il y a des valeurs "NA", la variable correspondante va avoir un poids nul.

Question 4

Code pour l'algorithme kNN avec cross validation et affichage de l'erreur de prédiction :

```
> pred <- list()
> for (i in 1:40){
+ pred[[i]] = knn(Tdata.train[,2:9], Tdata.eval[,2:9], Tdata.train[,1], k = i)
+ # display the confusion matrix
+ print(sum(diag((table(pred[[i]],Tdata.eval[,1]))))/nrow(Tdata.eval))
+ }

[1] 0.4588477
[1] 0.4748971
[1] 0.4888889
[1] 0.4921811
[1] 0.4995885
[1] 0.5045267
[1] 0.4860082
[1] 0.5045267
[1] 0.5041152
[1] 0.5037037
[1] 0.5213992
[1] 0.5127572
```

```
[1] 0.5201646
[1] 0.5176955
[1] 0.5222222
[1] 0.5234568
[1] 0.5218107
[1] 0.5234568
[1] 0.5255144
[1] 0.5316872
[1] 0.5353909
[1] 0.5378601
[1] 0.5444444
[1] 0.5506173
[1] 0.5567901
[1] 0.5691358
[1] 0.5847737
[1] 0.590535
[1] 0.6028807
[1] 0.6222222
[1] 0.6333333
[1] 0.6358025
[1] 0.6407407
[1] 0.6395062
[1] 0.6432099
[1] 0.6423868
[1] 0.6423868
[1] 0.6403292
[1] 0.6452675
[1] 0.645679
```

L'erreur du modèle augmente avec k. Il y a cependant un risque d'overfit pour k trop petit. Avec k=1, le modèle apprend toutes les irrégularités de l'échantillon d'apprentissage qui ne sont pas forcément généralisables, notamment lors de le periodes de forte volatilité du stock market price.

Question 5

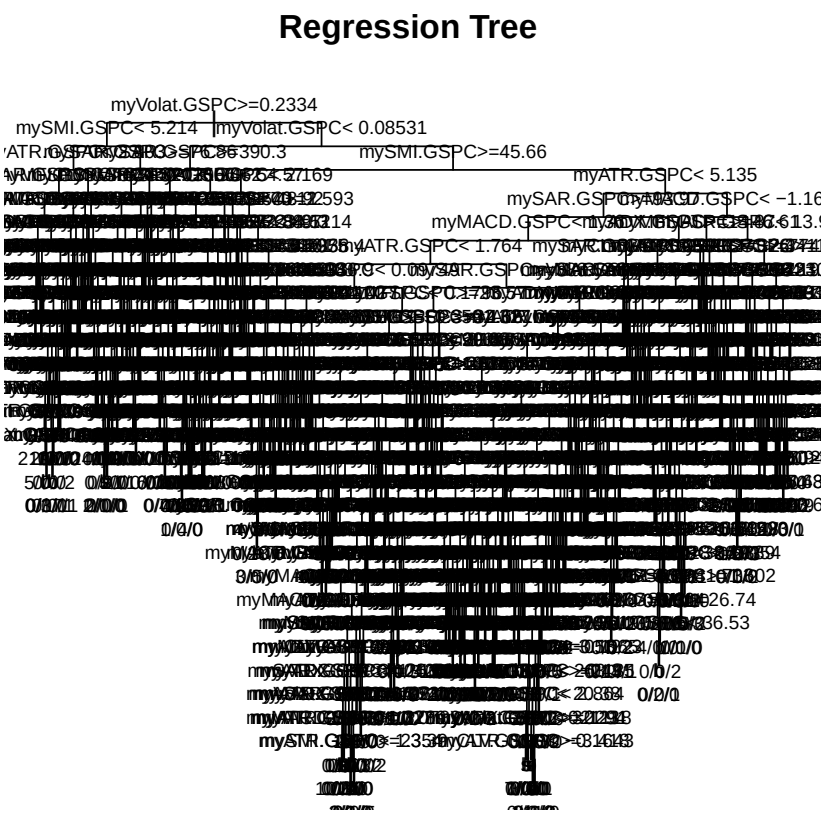
Code pour l'algorithme d'arbre de décision avec cross validation sur le paramètre cp :

```
> library(rpart)
> fit <- rpart(factor(signal) ~ ., method = "class",data= Tdata.train,
+              control = rpart.control(cp = 0.0001, minsplit = 2, minbucket = 1))
> printcp(fit)

Classification tree:
rpart(formula = factor(signal) ~ ., data = Tdata.train, method = "class",
    control = rpart.control(cp = 1e-04, minsplit = 2, minbucket = 1))

Variables actually used in tree construction:
[1] myADX.GSPC   myATR.GSPC   myCLV.GSPC   myMACD.GSPC  myMFI.GSPC
[6] mySAR.GSPC   mySMI.GSPC   myVolat.GSPC

Root node error: 2136/7542 = 0.28321

n= 7542

           CP nsplit rel error  xerror     xstd
1  0.00983146      0  1.000000 1.00000 0.018319
2  0.00499376      3  0.970506 0.98596 0.018240
3  0.00468165      6  0.955524 0.97706 0.018189
4  0.00374532      8  0.946161 0.97425 0.018173
```

```
5  0.00327715     10  0.938670 0.96536 0.018121
6  0.00304307     11  0.935393 0.96489 0.018119
7  0.00280899     19  0.910581 0.95506 0.018061
8  0.00249688     20  0.907772 0.94944 0.018027
9  0.00234082     43  0.837547 0.93773 0.017956
10 0.00187266     55  0.807116 0.91901 0.017840
11 0.00177903     78  0.761704 0.91058 0.017787
12 0.00163858    119  0.666199 0.88858 0.017644
13 0.00150853    126  0.653090 0.88811 0.017641
14 0.00148252    136  0.637172 0.85019 0.017384
15 0.00140449    153  0.607678 0.85019 0.017384
16 0.00131086    197  0.545880 0.84504 0.017347
17 0.00128745    202  0.539326 0.83567 0.017281
18 0.00124844    207  0.532303 0.83567 0.017281
19 0.00117041    210  0.528558 0.83661 0.017288
20 0.00109238    228  0.507491 0.83614 0.017284
21 0.00100321    253  0.473315 0.83474 0.017274
22 0.00093633    267  0.453652 0.81133 0.017104
23 0.00078027    395  0.326779 0.80665 0.017070
24 0.00074906    398  0.324438 0.80478 0.017056
25 0.00070225    407  0.316948 0.80665 0.017070
26 0.00065543    444  0.289794 0.80665 0.017070
27 0.00062422    449  0.286517 0.80665 0.017070
28 0.00058521    480  0.265449 0.80758 0.017077
29 0.00056180    488  0.260768 0.80946 0.017090
30 0.00046816    506  0.250000 0.78839 0.016932
31 0.00040128    868  0.079120 0.78699 0.016921
32 0.00037453    877  0.075375 0.79401 0.016975
33 0.00035112    887  0.071629 0.79401 0.016975
34 0.00031211    921  0.059457 0.79401 0.016975
35 0.00028090    939  0.053839 0.79494 0.016982
36 0.00023408    960  0.047753 0.80431 0.017052
37 0.00010000   1164  0.000000 0.80431 0.017052
```

```
> pfit<- prune(fit, cp=0.00037453) # from cptable
> rt.predictions.T = predict(pfit, Tdata.eval)
> rt.predict <- apply(rt.predictions.T,1,which.max)
> rt.predict <- colnames(rt.predictions.T)[rt.predict]
> rt.predict <- factor(rt.predict,levels=levels(Tdata.eval[,1]))
> print(sum(diag((table(rt.predict,Tdata.eval[,1])))/nrow(Tdata.eval))

[1] 0.4
```

Arbre de décision final :



On observe que l'arbre de décision fournit de meilleurs résultats que l'algorithme kNN.
L'arbre de décision est bien adapté à des problèmes de classification sur des valeurs discrètes (B, S, H dans notre cas).
En faisant une cross validation sur CP, l'erreur du modèle semble diminuer avec sa complexité (nombre de branches) jusqu'à un certain point. Ici, l'erreur du modèle est minimale pour cp=X (y itération).