Machine Learning Supervisé avec R

Florian Pothin

Classe : IAS-M2-DA-2

23/10/2024

Calcul des weights of evidence (WoE) II

- On a créé un data frame contenant les WoE des variables explicatives initiales

```
> head(credit_woe)
     Comptes Duree_credit Historique_credit Objet_credit Montant_credit    Epargne Anciennete_emploi
1 -0.8180987    0.4988765        0.73374058    0.1643031      0.2058521  0.7621401        0.29871667
2 -0.4013918   -0.9162907       -0.08786876    0.1643031     -0.5524983 -0.2524534       -0.03210325
3  1.1762632    0.4988765        0.73374058   -0.3109932      0.2058521 -0.2524534        0.29871667
4 -0.8180987   -0.9162907       -0.08786876    0.1643031     -0.5524983 -0.2524534        0.29871667
5 -0.8180987   -0.2035434       -0.08786876   -0.3592005     -0.5524983 -0.2524534       -0.03210325
6  1.1762632   -0.2035434       -0.08786876   -0.3109932     -0.5524983  0.7621401       -0.03210325
  Taux_effort Situation_familiale   Garanties Anciennete_domicile       Biens        Age Autres_credits
1 -0.15730029           0.1616414 -0.02797385        -0.001152738  0.46103496  0.1391971      0.1211786
2  0.15546647          -0.2353408 -0.02797385        -0.070150705  0.46103496 -0.5288441      0.1211786
3  0.15546647           0.1616414 -0.02797385         0.054941118  0.46103496  0.1391971      0.1211786
4  0.15546647           0.1616414  0.58778666        -0.001152738 -0.03188189  0.1391971      0.1211786
5  0.06453852           0.1616414 -0.02797385        -0.001152738 -0.58608236  0.1391971      0.1211786
6  0.15546647           0.1616414 -0.02797385        -0.001152738 -0.58608236  0.1391971      0.1211786
  Statut_domicile Nb_credits Type_emploi Nb_pers_charge   Telephone Cible
1       0.1941560  0.1157105  0.02278003    -0.00281611  0.09863759     0
2       0.1941560 -0.0748775  0.02278003    -0.00281611 -0.06469132     1
3       0.1941560 -0.0748775  0.09716375     0.01540863 -0.06469132     0
4      -0.4302047 -0.0748775  0.02278003     0.01540863 -0.06469132     0
5      -0.4302047  0.1157105  0.02278003     0.01540863 -0.06469132     1
6      -0.4302047 -0.0748775  0.09716375     0.01540863  0.09863759     0
```

- On crée un échantillon d'apprentissage et un de validation

```
> train  <- credit_woe[id,]
> valid  <- credit_woe[-id,]
```

Modèle logit sur WoE I

- On ajuste un modèle logit, qui est plus simple, avec un seul degré de liberté par variable

```
> logit <- glm(Cible~Comptes+Historique_credit+Duree_credit+Age+Epargne+Garanties+Autres_credits, data=train, family=binomial(link = "logit"))
> summary(logit)
Call:
glm(formula = Cible ~ Comptes + Historique_credit + Duree_credit +
    Age + Epargne + Garanties + Autres_credits, family = binomial(link = "logit"),
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9421  -0.7566  -0.4523   0.8537   2.6282

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)       -0.8672     0.1009  -8.595  < 2e-16 ***
Comptes           -0.8984     0.1295  -6.937 4.01e-12 ***
Historique_credit -0.8131     0.1822  -4.462 8.11e-06 ***
Duree_credit      -0.9974     0.2225  -4.482 7.38e-06 ***
Age               -0.6552     0.3434  -1.908  0.05640 .
Epargne           -1.0354     0.2451  -4.224 2.40e-05 ***
Garanties         -2.0730     0.7990  -2.595  0.00947 **
Autres_credits    -0.8773     0.4038  -2.172  0.02983 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Null deviance: 814.01  on 665  degrees of freedom
Residual deviance: 639.04  on 658  degrees of freedom
AIC: 655.04

Number of Fisher Scoring iterations: 5
```

Modèle logit sur WoE II

- Ce nouveau modèle a une aire sous la courbe ROC plus élevée : 0,7596 → 0,7657

```
> pred.logit <- predict(logit, newdata=valid, type="response")
> auc(valid$Cible, pred.logit, quiet=TRUE)
Area under the curve: 0.7657
```

- Le nombre de coefficients non significatifs au seuil de 5 % est passé de 3 à 1
  - c'est l'âge dont la p-value = 5,64 %

```
> sum(summary(logit)$coefficients[,4] >= 0.05)
[1] 1
```

Modèle logit sur WoE III

- La simplification du modèle peut permettre de prendre en compte des variables qualitatives discriminantes dont certaines modalités ont des coefficients non significativement ≠ 0
  - c'est le cas de l'objet de crédit dont la modalité « Intérieur » a une p-value = 96,6 %

```
> logit <-
glm(Cible~Comptes+Historique_credit+Duree_credit+Age+Epargne+Garanties+Autres_credits+Objet_credit,
data=train, family=binomial(link = "logit"))

> summary(logit)

Coefficients:
```

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	−0.68304	0.64957	−1.052	0.29302	
ComptesCC < 0 euros	0.16917	0.23478	0.721	0.47119	
ComptesCC > 200 euros	−1.36324	0.47385	−2.877	0.00402	**
ComptesPas de compte	−1.46689	0.27164	−5.400	6.66e−08	***
Historique_creditCrédits passés sans retard	−1.59679	0.37329	−4.278	1.89e−05	***
Historique_creditPas de crédits ou en cours sans retard	−0.90195	0.33336	−2.706	0.00682	**
Duree_credit(15,36]	0.68982	0.21622	3.190	0.00142	**
Duree_credit(36,Inf]	1.75035	0.37671	4.646	3.38e−06	***
Age(25,Inf]	−0.51624	0.23857	−2.164	0.03047	*
EpargnePas épargne ou > 500 euros	−1.12051	0.25616	−4.374	1.22e−05	***
GarantiesSans garant	1.34003	0.49197	2.724	0.00645	**
Autres_creditsCrédits extérieurs	0.57559	0.24826	2.318	0.02042	*
Objet_creditIntérieur	−0.01234	0.28661	−0.043	0.96566	
Objet_creditVoiture neuve	0.49118	0.31725	1.548	0.12157	
Objet_creditVoiture occasion	−0.86957	0.43685	−1.991	0.04653	*

Modèle logit sur WoE IV

- Avec les WoE, on a pu ajouter l'objet du crédit au modèle, dont tous les coefficients sont significativement ≠ 0 (y compris l'âge)

```
> logit <- glm(Cible~Comptes+Historique_credit+Duree_credit+Age+Epargne+Garanties+Autres_credits+Objet_credit,
data=train, family=binomial(link = "logit"))
> summary(logit)
Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)        -0.8646     0.1018  -8.493  < 2e-16 ***
Comptes            -0.8792     0.1306  -6.734 1.65e-11 ***
Historique_credit  -0.7876     0.1840  -4.280 1.87e-05 ***
Duree_credit       -1.0797     0.2279  -4.738 2.16e-06 ***
Age                -0.7849     0.3504  -2.240  0.02508 *
Epargne            -1.0257     0.2467  -4.157 3.23e-05 ***
Garanties          -1.9931     0.8010  -2.488  0.01284 *
Autres_credits     -0.8231     0.4061  -2.027  0.04271 *
Objet_credit       -0.8988     0.2983  -3.013  0.00258 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null deviance: 814.01  on 665  degrees of freedom
Residual deviance: 629.63  on 657  degrees of freedom
AIC: 647.63
> pred.logit <- predict(logit, newdata=valid, type="response")
> auc(valid$Cible, pred.logit, quiet=TRUE) # Area under the curve: 0.7657
Area under the curve: 0.7801
```

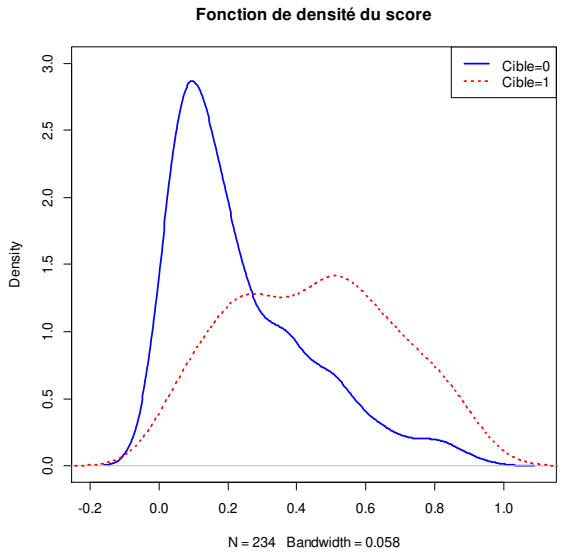
- On atteint une AUC = 0,780 plus élevée que celle des modèles précédents

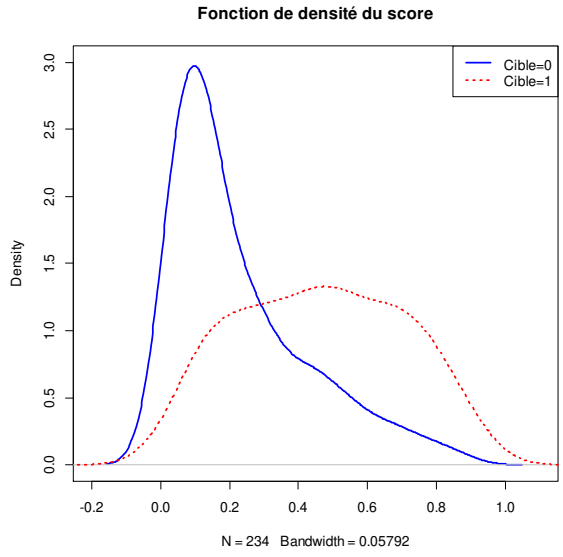
Fonction de densité du score

- Fonctions de densité du score sur les bons et mauvais dossiers

```
> plot(density(pred.logit[valid$Cible==0]), main="Fonction de densité du score",
col="blue", xlim = c(-0.2,1.1), ylim = c(0,3),lwd=2)
> lines(density(pred.logit[valid$Cible==1]), col="red", lty=3, lwd=2)
> legend("topright",c("Cible=0", "Cible=1"), lty=c(1,3), col=c("blue","red"), lwd=2)
```

- À gauche : modèle sur variables initiales – à droite : modèle sur WoE

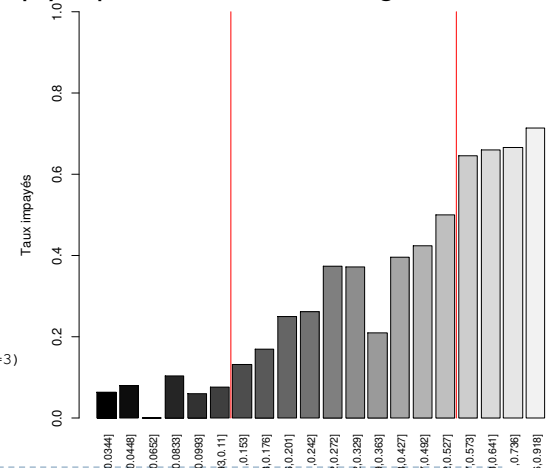




Seuils de score I

- On peut découper la note de score en tranches
- On constitue généralement deux ou trois tranches de score :
  - une tranche moins risquée, dans laquelle il suffit d'effectuer quelques vérifications indispensables (dans les fichiers de la Banque de France par exemple) et de demander au client les pièces minimales obligatoires
  - une tranche intermédiaire, dans laquelle il faut examiner attentivement le dossier et effectuer une analyse standard de risque
  - une tranche plus risquée, dans laquelle la demande est, sinon rejetée, du moins transmise à l'échelon hiérarchique supérieur pour un examen approfondi du dossier
- Nous appliquons le modèle à l'ensemble des données, nous découpons la note de score en vingtiles (tranches de 5 %), puis nous calculons et affichons les taux d'impayés pour chacun de ces vingtiles

```
> pred.logit <- predict(logit, newdata=credit2, type="response")
> q <- quantile(pred.logit, seq(0, 1, by=0.05))
> qscore <- cut(pred.logit, q)
> tab <- table(qscore, credit2$Cible)
> ti <- prop.table(tab,1)[,2]
> old <- par(no.readonly = TRUE)
> par(mar = c(7, 4, 2, 0))
> barplot(as.numeric(ti),col=gray(0:length(ti)/length(ti)),
+ names.arg=names(ti), ylab='Taux impayés', ylim=c(0,1),cex.names = 0.8, las=3)
> abline(v=c(7.3,19.3),col="red")
> par(old)
```



Seuils de score II

- Nous pouvons discerner deux seuils assez naturels à 0,11 et 0,527, que nous utilisons comme limites des tranches du score, et nous obtenons :
  - une tranche à faible risque, qui regroupe 29,6 % des dossiers et a un taux d'impayés de 6 %
  - une tranche à risque moyen, qui regroupe 50,2 % des dossiers et a un taux d'impayés de 29 %
  - une tranche à fort risque, qui regroupe 20,2 % des dossiers et a un taux d'impayés de 67 %
- Nous voyons que le score est très discriminant, puisque le taux d'impayés de la tranche à fort risque est 11 fois plus élevé que celui de la tranche à faible risque

```
> zscore <- recode(pred.logit, "lo:0.11='Faible'; 0.11:0.527='Moyen';
0.527:hi='Fort'")

> tab <- table(zscore,credit2$Cible)

> cbind(prop.table(tab,1), addmargins(tab,2))

                0          1   0   1 Sum

Faible 0.9391892 0.06081081 278  18 296

Fort   0.3316832 0.66831683  67 135 202

Moyen  0.7071713 0.29282869 355 147 502
```

Grille de score

- On peut transformer un modèle logistique en une grille de score avec pour chaque modalité un nombre de points ≥ 0
  - d'autant plus élevé que la modalité correspond à un profil plus risqué
  - et normalisé pour que chaque dossier ait un nombre total de points compris entre 0 et 100
- La transcription d'un modèle sous forme de « grille de score » est courante en *credit scoring*
- On parle de *scorecard*
- Avec des variables qualitatives ou discrétisées, et un coefficient par modalité, il suffit de :
  - substituer le logit à la probabilité $\frac{e^{logit}}{1+e^{logit}}$ comme valeur du score
  - puis normaliser le logit en sorte qu'il soit compris entre 0 et 100 (ou 1000 pour limiter l'effet des arrondis)
- Dans cette normalisation du logit, les coefficients de la régression logistique sont remplacés par de nouveaux coefficients, appelés « nombres de points », associés chacun à une modalité
- Le nombre de points est parfaitement corrélé au score logistique en termes de rangs (corrélation de Spearman = 1) et son pouvoir discriminant est exactement le même, puisque le classement des individus est conservé par la fonction croissante $\frac{e^x}{1+e^x}$
- Donc l'aire sous la courbe ROC de la grille de score est égale à celle du score logistique

Calcul de la grille de score

- Pour chaque variable qualitative ou discrète X_j, on note c_{jk} le coefficient du modèle associé à la k^e modalité, et a_j et b_j ses coefficients minimum et maximum dans la régression logistique :
  - $a_j = \min_k c_{jk}$ et $b_j = \max_k c_{jk}$
- On calcule ensuite le poids total sur l'ensemble des variables : $pt = \sum_j (b_j - a_j)$
- À chaque modalité k de X_j est associé un nombre de points égal à : $100 \times \frac{c_{jk} - a_j}{pt}$
- En présence de variables X_l quantitatives, la grille est plus complexe à établir : le poids total doit prendre en compte le coefficient β_l, le minimum m_l et le maximum M_l de chaque variable X_l :
  - $pt = \sum_j (b_j - a_j) + \sum_l |\beta_l| (M_l - m_l)$
- À une valeur x correspond alors un nombre de points : $100 \times \frac{|\beta_l|(x - m_l)}{pt}$
- La complexité vient de ce que le nombre de points n'est alors pas directement donné mais doit être calculé pour chaque valeur x

Calcul de la grille de score avec R I

- La fonction `glm` ne fournit pas directement un data frame contenant une colonne pour les variables (facteurs), une pour leurs modalités (niveaux) et une pour leurs coefficients $\Rightarrow$ nous devons le créer
- La composante `xlevels` de l'objet résultat est une liste contenant ses niveaux pour chaque facteur du modèle

```
> logit$xlevels
$Comptes
[1] "CC [0-200 euros[" "CC < 0 euros"     "CC > 200 euros"   "Pas de compte"
$Historique_credit
[1] "Crédits en impayé"                     "Crédits passés sans retard"
"Pas de crédits ou en cours sans retard"
$Duree_credit
[1] "(0,15]"   "(15,36]"  "(36,Inf]"
$Age
[1] "(0,25]"   "(25,Inf]"
$Epargne
[1] "< 500 euros"               "Pas épargne ou > 500 euros"
$Garanties
[1] "Avec garant" "Sans garant"
$Autres_credits
[1] "Aucun crédit extérieur" "Crédits extérieurs"
```

Calcul de la grille de score avec R II

- Avec `unlist(logit$xlevels)`, on concatène dans un vecteur les différents objets de la liste, les facteurs, dont avec `names(unlist(logit$xlevels))` on ne conserve que les noms

```
> names(unlist(logit$xlevels))
 [1] "Comptes1"           "Comptes2"           "Comptes3"
 [4] "Comptes4"           "Historique_credit1" "Historique_credit2"
 [7] "Historique_credit3" "Duree_credit1"      "Duree_credit2"
[10] "Duree_credit3"      "Age1"               "Age2"
[13] "Epargne1"           "Epargne2"           "Garanties1"
[16] "Garanties2"         "Autres_credits1"    "Autres_credits2"
```

- On supprime ensuite avec `gsub` les chiffres suffixant les noms de variables

```
> VARIABLE=c("", gsub("[0-9]", "", names(unlist(logit$xlevels))))
> VARIABLE
 [1] ""                  "Comptes"           "Comptes"           "Comptes"           "Comptes"
 [6] "Historique_credit" "Historique_credit" "Historique_credit" "Duree_credit"      "Duree_credit"
[11] "Duree_credit"      "Age"               "Age"               "Epargne"           "Epargne"
[16] "Garanties"         "Garanties"         "Autres_credits"    "Autres_credits"
```

- Extraction des modalités

```
> MODALITE=c("",unlist(logit$xlevels))
> MODALITE
                                          Comptes1                      Comptes2
                          ""       "CC [0-200 euros["             "CC < 0 euros"
                    Comptes3                Comptes4          Historique_credit1
…
```

Calcul de la grille de score avec R III

- On concatène ensuite variables et modalités dans une expression NOMVAR

```
> names = data.frame(VARIABLE, MODALITE, NOMVAR=c("(Intercept)",
paste(VARIABLE,MODALITE,sep="")[-1]))
```

- On récupère ensuite les coefficients du modèle dans un data frame regression que l'on va fusionner avec le précédent, pour avoir pour chaque coefficient la variable et la modalité qui lui correspondent
  - à noter que la fonction `as.numeric` ne récupère que les valeurs numériques

```
> (regression=data.frame(NOMVAR=names(coefficients(logit)),
COEF=as.numeric(coefficients(logit))))
```

	NOMVAR	COEF
1	(Intercept)	-0.5797981
2	ComptesCC < 0 euros	0.1584002
3	ComptesCC > 200 euros	-1.3485320
4	ComptesPas de compte	-1.5136834
5	Historique_creditCrédits passés sans retard	-1.5873249
6	Historique_creditPas de crédits ou en cours sans retard	-0.8970474
7	Duree_credit(15,36]	0.5766696
8	Duree_credit(36,Inf]	1.5115586
9	Age(25,Inf]	-0.4507556
10	EpargnePas épargne ou > 500 euros	-1.0999919
11	GarantiesSans garant	1.3194937
12	Autres_creditsCrédits extérieurs	0.5587985

Calcul de la grille de score avec R IV

- La fusion des deux data frames se fait sur la colonne qu'elles ont en commun : NOMVAR
  - pour un autre choix, il faudrait spécifier la clé par un `by`
- On élimine NOMVAR du résultat par [-1] (NOMVAR est la première colonne) et on remplace les valeurs manquantes dans les coefficients (ceux des modalités de référence) par 0
- Comme ces coefficients des modalités de référence n'apparaissent pas dans le data frame regression, il faut spécifier l'option `all.x=TRUE` pour que les lignes correspondantes soient ajoutées lors du merge (elles sont présentes dans le data frame names)

```
> param = merge(names, regression, all.x=TRUE)[-1]
> param$COEF[is.na(param$COEF)] <- 0
> param
           VARIABLE                                MODALITE       COEF
1                                                             -0.5797981
2               Age                                   (0,25]  0.0000000
3               Age                                 (25,Inf] -0.4507556
4    Autres_credits                  Aucun crédit extérieur  0.0000000
5    Autres_credits                      Crédits extérieurs  0.5587985
6           Comptes                      CC [0-200 euros[  0.0000000
7           Comptes                           CC < 0 euros  0.1584002
8           Comptes                         CC > 200 euros -1.3485320
9           Comptes                          Pas de compte -1.5136834
10     Duree_credit                                  (0,15]  0.0000000
11     Duree_credit                                 (15,36]  0.5766696
12     Duree_credit                                 (36,Inf]  1.5115586
13          Epargne                           < 500 euros  0.0000000
14          Epargne              Pas épargne ou > 500 euros -1.0999919
15        Garanties                            Avec garant  0.0000000
16        Garanties                            Sans garant  1.3194937
17 Historique_credit                      Crédits en impayé  0.0000000
18 Historique_credit              Crédits passés sans retard -1.5873249
19 Historique_credit Pas de crédits ou en cours sans retard -0.8970474
```

Calcul de la grille de score avec R V

- On crée ensuite un data frame qui contient le coefficient minimum de chaque variable, un autre qui contient le coefficient maximum, puis on les fusionne

```
> mini=aggregate(data.frame(min = param$COEF), by = list(VARIABLE = param$VARIABLE), min)
> maxi=aggregate(data.frame(max = param$COEF), by = list(VARIABLE = param$VARIABLE), max)
> total = merge(mini,maxi)
> total$diff = total$max - total$min
> total
           VARIABLE        min        max      diff
1                  -0.5797981 -0.5797981 0.0000000
2               Age -0.4507556  0.0000000 0.4507556
3    Autres_credits  0.0000000  0.5587985 0.5587985
4           Comptes -1.5136834  0.1584002 1.6720836
5      Duree_credit  0.0000000  1.5115586 1.5115586
6           Epargne -1.0999919  0.0000000 1.0999919
7         Garanties  0.0000000  1.3194937 1.3194937
8 Historique_credit -1.5873249  0.0000000 1.5873249
```

- Puis on calcule le poids total qui servira à normaliser le poids de chaque modalité

```
> poids_total = sum(total$diff)
> poids_total
[1] 8.200007
```

Calcul de la grille de score avec R VI

- On fusionne ensuite sur la colonne VARIABLE le data frame param avec le data frame mini pour ajouter en face de chaque modalité le coefficient minimum de la variable de cette modalité
- On calcule la différence entre le coefficient de chaque modalité et le coefficient minimum de la variable, puis le poids de la modalité

```
> grille = merge(param, mini, all.x=TRUE)
> grille$delta = grille$COEF - grille$min
> grille$POIDS = round((100*grille$delta) / poids_total)
```

- On affiche enfin la grille, après suppression de la ligne sans nom de variable, qui correspond à la constante

```
> grille[order(grille$VARIABLE,grille$MODALITE)[which(VARIABLE!="")], c("VARIABLE","MODALITE","POIDS")]
            VARIABLE                                MODALITE POIDS
2                Age                                   (0,25]     5
3                Age                                  (25,Inf]     0
4     Autres_credits                  Aucun crédit extérieur     0
5     Autres_credits                      Crédits extérieurs     7
6            Comptes                        CC [0-200 euros[    18
7            Comptes                             CC < 0 euros    20
8            Comptes                           CC > 200 euros     2
9            Comptes                            Pas de compte     0
10     Duree_credit                                    (0,15]     0
11     Duree_credit                                   (15,36]     7
12     Duree_credit                                  (36,Inf]    18
13          Epargne                                < 500 euros    13
14          Epargne             Pas épargne ou > 500 euros      0
15        Garanties                                 Avec garant     0
16        Garanties                                 Sans garant    16
17 Historique_credit                       Crédits en impayé    19
18 Historique_credit             Crédits passés sans retard     0
19 Historique_credit Pas de crédits ou en cours sans retard     8
```

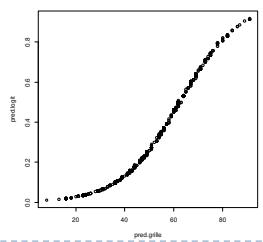
Application de la grille de score I

- On transforme la grille de score en une chaîne de caractères que la fonction `parse` transforme en une ligne de code R, que la fonction `eval` évalue, calculant ainsi le nombre de points de chaque dossier

```
> card <- function(base,i){
+ noquote(paste0("((",base,"$",grille[i,"VARIABLE"],"=='",grille[i,"MODALITE"],"')*",grille[i,"POIDS"],")"))
+ }
> card("credit2",2)
[1] ((credit2$Age=='(0,25]')*5)
> scorecard <- rbind(sapply(2:nrow(grille), function(x) card("credit2",x)))
> scorecard <- noquote(paste(scorecard, collapse = '+'))
> scorecard
[1] ((credit2$Age=='(0,25]')*5)+((credit2$Age=='(25,Inf]')*0)+((credit2$Autres_credits=='Aucun crédit
extérieur')*0)+ ((credit2$Autres_credits=='Crédits extérieurs')*7)+ …
> pred.grille <- eval(parse(text=scorecard))
```

- On vérifie que l'on obtient des nombres de points, quasiment parfaitement corrélés à la note du score logit, la liaison étant donnée par une fonction en S

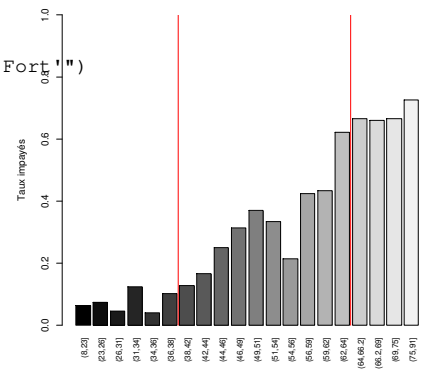
```
> head(pred.grille,10)
 [1] 36 78 29 59 64 31 31 62 24 54
> cor(pred.grille,pred.logit,method="spearman")
[1] 0.9995231
> plot(pred.grille,pred.logit)
```



Application de la grille de score II

- On peut déterminer des tranches du nombre de points, comme pour la note de score, et les seuils précédents de 0,11 et 0,527 de la note de score correspondent ici aux seuils de 37 et 63 points

```
> plot(pred.grille,pred.logit)
> q <- quantile(pred.grille, seq(0, 1, by=0.05))
> qscore <- cut(pred.grille, q)
> tab <- table(qscore, credit2$Cible)
> ti <- prop.table(tab,1)[,2] # affichage % en ligne
> old <- par(no.readonly = TRUE)
> par(mar = c(7, 4, 2, 0))
> barplot(as.numeric(ti), col=gray(0:length(ti)/length(ti)),
+ names.arg=names(ti), ylab='Taux impayés', ylim=c(0,1), cex.names = 0.8, las=3)
> abline(v=c(7.3,19.3),col="red")
> par(old)
> zscore <- recode(pred.grille, "lo:37='Faible'; 37:63='Moyen'; 63:hi='Fort'")
> tab <- table(zscore,credit2$Cible)
> cbind(prop.table(tab,1), addmargins(tab,2))
               0          1   0   1 Sum
Faible 0.9391892 0.06081081 278  18 296
Fort   0.3316832 0.66831683  67 135 202
Moyen  0.7071713 0.29282869 355 147 502
```



Sélection pas à pas I

- Le package `stats` contient une fonction `step` qui réalise la sélection pas à pas des variables en cherchant à minimiser un critère pénalisé du type AIC ou BIC
- La fonction `step` est associée à la fonction `glm` et peut donc être utilisée pour n'importe quel modèle linéaire généralisé, comme par exemple un modèle de régression linéaire ou de régression logistique
- La fonction `step` peut être utilisée pour la sélection ascendante (forward), descendante (backward), ou stepwise (both) (sélection ascendante avec suppression possible d'une variable déjà entrée dans le modèle)
- Pour une sélection pas à pas ascendante, nous partons du modèle minimal

```
> logit <- glm(Cible~1, data=train, family=binomial(link = "logit"))
> summary(logit)

Call:
glm(formula = Cible ~ 1, family = binomial(link = "logit"), data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.8451  -0.8451  -0.8451   1.5511   1.5511

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.84587    0.08453  -10.01   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 814.01  on 665  degrees of freedom
Residual deviance: 814.01  on 665  degrees of freedom
AIC: 816.01
```

Sélection pas à pas II

- La fonction `step` permet de spécifier le modèle initial, la direction ascendante de la sélection, l'affichage (`trace`) des étapes de la sélection, la plage (`scope`) de modèles examinée, et le multiple *k* du nombre *d* de degrés de liberté dans la pénalisation ajoutée à la déviance –2 log(vraisemblance)
- Cette pénalisation vaut 2.*d* pour le critère d'Akaïké (AIC) et log(effectif).*d* pour le critère de Schwartz (BIC), et on peut spécifier *k* = 2 ou *k* = log(nombre de lignes du data frame), ou encore d'autres valeurs
- Ici on a choisi le BIC (mais dans ses sorties la fonction `step` écrit AIC dans tous les cas)

```
> predicteurs <- -grep('(Cle|Cible)', names(train))
> formule <- as.formula(paste("y ~ ", paste(names(train[,predicteurs]), collapse="+")))
> formule
y ~ Comptes + Duree_credit + Historique_credit + Objet_credit +
    Montant_credit + Epargne + Anciennete_emploi + Taux_effort +
    Situation_familiale + Garanties + Anciennete_domicile + Biens +
    Age + Autres_credits + Statut_domicile + Nb_credits + Type_emploi +
    Nb_pers_charge + Telephone
```

- On teste les modèles compris entre le modèle initial (= constante) et le modèle contenant tous les prédicteurs
- Le BIC du modèle réduit à la constante vaut 820,51, et introduire la variable « Comptes » dans le modèle fait baisser le BIC à 743,90

```
> selection <- step(logit, direction="forward", trace=TRUE, k = log(nrow(train)),
scope=list(upper=formule))
Start:  AIC=820.51
Cible ~ 1

                     Df Deviance    AIC
+ Comptes             3   717.90 743.90
+ Historique_credit   2   769.66 789.16
…
+ Taux_effort         3   812.35 838.36
+ Type_emploi         3   812.88 838.88
```

Sélection pas à pas III

- Le BIC du modèle contenant la constante et la variable « comptes » vaut 743,90 et c'est ensuite l'historique de crédit qui fait le plus baisser le BIC, à 728,98
- Les lignes au-dessus de `<none>` correspondent aux variables susceptibles de procurer un BIC inférieur à celui du modèle atteint à cette étape, et donc d'être intégrées au modèle
- Quant à la déviance = 717,90 elle ne peut bien sûr que baisser à chaque ajout d'une variable dans le modèle

```
Step:  AIC=743.9
Cible ~ Comptes

                      Df Deviance    AIC
+ Historique_credit    2   689.97 728.98
+ Duree_credit         2   694.31 733.31
+ Epargne              1   701.77 734.28
+ Montant_credit       1   705.04 737.55
+ Autres_credits       1   710.36 742.86
<none>                     717.90 743.90
+ Garanties            1   711.41 743.92
+ Biens                2   707.53 746.53
+ Age                  1   715.54 748.04
+ Statut_domicile      1   715.76 748.27
+ Nb_pers_charge       1   716.54 749.04
+ Situation_familiale  2   710.66 749.66
+ Telephone            1   717.89 750.40
+ Anciennete_emploi    2   711.56 750.56
+ Objet_credit         3   708.80 754.31
+ Anciennete_domicile  3   708.82 754.33
+ Nb_credits           3   713.65 759.16
+ Taux_effort          3   714.41 759.92
+ Type_emploi          3   717.49 763.00
…
```

Sélection pas à pas IV

- On s'arrête quand le BIC ne diminue
- Quand le modèle contient déjà les variables « comptes », « historique de crédit », « épargne » et « durée de crédit », le BIC vaut 710,7
- On voit que la variable « Garanties » peut faire baisser le BIC à 709,80 mais que l'ajout d'une variable supplémentaire (« autres crédits ») fait remonter le BIC à 713,15

```
Step:  AIC=710.7
Cible ~ Comptes + Historique_credit + Epargne + Duree_credit

                      Df Deviance    AIC
+ Garanties            1   644.79 709.80
<none>                     652.19 710.70
+ Autres_credits       1   648.14 713.15
+ Age                  1   648.94 713.96
+ Montant_credit       1   650.93 715.94
+ Situation_familiale  2   644.47 715.99
+ Nb_pers_charge       1   651.12 716.13
+ Statut_domicile      1   651.90 716.91
+ Telephone            1   652.01 717.03
+ Anciennete_emploi    2   647.55 719.06
+ Objet_credit         3   641.20 719.21
+ Biens                2   649.10 720.62
+ Anciennete_domicile  3   643.53 721.54
+ Taux_effort          3   648.36 726.38
+ Nb_credits           3   649.07 727.09
+ Type_emploi          3   652.15 730.16
```

Sélection pas à pas V

- On a ajouté les garanties au modèle et la fonction s'arrête car il n'est plus possible de diminuer le BIC

```
Step:  AIC=709.8
Cible ~ Comptes + Historique_credit + Epargne + Duree_credit + Garanties

                     Df Deviance    AIC
<none>                    644.79 709.80
+ Autres_credits      1   639.87 711.39
+ Age                 1   641.30 712.81
+ Nb_pers_charge      1   643.59 715.10
+ Situation_familiale 2   637.17 715.18
+ Montant_credit      1   643.84 715.35
+ Telephone           1   644.28 715.79
+ Statut_domicile     1   644.60 716.12
+ Objet_credit        3   633.81 718.32
+ Anciennete_emploi   2   640.61 718.63
+ Anciennete_domicile 3   634.93 719.45
+ Biens               2   643.21 721.23
+ Taux_effort         3   641.24 725.76
+ Nb_credits          3   641.43 725.95
+ Type_emploi         3   644.72 729.23
```

- Le critère BIC nous a conduit à sélectionner pas à pas 5 variables
- Le critère AIC impose une pénalisation moins sévère et nous conduit à sélectionner 12 variables
- On peut aussi procéder à une sélection descendante :
  - le modèle initial contient toutes les variables
  - on n'a pas besoin de spécifier `scope=list(upper=formule))` puisque le modèle maximal est le modèle initial, mais on peut spécifier un modèle minimal (option `scope=list(lower=~Comptes+...)` pour contraindre certaines variables à apparaître dans le modèle

```
> logit <- glm(Cible~., data=train, family=binomial(link = "logit"))
> selection <- step(logit, direction="backward", trace=TRUE, k = log(nrow(train)))
```

Test de toutes les combinaisons de variables

- R permet de tester facilement tout un ensemble de combinaisons de variables (les calculs peuvent être très longs !)
- Ensemble des combinaisons de variables explicatives (ici : combinaisons de 1 à 5 variables)

```
> pos <- 20 # position variable à expliquer
> combis <- unlist(sapply(1:5, function(x) apply(combn(names(train)[-pos], x), 2, paste, collapse = " + ")))
```

- Calcul de l'ensemble des modèles logit s'appuyant sur ces combinaisons de variables (peut être très long)

```
> lst.model <- lapply(combis, function(x) glm(as.formula(paste("Cible ~", x)), data = train, family=binomial))
```

- Nombre de modèles

```
> length(lst.model)
[1] 16663
```

- Application de la liste des modèles à un échantillon de validation

```
> lst.pred <- lapply(lst.model, function(x) {predict(x, newdata=valid)})
```

- Calcul de l'aire sous la courbe ROC de ces modèles

```
> AUC <- sapply(lst.pred, function(x) {auc(valid$Cible,x,quiet=TRUE)})
> resultats <- data.frame(combis, AUC)
```

- Affichage des aires sous la courbe ROC les plus élevées et des combinaisons correspondantes

```
> head(resultats[order(resultats[,2], decreasing=T),])
```

	combis	AUC
6339	Comptes + Objet_credit + Montant_credit + Biens + Age	0.7713889
5723	Comptes + Historique_credit + Objet_credit + Montant_credit + Age	0.7713675
5725	Comptes + Historique_credit + Objet_credit + Montant_credit + Statut_domicile	0.7693376
5722	Comptes + Historique_credit + Objet_credit + Montant_credit + Biens	0.7691453
5036	Comptes + Duree_credit + Historique_credit + Objet_credit + Montant_credit	0.7689103
6360	Comptes + Objet_credit + Montant_credit + Statut_domicile + Telephone	0.7685684

Régression clusterwise

Le clustering de modèles

- Régression clusterwise : méthode de recherche simultanée des classes et des modèles de chaque classe
- C'est un modèle des moindres carrés ordinaires dans lequel on cherche à minimiser la somme des carrés des résidus suivante : $\sum_{i=1}^{n} \sum_{k=1}^{K} 1_k(i)(y_i - (\alpha_k + \beta_k x_i))^2$
  - où $1_k(i)$ est la fonction indicatrice de la k^e classe
- On peut y parvenir par application de l'algorithme suivant :
  - étape 1 : à partir d'une partition initiale, on estime séparément k modèles de régression
  - étape 2 : chaque observation est affectée à la classe et au modèle minimisant le carré du résidu
  - étape 3 : une fois toutes les observations reclassées, on a une nouvelle partition et on revient à l'étape 2
- Il peut arriver que l'on obtienne des classes de taille < nombre de variables $\Rightarrow$ recourir à une régression pénalisée du type ridge
- Package R : `flexmix`

Exemple de régression clusterwise I

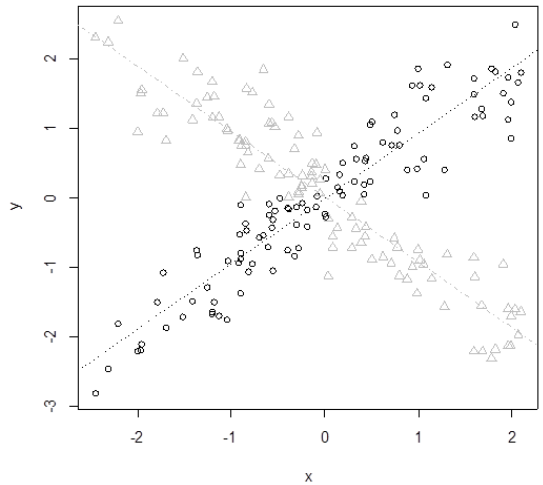
```
> set.seed(2)
> x1 <- rnorm(100)
> set.seed(3)
> y1 <- x1 + rnorm(100, sd=0.5)
> set.seed(5)
> y2 <- - x1 + rnorm(100, sd=0.5)
> x <- c(x1,x1)
> y <- c(y1,y2)
> modele <- lm(y ~ x)
> summary(modele)

Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-2.81971 -0.86755  0.02945  0.94792  2.54697

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.010835   0.083549   0.130    0.897
x           0.005453   0.072350   0.075    0.940

Residual standard error: 1.181 on 198 degrees of freedom
Multiple R-squared:  2.869e-05, Adjusted R-squared:  -0.005022
F-statistic: 0.00568 on 1 and 198 DF,  p-value: 0.94
```



- Nous avons obtenu une droite de régression de pente quasiment nulle

Exemple de régression clusterwise II

- Nous ajustons une régression clusterwise en spécifiant l'existence de 2 classes

```
> library(flexmix)
> clw <- flexmix(y ~ x, k=2)
> summary(clw)

Call:
flexmix(formula = y ~ x, k = 2)

       prior size post>0 ratio
Comp.1 0.501  101    156 0.647
Comp.2 0.499   99    159 0.623

'log Lik.' -218.5194 (df=7)
AIC: 451.0388   BIC: 474.127
```

- Les deux classes détectées sont presque de même taille
- Nous retrouvons les pentes correspondant aux deux classes que nous avons créées : l'une proche de +1 et l'autre proche de –1

```
> summary(refit(clw))
$Comp.1
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.010514   0.048656 -0.2161   0.8289
x            0.941763   0.038669 24.3547   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

$Comp.2
             Estimate Std. Error  z value Pr(>|z|)
(Intercept)  0.013144   0.050655   0.2595   0.7953
x           -0.941999   0.040461 -23.2815   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Lien avec la régression logistique

- Une approche courante consiste à se placer dans le cadre des modèles de mélanges et à rechercher l'estimateur du maximum de vraisemblance
- Les coefficients β de la régression logistique s'obtiennent à partir des fonctions de densité conditionnelles f_β en maximisant la log-vraisemblance calculée sur des observations $(x^1,y^1), (x^2,y^2), \ldots, (x^n,y^n)$: $\sum_{i=1}^{n} \log\left(f_\beta\left(y^i/x^i\right)\right)$
- Dans un modèle clusterwise, la fonction de densité conditionnelle est une somme de plusieurs fonctions $\sum_{k=1}^{K} \pi_k f_{\beta_k}$, où π_k est la probabilité *a priori* de la classe *k*, et il faut estimer les paramètres π_k et β_k en maximisant la log-vraisemblance : $\sum_{i=1}^{n} \log\left(\sum_{k=1}^{K} \pi_k f_{\beta_k}\left(y^i/x^i\right)\right)$
- La recherche directe des paramètres $(\pi_1, \ldots, \pi_K, \beta_1, \ldots, \beta_K)$ maximisant cette fonction devient vite très complexe, même en présence de deux classes seulement, car on ne sait pas à quelle classe appartient chaque individu
- L'estimation de ces paramètres est toutefois rendue possible par l'algorithme EM (expectation-maximization) de Dempster *et al.*

Les arbres de décision

Principe des arbres de décision

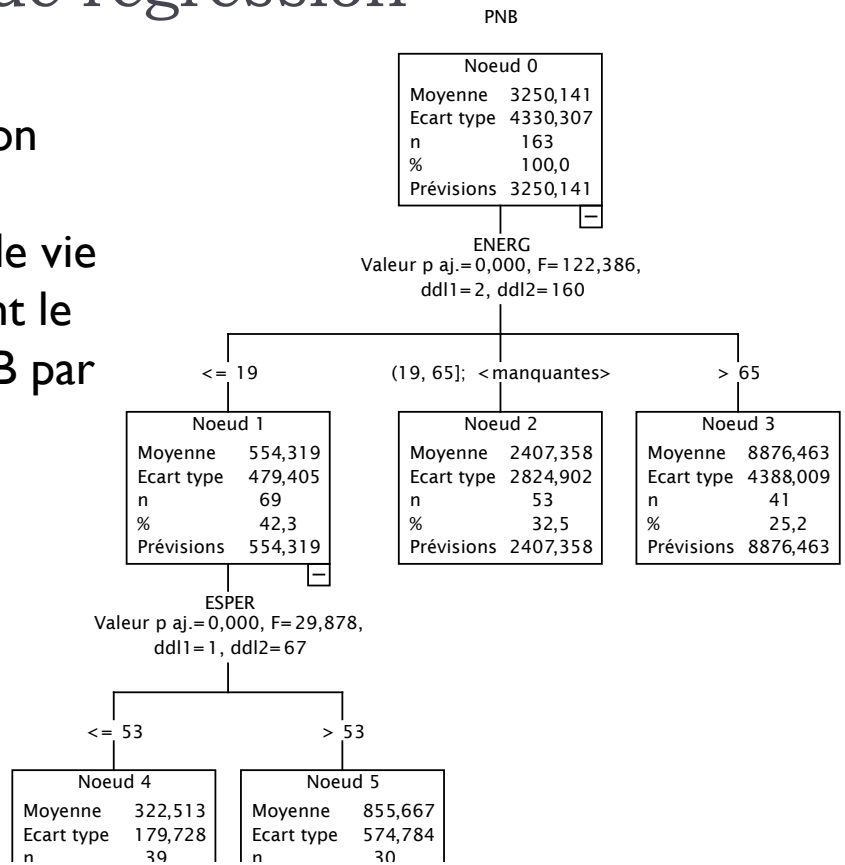
- Les arbres de décision sont une méthode de classification hiérarchique descendante supervisée
- Elle est appliquée itérativement pour scinder chaque ensemble d'individus en deux (ou plus de deux) sous-ensembles, selon un critère le plus lié possible à la variable à expliquer
- L'ensemble initial est constitué de l'ensemble des individus (« racine »), scindé en plusieurs sous-ensembles (« nœuds ») qui seront chacun scindés (en « nœuds-fils »), etc.
- Les nœuds terminaux sont les « feuilles » et chaque chemin entre la racine et une feuille est un ensemble de conditions = une règle
- Dans un arbre de régression, la moyenne de la variable à expliquer doit être la plus différente possible d'un nœud à l'autre
- Dans un arbre de classement, la probabilité d'appartenance à l'une des classes doit être la plus différente possible d'un nœud à l'autre

Arbres de régression

- La variable à expliquer Y est continue
- Les arbres de régression sont une alternative à la régression linéaire multiple
- Principe :
  - la variable Y doit avoir une variance plus faible dans les nœuds fils (modalités de la variable explicative créées par la scission) que dans le nœud père, la baisse de variance étant la plus importante possible
  - $\Leftrightarrow$ la variable Y doit avoir une moyenne la plus distincte possible d'un nœud fils à un autre
- On teste l'hypothèse nulle que la moyenne de la variable à expliquer est la même dans chaque nœud fils. On choisit la variable explicative, et la scission, pour laquelle la probabilité p associée au test de Fisher-Snedecor est minimale, et on scinde le nœud si p est inférieure au seuil fixé permettant de rejeter l'hypothèse nulle

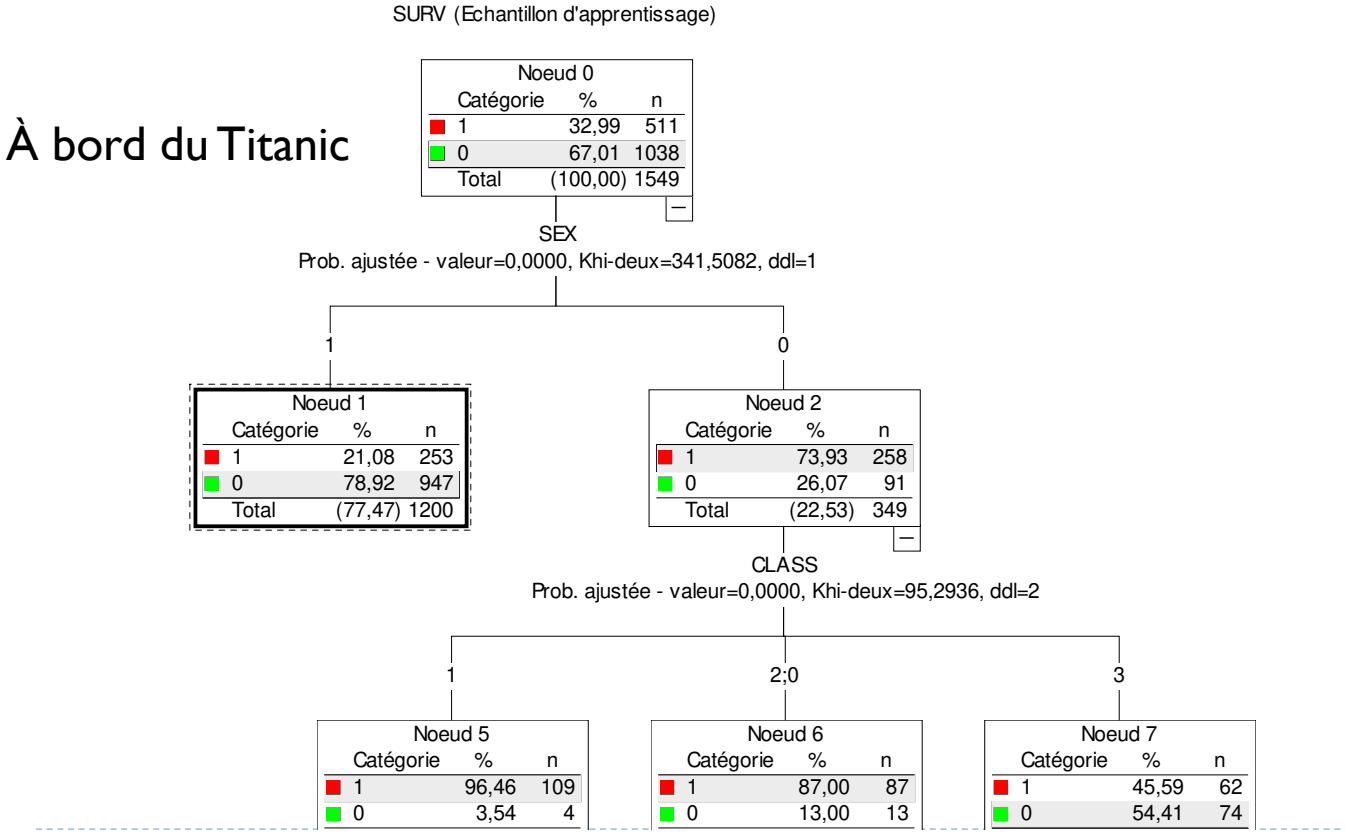
Arbre de régression

Ce sont la consommation d'énergie et l'espérance de vie qui expliquent le mieux le PNB par habitant (ou l'inverse !)



Arbre de classement

À bord du Titanic



Classement ou régression par arbre

- Détermination pour chaque nœud de l'ensemble des divisions possibles (variables et conditions sur ces variables)
- Application d'un critère (C1) permettant de sélectionner la meilleure division possible du nœud
- Application d'un (ou plusieurs) critère(s) d'arrêt (C2) des divisions
- Application d'une règle d'affectation de chaque nœud terminal
  - à une classe de Y si Y est qualitative
  - à une valeur de Y si Y est quantitative
  - $\Rightarrow$ prédiction de Y pour chaque individu
- Estimation de l'erreur, du coût associé à l'arbre

Critères d'arrêt

- Le critère d'arrêt (C2) peut combiner plusieurs règles :
  - la profondeur de l'arbre a atteint une limite fixée
  - ou le nombre de feuilles a atteint un maximum fixé
  - ou l'effectif de chaque nœud est inférieur à une valeur fixée en deçà de laquelle on estime qu'il ne faut plus diviser un nœud
  - ou la division ultérieure de tout nœud provoquerait la naissance d'un fils d'effectif inférieur à une valeur fixée
  - ou la qualité de l'arbre est suffisante (et en tout cas si tous les individus de chaque feuille sont dans la même classe, en classement)
  - ou la qualité de l'arbre n'augmente plus de façon significative
- Le critère de qualité dépend du type d'arbre
  - exemple : la pureté dans l'arbre CART

Critères de choix de scission d'un nœud

- Le critère du χ^2
  - lorsque les variables explicatives sont qualitatives
  - utilisé dans les arbres CHAID et QUEST
- Les critères basés sur une fonction d'impureté
  - pour tous types de variables explicatives
  - l'indice de Gini est utilisé dans l'arbre CART
  - l'indice Twoing est utilisé dans l'arbre CART lorsque la variable à expliquer a ≥ 3 modalités
  - l'indice Twoing ordonné est utilisé quand la variable à expliquer est ordinale
  - l'entropie est utilisée dans les arbres CART, C4.5 et C5.0
  - plus les classes sont uniformément distribuées dans un nœud, plus la fonction d'impureté est élevée ; plus le nœud est pur, plus elle est basse

Arbres de classement

- La variable à expliquer Y est une variable nominale (ou ordinale) à k modalités, définissant k groupes d'individus G_i d'effectifs n_i
- On peut considérer que les probabilités a priori $P(G_i)$ sont :
  - toutes égales
  - ou égales aux fréquences empiriques n_i/n
  - ou fixées *a priori* par une connaissance experte
- Pour tout nœud t, soient $P(t/G_i)$ la probabilité d'être dans le nœud t si on est dans la i^e classe, $P(t)$ la probabilité du nœud et $P(G_i/t)$ la probabilité d'être dans G_i si on est dans t
- On a : $P(G_i/t)P(t) = P(t/G_i)P(G_i)$ (formule de Bayes)
- Dans le 2^e cas ci-dessus ($P(G_i) = n_i/n$), on a :
  - $P(G_i/t) = (n_i/n)P(t/G_i)/P(t) = (n_i/n)(n_i(t)/n_i)/(n(t)/n) = n_i(t)/n(t)$
  - proportion d'individus du nœud t qui sont dans G_i = notée plus simplement f_i
  - c'est le cas le plus fréquent, dans lequel nous nous placerons pour la suite

Prédiction dans un arbre de classement

- Affectation d'un nœud terminal t au groupe G_i pour lequel $P(G_i/t)$ est maximale
  - quand $P(G_i/t) = n_i(t)/n(t) \Rightarrow$ affectation du nœud t au groupe de plus grand effectif dans t
- Affectation d'un individu :
  - d'abord à un nœud t, en fonction des valeurs de ses variables explicatives
  - ensuite au G_i auquel est affecté le nœud t
  - avec une prédiction $P(G_i/t)$ (la même pour tous les individus du nœud)
- On peut introduire des coûts de mauvais classement : C_{ij} = coût d'affectation d'un individu à G_i alors qu'il est dans G_j
  - on a $C_{ii} = 0$ et le cas le plus fréquent est celui où $C_{ij} = 1$ pour tous $i \neq j$
- Le coût d'affectation du nœud t à G_i est $\sum_{j=1}^{k} C_{ij} P\left(G_j/t\right)$
- On affecte alors t au G_i de coût minimal et non au $P(G_i/t)$ maximal

Fonction d'impureté

- Une fonction d'impureté est une fonction :
  - positive, concave, symétrique (elle ne dépend que de la proportion dans laquelle est présente chaque classe dans le nœud, et ne varie pas si l'on permute les classes par rapport aux proportions)
  - minimale lorsque tous les individus qui composent le nœud appartiennent à la même classe
  - maximale lorsque toutes les classes sont présentes dans la même proportion dans le nœud
- Principales fonctions d'impureté :
  - indice de diversité de Gini
  - entropie
  - plus petite proportion d'une classe dans un nœud (fonction non dérivable ☹)

Indice de diversité de Gini

- Indice de Gini d'un nœud = $1 - \Sigma_i f_i^2 = \Sigma_{i \neq j} f_i f_j = \Sigma_{i \neq j} P(G_i/t)P(G_j/t)$
  - où les $f_i = n_i(t)/n(t)$, i = 1 à p, sont les fréquences relatives dans le nœud des p classes à prédire (variable à expliquer), égales à $P(G_i/t)$ dans le cas le plus fréquent
  - = probabilité que 2 individus, choisis aléatoirement dans un nœud, appartiennent à 2 classes différentes
- Plus les classes sont uniformément distribuées dans un nœud, plus l'indice de Gini est élevé ; plus le nœud est pur, plus l'indice de Gini est bas
  - avec 2 classes, l'indice va de 0 (nœud pur) à 0,5 (mélange maximal) – avec 3 classes, l'indice va de 0 à 2/3
- Chaque scission en p nœuds fils (d'effectifs n_1, n_2 … n_p) doit provoquer la plus grande hausse de la pureté, donc la plus grande baisse de l'indice de Gini. Autrement dit, il faut minimiser : $\text{Gini(fils)} = \sum_{i=1}^{p} \frac{n_i}{n} \text{Gini}(i^e \text{ fils})$
- Par concavité, on a toujours Gini(père) ≥ Gini(fils)

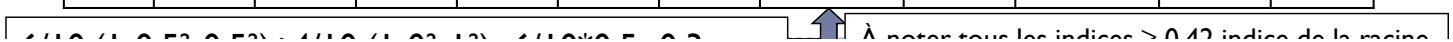
Mécanisme de scission des nœuds avec Gini

(exemple : catalogue avec prix des articles et achat O/N)

Article	Prix	Achat
1	125	N
2	100	N
3	70	N
4	120	N
5	95	O
6	60	N
7	220	N
8	85	O
9	75	N
10	90	O

Mécanisme de scission des nœuds avec Gini

Achat	N		N		N		O		O		O		N		N		N		N		N	
Prix	60		70		75		85		90		95		100		120		125		220			
Seuil	55		65		72,5		80		87,5		92,5		97,5		110		122,5		172,5		230	
	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>
O	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
N	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0,420		0,400		0,375		0,343		0,417		0,400		**0,300**		0,343		0,375		0,400		0,420	

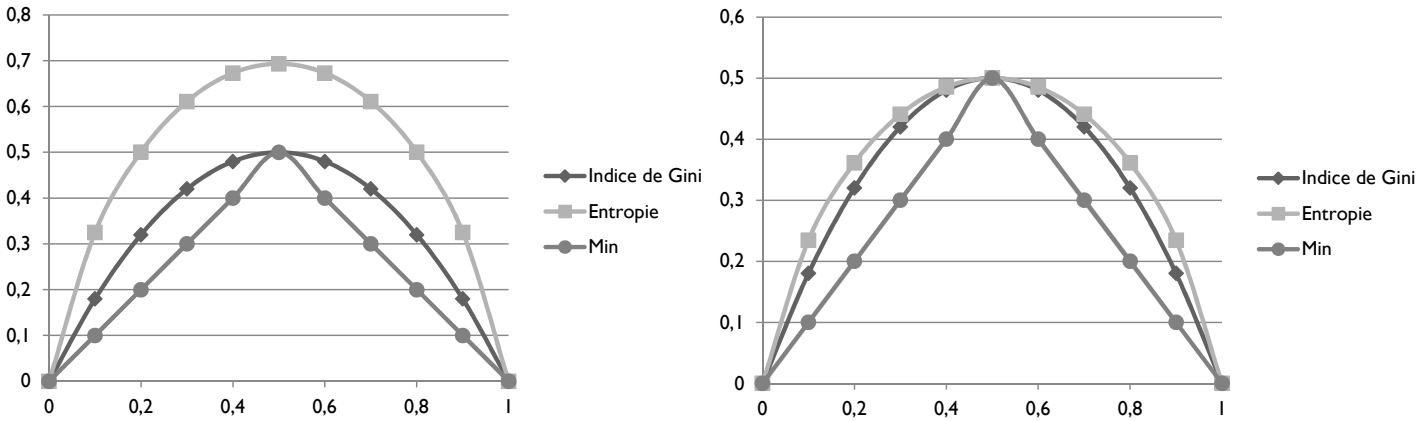


Entropie de Shannon

- Entropie (ou « information » ou « déviance ») d'un nœud t
  $= - \Sigma_i f_i.\log(f_i) = - \Sigma_i P(G_i/t)\log(P(G_i/t))$
  - où les f_i, $i = 1$ à p, sont les fréquences relatives dans le nœud des p classes à prédire
- Plus les classes sont uniformément distribuées dans un nœud, plus l'entropie est élevée ; plus le nœud est pur, plus l'entropie est basse
  - elle vaut 0 lorsque le nœud ne contient qu'une seule classe
- C'est une fonction d'impureté comme l'indice de Gini et elle produit des scissions peu différentes de l'indice de Gini
- Comme précédemment, il faut minimiser l'entropie dans les nœuds-fils

Comparaison des fonctions d'impureté

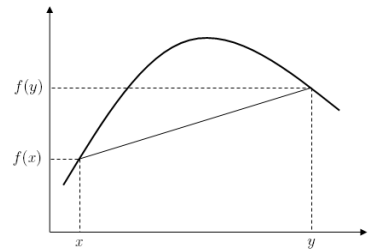
- L'entropie est toujours plus grande que l'indice de Gini, et leurs valeurs maximales sont 1-(1/*k*) pour l'indice de Gini et –log(–1/*k*) pour l'entropie (pour *k* classes)
- Elles valent par exemple 0,5 et environ 0,69 pour $k = 2$
- On peut mettre à l'échelle ces deux courbes $f(f_1)$ en multipliant l'entropie par [1-(1/*k*)] / [–log(–1/*k*)]



Utilité de la concavité

- La concavité d'une fonction $f \Rightarrow$ si A et B sont deux points du graphe G(f) de la fonction, le segment [A,B] est entièrement situé sous G(f)
- $f(tx + (1-t)y) \geq tf(x) + (1-t)f(y), t \in [0,1]$
- La concavité assure la diminution systématique de l'impureté lorsque l'on scinde un nœud père en plusieurs nœuds-fils
- Autrement dit, on a toujours (sachant que $n = n_g + n_d$) :

$$\text{Gini(père)} \geq \sum_{i=1}^{p} \frac{n_i}{n} Gini(i^e\ fils)$$



$$Gini\left(\frac{n_g}{n} f_1^g + \frac{n_d}{n} f_1^d\right) \geq \frac{n_g}{n} Gini\left(f_1^g\right) + \frac{n_d}{n} Gini\left(f_1^d\right)$$

- La diminution systématique de l'impureté garantit la convergence du processus de scission successive des nœuds
- Il est toutefois prévisible que, comme pour tout modèle trop complexe, un arbre de profondeur maximale ne puisse que conduire au sur-apprentissage $\Rightarrow$ nécessité d'une stratégie de limitation de la complexité (voir plus loin)

Choix d'une fonction d'impureté 1/2

- Pour l'optimisation, préférer une fonction dérivable comme l'indice de Gini ou l'entropie
- Ils décroissent plus fortement que le taux d'erreur quand les nœuds sont plus purs
- Exemple de 1000 individus dont 500 appartiennent à chaque classe, avec un arbre T1 répartissant ces individus en (400,100) dans un nœud et (100,400) dans un autre, et un arbre T2 répartissant ces individus en (315,25) dans un nœud et (185,475) dans un autre
- T2 est préférable car il conduit à un premier nœud très pur
- On peut vérifier que l'on a :
  - indice de Gini = 0,32, entropie = 0,50 et taux d'erreur = 0,20 pour T1
  - indice de Gini = 0,31, entropie = 0,48 et taux d'erreur = 0,21 pour T2
- Le critère du taux d'erreur conduirait donc à retenir T1 alors que l'arbre T2 que nous préférons est retenu par l'indice de Gini et l'entropie

Choix d'une fonction d'impureté 2/2

- Discussions sur le choix de l'indice de Gini ou l'entropie
- Breiman, Friedman, Olshen et Stone préfèrent l'indice de Gini
  - tout en reconnaissant que le choix d'un critère ou d'un autre influe peu sur l'arbre produit, et moins en tout cas que la stratégie d'élagage
- Mais l'indice de Gini :
  - est simple et donc rapide à calculer (un simple produit $2f_1f_2$ dans le cas de deux classes)
  - permet d'intégrer naturellement des coûts C_{ij} de mauvais classement dans la formule $\sum_{i \neq j} C_{ij} f_i f_j$
  - limite plus que l'entropie l'apparition de « end cut splits » (scissions déséquilibrées où l'un des nœuds fils est très pur mais très petit)
    - car l'entropie décroît plus fortement que l'indice de Gini quand les nœuds sont plus purs (voir précédemment)

Cas des arbres de régression

- Impureté = variance(Y) dans le nœud : l'objectif de baisse de l'impureté se traduit en objectif de variance la plus faible possible dans les nœuds-fils (variance intra-classe)
- Réduction de l'impureté = variance totale (nœud-père) – variance intra-classe (intra nœuds-fils) = variance inter-classe (inter-nœuds fils)
- Prédiction : les observations d'un nœud terminal se voient affecter comme valeur de Y la moyenne dans le nœud
- Coût (erreur) d'un nœud t = variance de Y dans t = impureté(t)
- On peut calculer la somme des coûts dans les nœuds (pondérés par les effectifs), et la normaliser par la variance de Y dans la racine

Les principaux arbres de décision 1/2

- CHAID (CHi-Square Automation Interaction Detection)
  - utilise le test du χ^2 pour déterminer la variable la plus significative pour chaque scission et le découpage de ses modalités
  - adapté à l'étude des variables explicatives discrètes (les variables explicatives continues peuvent être discrétisées)
- QUEST
  - variable à expliquer nominale
  - utilise le test du χ^2 comme CHAID mais pour produire des scissions binaires comme CART

Les principaux arbres de décision 2/2

- CART (Classification and Regression Tree)
  - cherche à maximiser la pureté des nœuds
  - adapté à l'étude de tout type de variables explicatives
  - dispositif d'élagage
  - utilisation de variables équidivisantes pour gérer les valeurs manquantes
  - généralement binaire (chaque nœud a au plus deux nœud-fils)
- C5.0
  - cherche à maximiser le gain d'information réalisé en affectant chaque individu à une branche de l'arbre
  - adapté à l'étude de tout type de variables explicatives
  - transformation de l'arbre en règles qui permet une simplification par suppression des règles redondantes

Arbre CHAID – Algorithme 1/2

- Cet arbre est de conception plus ancienne (principe : 1975, Hartigan ; algorithme : 1980, Kass)
- Il traite directement les variables explicatives discrètes ou qualitatives, et discrétise automatiquement les variables explicatives continues
- La variable à expliquer est qualitative à k modalités
- Utilise plusieurs fois la statistique du χ^2 :
  1. On construit pour chaque prédicteur X_i, le tableau de contingence X_i x Y et on effectue les étapes 2 et 3
  2. On sélectionne la paire de modalités de X_i dont le sous-tableau (2 x k) a le plus petit χ^2. Si ce χ^2 n'est pas significatif (p-value > à un seuil fixé), on fusionne les 2 modalités et on répète cette étape jusqu'à ce que toutes les paires de modalités (simples ou composées) aient un χ^2 significatif ou jusqu'à ce qu'il n'y ait plus qu'une seule modalité. Si X_i est ordinale ou quantitative, seules sont considérées les paires de modalités adjacentes

Arbre CHAID – Algorithme 2/2

3. Éventuellement, pour chaque modalité composée d'au moins 3 modalités originales, on détermine la division binaire au χ^2 le plus grand. S'il est significatif, on effectue cette division (même si le χ^2 de (A,B) x Y n'était pas significatif, ni celui de (A$\cup$B,C) x Y, le χ^2 de (A,B$\cup$C) x Y pourrait être significatif)
4. On calcule la significativité (p-value associée au χ^2) de chaque prédicteur X_i dont les modalités ont été précédemment regroupées et on retient le prédicteur le plus significatif. Si la p-value du χ^2 est inférieure au seuil choisi, on peut diviser le nœud en autant de nœuds-fils qu'il y a de modalités après regroupement. Si la p-value dépasse le seuil spécifié, le nœud n'est pas divisé

- Ajustement de Bonferroni
  - Lors du calcul de la significativité de tous les prédicteurs (étape 4), on peut multiplier la valeur de la probabilité du χ^2 par le coefficient de Bonferroni, qui est le nombre de possibilités de regrouper les m modalités d'un prédicteur en g groupes ($1 \leq g \leq m$)
  - Ce calcul permet d'éviter la surévaluation de la significativité des variables à modalités multiples

Arbre CHAID – Caractéristiques

- CHAID traite l'ensemble des valeurs manquantes comme une seule catégorie (qu'il fusionne éventuellement avec une autre)
  - pas d'utilisation de variables équidivisantes
- Il n'est pas binaire et produit des arbres souvent plus larges que profonds
  - utile pour la discrétisation de variables continues
- Le nombre de nœuds fils dépend des seuils fixés pour le test du χ^2
- Pas de dispositif d'élagage : la construction de l'arbre s'achève dès que les critères d'arrêt sont rencontrés
- R : package *CHAID* sur R-Forge

```
> install.packages("CHAID", repos="http://R-Forge.R-project.org")
> library("CHAID")
```

Discrétisation avec CHAID 1/4

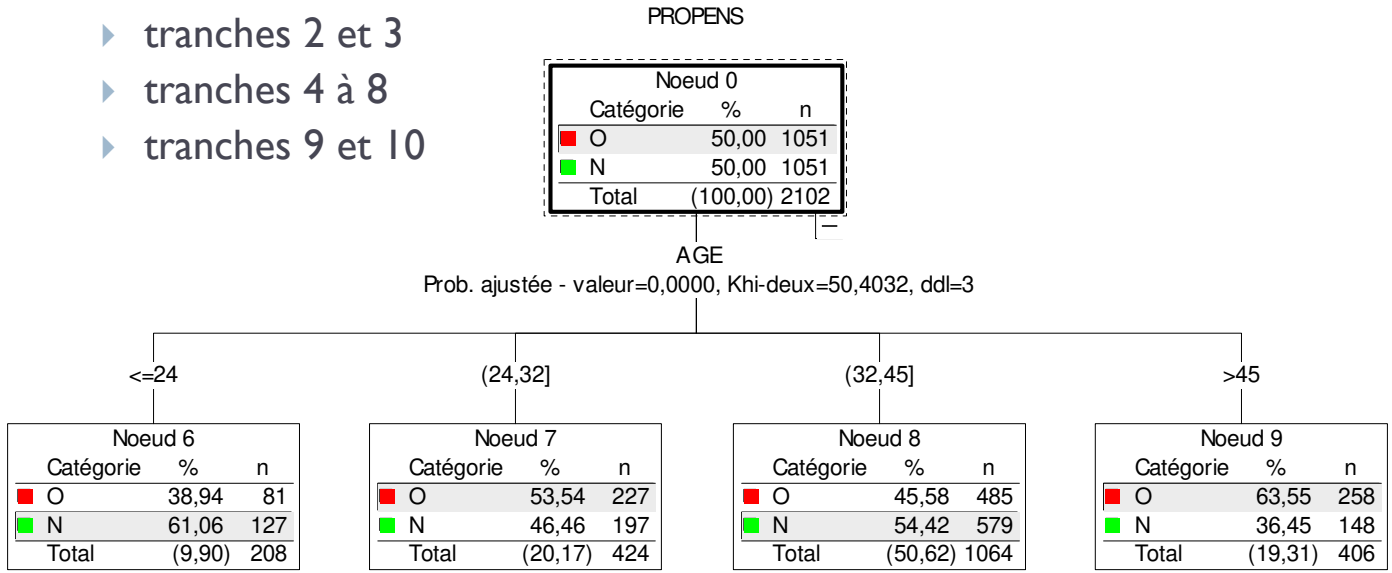
- Supposons que nous voulions prédire une variable cible à l'aide de certaines variables explicatives, dont l'âge, et que nous voulions découper l'âge en classes pour ces raisons :
  - prise en compte de la non-monotonie ou non-linéarité de la réponse en fonction de l'âge
  - suppression du problème des extrêmes
  - modèle plus robuste
- Nous allons découper l'âge en 10 tranches (ou plus, si le nombre d'individus est grand) et regarder le % d'individus dans la cible pour chaque classe d'âge

Discrétisation avec CHAID 2/4

			cible		
			non	oui	Total
tranche d'âge	18-25 ans	Effectif	127	81	208
		% dans tranche d'âge	61,1%	38,9%	100,0%
	25-29 ans	Effectif	104	126	230
		% dans tranche d'âge	45,2%	54,8%	100,0%
	29-32 ans	Effectif	93	101	194
		% dans tranche d'âge	47,9%	52,1%	100,0%
	32-35 ans	Effectif	113	99	212
		% dans tranche d'âge	53,3%	46,7%	100,0%
	35-38 ans	Effectif	93	94	187
		% dans tranche d'âge	49,7%	50,3%	100,0%
	38-40 ans	Effectif	149	123	272
		% dans tranche d'âge	54,8%	45,2%	100,0%
	40-42 ans	Effectif	108	72	180
		% dans tranche d'âge	60,0%	40,0%	100,0%
	42-45 ans	Effectif	116	97	213
		% dans tranche d'âge	54,5%	45,5%	100,0%
	45-51 ans	Effectif	77	113	190
		% dans tranche d'âge	40,5%	59,5%	100,0%
	> 51 ans	Effectif	71	145	216
		% dans tranche d'âge	32,9%	67,1%	100,0%
Total		Effectif	1051	1051	2102
		% dans tranche d'âge	50,0%	50,0%	100,0%

Discrétisation avec CHAID 3/4

- Nous voyons que certaines classes sont proches du point du vue du % dans la cible :
  - tranches 2 et 3
  - tranches 4 à 8
  - tranches 9 et 10

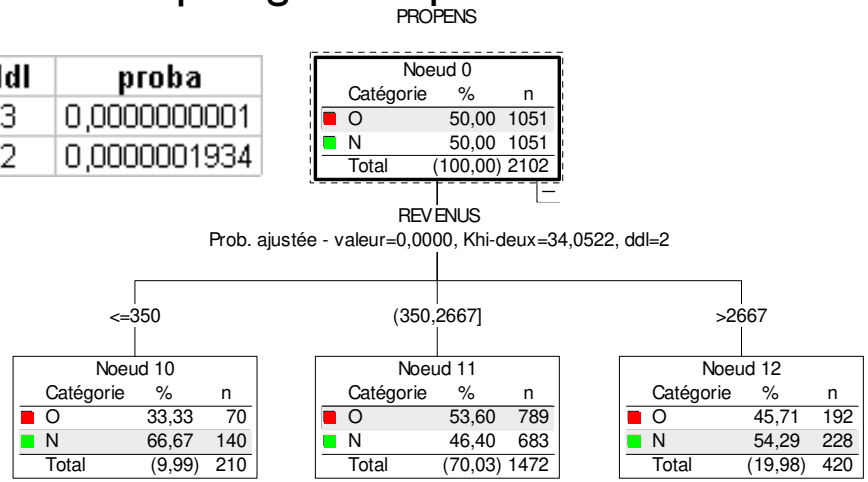


- CHAID a fait automatiquement ce que nous avons fait manuellement

Discrétisation avec CHAID 4/4

- Pour la scission de la racine de l'arbre, la variable AGE est retenue devant la variable REVENUS car la probabilité associée au χ^2 des REVENUS est plus grande que celle associée à l'AGE

variable	chi-2	ddl	proba
âge	50,40	3	0,0000000001
revenus	34,05	2	0,0000001934



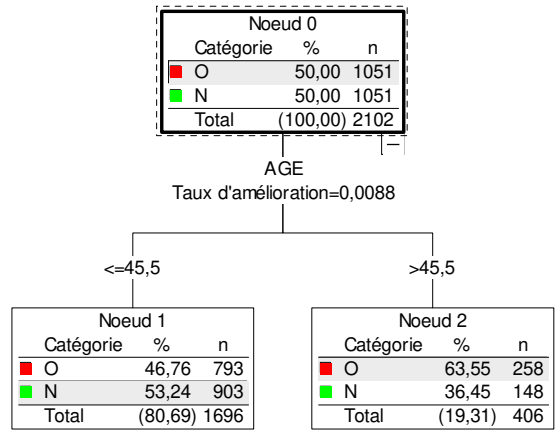
- Si le nombre de degrés de liberté n'est pas le même pour deux variables, il faut comparer les p-values et non les χ^2

Arbre CART

- Décrit dans l'ouvrage classique de Breiman, Friedman, Olshen et Stone (1984) : *Classification and Regression Trees*
- Le critère de division est basé sur une fonction d'impureté
- Optimal : toutes les scissions possibles sont examinées
- Optimal : dispositif d'élagage performant
  - l'arbre maximum construit, l'algorithme en déduit une séquence de sous-arbres par élagages successifs, et retient celui qui optimise un certain critère
- Général : variable à expliquer quantitative ou qualitative
  - ⇒ CART sert à la régression comme au classement
- Général : CART permet la prise en compte de coûts C_{ij} de mauvaise affectation (d'un individu de la classe j dans la classe i) en les intégrant dans le calcul de l'indice de Gini $\sum_{i \neq j} C_{ij} f_i f_j$
- Dans sa version la plus courante, CART est binaire
  - pour segmenter moins rapidement les données
- Gère les valeurs manquantes en recourant aux variables équidivisantes (« surrogate variables »)
  - différent de CHAID
- R : packages `rpart`, `tree` et `rpartOrdinal` (Y ordinale)

Scission avec CART

- La scission de la racine se fait par l'âge, comme avec CHAID, mais l'arbre binaire est moins équilibré :



- On peut aussi pénaliser les scissions déséquilibrées
  - multiplier la baisse d'impureté par un coefficient $(p_G p_D)^\alpha$ dépendant de la proportion d'individus allant à gauche et de la proportion d'individus allant à droite, avec $\alpha \geq 0$ (0 pour la scission habituelle)
- Ces scissions déséquilibrées sont moins rares avec l'entropie
- CART est apte à détecter rapidement des profils très marqués

CART et complexité du choix (C1)

- Si une variable explicative qualitative X a un ensemble E de n valeurs possibles x_1, …, x_n, toute condition de séparation sur cette variable sera de la forme
  - $X \in E'$, où $E' \subset E - \{0\}$
- 2^{n-1} — 1 conditions de séparation possibles

- Pour une variable explicative continue X, la complexité est liée au tri des valeurs x_1, …, x_n de X, puisqu'une fois les variables dans l'ordre $x_1 \leq \ldots \leq x_n$, il suffit de trouver l'indice k tel que la condition
  - $X \leq$ moyenne (x_k , x_{k+1})

  soit la meilleure (selon le critère choisi, par exemple Gini)

Variables équiréductrices

- Variables assurant la réduction de l'impureté des nœuds la plus proche possible de celle de la variable retenue
- Variables « concurrentes » (« primary splits ») utilisées lors de la construction de la variable pour repérer d'éventuelles variables plus intéressantes que celles retenues en première approche
  - plus fiables
  - mieux acceptées par les experts métiers
  - moins coûteuses à collecter

Variables équidivisantes

- Variables répartissant les individus de la façon la plus proche possible de celle de la variable retenue avec sa scission
  - si le nœud est scindé par la condition $\{X < x\}$, on recherche les variables équidivisantes avec des arbres de profondeur 1 ayant la condition $\{X < x\}$ pour variable à expliquer (et sans coûts de mauvais classement)
  - soit n le nombre d'individus, n_1 le nombre d'individus bien classés par l'arbre précédent et n_0 le nombre d'individus bien classés par la règle majoritaire :
  - la concordance de la variable équidivisante est $(n_1\text{-}n_0)/(n\text{-}n_0)$
  - on ne s'intéresse qu'aux variables équidivisantes de concordance > 0
- Variables « de rechange » ou « suppléantes » (« surrogate splits ») utilisées en cas de valeur manquante pour un individu de la variable retenue

Variables équidivisantes : exemple

- Si la scission d'un nœud est produite par la condition {X < x}, et si une variable équidivisante est {Y < y}, avec un croisement

	Y < y	Y ≥ y
X < x	40	20
X ≥ x	4	80

- la concordance vaut (120-84)/(144-84) = 0,6
- C'est mieux que la règle de la majorité consistant à affecter tous les individus à la modalité {X ≥ x}
- On ne calcule pas la concordance 120/144 (0,833) car on veut mesurer le gain de concordance par rapport à la règle majoritaire
  - ainsi un bon classement de 72 individus sur 144 n'est pas affecté d'un taux de 0,5 mais de -0,2 (72-84/144-84) car il est inférieur à la règle majoritaire qui classe bien 84 individus

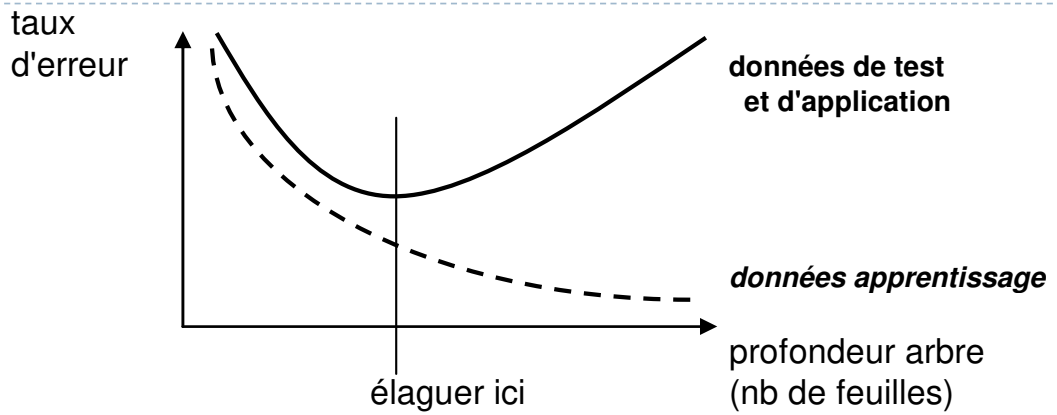
Traitement des valeurs manquantes

- Apprentissage :
  - un individu participe si sa variable à expliquer et au moins une variable explicative ont une valeur non manquante
  - l'indice de Gini de la scission $\{X < x\}$ d'un nœud est calculé sans prendre en compte les individus avec X manquant
  - on ajuste les n_k/n pour que leur somme = 1
- Application :
  - si X est manquant pour un individu, il :
    - n'est pas affecté à un nœud-fils et reste au niveau du père (paramètre `usesurrogate` = 0 de `rpart`)
    - est affecté à un nœud-fils à l'aide d'une variable équidivisante (paramètre `usesurrogate` = 1 ou 2 de `rpart`)
      - si aucune variable équidivisante, application de la règle majoritaire (`usesurrogate` = 2, valeur par défaut) ou on reste au niveau père (`usesurrogate` = 1)
  - on a toujours une prédiction ≠ NA car même si tous les prédicteurs impliqués dans des scissions sont manquants pour un individu, il est affecté à la racine avec les probabilités *a priori*
    - sauf si `na.action = na.omit` ou `na.action = na.fail`

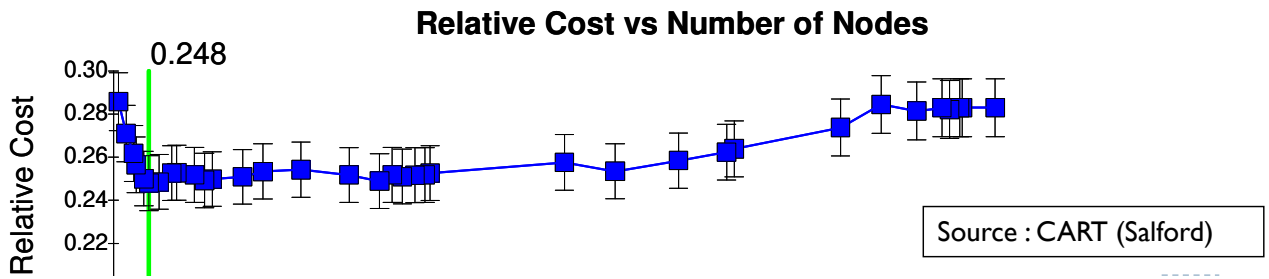
Importance d'une variable

- Une variable X1 peut ne jamais apparaître dans l'arbre, tout en étant très corrélée à une variable X2 qui apparaît souvent et la masque
  - $\Rightarrow$ il ne faut pas évaluer l'importance de X1 en ne considérant que les nœuds de l'arbre où X1 a été retenue pour la scission
- Importance d'une variable dans un nœud :
  - = baisse d'impureté de la scission produite par la variable si elle a été retenue pour la scission du nœud
  - = (baisse d'impureté x concordance avec la variable retenue), si la variable est équidivisante pour ce nœud (de concordance > 0)
  - = 0 sinon
- Importance d'une variable dans un arbre :
  - somme de l'importance de la variable dans l'ensemble des nœuds
  - normalisée pour que la somme des importances des variables = 100
  - `rpart` met l'importance d'une variable à 0 si elle est < 0,01

Élagage et sur-apprentissage



- Il est préférable d'élaguer un arbre pour éviter la remontée du taux d'erreur due au sur-apprentissage



Élagage dans CART

- Il faut trouver le bon compromis entre l'ajustement de l'arbre en apprentissage (biais) et sa capacité de généralisation (variance)
- Le Lasso traite ce problème par l'ajout d'une pénalisation dont l'augmentation progressive annule un à un les coefficients de la régression
- La stratégie inventée par les inventeurs de CART consiste à commencer par construire un arbre maximal avant de l'élaguer en coupant les branches jugées les moins généralisables
- Ce n'est pas l'indice de Gini ni l'entropie qui sont utilisés pour contrôler l'élagage, mais le taux d'erreur

Coût-complexité d'un arbre

- Complexité (ou taille) de l'arbre : nombre de feuilles (≠ racine)
- Coût-complexité C_{CP}(arbre) = *CR*(arbre) + *CP*.(# feuilles)
- *CR* : coût de l'arbre calculé par resubstitution (sur les données d'apprentissage)
  - ce coût est généralement un taux d'erreur dans un arbre de classement, et une erreur quadratique en régression (mais il peut intégrer des coûts de mauvaise affectation)
- Minimum de *CR* atteint pour l'arbre maximal T_{max} : attention au sur-apprentissage !
- Solution : augmenter progressivement le paramètre de complexité *CP* et chercher un sous-arbre $T_{CP} \subset T_{max}$ minimisant le coût-complexité C_{CP}
- Pour *CP* = 0, le sous-arbre minimisant le coût-complexité est l'arbre maximal, mais lorsque *CP* augmente, le nombre de feuilles représente un coût et la somme « *CR*(arbre) + *CP*.(# feuilles) » est minimisée pour un nombre plus petit de feuilles

Premier résultat sur l'élagage

- Quand *CP* augmente, le sous-arbre T_{CP} minimisant C_{CP} est de plus en plus petit $\Rightarrow$ on a une séquence descendante des tailles des arbres
  - à la complexité *CP* = 0 correspond l'arbre maximal
  - au-delà d'un certain seuil de *CP*, il faut que le nombre de feuilles soit nul, car le coût par resubstitution de l'arbre réduit à la racine est inférieur à celui de n'importe quel arbre augmenté de *CP*
- 1er résultat : si deux sous-arbres minimisent le coût-complexité pour une valeur *CP*, alors ils sont égaux ou l'un contient l'autre $\Rightarrow$ il existe un arbre T_{CP} de taille minimale qui minimise le coût-complexité C_{CP}
- Quand *CP* parcourt l'ensemble infini des valeurs réelles ≥ 0, l'ensemble des arbres T_{CP} est fini, et un arbre T_{CP} minimise aussi $C_{CP'}$ pour une valeur *CP'* > *CP* jusqu'à ce que *CP'* atteigne une valeur à laquelle corresponde un arbre $T_{CP'}$ strictement plus petit que T_{CP}
- On obtient ainsi une séquence croissante des *CP*, à laquelle correspond une séquence décroissante des T_{CP}

Second résultat sur l'élagage

- 2e résultat : ces arbres sont inclus les uns dans les autres, ce qui signifie que chacun est obtenu par élagage du précédent
- De plus, il existe une procédure simple et rapide pour déterminer la branche de T_{CP} à élaguer à chaque étape : il faut trouver la plus petite valeur $CP' > CP$ pour laquelle T_{CP} a un nœud {t} tel que $C_{CP'}(\{t\}) = C_{CP'}$(branche de T_{CP} ayant {t} pour racine)
  - cette valeur existe puisqu'une pénalisation suffisamment grande finira par donner à la branche issue d'un nœud {t} un coût-complexité plus grand qu'au nœud lui-même
- On peut donc élaguer cette branche puisque l'arbre élagué en {t} conduit au même coût-complexité pour CP'
- Ce processus est effectif et assez rapide à mettre en œuvre
  - bien plus que s'il fallait effectuer une recherche exhaustive des sous-arbres de T_{max}

Synthèse

- Par élagage successif, nous avons obtenu une séquence de sous-arbres emboîtés

$$T_{max} \supset T_1 \supset T_2 \supset \ldots \supset T_m = \{\text{racine}\}$$

- de complexités croissantes

$$0 = CP_0 < CP_1 < CP_2 < \ldots CP_m$$

- et dans laquelle chaque T_i est le sous-arbre dont le coût par resubstitution est le plus bas de tous les sous-arbres de T_{max} de même taille
- Cette séquence de sous-arbres étant obtenue, il reste bien sûr à déterminer le niveau optimal de complexité pour l'ajustement et la généralisation du modèle
- En pratique (voir le package `rpart`), on peut limiter la séquence explorée en fixant une valeur minimale $CP_0 > 0$ (c'est le paramètre `cp`)

Mise en œuvre avec R

- Package le plus utilisé : `rpart` (Recursive PARTitioning)
  - à la base de `ipred` pour le bagging et `ada` pour le boosting
  - ne gère pas d'échantillon de test (contrairement à `randomForest`)
  - autre package (moins rapide) : `tree`
- Pour le classement (`method = class`) et la régression (`method = anova`)
- Avec l'indice de Gini (`split = gini`) et l'entropie (`split = information`)
- Permet de spécifier des coûts, l'utilisation de variables équidivisantes, la pénalisation minimale de la complexité dans le processus d'élagage (`cp`) ou la profondeur maximale (`maxdepth`), l'effectif minimum d'un nœud pour être scindé (`minsplit`) et l'effectif minimum d'une feuille (`minbucket`, valeur par défaut = `minsplit/3`)
- Arbre de profondeur maximale :

```
> cart <- rpart(y ~ ., data = train, method="class",
parms=list(split="gini"), cp=0)
```

Mise en œuvre avec R (suite)

- Arbre avec contraintes sur l'élagage :

```
> cart <- rpart(y ~ . , data = train, method="class",
parms=list(split="gini"), control=list(minbucket=30,
minsplit=30*2, maxdepth=4))
```

- « Stump » (arbre à deux feuilles) :

```
> cart <- rpart(y ~ . , data = train, method="class",
parms=list(split="gini"), control=list(maxdepth=1,cp=-
1,minsplit=0))
```

- Évolution de l'élagage en fonction de la pénalisation

```
> printcp(cart)
```

- Affichage des informations précédentes + variables concurrentes et suppléantes, et importance des variables

```
> summary(cart, digits=3)
```

- Élagage à un niveau de pénalisation fixé :

```
> prunedcart4f <- prune(cart, cp=0.0328152)
```

  - à noter : pas d'application automatique dans `rpart` de la règle 1 SE ou 0 SE

Détail de l'arbre

```
> cart # commande équivalente à "print(cart)"
n= 666

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 666 200 0 (0.69969970 0.30030030)
    2) Comptes=CC > 200 euros,Pas de compte 300  35 0 (0.88333333 0.11666667)
      4) Montant_credit< 9504 291  30 0 (0.89690722 0.10309278)
        8) Objet_credit=Autres,Electroménager,Formation,Mobilier,Vidéo HIFI,Voiture
neuve,Voiture occasion 247  18 0 (0.92712551 0.07287449) *
        9) Objet_credit=Business,Etudes,Travaux 44  12 0 (0.72727273 0.27272727)
         18) Autres_credits=Aucun crédit 33   5 0 (0.84848485 0.15151515) *
         …
```

- Pour chaque nœud, nous avons :
  - son numéro *node*, par exemple « 2 »
  - sa règle de scission *split*, par exemple « Comptes=CC > 200 euros,Pas de compte »
  - le nombre *n* d'individus dans ce nœud, par exemple « 300 »
  - le nombre *loss* d'individus mal classés ($\notin$ classe majoritaire), ici « 35 »
  - la valeur prédite *yval*, qui est la classe majoritaire, ici « 0 »
  - la probabilité d'appartenance à chaque classe, ici (0,88333333 et 0,11666667)
  - le cas échéant, l'indication par un * à la fin de la ligne que le nœud est terminal

Variables concurrentes et suppléantes

- Affichage de la fonction `summary` (primary splits : variables concurrentes, surrogate splits : suppléantes)

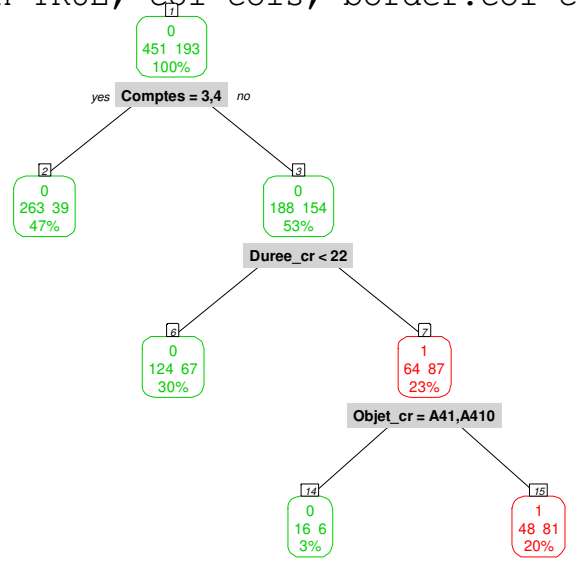
```
Node number 1: 666 observations,    complexity param=0.07
  predicted class=0  expected loss=0.3  P(node) =1
    class counts:   466   200
   probabilities: 0.700 0.300
  left son=2 (300 obs) right son=3 (366 obs)
  Primary splits:
      Comptes           splits as  RRLL,      improve=36.8, (0 missing)
      Epargne           splits as  RLLRL,     improve=14.0, (0 missing)
      Historique_credit splits as  LLRRL,     improve=14.0, (0 missing)
      Montant_credit    < 8020  to the left,  improve=12.5, (0 missing)
      Duree_credit      < 34.5  to the left,  improve=11.3, (0 missing)
  Surrogate splits:
      Epargne             splits as  RLLRL,      agree=0.629, adj=0.177, (0 split)
      Objet_credit        splits as  RRRRRRLLRL, agree=0.581, adj=0.070, (0 split)
      Historique_credit   splits as  RLRRR,      agree=0.578, adj=0.063, (0 split)
      Age                 < 40.5  to the right,  agree=0.575, adj=0.057, (0 split)
      Anciennete_domicile splits as  RRLR,       agree=0.572, adj=0.050, (0 split)

Node number 2: 300 observations,    complexity param=0.00667
  predicted class=0  expected loss=0.117  P(node) =0.45
    class counts:   265    35
   probabilities: 0.883 0.117
```

Affichage d'un arbre dans R avec `rpart.plot`

- Affichage amélioré par rapport à la fonction `plot` de `rpart` :

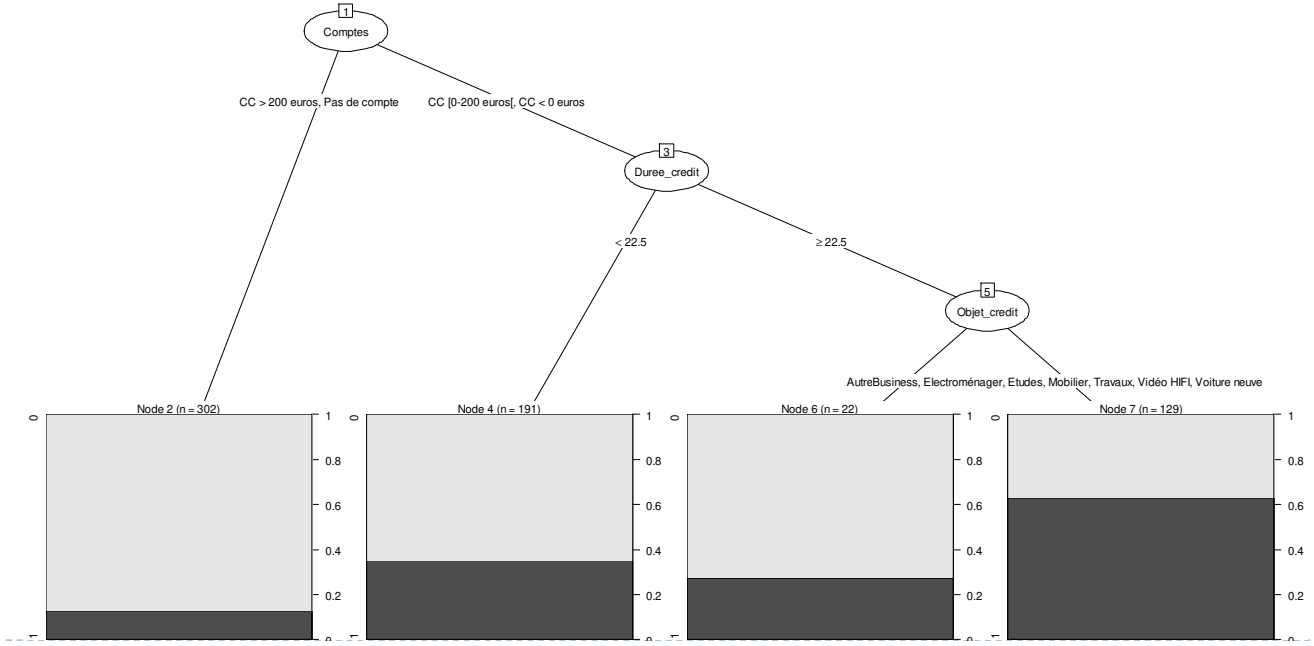
```
> library(rpart.plot)
> cols <- ifelse(prunedcart4f$frame$yval == 1,"green3","red")
> prp(prunedcart4f, type=2, extra=101,
split.box.col="lightgray", nn=TRUE, col=cols, border.col=cols)
```



Affichage d'un arbre dans R avec `partykit`

- Affichage amélioré par rapport à la fonction `plot` de `rpart` :

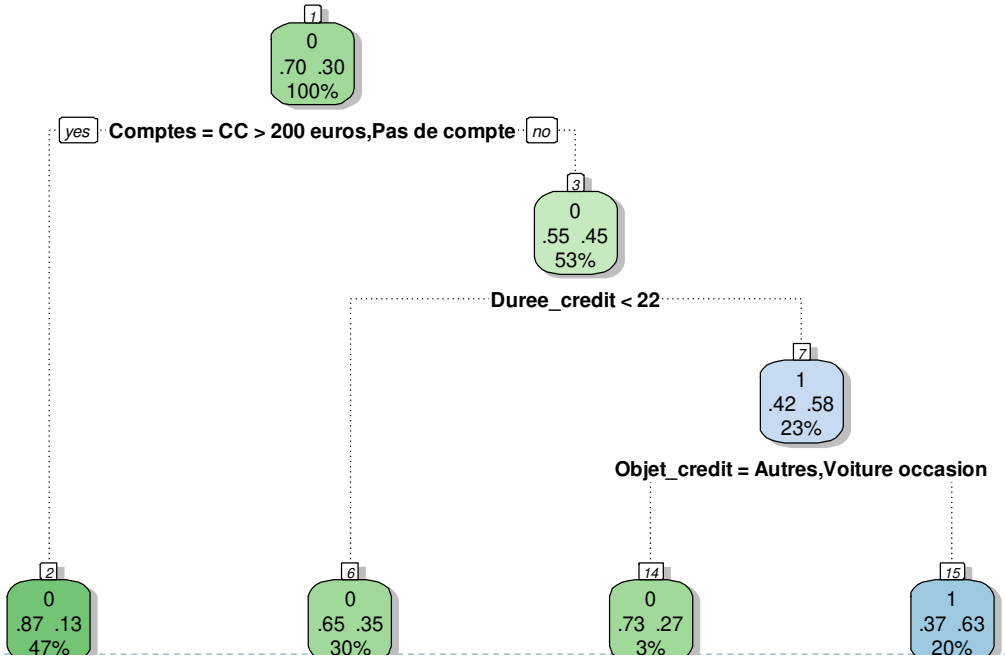
```
> library(partykit)
> plot(as.party(prunedcart4f))
```



Affichage d'un arbre dans R avec `rattle`

- Affichage amélioré par rapport à la fonction `plot` de `rpart` :

```
> library(rattle)
> fancyRpartPlot(prunedcart4f, sub=" ")
```



Évolution du paramètre de complexité

- Avec `cp` = 0, la fonction `printcp` de `rpart` affiche :

	CP	nsplit	rel error	xerror	xstd
1	0.0700000	0	1.000	1.000	0.059148
2	0.0650000	3	0.790	1.000	0.059148
3	0.0350000	4	0.725	0.810	0.055361
4	0.0183333	5	0.690	0.770	0.054404
5	0.0150000	8	0.635	0.840	0.056041
6	0.0133333	10	0.605	0.825	0.055705
7	0.0100000	14	0.550	0.805	0.055245
8	0.0075000	15	0.540	0.880	0.056897
9	0.0066667	17	0.525	0.880	0.056897
10	0.0033333	20	0.505	0.915	0.057601
11	0.0000000	23	0.495	0.960	0.058448

Explication du tableau

- Ce tableau affiche, en fonction de *CP* :
  - *rel error* : erreur calculée par resubstitution
  - *xerror* : erreur calculée par 10-validation croisée
  - *xstd* : écart-type de l'erreur par validation croisée = $\sqrt{\frac{xerror \times (1-xerror)}{taille\ échantillon}}$
  - *nsplit* : nombre de scissions (= nombre de feuilles – 1)
- Les taux d'erreur affichés sont relatifs : ils ont été mis à l'échelle afin de valoir 1 pour l'arbre réduit à la racine
- Or l'arbre réduit à la racine n'a pas un taux d'erreur égal à 1 mais à ε (en affectant tous les individus à la classe majoritaire) ⇒ chaque ligne du tableau doit être multipliée par ε pour obtenir l'erreur absolue

```
           CP nsplit rel error xerror     xstd
1  0.0700000      0     1.000  1.000 0.059148
…
11 0.0000000     23     0.495  0.960 0.058448
```

- Ici ε = 200/666 et l'erreur par resubstitution de l'arbre maximal = ε x 0,495 = 0,1486486
- On peut vérifier ce calcul :

```
> sum(predict(cart,type="class") != train$Cible)/nrow(train)
[1] 0.1486486
```

- Et calculer l'écart-type relatif de l'erreur par validation croisée :

```
> x <- 0.960*200/666 # xerror absolue
> sqrt((x*(1-x))/666)/(200/666)
[1] 0.05844841
```

Lecture du tableau

- Reprenons le tableau

```
           CP nsplit rel error xerror     xstd
1  0.0700000      0     1.000  1.000 0.059148
2  0.0650000      3     0.790  1.000 0.059148
...
10 0.0033333     20     0.505  0.915 0.057601
11 0.0000000     23     0.495  0.960 0.058448
```

- On a C_{CP} = (0,0033333 x 20) + 0,505 = (0,0033333 x 23) + 0,495 $\Rightarrow$ l'arbre élagué correspondant à CP = 0,0033333 est l'arbre à 21 feuilles (20 scissions), puisque son coût par resubstitution, un peu plus élevé que celui de l'arbre maximal à 24 feuilles, est compensé par une pénalisation moins grande (et qu'on retient le plus petit des arbres minimisant le coût-complexité C_{CP})
- On a (0,07 x 3) + 0,79 = 1 = (0,07 x 0) + 1 (coût-complexité de l'arbre réduit à la racine) $\Rightarrow$ l'arbre élagué correspondant à CP = 0,07 est la racine

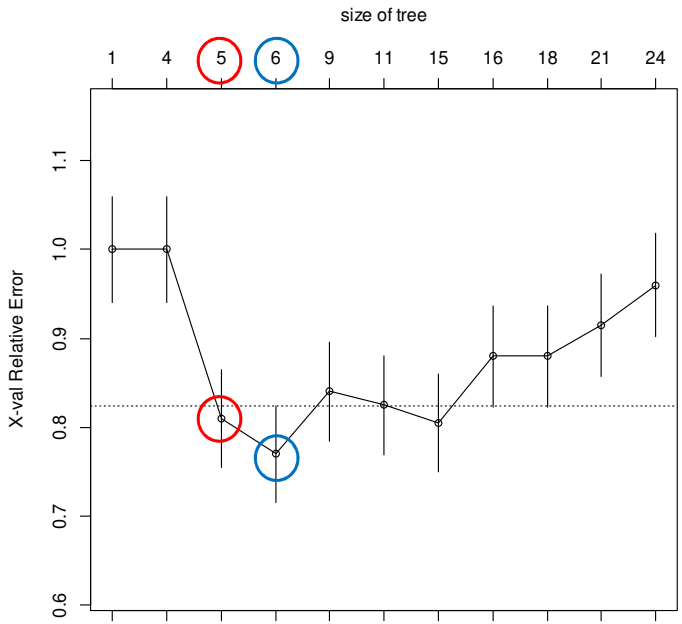
Règles d'élagage

- « 0 SE » : élaguer l'arbre au minimum du taux d'erreur (plus généralement, du coût) calculé par validation croisée ou sur un échantillon de test
- « 1 SE » : élaguer l'arbre au niveau du plus petit arbre (c'est-à-dire la plus grande valeur de *CP*) dont l'erreur soit inférieure à l'erreur minimale plus un écart-type
- La règle « 1 SE » conduit à retenir un arbre moins complexe que la règle « 0 SE », et elle est cohérente avec le fait qu'il faut tenir compte de la variabilité de l'erreur calculée par validation croisée
- Dans les tests sur données simulées effectués par Breiman *et al.*, la règle « 1 SE » conduit à une plus grande stabilité dans la taille des arbres élagués

Graphique

- La fonction `plotcp` produit un graphique représentant l'évolution de l'erreur par validation croisée et du nombre de feuilles, en fonction de la pénalisation

L'erreur minimale plus un écart-type est représentée par une ligne horizontale en pointillés



Application de « 0 SE »
Application de « 1 SE »

Erreur minimale + 1 écart-type =

```
> 0.770 +
0.054404
[1] 0.824404
```

Élagage automatique

- Application de la règle « 1 SE » :

```
> xerr <- cart$cptable[,"xerror"]
> minxerr <- which.min(xerr)
> seuilerr <- cart$cptable[minxerr, "xerror"] +
cart$cptable[minxerr, "xstd"]
> xerr [xerr < seuilerr][1]
   3
0.81
> mincp <- cart$cptable[names(xerr [xerr < seuilerr][1]),
"CP"]
> mincp
[1] 0.035
> prunedcart <- prune(cart,cp=mincp)
```

- Application de la règle « 0 SE » :

```
prunedcart <- prune(cart,
cp=cart$cptable[which.min(cart$cptable[,"xerror"]),"CP"])
```

Prédiction d'un arbre

- Application de l'arbre élagué à l'échantillon de validation :

```
> prunedcart5f <- prune(cart, cp=0.035)
> pred.cart    <- predict(prunedcart5f, type="prob", valid)
```

- Création d'une matrice avec une colonne par modalité de la variable à expliquer, chaque colonne contenant la probabilité associée
- Exemple des cinq premières valeurs dans l'échantillon de test :

```
> head(pred.cart,5)
          0         1
1 0.7119565 0.2880435
2 0.3047619 0.6952381
3 0.8833333 0.1166667
4 0.3047619 0.6952381
6 0.8833333 0.1166667
```

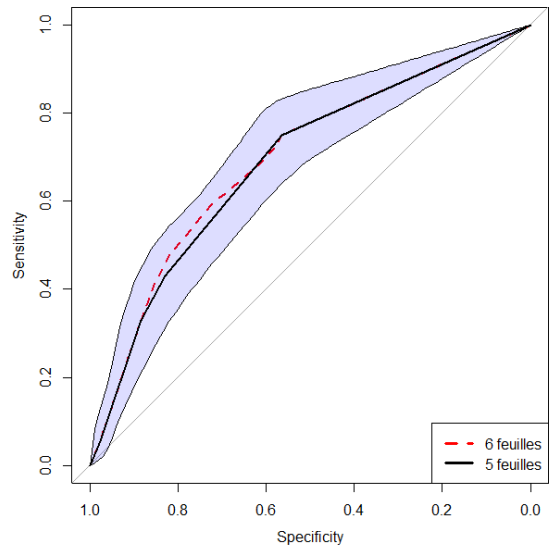
- Aire sous la courbe ROC :

```
> library(pROC)
> auc(valid$Cible, pred.cart[,2], quiet=TRUE)
Area under the curve: 0.6894
```

Aire sous la courbe ROC d'un arbre

- Élagage de l'arbre maximal à 5 et 6 feuilles, application à l'échantillon de validation et superposition des aires sous la courbe ROC

```
> prunedcart5f  <- prune(cart, cp=0.035)
> pred.cart  <- predict(prunedcart5f,
type="prob", valid)
> prunedcart6f <- prune(cart, cp=0.0183333)
> pred.cart6f  <- predict(prunedcart6f,
type="prob", valid)
> roc <- plot.roc(valid$Cible, pred.cart[,2],
col='black', lty=1, ci=TRUE, quiet=TRUE)
> plot.roc(valid$Cible, pred.cart6f[,2],
add=TRUE, col='red', lty=2, ci=TRUE, quiet=TRUE)
> roc.se <- ci.se(roc,
specificities=seq(0,1,.01), boot.n=2000)
> plot(roc.se, type="shape", col="#0000ff22")
> legend("bottomright",c('6 feuilles','5
feuilles'),col=c('red','black'),lty=c(2,1),lwd=3)
```



- La courbe ROC de l'arbre à 6 feuilles est à l'intérieur de l'intervalle de confiance à 95 % de celle de l'arbre à 5 feuilles
- On constate généralement que les courbes ROC des meilleurs modèles ajustés sur un même échantillon sont à l'intérieur d'un intervalle de confiance à 95 %

Calcul des erreurs et des aires sous la courbe ROC

- Ajout de l'aire sous la courbe ROC à la table des erreurs en fonction du paramètre de complexité :

```
> set.seed(235)
> auc <- matrix(NA,nrow(cart$cptable)-1,4)
> for(i in 2:nrow(cart$cptable))
+ {
+ cartp <- prune(cart, cp=cart$cptable[i,"CP"])
+ predc <- predict(cartp, type="prob", valid)[,2]
+ auc[i-1,1] <- cart$cptable[i,"CP"]
+ auc[i-1,2] <- cart$cptable[i,"nsplit"]+1
+ auc[i-1,3] <- cart$cptable[i,"xerror"]
+ auc[i-1,4] <- auc(valid$Cible, predc, quiet=TRUE)
+ }
> colnames(auc) <- c("CP","nfeuilles","erreur","AUC")
> auc
               CP nfeuilles erreur       AUC
 [1,] 0.065000000         4  1.000 0.6863462
 [2,] 0.035000000         5  0.810 0.6894231
 [3,] 0.018333333         6  0.770 0.6869658
 [4,] 0.015000000         9  0.840 0.6955128
 [5,] 0.013333333        11  0.825 0.6846581
 [6,] 0.010000000        15  0.805 0.6927350
 [7,] 0.007500000        16  0.880 0.6864744
 [8,] 0.006666667        18  0.880 0.6925427
 [9,] 0.003333333        21  0.915 0.6822863
[10,] 0.000000000        24  0.960 0.6820299
```

Arbre C5.0

- C5.0 est adapté comme CART à tout type de variables
  - C5.0 est plus rapide et gère mieux la mémoire que C4.5 aussi inventé par J.R. Quinlan
- Dispositif d'optimisation de l'arbre par construction puis élagage d'un arbre maximum
  - le procédé d'élagage est différent de celui de CART et il est lié à l'intervalle de confiance du taux d'erreur donc à l'effectif du nœud
- Utilisation de l'entropie comme fonction d'impureté
- C5.0 n'est pas binaire. Les variables qualitatives, au niveau d'un nœud père, donnent naissance à un nœud fils par modalité
  - inconvénient : les nœuds voient plus rapidement leurs effectifs baisser
- Transformation de l'arbre en règles qui permet une simplification par suppression de règles redondantes mais fait perdre la structure d'arbre
- R : packages `C50` et `RWeka`

Prédiction d'une variable ordinale I

- Les coûts de mauvais classement permettent de prédire une variable ordinale
- Soient $s_1 < s_2 < \ldots < s_J$ les valeurs ordonnées de Y
- Un coût de mauvais classement peut être $C_{ij} = |s_i - s_j|$
- Une erreur de classement dans une classe adjacente a moins de poids que dans une classe distante
- L'indice de Gini devient alors $\sum_{i=1}^{J}\sum_{j=1}^{J}|s_i - s_j|f_i f_j$
- $= 2\ \sum_{j=1}^{J-1}(s_{j+1} - s_j)F_j(1 - F_j)$ avec $F_j = \sum_{h=1}^{j} f_h$ (proportion cumulée) d'après Piccarreta (2001)
- $= 2\ \sum_{j=1}^{J-1} F_j(1 - F_j)$ dans le cas (fréquent) où $(s_{j+1} - s_j) = 1$ pour tout $j \geq 1$
- $\sum_{j=1}^{J-1} F_j(1 - F_j)$ est l'indice de Gini ordinal de Piccarreta (2008)

Prédiction d'une variable ordinale II

- Pour l'élagage d'un arbre avec Y ordinale, le taux d'erreur $\frac{1}{n}\sum_{i=1}^{n} 1_{s_i \neq \hat{s}(x_i)}$ peut être remplacé par un coût total de mauvais classement $\sum_{i=1}^{n} |s_i - \hat{s}(x_i)|$ faisant intervenir la distance entre la valeur ordinale observée s_i et prédite $\hat{s}(x_i)$
- Ce coût peut être utilisé dans le calcul du coût-complexité utilisé pour l'élagage de l'arbre
- Avec un coût de mauvais classement $C_{ij} = (s_i - s_j)^2$, on peut montrer que l'indice de Gini est proportionnel à la variance des valeurs $s_1, s_2, \ldots, s_J$, et que l'objectif de réduction de l'indice de Gini équivaut à la réduction de la variance des s_j dans les nœuds-fils, donc au développement d'un arbre de régression dont les valeurs numériques sont les valeurs s_j

Prédiction d'une variable ordinale III

- Le package R `rpartScore` met en œuvre les arbres de décision ordinaux sur la base de CART avec :
  - les coûts $C_{ij} = |s_i - s_j|$ (`split="abs"`) et $C_{ij} = (s_i - s_j)^2$ (`split="quad"`)
  - le coût total de mauvais classement $\sum_{i=1}^{n} |s_i - \hat{s}(x_i)|$ (`prune="mc"`)
- La fonction `rpartScore` renvoie un objet de la classe `rpart`
- NB : le package `rpart` n'implémente pas les coûts de mauvaise classification car il suppose que le coût C_{ij} d'affectation à G_i d'un individu qui est dans G_j ne dépend que de G_j et quand on lui spécifie une matrice de coûts comme paramètre `loss`, il remplace C_{ij} par $\Sigma_i C_{ij}$

Avantages des arbres de décision 1/2

- Ils fournissent des règles :
  - explicites
  - visuelles
  - qui s'écrivent directement avec les variables d'origine
- Méthode non paramétrique, non perturbée par :
  - la distribution non linéaire ou non monotone des prédicteurs par rapport à la variable à expliquer
  - la colinéarité des prédicteurs
  - les interactions entre les prédicteurs
  - les individus hors norme (isolés dans des règles spécifiques)
  - les fluctuations des prédicteurs non discriminants (l'arbre sélectionne les plus discriminants)

Avantages des arbres de décision 2/2

- Beaucoup traitent (sans recodification) des données hétérogènes (numériques et non numériques, voire manquantes)
  - CART traite les valeurs manquantes en remplaçant les variables concernées par des variables équidivisantes
  - CHAID traite l'ensemble des valeurs manquantes d'une variable comme une modalité à part ou pouvant être associée à une autre
  - éviter d'avoir trop de valeurs manquantes
- Temps de calcul assez rapide

Limites des arbres de décision 1/2

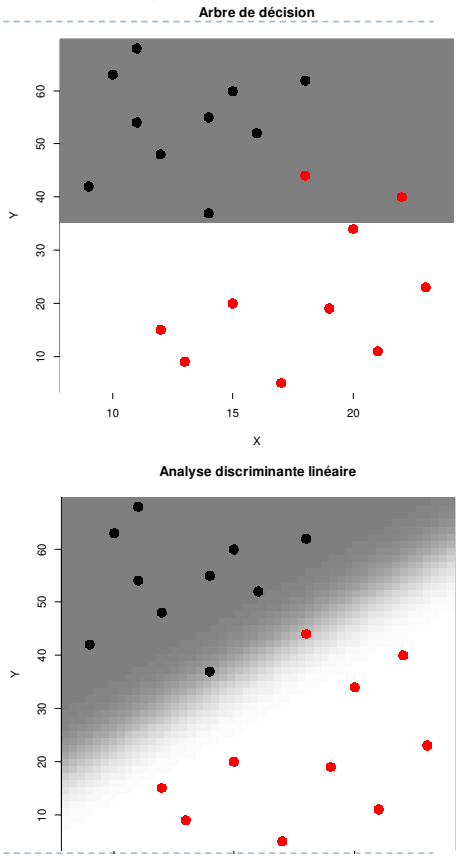
- Les nœuds du niveau $n+1$ dépendent fortement de ceux du niveau n
  - les variables sont testées séquentiellement et non simultanément

  $\Longrightarrow$ la modification d'une seule variable, si elle est placée près du sommet de l'arbre, peut entièrement modifier l'arbre
  - un arbre détecte des optimums locaux et non globaux
  - un arbre est sensible au franchissement d'un seuil de scission

  $\Longrightarrow$manque de robustesse
- L'apprentissage nécessite un nombre suffisant d'individus
  - pour avoir si possible au moins une trentaine individus par nœud
- Même avec des variables explicatives continues, la prédiction est distribuée de façon discontinue puisqu'elle dépend des feuilles
  - nombre de valeurs prédites distinctes $\leq$ nombre de feuilles

Limites des arbres de décision 2/2

- La forme des modèles obtenus, ($X \leq n$) et ($X \in \{a,b,c\ldots\}$), conduit à délimiter des régions rectangulaires de l'espace des variables qui ne correspondent pas forcément à la distribution des individus
- Les arbres obliques remédient à cet inconvénient en substituant aux règles simples de division des nœuds, de la forme ($X \leq n$), des règles sur plusieurs variables du type ($aX + bY + \ldots \leq n$) et permettent un classement au moins aussi précis que si l'arbre avait beaucoup plus de nœuds



La classification automatique

Terminologie : de nombreux synonymes

- **Classification**, ou classification automatique, terme généralement employé par les auteurs français
  - attention : il est employé dans un autre sens par les anglo-saxons (qui disent « classification » pour désigner la technique prédictive que les français appellent « classement »)
- **Segmentation** : terme employé en marketing (les « segments de clientèle ») et assez explicite
- **Typologie**, ou **analyse typologique**
- **Clustering** : terme anglo-saxon le plus courant
- **Taxinomie** ou **taxonomie** (biologie, zoologie)
- **Nosologie** (médecine)
- **Reconnaissance de forme non supervisée** (réseaux de neurones)
- ...

Structure des données à classer

- Soit une matrice rectangulaire dont :
  - lignes = individus
  - colonnes = variables
- Cette structure permet de classer individus ou variables

- Soit une matrice carrée de similarités, distances entre :
  - individus
  - ou variables (par exemple : la matrice des corrélations)
- Cette structure permet aussi de classer individus ou variables

Structure des classes obtenues

- Soit 2 classes sont toujours disjointes : méthodes de partitionnement :
  - généralement, le nombre de classes est défini *a priori*
  - certaines méthodes permettent de s'affranchir de cette contrainte (méthodes basées sur la densité comme DBSCAN ou OPTICS)
- Soit 2 classes sont disjointes ou l'une contient l'autre : méthodes hiérarchiques :
  - ascendantes (agglomératives : agglomération progressive d'éléments 2 à 2)
  - descendantes (divisives)
- Soit 2 classes peuvent avoir plusieurs objets en commun (classes « empiétantes » ou « recouvrantes ») :
  - analyse « floue », où chaque objet a une certaine probabilité d'appartenir à une classe donnée

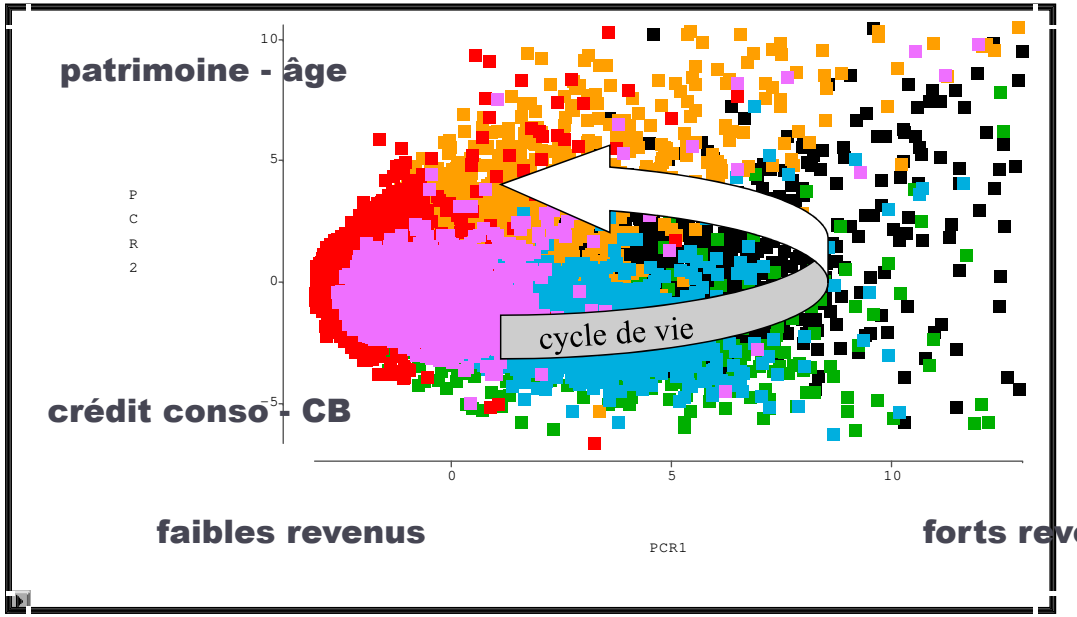
Les différentes méthodes de classification

- Méthodes de partitionnement
  - centres mobiles, *k*-means et nuées dynamiques
  - *k*-modes, *k*-prototypes, *k*-représentants (*k*-medoids)
  - réseaux de Kohonen
  - méthodes basées sur la densité
  - méthode d'agrégation des similarités
- Méthodes hiérarchiques
  - ascendantes (agglomératives)
    - basées sur une notion de distance ou de densité
  - descendantes (divisives)
- Méthodes mixtes
- Analyse floue (fuzzy clustering)

Applications de la classification

- Marketing : répartir la clientèle en segments dotés chacun d'une offre et d'une communication spécifique – autres utilisations pour :
  - les ciblages des actions commerciales
  - l'évaluation du potentiel commercial
  - l'affectation des clients aux différents types de commerciaux
- Commercial : répartir l'ensemble des magasins d'une enseigne en établissements homogènes du point de vue type de clientèle, CA, CA par rayon (selon type d'article), taille du magasin…
- Médical : déterminer des groupes de patients susceptibles d'être soumis à des protocoles thérapeutiques déterminés, chaque groupe regroupant tous les patients réagissant identiquement
- Sociologie : répartir la population en groupes homogènes du point de vue sociodémographique, style de vie, opinions, attentes…
- …

Exemple de segmentation de clientèle (bancaire)



S1	(rouge)	: peu actifs	S4	(orange)	: seniors
S2	(rose)	: jeunes	S5	(noir)	: aisés
S3	(bleu)	: consommateurs	S6	(vert)	: débiteurs

Interprétation des classes

- Statistiques descriptives des classes (comparaison des moyennes ou des modalités par un test statistique)
- Analyse factorielle représentant les classes obtenues et les variables initiales
- Classification des variables : variables initiales + indicatrices des classes obtenues
- Arbre de décision avec la classe obtenue comme variable à expliquer