Arbres de décision
et
Forêts aléatoires

Arbres de décision

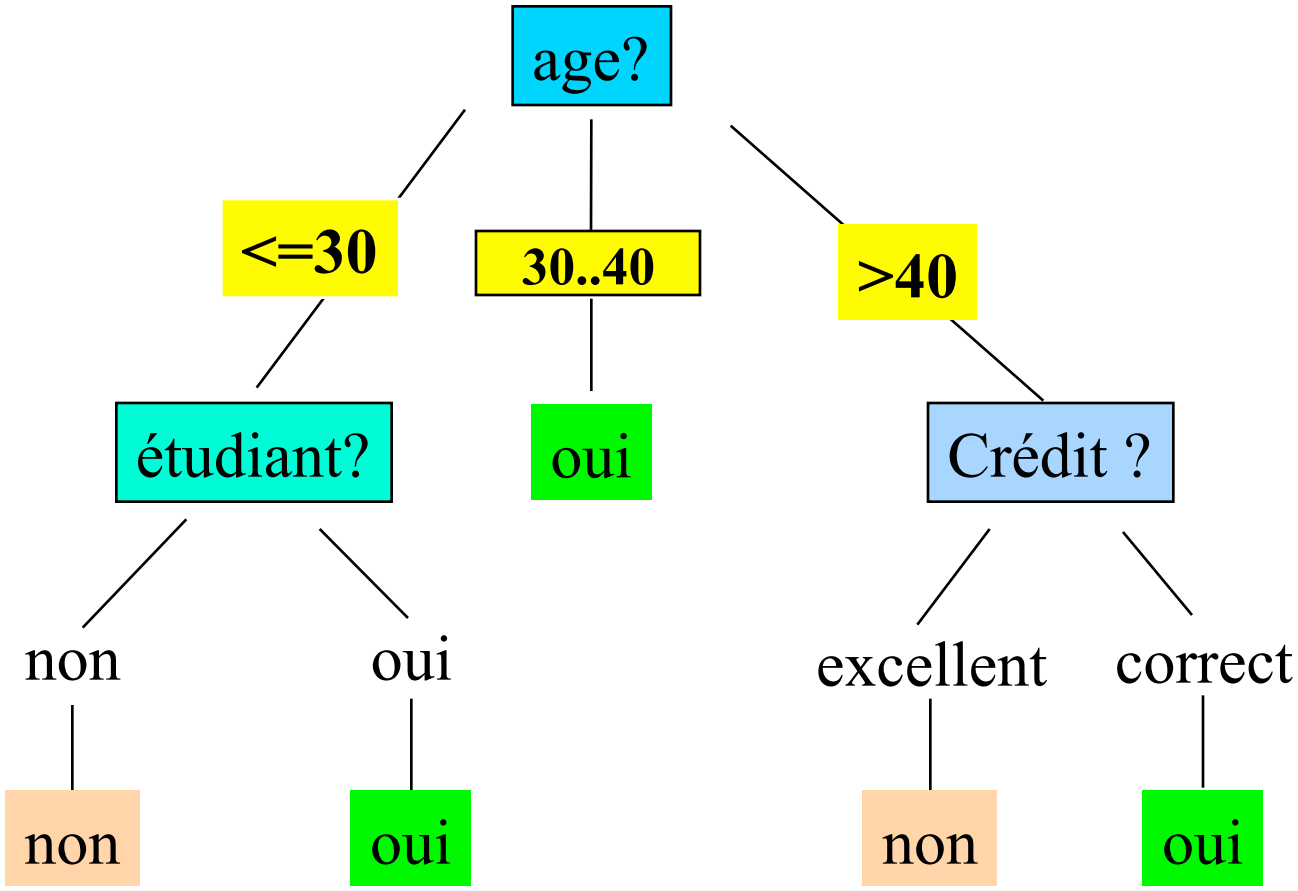
Classification avec arbres de décision

- Arbre de Décision
  - Les nœuds internes correspondent à des tests
  - Un arc correspond au résultat d'un test
  - Les nœuds feuilles représentent des classes
- La génération se fait en 2 phases
  - Construction de l'arbre
    - Au début tous les tuples se trouvent sur la racine
    - Partitionner les tuples récursivement en se basant à chaque fois sur un attribut sélectionné
  - Simplification de l'arbre
    - Identifier et supprimer les branches qui correspondent à des exceptions
- Utilisation:
  - Tester les attributs du tuple par rapport à l'arbre pour trouver la branche et qu'il satisfait donc sa classe

Training set

age	salaire	etudiant	crédit	achète_ordinateur
<=30	élévé	non	correct	non
<=30	élévé	non	excellent	non
30…40	élévé	non	correct	oui
>40	moyen	non	correct	oui
>40	faible	oui	correct	oui
>40	faible	oui	excellent	non
31…40	faible	oui	excellent	oui
<=30	moyen	non	correct	non
<=30	faible	oui	correct	oui
>40	moyen	oui	correct	oui
<=30	moyen	oui	excellent	oui
31…40	moyen	non	excellent	oui
31…40	élévé	oui	correct	oui
>40	moyen	non	excellent	non

Output: Un arbre de décision pour “achète_ordinateur”



Création de l'arbre de décision

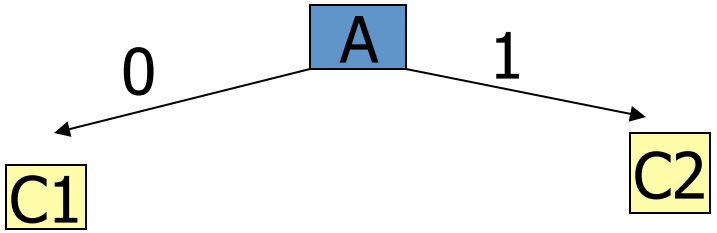
- L'arbre est construit top-down récursivement
  - Au début, tous les tuples sont sur la racine
  - Les attributs sont qualitatifs (discrétisation s'il le faut)
  - Les tuples sont ensuite partitionnés en fonction de l'attribut sélectionné
  - L'attribut de test est sélectionné en utilisant des heuristiques ex: gain informationnel (on y reviendra)
- Conditions d'arrêt du partitionnement
  - Tous les tuples d'un nœud se trouvent dans la même classe

Choix de l'attribut de partionnement (1)

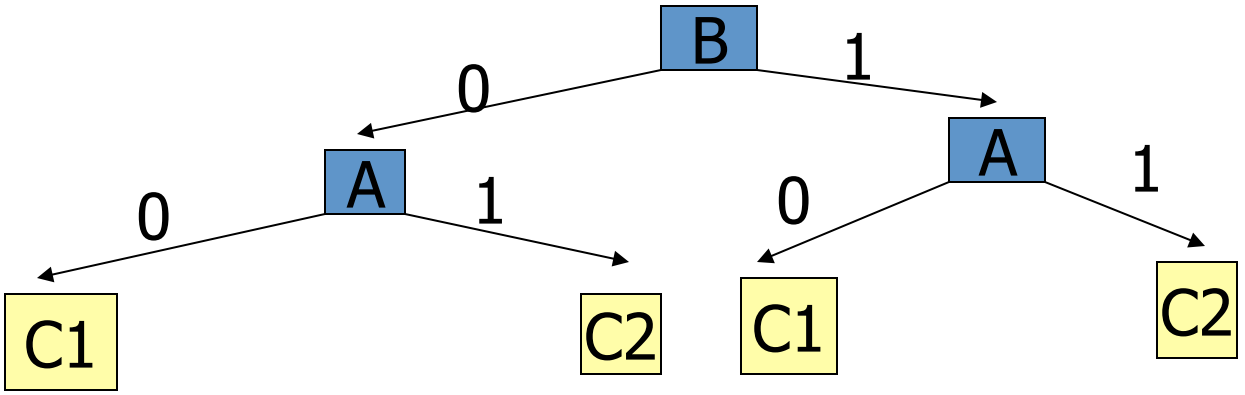
- Soit le training set suivant

A	B	Classe
0	1	C1
0	0	C1
1	1	C2
1	0	C2

Si c'est A qui est choisi en premier



Si c'est B qui est choisi en premier



Choix de l'attribut de partionnement (2)

- Un arbre de décision représente la suite de questions à poser pour pouvoir classifier un nouvel exemple.
- Le but consiste à obtenir une classification en posant le moins possible de questions
- Dans l'exemple précédent, on dira que l'attribut A apporte plus d'information, respectivement à la classification des exemples, que B
- Nous avons donc besoin de quantifier l'information apportée par chaque attribut

Notions sur la théorie de l'information(1)

- Intuitivement : Plus un événement est probable, moins il nous apporte d'information
  - Exemple : Vous êtes dans le désert et on vous annonce que le lendemain, il fera beau. C'est un événement très probable, ce message n'apporte donc presque aucune information

- La quantité d'information associée à un événement x sera considérée comme une fonction croissante sur son *improbabilité*

$$h(X) = f\left(\frac{1}{\text{Proba}(X)}\right)$$

- Un événement certain apporte une quantité d'information nulle, ainsi *f(1)* doit être nulle

Notions sur la théorie de l'information(2)

- La réalisation de 2 événements indépendants apporte une quantité d'information égale à la somme de leurs informations respectives, i.e

$$h(X,Y) = f\left(1/\text{Proba}(X,Y)\right) = f\left(1/\text{Proba}(X) * \text{Proba}(Y)\right) = h(X) + h(Y)$$

- C'est la fonction log en base 2 qui a été choisie. Ainsi,

$$h(X) = -\log_2\left(\text{Proba}(X)\right)$$

- La fonction h satisfait les 2 conditions: croissante et l'info de deux événements indépendants est la somme des infos

Notions sur la théorie de l'information(3)

- Supposons maintenant qu'il y a deux classes, *P* et *N*
  - Soit *S* un ensemble qui contient *p* éléments de *P* et *n* éléments de *N*
  - La probabilité qu'un élément soit dans P est *p/(p+n)*
  - La quantité d'information nécessaire pour décider si un élément quelconque de *S* se trouve dans *P* ou dans *N* est définie par

$$I(p, n) = -p \log_2 \frac{p}{p + n} - n \log_2 \frac{n}{p + n}$$

Cas particulier

- Supposons que p soit nul. Cela veut dire que I(n,p)=0:
  - p=0 et log(p/[n+p])=-∞ le produit donne 0 (pour être précis, la limite du produit tend vers 0 quand p tend vers 0)
  - log(n/[n+p])=0 donc le produit donne 0
- Ce qui est conforme à l'intuition: On n'a pas besoin d'info pour décider si un élément est dans N ou P; on est sûr qu'il est dans N

Intuition de l'expression I(n,p)

- Chaque élément apporte une information qui est
  - $-\log\left(\frac{p}{n+p}\right)$ si il est dans P
  - $-\log\left(\frac{n}{n+p}\right)$ si il est dans N
- Si l'on fait le total des infos, on obtient I(n,p)

Gain d'information et arbre de décision

- Supposons qu'en utilisant l'attribut A, S est partitionné en {S_1, S_2 , …, S_v} (ça veut dire que A prend v valeurs)
  - Si S_i contient p_i *tuples* de *P* et n_i tuples de *N*, l'entropie, ou la quantité d'information nécessaire pour classifier les objets de tous les sous arbres S_i est

$$E(\mathrm{A}) = \sum_{i=1}^{\nu} I(p_i, n_i)$$

  - L'entropie mesure la « quantité de désordre » qui reste après le choix de A

- L'information de codage gagnée en utilisant A sera

$$Gain(\mathrm{A}) = I(p, n) - E(\mathrm{A})$$

Intuition derrière Gain(A)

- Pour classer les éléments dans S_i, nous avons besoin d'une quantité d'info égale à $I(n_i,p_i)$
- Pour classer les éléments dans tous les S_i, on fait la somme des $I(n_i,p_i)$ ce qui donne E(A)
- On sait que pour classifier les éléments, nous avons besoin d'une quantité d'info égale à I(n,p)
- Suite au pationnement des n+p éléments selon les valeurs de A, nous aurons besoin d'une quantité d'info égale à E(A)
- Donc, il nous manquera que I(n,p)-E(A) pour pouvoir classer

Application à l'exemple

- Il y a 2 classes C1 (P) et C2 (N)
- En choisissant A, S est partitionné en S_1 et S_2
- $p_1=2$, $n_1=0$, $p_2=0$ et $n_2=2$

A	B	Classe
0	1	C1
0	0	C1
1	1	C2
1	0	C2

- E(A)=(I(2,0)+I(0,2))
  - I(2,0)=-log(1)-0*log(0)=0
  - I(0,2)=0
  - E(A)=0

- Gain(A)=I(2,2)-E(A)
  - I(2,2)=-2*log(2/4)-2*log(2/4)=4

- Gain(A)=4

Application à l'exemple

- En choisissant B, S est partitionné en S_1 et S_2
- $p_1=1$, $n_1=1$, $p_2=1$ et $n_2=1$

- E(B)=(I(1,1)+I(1,1))
  - I(1,1)=-log(1/2)-log(1/2)=2
  - E(B)=4

- Gain(B)=I(2,2)-E(B)=0

- Il vaut mieux choisir A!!

A	B	Classe
0	1	C1
0	0	C1
1	1	C2
1	0	C2

S1

S2

Gain d'information: Exemple

- Classe P: achète_ordinateur = "oui"
- Classe N: achète_ordinateur = "non"
- I(p, n) = I(9, 5) =13,16
- L'entropie de l'attribut *age*:

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	4,855
30…40	4	0	0
>40	3	2	4,855

$$E(age) = I(2,3) + I(4,0)$$

$$+ I(3,2) = 9,71$$

Ainsi,

Gain(age)=I(p,n)-E(age)=3,45

C'est l'attribut qui maximise le gain

Remarquer que le salaire n'a pas du tout été utilisé

Extraction de règles de classification

- De la forme SI-ALORS
- Chaque chemin partant de la racine et atteignant une feuille donne lieu à une règle
- Chaque paire attribut-value le long d'un chemin forme une conjonction
- Les feuilles constituent la classe
- Exemple

SI *age* = "<=30" ET *étudiant* = "*non*" ALORS
achète_ordinateur = "*non*"

SI *age* = "<=30" ET *étudiant* = "*oui*" ALORS
achète_ordinateur = "*oui*"

Problème de l'overfitting

- En appliquant la méthode décrite jusque là, on obtient des arbres qui classent correctement les exemples du training set
  - Aucune erreur (normalement)

- Mais rien ne dit qu'ils seront efficaces pour l'autre partie des exemples

- Lorsque l'arbre « colle trop au training set » on parle d'overfitting

- Pour résoudre le problème, on va autoriser des erreurs sur le training set pour obtenir des arbres assez généraux

Généraliser l'arbre induit

- 2 Approches
  - Prepruning: ne pas découper un nœud si le partage fait basculer la mesure de pertinence en dessous d'un certain seuil.
    - Par exemple si le gain est inférieur à un certain seuil
    - Difficile de choisir un seuil approprié
  - Postpruning: supprimer des banches d'un arbre déjà construit. Obtenir un ensemble d'arbres réduits
    - Utiliser un ensemble de données différent du training set pour choisir le meilleur arbre réduit

Gestion des erreurs

- Supposons que le taux d'erreurs soit de 25%. Comment peut-on prédire le taux d'erreurs sur les nouveaux exemples?
- Cette prédiction ressemble au problème suivant: On jette une pièce un certain nombre de fois.
  - Quand c'est pile alors on a un succès (l'exemple est bien classifié)
  - Quand c'est face, alors échec (l'exemple est mal classifié)
  - Le taux d'erreurs sur N exemples correspond au taux d'échecs après N jets de la pièce

Gestion des erreurs

- La répétition des jets correspond à une expérience de Bernouilli (voir cours de statistique)

- En utilisant la notion des « intervalles de confiance » on peut estimer, à partir d'un taux calculé, la probabilité d'un échec

  – Si à la suite de 1000 jets, on obtient 750 fois face alors on en déduit qu'on est confient à 80% que le taux de faces en général soit dans l'intervalle [0.73, 0.76]

  – Si à la suite de 100 jets, on obtient 75 fois face, alors dans ce cas, l'intervalle de confiance à 80% devient [0.69,0.80]

Prise en compte des erreurs

- La moyenne et la variance d'une variable de Bernouilli sont resp. p et p(1-p)
- Le taux d'échecs est f=S/N (où S est le nombre de d'échecs après N essais)
- La moyenne et la variance de f sont p et p(1-p)/N
- Le problème consiste à trouver p en fonction de f
- Pour un N *assez grand* (>100) f suit une loi normale
- La variable $Y = \frac{f - p}{\sqrt{p(1-p)/N}}$ suit une loi normale centrée réduite
- étant donné c (seuil de confiance) on peut trouver z tel que Probabilité[-z ≤Y≤z]=c à partir de la table de la loi normale

Prise en compte des erreurs

- Une fois qu'on a trouvé z, il suffit de résoudre l'équation

$$p = \left(f + \frac{z^2}{2N} \pm z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \right) \Big/ \left(1 + \frac{z^2}{N} \right)$$

- Les systèmes (e.g C4.5) adoptent une approche pessimiste en prenant la plus grande valeur de p

Forêts aléatoires
(Random Forests)

Agrégation de modèles

- Algorithmes très récents (années 2000) pour la classification et la régression

- Utilise des stratégies adaptatives (*boosting*) ou aléatoires (*bagging*)

- Idée : Utiliser une combinaison ou une agrégation d'un grand nombre de modèles tout en évitant l'*overfitting*

Agrégation de modèles

- Bagging ou Random Forests : utiliser le hasard pour améliorer les performances d'algorithmes de « faibles» performances.
- Boosting : utiliser une stratégie adaptative pour *booster* des performances, applicable là aussi à tout type d'algorithme (Réseau de neurones, DT, etc.)
- Ces stratégies améliorent principalement les performances des algorithmes non linéaires plus instables. Ils sont en général moins efficaces dans le cas des méthodes linéaires stables.

Bagging

- *Y* variable à expliquer, *X*1, . . . *Xp* variables explicatives
- ϕ modèle appris sur un échantillon *z* = {(*x*1,*y*1)...(*xn*,*yn*)}
- On considère *B* échantillons bootstrap *z*1,...*zB* d'individus étiquetés, ces échantillons sont issus de *z* par tirage aléatoire avec remise.
- Sur chacun des échantillons, on apprend un modèle ϕ*zi*
- On *prédit Y* en agrégeant les différentes décisions sur chacun des *zi* par

$$\hat{\phi}(x) = \frac{1}{B}\sum_{i=1}^{B}\phi_{z_i}(x)$$

  pour une variable quantitative
- On *prédit Y* en agrégeant les différentes décisions sur chacun des *zi* par

  ϕˆ(*x*)=Décision majoritaire parmi les ϕ*zi*(*x*)

  pour une variable qualitative

Random Forests

- $z = \{(x_1, y_1) \ldots (x_n, y_n)\}$ échantillon d'apprentissage, x_i décrit par p variables explicatives
- Pour $b = 1 \ldots B$ (B représente le nombre d'arbres formés dans la forêt)
  - Tirer un échantillon aléatoire z_b avec remise parmi z
  - Estimer un arbre sur z_b avec *randomisation* des variables :
  - Pour la construction de chaque noeud de chaque arbre, on tire uniformément q variables parmi p pour former la décision associée au noeud
- En fin d'algorithme, on possède B arbres que l'on moyenne ou qu'on fait voter pour la régression ou la classification
- En général, un choix optimal pour q est à peu près $q = \sqrt{p}$