Analyse, Classification et Indexation des données (ACID)

Apprentissage supervisé : Arbres de décision et forêts aléatoires

Akka Zemmari

LaBRI, Université de Bordeaux

2023 - 2024

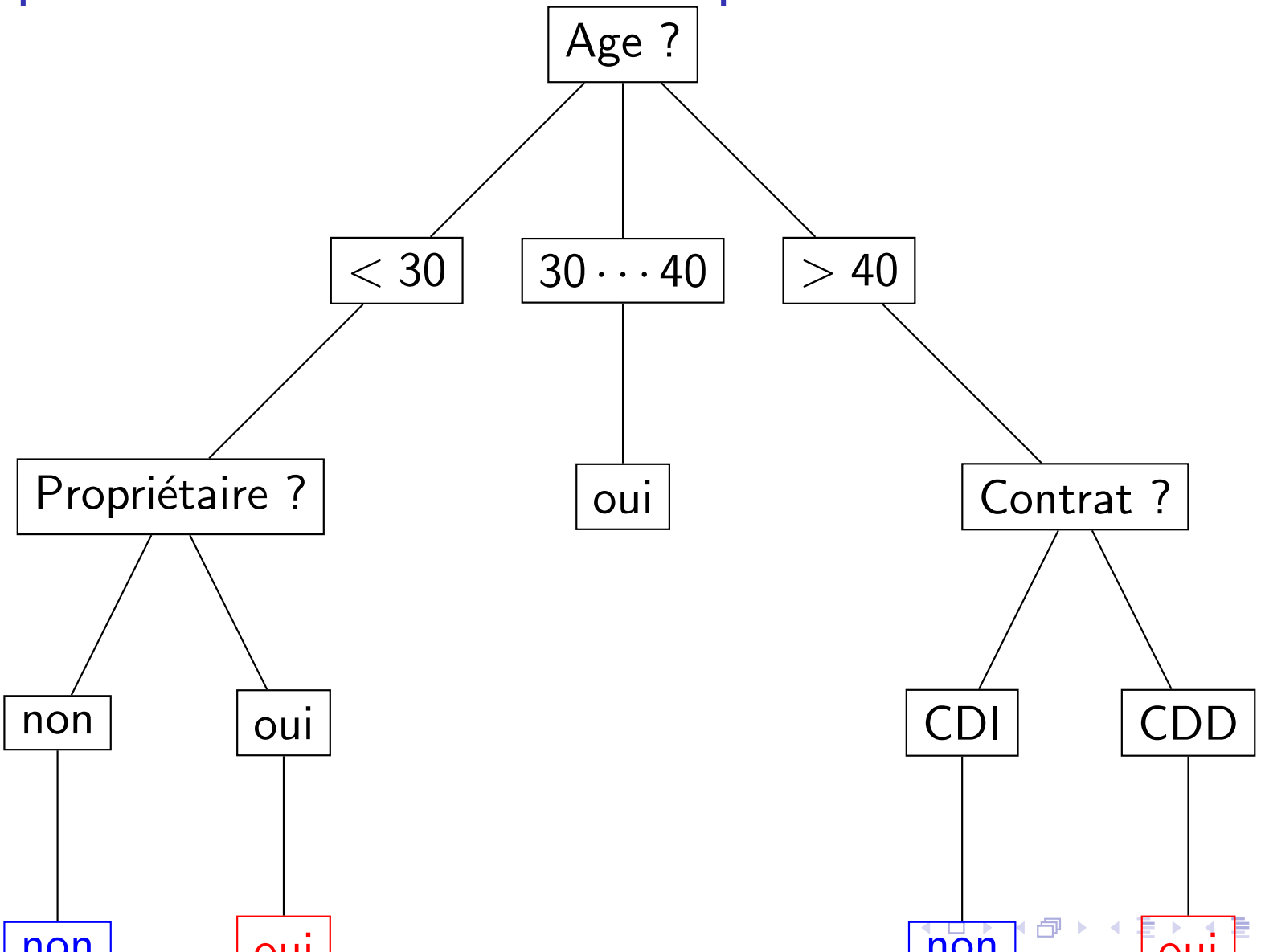
Arbres de décision et classification

Méthode supervisée : Donnés d'entrainement : (x_i, y_i)

Input : Training set

Age	Salaire	Propriétaire	Contrat	Prêt accordé
< 30	élevé	non	CDD	non
< 30	élevé	non	CDI	non
$30 \cdots 40$	élevé	non	CDD	oui
> 40	moyen	non	CDD	oui
> 40	faible	oui	CDD	oui
> 40	faible	oui	CDI	non
$30 \cdots 40$	faible	oui	CDI	oui
< 30	moyen	non	CDD	non
< 30	faible	oui	CDD	oui
> 40	moyen	oui	CDD	oui
< 30	moyen	oui	CDI	oui
$30 \cdots 40$	moyen	non	CDI	oui
$30 \cdots 40$	élevé	oui	CDD	oui
> 40	moyen	non	CDI	non

Output : Un arbre de décision pour "Prêt accordé"



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 noeud se trouvent dans la même classe

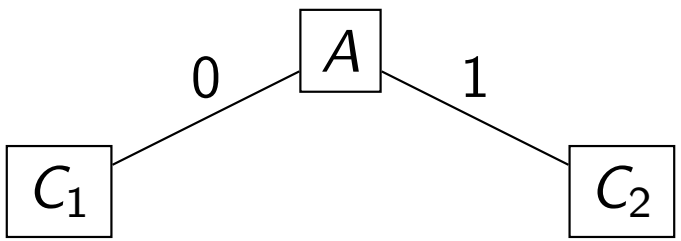
Choix de l'attribut de partionnement (1)

Soit le training set suivant :

A	B	Classe
0	1	C_1
0	0	C_1
1	1	C_2
1	0	C_2

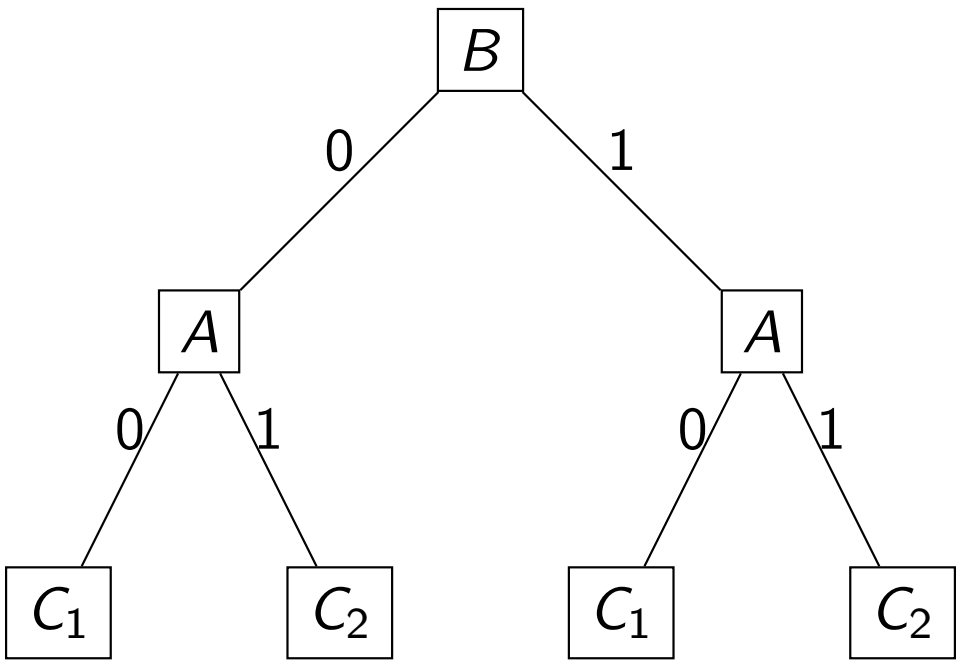
Choix de l'attribut de partionnement (2)

Si c'est A qui est choisi en premier :



Choix de l'attribut de partionnement (3)

Si c'est B qui est choisi en premier :



Choix de l'attribut de partionnement (4)

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

$$h(x) = f\left(\frac{1}{\mathbb{P}r(x)}\right).$$

- 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

$$\begin{aligned} h(x,y) &= f\left(\frac{1}{\mathbb{P}r(x,y)}\right) = f\left(\frac{1}{\mathbb{P}r(x)\times\mathbb{P}r(y)}\right) \\ &= h(x)+h(y). \end{aligned}$$

- Question : Quelle fonction vérifie toutes ces propriétés ?
- Réponse : C'est la fonction log en base 2 qui a été choisie :

$$h(x) = -\log_2\left(\mathbb{P}r(x)\right).$$

- La fonction h satisfait les 2 conditions : croissante et l'information apportée par deux événements indépendants est la somme des informations apportées par chacun.

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 $\frac{p}{p+n}$
  - La quantité d'information nécessaire pour décider si un élément quelconque de S se trouve dans P ou dans N est définie par

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

Cas particulier

Supposons que p soit nul. Cela veut dire que $I(n, p) = 0$:

- $p = 0$ et $\log_2\left(\frac{p}{n+p}\right) = -\infty$ le produit donne 0 (pour être précis, la limite du produit tend vers 0 quand p tend vers 0)
- $\log_2\left(\frac{n}{n+p}\right) = 0$ donc le produit vaut 0, ce qui est conforme à l'intuition : on n'a pas besoin d'information 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
  - $\log_2\left(\frac{p}{n+p}\right)$ s'il est dans P,
  - $\log_2\left(\frac{n}{n+p}\right)$ s'il est dans N.
- Si l'on fait le total des informations, on obtient $I(n, p)$.

Gain d'information et arbre de décision

- Supposons qu'en utilisant l'attribut A, S est partitionné en $\{S_1, S_2, \cdots, S_v\}$ (ce qui 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 :

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

  - $\Rightarrow$ L'entropie mesure la quantité de désordre qui reste après le choix de A
- L'information de codage gagnée en utilisant A sera donc

$$G(A) = Gain(A) = I(p, n) - E(A).$$

Intuition derrière Gain(A)

- Pour classer les éléments dans S_i, nous avons besoin d'une quantité d'information 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'information égale à $I(n, p)$
- Suite au pationnement des $n + p$ éléments selon les valeurs de A, nous aurons besoin d'une quantité d'information égale à $E(A)$
- Donc, il ne nous manquera que $I(n, p) - E(A)$ pour pouvoir classer

Retour à l'exemple

A	B	Classe
0	1	C_1
0	0	C_1
1	1	C_2
1	0	C_2

- Il y a 2 classes C_1 (P) et C_2 (N)

Retour à l'exemple

- En choisissant A, S est partitionné en S_1 et S_2

A	B	Classe
0	1	C_1
0	0	C_1
1	1	C_2
1	0	C_2

$$p_1 = 2,\ n_1 = 0,\ p_2 = 0 \text{ et } n_2 = 2$$

- $E(A) = I(2,0) + I(0,2)$
  - $I(2,0) = -\log_2(1) - 0 \times \log_2(0) = 0$
  - $I(0,2) = 0$
  - $\Rightarrow E(A) = 0$
- $G(A) = I(2,2) - E(A)$
  - $I(2,2) = -2 \times \log_2(2/4) - 2 \times \log_2(2/4) = 4$
- $\Rightarrow G(A) = 4$

Retour à l'exemple

- En choisissant B, S est partitionné en S_1 et S_2

A	B	Classe
0	1	C_1
0	0	C_1
1	1	C_2
1	0	C_2

$$p_1 = 1,\ n_1 = 1,\ p_2 = 1 \text{ et } n_2 = 1$$

- $E(B) = I(1,1) + I(1,1)$
  - $I(1,1) = -\log_2\left(\frac{1}{2}\right) - \log_2\left(\frac{1}{2}\right) = 2$
  - $\Rightarrow E(B) = 4$
- $G(B) = I(2,2) - E(B) = 0$
- $\Rightarrow$ Il vaut mieux choisir A.

Retour à l'exemple jouet

Age	Salaire	Propriétaire	Contrat	Prêt accordé
< 30	élevé	non	CDD	non
< 30	élevé	non	CDI	non
$30 \cdots 40$	élevé	non	CDD	oui
> 40	moyen	non	CDD	oui
> 40	faible	oui	CDD	oui
> 40	faible	oui	CDI	non
$30 \cdots 40$	faible	oui	CDI	oui
< 30	moyen	non	CDD	non
< 30	faible	oui	CDD	oui
> 40	moyen	oui	CDD	oui
< 30	moyen	oui	CDI	oui
$30 \cdots 40$	moyen	non	CDI	oui
$30 \cdots 40$	élevé	oui	CDD	oui
> 40	moyen	non	CDI	non

Retour à l'exemple jouet

- Classes :
  - $P = \{\text{Prêt accordé} = \text{oui}\}$
  - $N = \{\text{Prêt accordé} = \text{non}\}$
- $p = 9$, $n = 5$ et $I(p, n) = 13,16$
- Calcul de l'entropie de *Age* :

Age	p_i	n_i	$I(p_i, n_i$
≤ 30	2	3	4.855
$30 \cdots 40$	4	0	0
> 40	3	2	4.855

Retour à l'exemple jouet

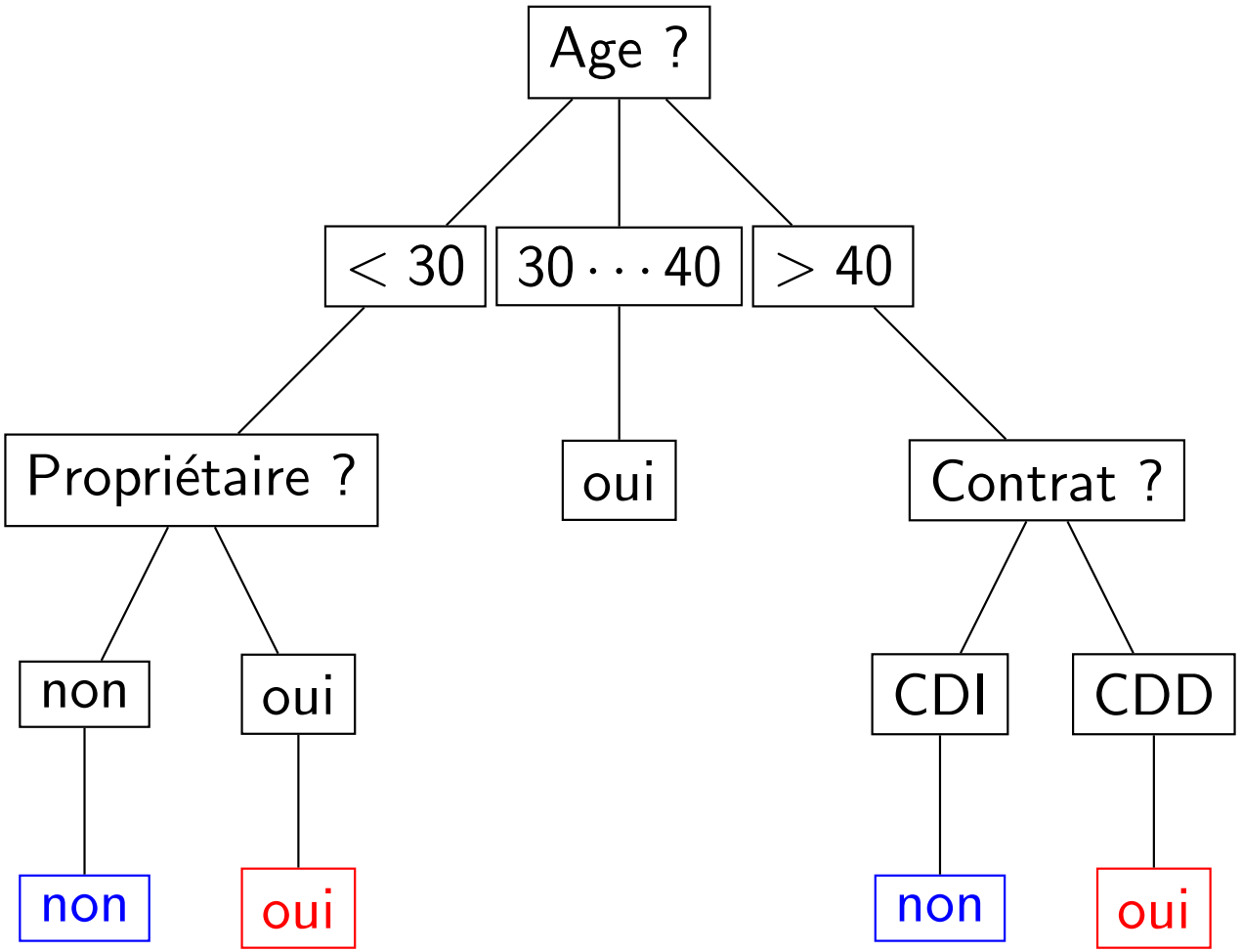
- Calcul de l'entropie de *Age* :

Age	p_i	n_i	$I(p_i, n_i$
≤ 30	2	3	4.855
$30 \cdots 40$	4	0	0
> 40	3	2	4.855

$$\Rightarrow E(Age) = I(2,3) + I(4,0) + I(3,2)$$

- $G(Age) = I(p,n) - E(Age) = 3.45$
- Tous calculs faits, il s'agit de l'attribut qui maximise le gain !!

Output : Un arbre de décision pour "Prêt accordé"



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** Propriétaire = "non" **ALORS**
  Prêt accordé = "non";
  SI Age = "< 30" **ET** Propriétaire = "oui" **ALORS**
  Prêt accordé = "oui"

Autres critères pour le choix de l'attribut

- Indice Gini :

Autres critères pour le choix de l'attribut

- Indice Gini :

$$Gini = 1 - \sum_{i=1}^{v} f_i^2$$

où f_i désigne la fraction des éléments de l'ensemble avec l'étiquette i,dans l'ensemble

Problème de l'overfitting

L'overfitting, c'est quoi ?

- Par la construction ci-dessus, on obtient des arbres qui classent correctement les exemples du training set
  Aucune erreur (normalement)
- Mais rien ne dit qu'ils seront efficaces pour les exemples de l'ensemble de test ...
- 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 qui généralisent mieux ...

L'overfitting, comment l'éviter ?

Deux approches :

- Prepruning
- Postpruning

L'overfitting, comment l'éviter ?

Pré-élagage (Prepruning) :

- ne pas découper un nœud si le partage fait basculer la mesure de pertinence en dessous d'un certain seuil.
- Arrêter quand il y a une classe majoritaire dans le noeud : utiliser un seuil pour détecter une classe dominantes.
- Inconvénients:Arrêter la construction de l'arbre peut donner un arbre sous optimal.

L'overfitting, comment l'éviter ?

Post-élagage (Postpruning) :

- On construit d'abord l'arbre de décision (le plus parfait possible)
- On coupe des sous arbres entiers et en les remplace par des feuilles représentant la classe la plus fréquente dans l'ensemble des données de cet arbre.
- On commence de la racine et on descend, pour chaque noeud interne (non feuille), on mesure sa complexité avant et après sa coupure (son remplacement par une feuille), si la différence est peu importante, on coupe le sous arbre et on le remplace par une feuille.

Synthèse des algorithmes de construction de l'arbre

- ID3 (Inductive Decision Tree, Quinlan 1979) :
  - arbres de discrimination (i.e. variables uniquement qualitatives)
  - critère = entropie
- C4.5 (Quinlan 1993) :
  - Amélioration ID3, permettant notamment arbres de régression (gestion des variables continues) et valeurs manquantes
- CART(Classification And Regression Tree, Breimanetal. 1984) :
  - critère = Gini

Inconvénient principal des arbres de décision :

- Sensibilité au bruit et points aberrants
- Une solution : Forêts aléatoires

Agrégation de modèles

- Algorithmes récents (années 2000) pour la classification et la régression.
- Utilisent 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: 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, $X_1, X_2, \cdots, X_p$ variables explicatives
- ϕ modèle appris sur un échantillon $z = \{(x_1, y_1, (x_2, y_2), \cdots, (x_n, y_n)\}$
- On considère B échantillons bootstrap $z_1, z_2, \cdots, z_B$ d'individus étiquetés. Ces échantillons sont issus de z par tirage aléatoire avec remise.
- Sur chacun des échantillons, on apprend un modèle ϕ_{z_i}

Bagging

- On prédit Y en agrégeant les différentes décisions de chacun des ϕ_{z_i} :

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

pour une variable quantitative.

- On prédit Y en agrégeant les différentes décisions de chacun des ϕ_{z_i} :

$$\hat{\phi}(x) = \text{Décision majoritaire parmi les décisions des } \phi_{z_i}$$

pour une variable qualitative.

Forêts Aléatoires (Random Forests)

Un algorithme de bagging.
Principe : l'union fait la force

- Forêt = ensemble d'arbres
- Forêt Aléatoire (Breiman & Cutter, 2001):
  - apprendre grand nombre T d'arbres simples,
  - utilisation par vote des arbres (classe majoritaire, voire probabilités des classes par % des votes) si classification, ou moyenne des arbres si régression.

Apprentissage de Forêt Aléatoire (1)

Objectif : obtenir des arbres les plus décorrelés possible

- chaque arbre appris sur un sous-ensemble aléatoire différent de s exemples d'apprentissage
- chaque noeud de chaque arbre choisi comme "split" optimal parmi k variables tirées aléatoirement dans les entrées (avec $k << d$ la dim des entrées)

Apprentissage de Forêt Aléatoire (2)

- chaque arbre appris avec l'algorithme CART
- on limite fortement la profondeur p des arbres ($\sim$2 à 5)

Apprentissage de Forêt Aléatoire (3)

Plus ou moins formellement :

- $S = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$: base d'apprentissage, chaque x_i de dimension d
- Pour $t = 1, \cdots, T$:
  1. Tirer aléatoirement (avec remise) m exemples dans S;
  2. Sur les d features (attributs, variables), on ne retient qu'un échantillon de cardinal $k << d$ (typiquement $k \sim \sqrt{d}$)
  3. Sur chaque échantillon, on entraîne un arbre de décision, en limitant sa croissance par validation croisée

Success story

Squelettisation de personne avec Kinect[1] :

[1] https://pterneas.com/2014/03/13/kinect-for-windows-version-2-body-tracking/

En pratique

Voir le jupyter notebook