Arbres de décision : motivation, définitions

Définitions

En théorie des graphes, un arbre est un graphe non orienté, acyclique et connexe. L'ensemble des nœuds se divise en trois catégories :

- Nœud racine (l'accès à l'arbre se fait par ce nœud),
- Nœuds internes : les nœuds qui ont des descendants (ou enfants), qui sont à leur tour des nœuds,
- Nœuds terminaux (ou feuilles) : nœuds qui n'ont pas de descendant.

Les arbres de décision (AD) sont une catégorie d'arbres utilisée dans l'exploration de données et en informatique décisionnelle. Ils emploient une représentation hiérarchique de la structure des données sous forme des séquences de décisions (tests) en vue de la prédiction d'un résultat ou d'une classe.

Chaque individu (ou observation), qui doit être attribué(e) à une classe, est décrit(e) par un ensemble de variables qui sont testées dans les nœuds de l'arbre.

Les tests s'effectuent dans les **nœuds internes** et les décisions sont prise dans les nœuds feuille.

Exemple : comment répartir une population d'individus (e.g. clients, produits, utilisateurs, etc.) en groupes homogènes selon un ensemble de variables descriptives (e.g. âge, temps passé sur un site Web, etc.) et en fonction d'un objectif fixé (variable de sortie ; par exemple : chiffre d'affaires, probabilité de cliquer sur une publicité, etc.).

Arbres de décision : exemples

Voici quelques exemples où on doit classer une situation ou individu en suivant une séquence de tests. Le processus de décision est équivalent à une « descente » dans l'arbre (de la racine vers une des feuilles) : à chaque étape un attribut est testé et un sous-arbre est choisi, la parcours s'arrête dans une feuille (une décision est prise).

Prêt bancaire

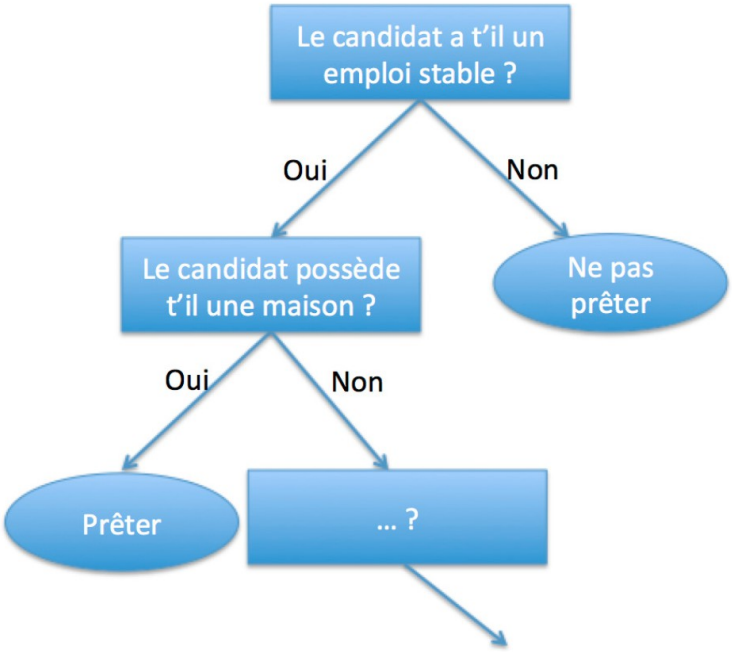


Fig. 32 Accorder ou non un prêt bancaire. Chaque individu est évalué sur un ensemble de variables testées dans les nœuds internes. Les décisions sont prises dans les feuilles. Source : https://maximilienandile.github.io

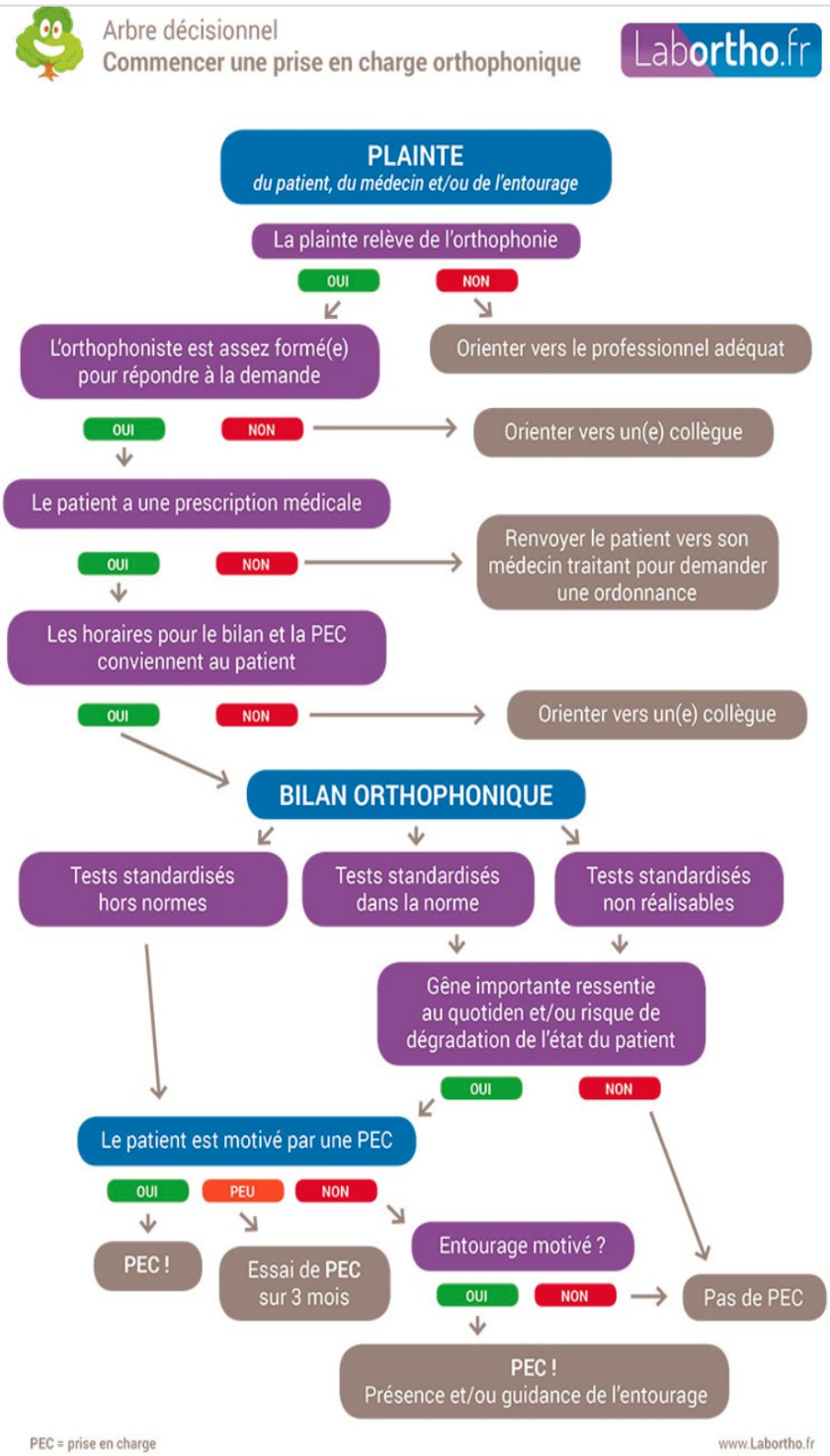


Fig. 33 Modalités de prise en charge orthophonique d'un patient. Source : http://www.labortho.fr

La construction d'un arbre de décision

. Pour réaliser la construction d'un arbre de décision il faut s'appuyer seulement sur les données d'apprentissage.

But : « apprendre » un arbre de décision performant en terme de généralisation

Méthode naïve : Explorer tous les arbres possibles et ne garder que celui qui produit les meilleurs résultats sur des données de test.

Problème : Il y a explosion combinatoire du nombre d'arbres possible !!!

Solution : Il faut donc une méthode *« intelligente »* pour explorer l'espace de tous les arbres possibles et trouver l'arbre de décision le ***« plus efficace ».***

Apprentissage avec les arbres de décision

Chaque élément x de la base de données d'apprentissage est représenté par un vecteur multidimensionnel $(x_1,x_2,...x_n)$ correspondant à l'ensemble de variables descriptives des élements à classer.

. Chaque nœud interne de l'arbre correspond à un test fait sur une des variables xi :

- Variable catégorielle : génère une branche (un descendant) par valeur de l'attribut : ***cas de la classification***
- Variable numérique : test par intervalles (tranches) de valeurs : ***cas de la régréssion***

Les feuilles de l'arbre spécifient les classes.

. Une fois l'arbre construit, classer un nouveau candidat se fait par une descente dans l'arbre, de la racine vers une des feuilles (qui encode la décision ou la classe).

. A chaque niveau de la descente on passe un nœud intermédiaire où une variable xi est testée pour décider du chemin (ou sous-arbre) à choisir pour continuer la descente.

Principe de la construction

. Au départ, les points des la base d'apprentissage sont tous placés dans le nœud racine.

. Une des variables de description des points est la classe : *«variable cible».*

. La variable cible peut être catégorielle (problème de classement) ou valeur réelle (problème de régression).

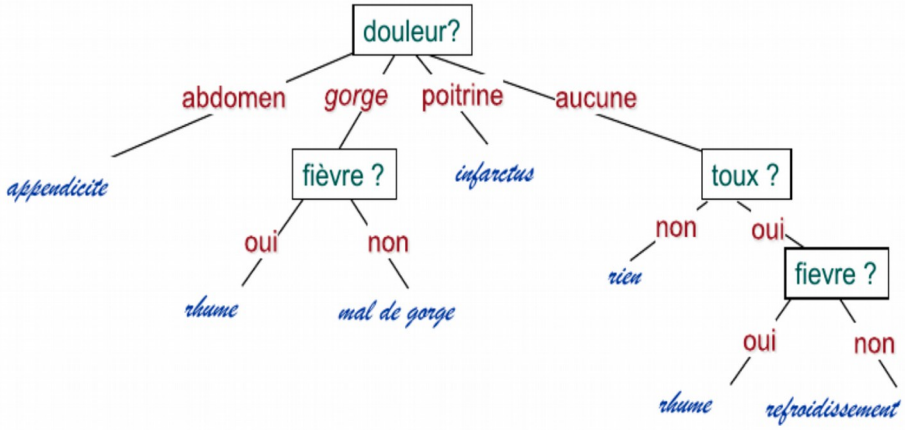
. Chaque nœud est coupé (opération *« split »*) donnant naissance à plusieurs nœuds descendants.

. Un élément de la base d'apprentissage situé dans un nœud se retrouvera dans un seul de ses descendants.

- L'arbre est construit par partition récursive de chaque nœud en fonction de la valeur de l'attribut testé à chaque itération (top-down induction).
- Le critère optimisé est la homogénéité des descendants par rapport à la variable cible (***Entropie***).
- La variable qui est testée dans un nœud sera celle qui maximise cette homogénéité (***Gain d'information***)
- Le processus s'arrête quand les éléments d'un nœud ont la même valeur pour la variable cible (homogénéité).

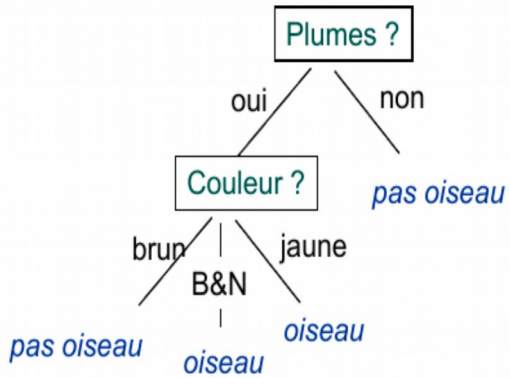
Exemple 1 : maladies

	Toux	Fièvre	Poids	Douleur
Marie	non	oui	normal	gorge
Fred	non	oui	normal	abdomen
Julie	oui	oui	maigre	aucune
Elvis	oui	non	obese	poitrine



Exemple 2 : oiseaux

	Couleur	Ailes	Plumes	Sonar	Concept
Faucon	jaune	oui	oui	non	*oiseau*
Pigeon	B&N	oui	oui	non	*oiseau*
chauve-souris	brun	oui	non	oui	*pas oiseau*



(Si Plumes = « non » Alors Classe= « pas-oiseau)
ou (Si Plumes = oui & Couleur= brun Alors Classe= pas-oiseau)
ou (Si Plumes = oui & Couleu r= B&N Alors Classe= oiseau)
ou (Si Plumes = oui & Couleu r= jaune Alors Classe= oiseau)

La méthode ID3 (Iterative Dichotomiser 3)

. L'algorithme commence par le placement de tous les exemples d'apprentissage dans le nœud racine.

. Ensuite, chaque nœud est coupé sur un des attributs restants (qui n'a pas encore été testé).

. Le choix de cet attribut se fait à travers une mesure d'homogénéité par rapport à la variable cible. Cette mesure est le gain d'information obtenu par le découpage.

. On suppose que la variable cible (classe) a m valeurs distinctes (les étiquettes de classe).

. Pour un nœud S (interne ou feuille) on calcule son entropie par rapport à la cible comme suit :

- Partitionner S sur les valeurs de la classe en m groupes : C1,...,Cm.
- Calculer pi,i=1...m, la probabilité qu'un élément de S se retrouve dans C_i ($pi \approx |C_i|/|S|$ ou $|C_i|$ est la taille du groupe C_i).
- $H(S)=-\sum p_i * \log(p_i)$ est l'entropie de S.

. H(S) mesure l'écart de la distribution de la variable cible par rapport à la distribution uniforme :

- H(S)=0 si S est homogène (tous les éléments sont dans la même classe : toutes les probabilités pi sont égales à 0, sauf une - qui est égale à 1).
- H(S)=max si toutes les probabilités pi sont égales (tous les groupes Ci ont la même taille : $p_1 = \square = p_n = 1/m$).

Pour calculer le gain d'information dans un nœud interne S sur l'attribut a :

- Partitionner S sur les valeurs de l'attribut a en k sous-groupes : $S1,...,S_k$ (k est le nombre de valeurs distinctes de l'attribut a),
- p_i : la probabilité qu'un élément de S appartient à S_i ($p_i \approx |S_i|/|S|$),
- Calculer $GI(S;a)=H(S)-\sum p_{i*}H(S_i)$ le gain d'information sur l'attribut a.

Avec ces précisions, l"algorithme ID3 commence par la racine. Ensuite pour le nœud S en train d'être testé :

- Calculer le gain d'information pour chaque attribut pas encore utilisé,
- Choisir l'attribut a de gain d'information maximal,
- Créer un test (décision) sur cet attribut dans le nœud S et générer les sous-nœuds $S_1,...S_k$ correspondant à la partition sur l'attribut choisi a,
- Récurrence sur les nœuds qui viennent d'être crées.

Sortie de la récursivité :

- Tous les éléments de S sont dans la même classe ($H(S)=0$) : S devient nœud feuille,
- Pas d'attributs non utilisés : nœud feuille sur la classe majoritaire ;

Exemple :

Classification d'un ensemble de jours (J1, …, J14) en deux catégories : « Adapté pour jouer au golf », et « Pas adapté ».

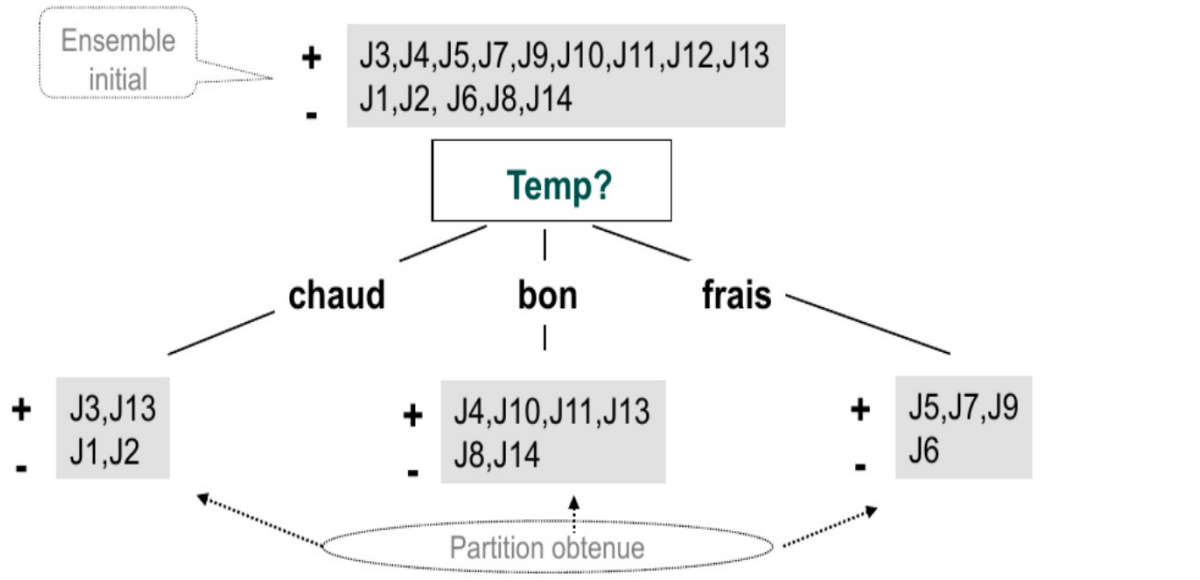
Exemple *[Quinlan,86]*

Attributs Valeurs possibles	Pif soleil,couvert,pluie	Temp chaud,bon,frais	Humid normale,haute	Vent vrai,faux

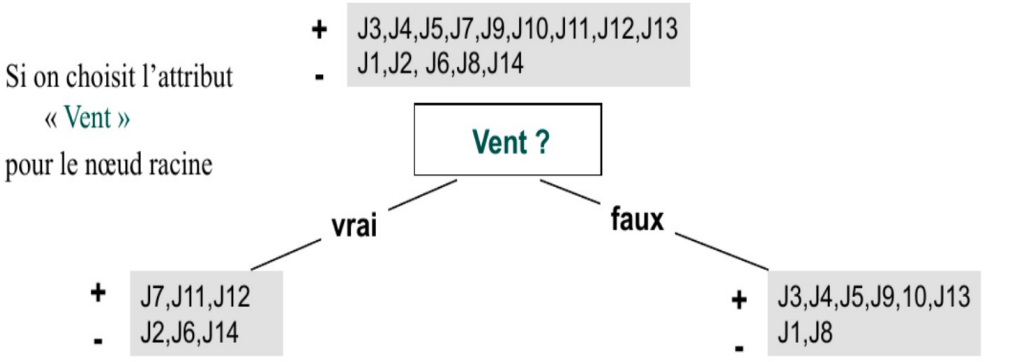
N°	Pif	Temp	Humid	Vent	Golf
1	soleil	chaud	haute	faux	NePasJouer
2	soleil	chaud	haute	vrai	NePasJouer
3	couvert	chaud	haute	faux	Jouer
4	pluie	bon	haute	faux	Jouer
5	pluie	frais	normale	faux	Jouer
6	pluie	frais	normale	vrai	NePasJouer
7	couvert	frais	normale	vrai	Jouer
8	soleil	bon	haute	faux	NePasJouer
9	soleil	frais	normale	faux	Jouer
10	pluie	bon	normale	faux	Jouer
11	soleil	bon	normale	vrai	Jouer
12	couvert	bon	haute	vrai	Jouer
13	couvert	chaud	normale	faux	Jouer
14	pluie	bon	haute	vrai	NePasJouer

la classe

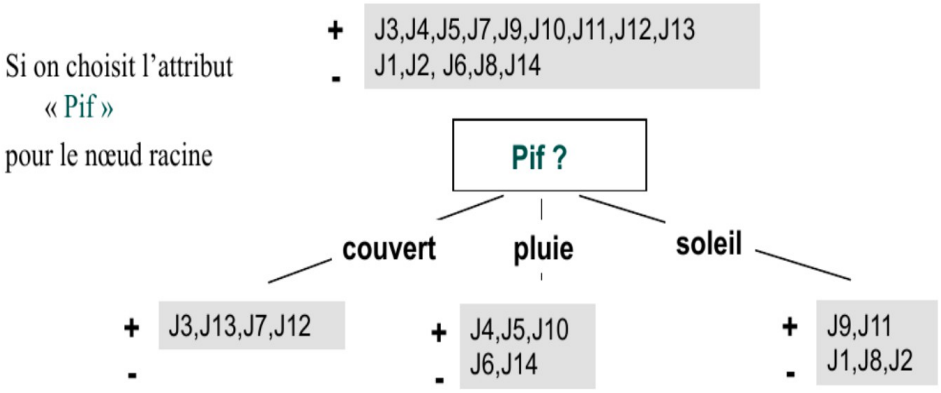
Choix de l'attribut ***Temp*** pour la décision



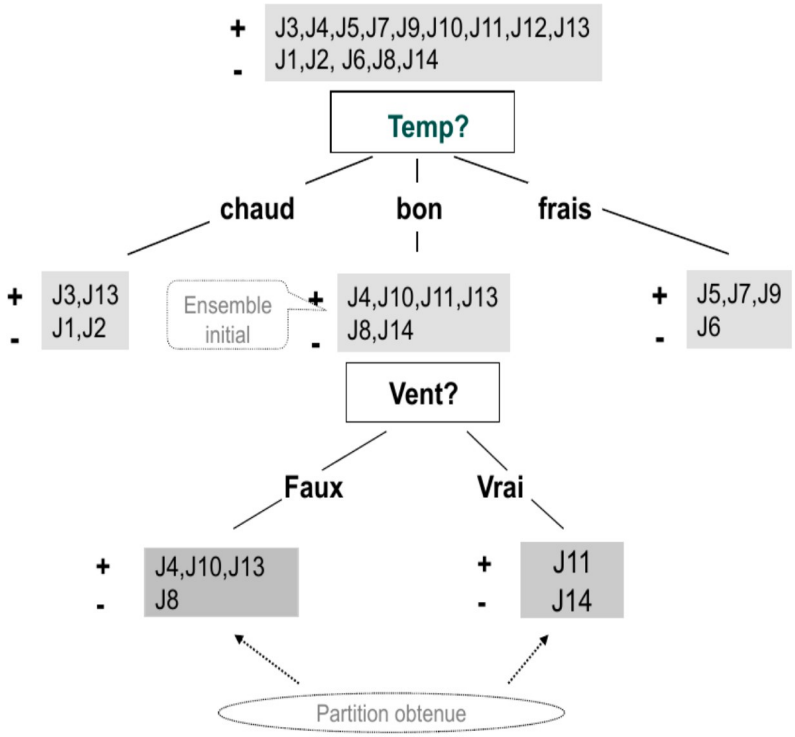
Si on choisit l'attribut « Vent » pour le nœud racine



Si on choisit l'attribut « Pif » pour le nœud racine



Développement de l'arbre à partir d'un noeud pendant (feuille)
Exemple



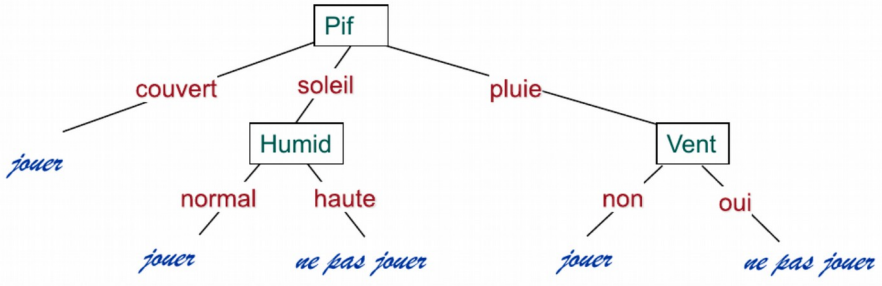
Application numérique

Gain(Pif) = 0.940 - 0.694 = 0.246 bits

- Gain(Temp) = 0.029 bits
- Gain(Humid) = 0.151 bits
- Gain(Vent) = 0.048 bits

Choix de l'attribut Pif pour le premier test

- Arbre final obtenu :



Elagage d'un arbre de décision

. L'arbre construit par la méthode ID3 est d'erreur apparente nulle (tous les exemples de la base d'apprentissage sont bien classés).

. Mais il se peut que le classifieur possède un pouvoir d'apprentissage *« par coeur »* des données d'apprentissage mais un pouvoir de *généralisation* faible (commet beaucoup d'erreurs sur des données de test non vues durant la phase d'apprentissage :
Il s'agit du phénomène de *sur_apprentissage (Overfeting)*

. Pour éviter de construire de arbres trop grands et déséquilibrés : on procède à l'élagage de l'arbre construit par la méthode ID3.

. L'élagage consiste à supprimer des sous arbres de l'arbre complet obtenu par la méthode ID3 tout en minimisant l'erreur de classification commise par le nouvel arbre obtenu et qui conserve un pouvoir de *généralisation et qu'il soit équilibré*.

La méthode d'élagage CART

. Soit V un ensemble de données d'apprentissage appelé ensemble de validation (exemples non vus par le classifieur durant la phase d'apprentissage)

. Soit S l'ensemble des données d'apprentissage

. soit T_0 l'arbre appris sur les données S à l'aide de la méthode ID3

. Pour chaque noeud interne p de T_0, on définit le terme suivant :

alpha(p)=delta(p)/(nb$_f$-1)

où :

delta(p) = nombre d'erreurs supplémentaires que commet l'arbre sur l'ensemble S

nb$_f$ = nombre de feuilles supprimées après élagage au niveau du noeud p

. L'arbre T_{i+1} est obtenu en élaguant T_i en son nœud qui porte la plus petite valeur de *alpha(p)*

. On obtient ainsi une suite d'arbres : $T_0, T_1, \ldots T_i, \ldots T_{max}$ où T_{max} est réduit à un nœud feuille

. Dans une deuxième phase: on calcule l'arbre élagué optimal en choisissant l'arbre parmi $T_0, T_1, \ldots T_i, \ldots T_{max}$ qui *fait le moins d'erreurs* sur l'ensemble de validation V.

Algorithme Elaguer (T_0, S, V)

{

k=0

T_k=T

Tant que T_k a plus d'un nœud (ce n'est pas une feuille)

{

pour chaque nœud p interne de T_k

calculer delta(p) sur S

fin pour

choisir le nœud p_{opt} pour lequel delta(p_{opt}) est minimal

T_{k+1} se déduit de T_k en remplaçant p_{opt} par une feuille

k=k+1

}

Dans l'ensemble des arbres (T_0, T_1, ...T_i,....T_{max}) choisir celui qui fait le moins d'erreurs de classification sur l'ensemble de validation V

}

Exemple :

. Soit la Base d'apprentissage suivante (décision de jouer ou pas au Football?)

Match à domicile ?	Balance positive ?	Mauvaises conditions climatiques ?	Match précédent gagné ?	Match gagné
V	V	F	F	V
F	F	V	V	V
V	V	V	F	V
V	V	F	V	V
F	V	V	V	F
F	F	V	F	F
V	F	F	V	F
V	F	V	F	F

. L'application de l'algorithme ID3 nous donne l'arbre de décision suivant:

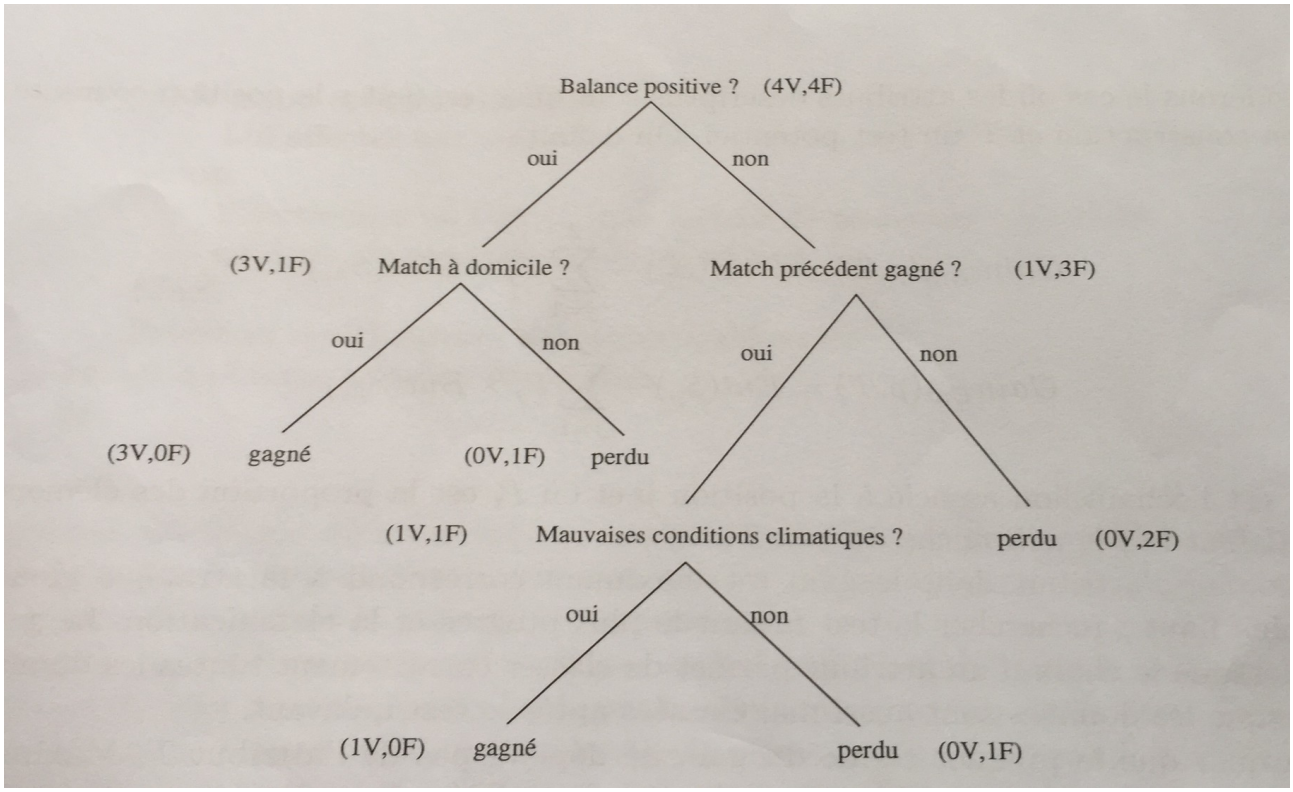


figure1

. Soit l'ensemble de validation suivant:

Match à domicile ?	Balance positive ?	Mauvaises conditions climatiques ?	Match précédent gagné ?	Match gagné
V	V	V	F	V
F	V	V	F	V
F	F	F	V	F
V	F	V	F	F
V	F	V	F	V

Elaguage

*Itération1 :soit T_0 l'arbre de la **figure1**. Pour cet arbre on obtient :*

alpha(racine)=4/4=1
alpha(1)=1
alpha(2)=1/2
alpha(2.1)=1

On obtient T_1 en élaguant de T_0 la branche à partir du noeud 2

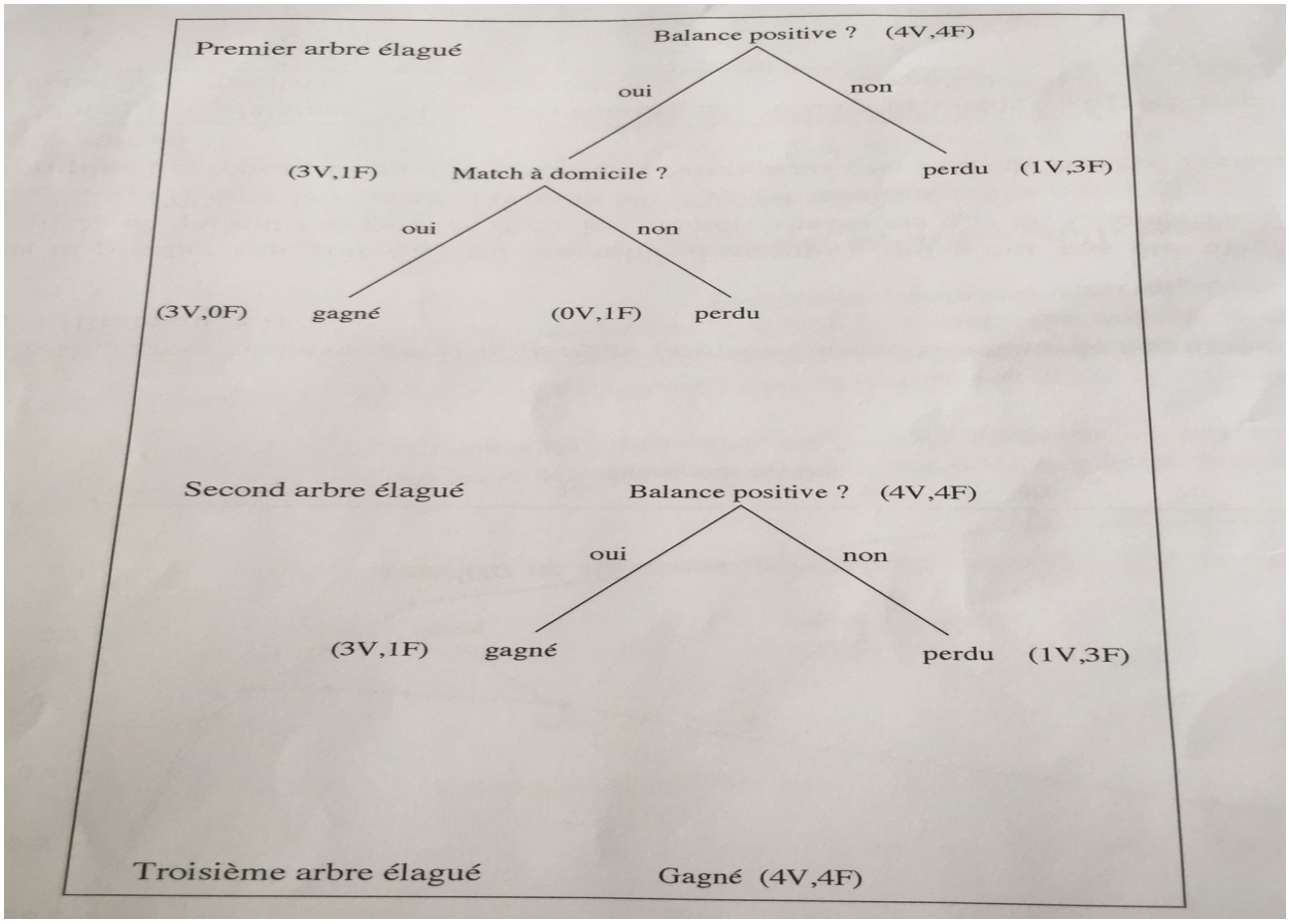
Itération2 : même processus sur l'arbre T1

alpha(racine)=3/2
alpha(1)=1

On obtient T_2 en élaguant de T_1 la branche à partir du noeud 1

Itération3 : même processus sur l'arbre T_2

On obtiens T3 qui est se réduit à une feuille avec la classe gagné.



L'ensemble de validation sur les arbres T0,T1,T2 et T3 nous donne :

$nb\_erreur(T_0)=1,$
$nb\_erreur(T_1)=1,$
$nb\_erreur(T_2)=1$
$nb\_erreur(T_3)=2$

L'algorithme renvoie l'arbre T_2 (plus petite erreur d'un arbre le plus équilibré)

Bagging et Forêts aléatoires

Défauts des arbres de décision

- Parfois les arbres générés ne sont pas équilibrés (ce qui implique que le temps de parcours n'est plus logarithmique). Il est donc recommandé d'équilibrer la base de données avant la construction, pour éviter qu'une classe domine (en termes de nombre d'exemples d'apprentissage).
- Sur-apprentissage : parfois les arbres générés sont trop complexes et généralisent mal (solution : élagage, le contrôle de la profondeur de l'arbre)
- Ils sont **instables** : des changements légers dans les données produisent des arbres très différents. Les changements des nœuds proches de la racine affectent beaucoup l'arbre résultant. On dit que les arbres produisent des **estimateurs de variance élevée**.

. Le besoin de répondre à ce troisième problème, qui n'admet pas de solution par optimisation algorithmique, a conduit aux approches de type *Bagging* et « Forêts aléatoires ».

. L'idée derrière et celle de la **Réduction de variance** : on utilise pour cela la moyenne de plusieurs estimateurs, calculés sur des données légèrement différentes, en somme utiliser le hasard pour améliorer les performances des algorithmes de base (Algorithme d'élagation de CART).

Bagging

Nous commençons par le *Bagging* (*Bootstrap Agregating*). Soit la base d'apprentissage décrite comme suit :

- Les données sont décrites par les attributs : $A_1,...,A_p$, et Classe C,
- Données d'apprentissage : (x_i,y_i)
- y_i peuvent être des valeurs continues ou discrètes (étiquettes des classes),

On considère G(x) un modèle de prédiction appris sur un échantillon de données $z=\{(x_i,y_i)\}$ (arbre de décision CART).

Algorithme bagging :

- On tire au hasard dans la base d'apprentissage B échantillons avec i=1,…,B (chaque échantillon ayant n points) — *appelés échantillons bootstrap*
- Pour chaque échantillon i on calcule le modèle $G_i(x)$;
- Régression : *agrégation par la moyenne*
  $G(x)=1/B*\sum Gi(x)$
- Classement : agrégation par vote
  $G(x)=Votemajoritaire(G1(x),...,GB(x))$.

Les défauts du bagging :
Les estimateurs Gi ne sont pas en réalité indépendants. En effet, Gi sont calculés sur des échantillons qui se recouvrent fortement (tirage avec remise) et donc ils sont corrélés.

L'amélioration proposée par les forêts aléatoires est de baisser la corrélation entre les Gi à l'aide d'une étape supplémentaire de randomisation.

Forêts aléatoires

Les *forêts aléatoires* sont donc une amélioration du *bagging* pour les arbres de décision CART dans le but de rendre les arbres utilisés plus indépendants (moins corrélés).
Caractéristiques :

- Elles donnent de bons résultats surtout en grande dimension,
- Elles sont très simples à mettre en œuvre,
- Elles ont peu de paramètres.

Algorithme « forêts aléatoires »:

- On tire au hasard dans la base d'apprentissage B échantillons avec remise i=1,…,B (chaque échantillon ayant n points).
- Pour chaque échantillon i on construit un arbre CART Gi(x) selon un algorithme légèrement modifié : a chaque fois qu'un nœud doit être coupé (étape "split") on tire au hasard une partie des attributs (*q* parmi les *p* attributs) et on choisit le meilleur découpage dans ce sous-ensemble.
- Régression : agrégation par la moyenne
  $G(x)=1/B*\sum Gi(x)$.
- Classement : agrégation par vote
  $G(x)=Votemajoritaire(G1(x),...,GB(x))$.

Les arbres sont moins corrélés car :

- Ils sont appris sur un ensemble différent d'attributs.
- Ils sont construits sur des échantillons différents.

Remarque : Généralement on choisit q= racine_carée(n) où n est la dimension de chaque entrée