Une introduction aux arbres de décision

Stéphane Caron
http://scaron.info

31 août 2011

Les arbres de décision sont l'une des structures de données majeures de l'apprentissage statistique. Leur fonctionnement repose sur des heuristiques qui, tout en satisfaisant l'intuition, donnent des résultats remarquables en pratique (notamment lorsqu'ils sont utilisés en « forêts aléatoires »). Leur structure arborescente les rend également lisibles par un être humain, contrairement à d'autres approches où le prédicteur construit est une « boîte noire ».

L'introduction que nous proposons ici décrit les bases de leur fonctionnement tout en apportant quelques justifications théoriques. Nous aborderons aussi (brièvement) l'extension aux *Random Forests*. On supposera le lecteur familier avec le contexte général de l'apprentissage supervisé. [1]

Table des matières

1. Wikipédia : http://fr.wikipedia.org/wiki/Apprentissage_supervisé

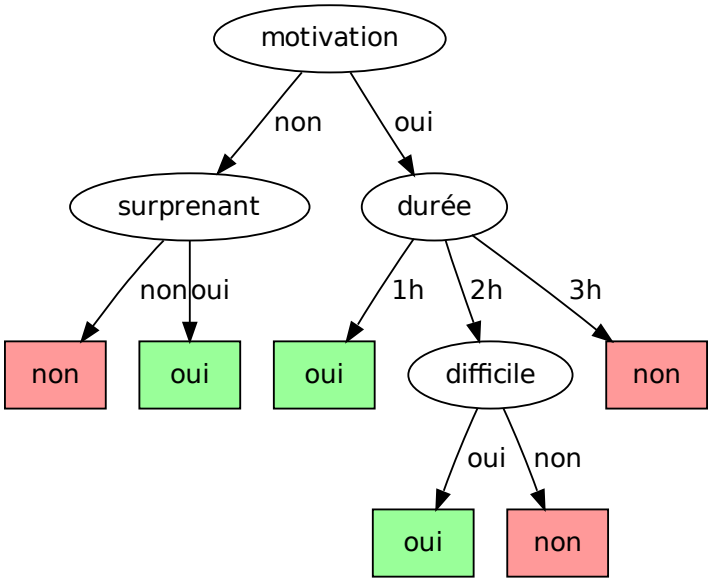

FIGURE 1 – Exemple d'arbre de décision pour la question « Cette présentation est-elle intéressante ? »

Un arbre de décision modélise une hiérarchie de tests sur les valeurs d'un ensemble de variables appelées *attributs*. À l'issue de ces tests, le prédicteur produit une valeur numérique ou choisit un élément dans un ensemble discret de conclusions. On parle de *régression* dans le premier cas et de *classification* dans le second. Par exemple, l'arbre de la figure 1 ci-dessus décide une réponse booléenne (classification dans l'ensemble {oui, non}) en fonction des valeurs discrètes des attributs {difficile, durée, motivation, surprenant}.

Un ensemble de valeurs pour les différents attributs est appelé une « instance », que l'on note généralement $(\mathbf{x}, y)$ où y est la valeur de l'attribut que l'on souhaite prédire et $\mathbf{x} = x_1, \ldots, x_m$ désignent les valeurs des m autres attributs. L'apprentissage d'un arbre de décision se fait sur un ensemble d'instances $T = \{(\mathbf{x}, y)\}$ appelé « ensemble d'entraînement ».

1 Construction d'un arbre de décision

À partir d'un ensemble d'observations $T = \{(\mathbf{x}, y)\}$, on souhaite construire un arbre de décision prédisant l'attribut y en fonction de nouvelles instances $\mathbf{x}$.

date	motivation	durée (h)	difficile	surprenant	température (K)
11/03	oui	1	non	oui	271
14/04	oui	4	oui	oui	293
03/01	non	3	oui	non	297
25/12	oui	2	oui	non	2911
⋮	⋮	⋮	⋮	⋮	⋮

TABLE 1 – Exemple d'ensemble d'entraînement pour l'arbre de la figure 1

Pour ce faire, il existe essentiellement deux familles d'algorithmes à ce jour : les arbres de QUINLAN [10, 7] et les arbres CART [2]. Les deux approches suivent le paradigme diviser-pour-régner, que l'on peut schématiser (dans le cas d'attributs à valeurs discrètes) par le pseudo-code suivant :

```
1:  ArbreDecision(T)
2:      si "condition d'arret"
3:          retourner feuille(T)
4:      sinon
5:          choisir le "meilleur" attribut i entre 1 et m
6:          pour chaque valeur v de l'attribut i
7:              T[v] = {(x, y) de T tels que x_i = v}
8:              t[v] = ArbreDecision(T[v])
9:          fin pour
10:         retourner noeud(i, {v -> t[v]})
11:     fin si
```

FIGURE 2 – Constructeur pour des attributs à valeurs discrètes

Où $\mathsf{noeud}(i, \{v \to t_v\})$ désigne le constructeur d'un nœud qui teste l'attribut i et possède un descendant t_v pour chaque valeur v possible. Les parties entre guillemets correspondent à des choix heuristiques propres à chaque algorithme :

Condition d'arrêt : elle influe sur la profondeur et la précision du prédicteur produit. Par exemple, la condition $|T| = 1$ produira des arbres très précis sur l'ensemble d'entraînement (*i.e.* pour une instance $(\mathbf{x}, y) \in T$, la prédiction pour $\mathbf{x}$ sera exactement y) mais également très profonds, donc longs à calculer, et qui risquent de *surapprendre* les données d'entraînement.

Meilleur attribut : il s'agit d'évaluer localement quel attribut apporte « le plus d'information » (ou encore « est le plus corrélé ») au résultat à prédire. On verra plusieurs tels critères par la suite.

Lorsque l'attribut x_i est à valeurs réelles, on adapte l'algorithme ci-dessus en choisissant une valeur de partage (*split value*) v et en effectuant le test $x_i \leq v$. On notera « $\mathsf{noeud}(i, v, t_{\leq}, t_{>})$ » le constructeur associé. En particulier, si tous les attributs sont réels, l'arbre de décision obtenu est binaire.

2 Régression

Considérons dans un premier temps le cas de la régression, *i.e.* lorsque la valeur à prédire est un nombre réel. Une fois l'arbre construit, la régression d'une nouvelle instance $\mathbf{x}$ explore l'une de ses branches comme suit :

```
1:  Regresser(x, t)
2:      si t = feuille(Tf)
3:          retourner la moyenne des y de Tf
4:      sinon si t = noeud(i, v, t_left, t_right)
5:          si x[i] <= v
6:              retourner Regresser(x, t_left)
7:          sinon, x[i] > v
8:              retourner Regresser(x, t_right)
9:      sinon, t = noeud(i, {v -> t[v]})
10:         retourner Regresser(x, t[x[i]])
11:     fin si
```

L'exploration aboutit à une feuille f construite sur un ensemble d'entraînement T_f. Notons $\mathsf{Y}_f := \{y | \exists(\mathbf{x}, y) \in T_f\}$. La valeur prédite est la moyenne $\overline{y}$ de $\mathcal{Y}_f$, prédiction qui sera d'autant meilleure que la distribution des $y \in \mathcal{Y}_f$ est concentrée autour de $\overline{y}$. De manière équivalente, plus la dispersion des $y \in \mathcal{Y}_f$ est grande, plus la prédiction sera mauvaise. Dans des solutions comme CART [2], on utilise la variance $\overline{\sigma}^2$ des $y \in \mathcal{Y}_f$ pour mesurer cette dispersion, et par ce biais estimer la « qualité » de la feuille f.

2.1 Choix de l'attribut de partage

CART utilise également ce critère de variance pour choisir le meilleur attribut de partage à la ligne 4 du constructeur 1. Un attribut i produit une partition $T = \cup_{v_i} T_{v_i}$, chaque sous-ensemble ayant sa propre variance $\mathbf{V}(T_{v_i})$. La variance attendue après un branchement sur l'attribut i (pour une instance $(\mathbf{x}, y)$ tirée

uniformément au hasard dans T) est alors

$$V_i \;=\; \sum_{v_i} \frac{|T_{v_i}|}{|T|} \mathbf{V}(T_{v_i}),$$

et l'attribut i^* qui minimise cette valeur est alors considéré (heuristiquement) comme le meilleur choix possible.

Nous allons voir dans la section suivante que ce choix de minimiser la variance n'est pas seulement heuristique : il est également intrinsèquement lié à la minimisation de l'erreur quadratique de prédiction.

2.2 Pourquoi chercher à minimiser la variance ?

L'une des principales hypothèses faites dans les travaux de *Machine Learning* est que les instances $(\mathbf{x}, y)$ sont toutes tirées indépendamment selon une même loi de probabilité $\mathcal{P}$ inconnue. On peut alors voir les $(\mathbf{x}, y) \in T$ comme les réalisations i.i.d. de variables aléatoires $\mathbf{X}$ et Y (éventuellement corrélées). De même, les $y \in \mathcal{Y}_f$ sont des réalisations i.i.d. d'une certaine variable aléatoire Y_f, qui n'est autre que Y conditionnée par les événements $\{X_i \leq \text{ou} > v\}$ testés le long de la branche menant à f.

Dans ce contexte, $\overline{y}$ est un estimateur de $\mathbf{E}(Y_f)$ et l'erreur quadratique commise en prédisant $\overline{y}$ pour une nouvelle instance de Y_f s'écrit

$$\begin{aligned} \mathbf{E}(Y_f - \overline{y})^2 &= \mathbf{E}(Y_f - \mathbf{E}(Y_f) + \mathbf{E}(Y_f) - \overline{y})^2 \\ &= (\overline{y} - \mathbf{E}(Y_f))^2 + \mathbf{E}(Y_f - \mathbf{E}(Y_f))^2 \end{aligned}$$

($\overline{y}$ et $\mathbf{E}(Y_f)$ sont ici constantes.) Le second terme de cette erreur n'est autre que la variance σ^2 de Y_f, dont $\overline{\sigma}^2$ est un estimateur. Minimiser $\overline{\sigma}$ pour la feuille f vise donc à minimiser ce second terme, mais également le premier. En effet, $\overline{y}$ étant une moyenne emprique $\overline{y} = \frac{1}{k}\sum_{i=1}^{n} Y_f^{(i)}$ de réalisations de Y_f, on a $\mathbf{E}_T(\overline{y}) = \mathbf{E}(Y_f)$ et $\mathbf{V}_T(\overline{y}) = \frac{1}{n}\sigma^2$.[2] L'inégalité de Markov s'écrit alors

$$\mathbf{P}_T[(\overline{y} - \mathbf{E}(Y_f))^2 > x] < \frac{\sigma^2}{kx},$$

et l'on voit que réduire σ a pour effet de concentrer la distribution de $(\overline{y} - \mathbf{E}(Y_f))$ en zéro.

3 Classification

Le raisonnement pour classifier une instance est similaire à la régression :

2. L'indice T indique que l'intégration a lieu sur tous les ensembles d'entraînement T possibles, pour une taille $|T|$ fixée.

```
1:  Classifier(x, t)
2:      si t = feuille(Tf)
3:          retourner la classe majoritaire de Tf
4:      sinon si t = noeud(i, v, t_left, t_right)
5:          si x[i] <= v
6:              retourner Classifier(x, t_left)
7:          sinon, x[i] > v
8:              retourner Classifier(x, t_right)
9:      sinon, t = noeud(i, {v -> t[v]})
10:         retourner Classifier(x, t[x[i]])
11:     fin si
```

FIGURE 3 – Constructeur pour des attributs à valeurs discrètes

Si toutes ou la grande majorité des instances de T_f ont la même classe $c \in \{1, \ldots, C\}$, c apparaît comme la meilleure prédiction possible, et la fraction des instances de classe c dans T_f est un indicateur de la « sûreté » de cette prédiction. Dans le cas contraire, prédire la classe la plus fréquente reste une option, mais la sûreté de la prédiction n'en sera que moins bonne.

3.1 Mesurer l'homogénéité des feuilles

Les observations précédentes soulignent qu'un arbre de classification est d'autant meilleur que (les instances de) ses feuilles sont de classes homogènes. On souhaite alors définir une mesure de l'hétérogénéité d'une feuille qui jouera le même rôle que la variance dans le cas de la régression. Soient $p_1, \ldots, p_C$ les fréquences relatives des classes $1, \ldots, C$ dans T_f, et c^* la classe la plus fréquente. Voici trois mesures fréquemment rencontrées dans la littérature :

Taux d'erreur : $e(T_f) := 1 - p_{c^*}$
Il s'agit du taux d'erreurs de classification sur l'ensemble d'entraînement.

Critère de Gini : $e(T_f) = \sum_c p_c(1 - p_c)$
Il s'agit du taux d'erreur sur l'ensemble d'entraînement d'un algorithme randomisé qui retournerait la classe c avec probabilité p_c (au lieu de toujours retourner la classe c^*). C'est la mesure d'erreur utilisée dans CART [2].

Entropie : $e(T_f) = -\sum_c p_c \log p_c$
Il s'agit d'un estimateur de l'entropie [3] de la classe d'une instance de T_f tirée uniformément au hasard. C'est la mesure d'erreur utilisée dans les arbres ID3 et C4.5 [10, 7].

3. Wikipédia : `http://fr.wikipedia.org/wiki/Entropie`

3.2 Choix de l'attribut de partage

La mesure d'erreur est utilisée de même pour choisir le meilleur attribut de partage : si un attribut i partitionne T en $T = \cup_{v_i} T_{v_i}$, chaque ensemble T_{v_i} a sa propre erreur $e(T_{v_i})$ et l'erreur attendue après un branchement sur cet attribut (pour un élément tiré uniformément au hasard de T) est

$$e_i \;=\; \sum_{v_i} \frac{|T_{v_i}|}{|T|} e(T_{v_i}).$$

L'attribut qui minimise cette erreur est (heuristiquement) considéré comme le meilleur choix possible.

4 Extension des arbres de décision

Les arbres tels que nous venons de les aborder présentent plusieurs limitations :

- le problème d'optimisation global est NP-complet pour de nombreux critères d'optimalité [6], conduisant à l'emploi d'heuristiques ;
- la procédure d'apprentissage est statique : ils ne sont pas prévus pour apprendre de manière incrémentale de nouvelles instances qui viendraient s'ajouter à l'ensemble d'entraînement ;
- ils sont sensibles au bruit et ont une forte tendance à surapprendre les données, *i.e.* à apprendre à la fois les relations entre les données et le bruit présent dans l'ensemble d'apprentissage.

Il existe des solutions à ces problèmes qui travaillent directement sur les arbres, notamment l'élaguage [7, 5]. En pratique, on choisit souvent de « bagger » (voir ci-après) les arbres plutôt que de les utiliser comme prédicteurs individuels, un contexte dans lequel le surapprentissage et la sous-optimalité dû aux heuristiques posent moins de problèmes.

4.1 Bagging

Le « bagging », acronyme pour « **b**ootstrap **agg**regat**ing** », est un méta-algorithme qui combine les deux techniques sus-acronymées :

Bootstrapping : un bootstrap d'un ensemble T est l'ensemble obtenu en tirant $|T|$ fois des éléments de T uniformément au hasard et *avec remise*. Le bootstrapping d'un ensemble d'entraînement T produit un nouvel ensemble T' qui présente en moyenne $1 - e^{-1} \approx 63\%$ instances uniques différentes de T quand $|T| \gg 1$.

Aggrégation : on produit plusieurs bootstraps $T'_1, \ldots, T'_m$, chaque bootstrap T'_i étant utilisé pour entraîner un prédicteur t_i (penser ici à un arbre de régression, mais la technique s'applique à n'importe quelle famille de prédicteurs). Étant donnée une instance $(\mathbf{x}, y)$, on fait régresser chaque arbre, ce qui nous donne un ensemble de valeurs $y_1, \ldots, y_m$ prédites. Celles-ci sont alors aggrégées en calculant leur moyenne $\bar{y} = \frac{1}{m} \sum_i y_i$.

Le bagging corrige plusieurs défauts des arbres de décision, notamment leur instabilité (de petites modifications dans l'ensemble d'apprentissage peuvent entraîner des arbres très différents) et leur tendance à surapprendre. La contrepartie à payer est une perte de lisibilité : les prédictions d'une forêts d'arbres « baggés » ne sont plus le fruit d'un raisonnement, mais un consensus de raisonnements potentiellement très différents.

L'analyse théorique du bagging demeure incomplète à ce jour : plusieurs arguments aident à comprendre son impact et suggèrent des conditions dans lesquelles il améliore les prédictions, mais l'élaboration d'un modèle dans lequel cet impact est compris et mesurable reste un sujet de recherche.

4.2 Random Forests

Proposées en 2002 par Leo Breiman et Adele Cutler [1], les forêts aléatoires (*Random Forests*) modifient l'algorithme 1 pour que les arbres construits soient adaptés au bagging. Les principales modifications sont les suivantes :

Échantillonage des attributs : à la ligne 4 de l'algorithme 1, au lieu de choisir l'attribut i dans l'ensemble $A = \{1, \ldots, m\}$, on tire uniformément au hasard un sous-ensemble $A' \subset A$ de taille $m' \leq m$ (pour la régression, les auteurs suggèrent $m' \approx \frac{m}{3}$) dont on choisira le meilleur attribut. L'objectif de cette opération est de décorréler les arbres produits, la corrélation entre les prédicteurs réduisant les gains potentiels en précision dus au bagging.

Critère d'arrêt : les arbres construits sont aussi profonds que possible, *i.e.* le critère d'arrêt est une condition forte ($|T_f| < 10$, $|\mathcal{Y}_f| = 1$, ...), dans les limites d'un temps de calcul raisonnable. Ainsi, un prédicteur apprendra toutes les relations entre les données à sa portée, le surapprentissage et la sensibilité au bruit qui en résultent étant compensés par le bagging.

Pour une description plus approfondie des forêts aléatoires et des outils qui les accompagnent, voir [1, 5].

Conclusion

Vous trouverez un exemple d'implémentation en Python/SQLite de l'intégralité de ces notions dans la bibliothèque PyDTL [3], open source et disponible sous

licence LGPL. Celle-ci s'inspire aussi bien de CART [2] que de C4.5 [10, 7] tout en adaptant la construction aux forêts aléatoires. Certains logiciels intègrent également des modules pour les arbres de décision, notamment *Weka* [9], *milk* [4] et *Orange* [8].

Ici ce termine notre introduction à l'apprentissage d'arbres de décision. Il y a sans doute de nombreux points à améliorer, et je serais enchanté d'y remédier si vous trouvez le temps de m'écrire pour me les signaler. Pour une étude plus approfondie de ces techniques, ou plus généralement de l'apprentissage supervisé et du *Data Mining*, je vous recommande vivement la lecture des *Elements of Statistical Learning* de HASTIE et al. [5].

Licence

Références

[1] Leo Breiman. Random forests. *Machine Learning*, 45(1) :5–32, 2001.

[2] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1 edition, January 1984.

[3] Stéphane Caron. Pydtl – decision tree learning for python. `http://scaron.info/pydtl`.

[4] Luis Pedro Coelho. milk – machine learning toolkit. `http://pypi.python.org/pypi/milk`.

[5] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, corrected edition, July 2003.

[6] Laurent Hyafil and R. L. Rivest. Constructing Optimal Binary Decision Trees is NP-complete. *Information Processing Letters*, 5(1) :15–17, 1976.

[7] Ron Kohavi and Ross Quinlan. Decision tree discovery. In *Handbook of Data Mining and Knowledge Discovery*, pages 267–276. University Press, 1999.

[8] University of Ljubljana. Orange. `http://orange.biolab.si`.

[9] University of Waikato. Weka. `http://www.cs.waikato.ac.nz/~ml/weka`.

[10] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1) :81–106, 1986.