Complexité temporelle

Antoine Gennart

23 novembre 2015

1 BuildDecisionTree

1.1 Cas d'un arbre non optimal

L'arbre non-optimal lit une liste de question toujours dans le même ordre quel que soit la base de donnée, à chaque noeuds de l'arbre pose une question dont la réponse peut être soit *true* soit *false*. Il insère les feulles (*leaf()*) seulement après avoir lu toutes les questions.

La complexité temporelle de cet arbre est de $O(n) = 2^n$. Effectivement, il y a n questions binaire différentes, laissant à chaque noeuds deux choix.

1.2 Cas d'un arbre optimal

Dans la constructions de l'arbre optimal, on devrait être capable de lire les questions directement depuis la base de donnée (malheureusement je ne sais pas comment faire cela). Cela permettrait de nous fournir la liste des questions quelle que soit la base de donnée donnée.

Pour la construction de cet arbre, nous allons créer une sous fonction qui va déterminer combien de *person* vont répondre true à une question et combien vont répondre false. Au plus la différence en valeur absolue de ces deux valeurs est petite, au plus nous seront intéressé par poser la questions (de cette manière, quelque soit la question nous seront sur d'éliminer un maximum de candidats).

Un problème se pose : Il faudrait réitérer l'opération de savoir quelle questions à la plus petite différence en valeur absolue de réponse true et false. Cela aurait pour conséquence qu'on ne posera pas les questions dans le même ordre pour ceux qui auraient répondu true à une question que pour ceux qui y aurait répondu false. Rien de bien grave me diriez vous, mais cela va accentuer la difficulté de la tâche de la gestion du bouton *unknown*.

Partons coder cet algorithme ....

2 GameDriver

3 Cas basique

Le GameDriver basique ne donne que le choix true ou false. Il suffit donc de se ballader dans l'arbre en répondant à chaque fois true ou false. La complexité temporelle de cet arbre (associé à l'arbre construit par le BuildDecisionTree non optimal) est donc : $\Theta(n)$

Si il est associé à l'arbre construit par le BuildDecisionTree optimal : $O(n) = n$ $\Omega(n) =$ plus petite branche de l'arbre

4 Cas Améliorer +++++

La complexité de cette fonction augmente, car elle augmente le nombre de choix que l'utilisateurs peut faire, et ajoute un *degré de liberté* au programme.

4.1 Le bouton OOPS

Ce bouton augmente la complexité temporelle de la fonction de 1, étant donné quelle est defini à un multiple près, ceci ne change rien à la complexité temporelle.

4.2 le bouton "Je ne sais pas"

Ce bouton augmente la complexité temporelle car il demande d'additionner les deux arbre