Machine Learning et Scoring à valeurs discrètes

Félix Baschenis

23 Juin 2010

Table des matières

1 Introduction

Mon stage de L3 s'est effecuté au CMLA, sur un sujet proposé par Nicolas Vayatis. Il s'agissait de plusieurs sujets d'études, concentrés autour de l'ordonnancement, c'est à dire de classer un ensemble continu à partir d'un nombre fini de notations sur des éléments pris au hasard dans cet ensemble. Ce domaine fait écho au Challenge Netflix [8], dont nous parlerons plus

tard, et à plusieurs articles de Stéphan Clémençon et Nicolas Vayatis [2], [3] concernant l'ordonnançement binaire (les notes sont prises dans $\{-1, 1\}$), ayant nottament débouchés sur l'implémentation d'un algorithme de scoring via une interface R, par Nicolas Baskiotis. Je me suis plus particulièrement interessé à l'exploitation de cet algorithme, pour traiter un domaine peu abordé, l'ordonnancement à valeurs discrètes (les notes appartiennent à un ensemble discret ordonné, par exemple $\{1, 2, 3, 4, 5\}$).

1.1 Le Challenge Netflix

Le Challenge Netflix [8] est un concours organisé par une agence américaine de location de vidéo, qui a proposé une belle récompense en échange de la recherche et la production d'un algorithme meilleur que le leur. L'objectif de cet algorithme est de proposer de nouveaux films à louer à des utilisateurs réguliers. Les équipes du concours disposaient d'un base de données de notations (un client donne une note à film, typiquement entre 1 et 5) et devaient prédire quels films proposer à un client ayant déjà visionné et noté certains films.

Le problème était originellement décrit comme une question de prédiction, mais je l'étudierais comme un problème de scoring : je ne cherche pas à prédire la note d'un film pour un utilisateur quelquonque, mais à ordonner l'ensemble de mes données à partir d'un échantillon restreint.

1.2 Formalisation du problème

Ici, je verrai les données à trier comme des éléments $X \in \mathbb{R}^{\mathrm{d}}$. Dans le cas de Netflix, on peut par exemple imaginer que chaque film est décrit par d caractéristiques (comme le genre, le réalisateur, la date de sortie, etc...) et que chacune d'entre elles est représentée sur la droite réelle. Les notes sont elles des éléments $Y \in \{1, 2, 3, 4, 5\}$. L'algorithme recoit en entrée un nombre fini de notations, qui sont des couples (X_i, Y_i)

L'objectif est de donner une fonction de score $s : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}$. Cette fonction induit un ordre sur $\mathbb{R}^{\mathrm{d}}$:

Définition 1. $\forall\ s : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}$ on définit $<_s$ tel que

$$X_1 <_s X_2 \iff s(X_1) < s(X_2)$$

On dit alors que X_1 est plus petit que X_2 selon s.

Pour que l'algorithme soit efficace, il faut que la fonction de score renvoyée soit cohérente avec les notes obtenues. Ainsi, pour deux éléments X_1 et X_2

on souhaite que X_1 soit plus petit que X_2 selon s si et seulement si X_2 est probablement mieux noté que X_1. Bien que correctement maitrisée dans le cas binaire, cette notion de "probablement mieux noté" est assez flou dans le cas où $Y \in \{1, 2, 3, 4, 5\}$, et je la détaillerai plus tard. C'est d'ailleurs tout le sens du travail présenté ici, mais j'aurai d'abord besoin de m'intéresser à une méthode de résolution du cas binaire.

2 L'Algorithme TreeRank

Développé par Stéphan Clémençon et Nicolas Vayatis [3], et implémenté dans R par Nicolas Baskiotis, cet algorithme de TreeRanking fonctionne dans le cadre binaire, où $Y \in \{-1, 1\}$. C'est un algorithme d'apprentissage, qui se base donc sur un critère optimisé à chaque étape de l'algorithme (principe du Machine Learning) et qui fournit un arbre de tri.

2.1 La courbe ROC et l'AUC [1], [5]

Soit (X, Y) un couple de variables aléatoires, $X \in \mathbb{R}^{\mathrm{d}}$ et $Y \in \{-1, 1\}$. On définit la loi de probabilité η telle que :

$$\eta(x) = P(Y = +1 \mid X = x)$$

La courbe ROC (pour Recever Operating Characteristic) d'une fonction de scoring s permet de vérifier que les données d'étiquette $+1$ sont le plus possible en haut de la liste. On rappelle que les Y_i sont dans $\{-1, 1\}$, ce qui est cohérent avec l'idée de "bien triée". Pour cela on définit les taux de faux positifs et de vrais positifs.

Définition 2. On définit τ_+ et τ_- respectivement les taux de *vrais positifs* et de *faux positifs* tel que :

$$\tau_+(t) = P(s(X) \geq t \mid Y = +1)$$

$$\tau_-(t) = P(s(X) \geq t \mid Y = -1)$$

La courbe ROC met en parallèle ces deux taux, afin de visualiser les "erreurs" commises par la fonction s

Définition 3. On définit la courbe paramétrée ROC_s tel que :

$$ROC_s : t \mapsto (\tau_-(t), \tau_+(t))$$

Remarque. – Pour toute fonction de score s, ROC_s est de classe C^0, différentiable et concave.

– Son graphe est contenu dans le carré $[0,1] \times [0,1]$ car son abcisse et son ordonnée sont des probabilités.

– En $t = +\infty$, $ROC_s(t) = (0,0)$ et en $t = -\infty$, $ROC_s(t) = (1,1)$ car $\forall\ X \in \mathbb{R}^{\mathrm{d}}\ s(X) \in \mathbb{R}$ donc $P(s(X) \geq -\infty) = 1$ et $P(s(X) \geq +\infty) = 0$.

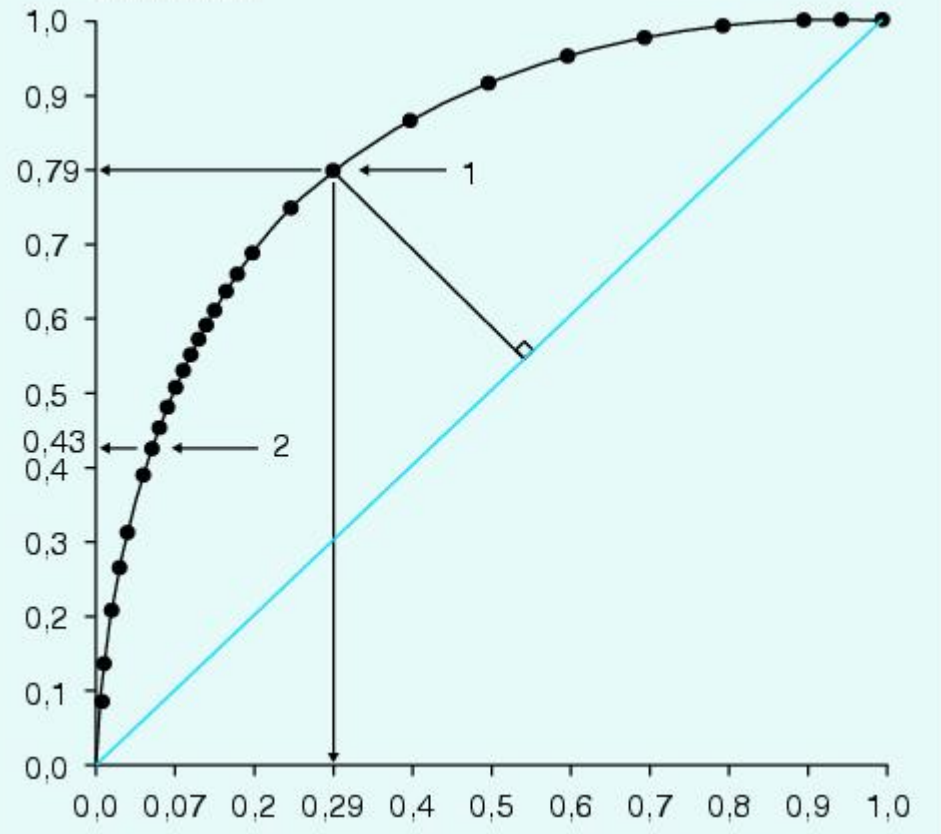

FIGURE 1 – Exemple d'une courbe ROC

La Figure 1 montre deux courbes ROC. Celle en bleu représente la courbe associée à la fonction de score qui répartir aléatoirement les X_i, indépendament des Y_i puisque $\forall t \in \mathbb{R}\ \tau_+(t) = \tau_-(t)$ ce qui signifie que $P(s(X) \geq t)$ ne dépend pas du label de X. En noir, on a la courbe ROC d'une fonction de score s puisque $\forall t \in \mathbb{R}\ \tau_+(t) \geq \tau_-(t)$, alors qu'on cherche justement à maximiser $\frac{\tau_+(t)}{\tau_-(t)}$.

Je vais maintenant montrer comment utiliser la courbe ROC comme critère d'optimisation. D'abord, la proposition suivante illustre parfaitement l'idée que la courbe ROC traduit l'ordre induit par la fonction s. Sa démonstration

est d'ailleurs immédiate.

Proposition 1. *Soit π une fonction croissante de $\mathbb{R} \to \mathbb{R}$. Alors :*

$$ROC_{\pi \circ s} = ROC_s$$

On peut également remarquer que la courbe ROC d'une fonction de score s est le graphe d'une fonction $f_s : [0,1] \to [0,1]$. On note $ROC(s,x) = f_s(x)$. Cette vision des choses permet de donner un ordre partiel sur les fonctions de score, interprétant l'idée qu'une fonction de score est "bonne" si elle maximise le rapport $\frac{\tau_+(t)}{\tau_-(t)}$.

Définition 4. On dit que s_1 est *uniformément meilleure* que s_2, note $s_1 \geq_{ROC} s_2$ si :

$$\forall x \in [0,1] \; ROC(s_1,x) \geq ROC(s_2,x)$$

On peut maintenant établir un théorème extrèmement important, dont la preuve découle du lemme de Neyman-Pearson.

Théorème 1.

$$\forall \; s : \mathbb{R}^d \to \mathbb{R} \quad \eta \geq_{ROC} s$$

Remarque. D'apres la proposition 1, on déduit qu'on peut remplacer η par n'importe quelle transformée strictement croissante de η.

Ce théorème permet donc de montrer l'existence d'une fonction de score optimale, qui est logiquement η, puisqu'étant la fonction de score *à posteriori*, se basant sur une connaissance parfaite de la répartition des données. On peut ainsi développer un critère d'optimisation de notre fonction de score s, en cherchant à la faisant tendre vers η. On va procéder par itération, mais faire en sorte que s_{i+1} soit *uniformément meilleure* que s_i étant impossible, on se contentera d'optimiser un critère global, rendre s_{i+1} *globalement meilleure* que s_i, en prenant en compte l'aire sous la courbe, appelé AUC.

Définition 5. On considères donc :

$$AUC(s) = \int_0^1 COR(s,x)dx$$

Proposition 2. *On remarque que l'AUC peut s'exprimer de manière probabiliste : Soit (X,Y) et (X',Y') deux variables indépendantes. Alors :*

$$AUC(s) = P\Big(s(X) < s(X') \mid (Y,Y') = (-1,+1)\Big) + \frac{1}{2}P\Big(s(X) = s(X') \mid (Y,Y') = (-1,+1)\Big)$$

On peut maintenant s'intéresser à l'algorithme de TreeRanking en luimeme qui optimise l'AUC a chaque itération.

2.2 Présentation de l'algorithme

Le TreeRanking est un algorithme d'apprentissage (Machine Learning) implémenté dans R par Nicolas Baskiotis. Il reçoit en entrée un echantillon d'apprentissage $D_n = \{(X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n)\}$ provenant du couple de variables aléatoires (X, Y) et renvoie un arbre de tri par partitions successives, en optimisant à chaque étape l'AUC. Comme on effectue un partionnement récursif, on introduit les taux empiriques de faux positifs et de vrais positifs d'un classifieur qui vaudrait toujours +1 sur chaque sous-ensemble de $\mathbb{R}^{\mathrm{d}}$.

Définition 6. Soit C un sous-ensemble de $\mathbb{R}^{\mathrm{d}}$. On définit $\hat{\alpha}(C)$ et $\hat{\beta}(C)$ tel que :

$$\hat{\alpha}(C) = \frac{1}{n_-} \sum_{i=1}^{n} \mathbb{I}\{X_i \in C \ , \ Y_i = -1\}$$

$$\hat{\beta}(C) = \frac{1}{n_+} \sum_{i=1}^{n} \mathbb{I}\{X_i \in C \ , \ Y_i = +1\}$$

où $n_- = \sum_{i=1}^{n} \mathbb{I}\{Y_i = -1\}$ et $n_+ = \sum_{i=1}^{n} \mathbb{I}\{Y_i = +1\}$ représentent respectivement le nombre d'échantillons à label négatifs et positifs.

Ces définitions sont intéressantes car elles montrent qu l'algorithme cherche à séparer l'ensemble C en sous ensembles pour lesquels $\alpha(C)$ soit grand d'un coté, et petit de l'autre. Cependant, je ne détaillerai pas l'algorithme, qui est expliqué en détail dans l'article de Stéphan Clémençon et Nicolas Vayatis. Il est quand même bon de regarder ce que l'algorithme fournit en sortie.

2.3 Interprétation des résultats

La sortie du TreeRanking est un arbre binaire orienté. C'est une structure classique d'algorithmique qui permet d'organiser un ensemble de données suivant des règles et un ordre précis. Si D est la profondeur de l'arbre et $k \in \{0, \cdots, 2^D - 1\}$, le noeud de profondeur D et occupant la $k^{\text{ème}}$ place en partant de la gauche correspond à un ensemble, noté $C_{D,k}$. Chaque ligne, c'est à dire l'ensemble $\{C_{D,k} \mid k \in \{0, \cdots, 2^D - 1\} \}$ pour une profondeur D donnée est une partition de l'ensemble C des films. De plus, l'algorithme garantit l'égalité suivante :

$$\forall D \ \forall k \in \{0, \cdots, 2^D - 1\} \ C_{D,k} = C_{D+1,2k} \cup C_{D+1,2k+1}$$

Ainsi l'algorithme sépare bien chaque noeuds en deux fils qui forment une partition de l'ensemble associé à leur père. La figure 2 montre un tel arbre, de profondeur $D = 4$.

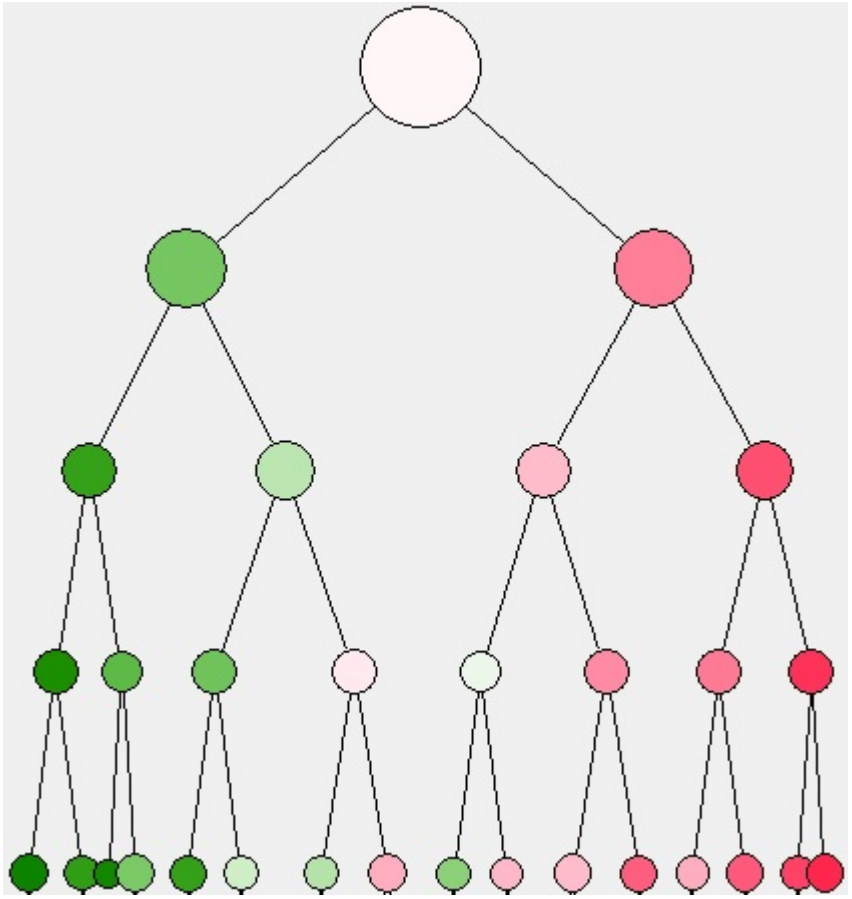

FIGURE 2 – Exemple d'un arbre de TreeRanking

Chaque noeud est coloré, par une teinte allant du vert foncé au rose pourpre. Elle dépend du rapport entre le nombre d'éléments de labels positifs et négatifs contenus dans l'ensemble correspondant au noeud coloré. Une couleur verte (*respectivement* rose) dans le noeud C indique que $n_{C,+} = \sum_{i=0}^{n} \mathbb{I}\{X_i \in C \ , \ Y_i = +1\}$ est très grand (*respectivement* petit) devant $n_{C,-} = \sum_{i=0}^{n} \mathbb{I}\{X_i \in C \ , \ Y_i = -1\}$. Les noeuds correspondants aux ensembles contenant des données biens classées doivent donc être le plus possible vert.

Par ailleurs, l'arbre de tri renvoyé par l'algorithme définit implicitement une fonction de score. Le TreeRanking donne la partition $\{C_{D,k} \mid k \in \{0, \cdots, 2^D - 1\}\}$, on donne le même rang à tout les éléments de chaqun des $C_{D,k}$. Comme l'algorithme sépare chaque noeud entre ensembles bien notés et mal notés, et place l'ensemble des noeuds bien notés comme le fils gauche, et l'ensemble des noeuds mal notés comme le fils droit, on trie la partition de gauche à droite, de cette manière :

Définition 7. Soit T un arbre de TreeRanking. On définit la fonction de score :

$$s_T(X) = s_T(X) = \sum_{k=0}^{2^D-1} (2^D - k)\mathbb{I}\{X \in C_{D,k}\}$$

Enfin, l'algorithme nous donne aussi la manière dont les partitions sont faites, ce qui permet de placer facilement un nouvel élément dans l'ensemble de la partition qui lui convient. La Figure 3 donne un exemple d'arbre de ce type, LeafRank, associé a l'arbre de la figure 2 .

J'ai ainsi détaillé les résultats de l'algorithme de TreeRanking, dans le cas binaire, c'est à dire où $Y_i \in \{-1, 1\}$. Je vais maintenant montrer l'utilisation de cet algorithme dans le cas où $Y_i \in \{1, 2, 3, 4, 5\}$.

3 Utilisation du TreeRanking pour le Scoring à valeurs discrètes

L'idée générale est d'utiliser un critère adapté, le DCG, combiné à l'algorithme de TreeRanking.

3.1 Le DCG : Discounted Cumulated Gain [7]

Le DCG est un outil permettant d'évaluer l'efficacité d'une fonction de scoring, lorsque ses éléments ont des labels à valeur dicrète. Le but est de donner plus d'important aux éléments bien classés qu'aux éléments mal classés, et donc de négliger les erreurs faites en bas de liste, pour améliorer les résultats au début de celle-ci. Cette démarche est pertinente puisqu'elle traduit souvent les besoins de l'utilisateurs, dans des domaines de l'ordonnancement aussi variés que l'aide au diagnostic ou les moteurs de recherche : on ne s'interesse que rarement à la $10^{\text{ème}}$ page de la réponse à une requête sur un moteur de recherche. L'idée est de pénaliser l'important des éléments mal classés dans l'évaluation de la fonction de score, par des poids, souvent définis de manière logarithmique, ce qui donne la définition naturelle suivante :

Définition 8. Soit $s : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}$ une fonction de score. En réorganisant les X_i suivant s, tel que $X_1 >_s X_2 >_s \cdots >_s X_n$ on définit :

$$DCG_s(X_i) = \begin{cases} Y_i & \text{si} \quad i = 1 \\ DCG_s(X_{i-1}) + \frac{Y_i}{\log(i)} & \text{sinon} \end{cases}$$

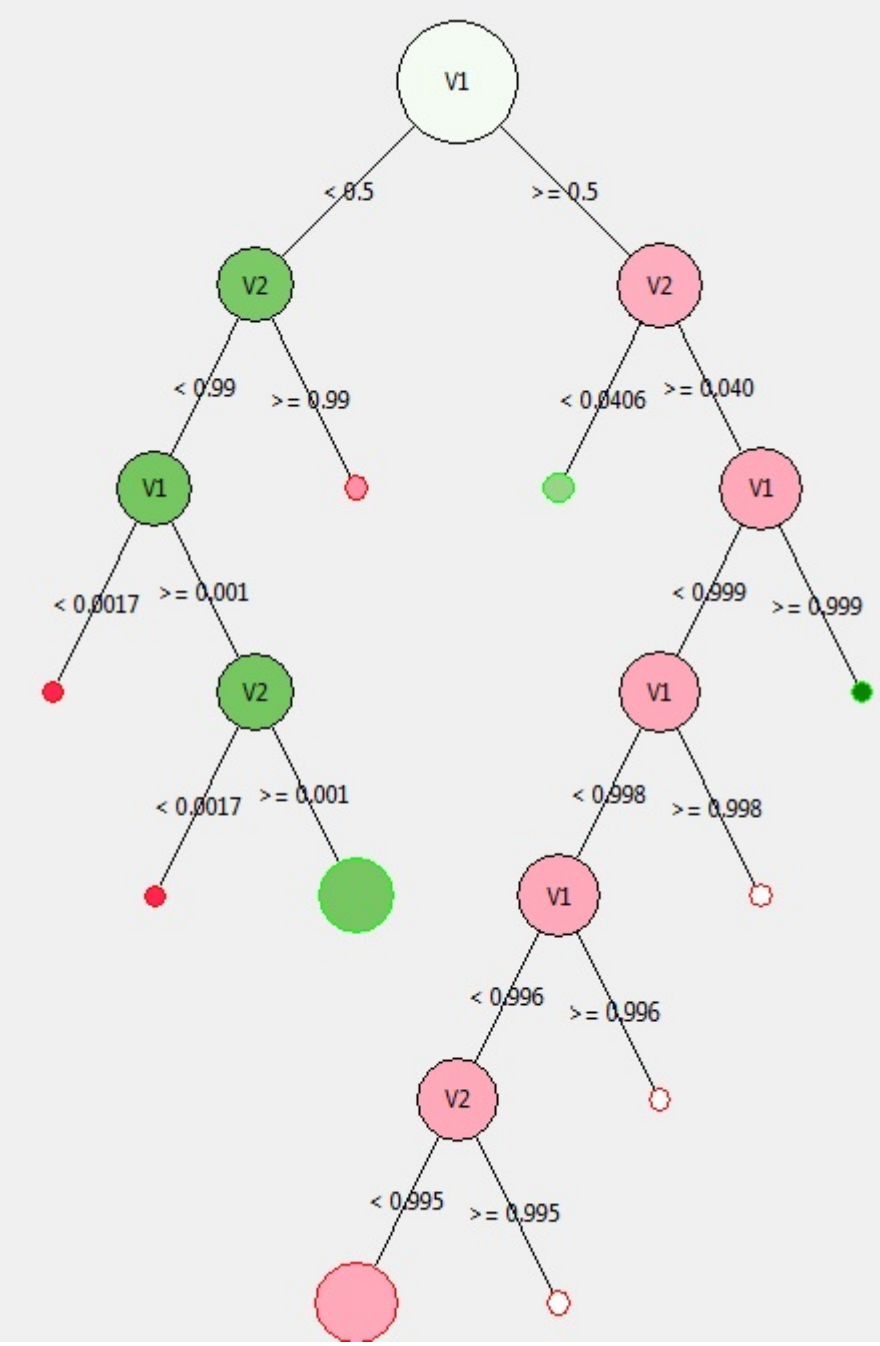

FIGURE 3 – Exemple d'un LeafRanking

3.2 Méthode de résolutions

Pour pouvoir appliquer l'algorithme de TreeRanking, on a besoin de labels binaires. Il faut donc transformer notre jeu de données, pour passer de labels

discrets à des labels binaires. On réalise cela en posant des limites arbitraires. On crée ainsi un nouveau jeu de données, utilisables par l'algorithme.

Définition 9. Soit $D = \{(X_i, Y_i) \mid 1 \leq i \leq n\}$ un jeu de données. On construit alors les jeus de données D_j pour $j \in \{1, 2, 3, 4\}$:

$$D_j = \{(X_i, -1) \mid (X_i, Y_i) \in D \wedge Y_i \leq j\} \cup \{(X_i, +1) \mid (X_i, Y_i) \in C \wedge Y_i > j\}$$

Remarque. Considérer l'ensemble D_5 n'a aucun interêt puisqu'alors toutes les données auraient un label négatif ce qui conduirait nécessairement à une fonction de score classant tout les films de la même manière.

Une fois que nous possédons les ensembles D_j on peut appliquer l'algorithme de TreeRanking à ces ensembles, ce qui nous donne quatre fonctions de score différentes, notées s_j. On applique ensuite le principe du DCG à ces fonctions de score afin d'en générer une nouvelle, prenant en compte toutes les valeurs possibles du label discret.

On remarque que la fonction de score s_1 ne donne des informations que sur les films de label 1 : tout les autres sont confondus. De même s_4 ne donne des informations que sur ceux de label 5. Or, le principe du DCG veut que l'on privilégie les *bons* films, et donc ceux qui sont le mieux notés. Ainsi, le classement selon la meilleur note, c'est à dire s_4 est plus important que les autres. Ici, on utilisera des poids logarithmiques pour traduire cette prévalence.

Définition 10. Soit D un ensemble de données à valeurs discrètes. En utilisant les notations précédentes, on définit :

$$s(X) = \sum_{i=1}^{4} \frac{s_i(X)}{\log(6-i)} = s_4(X) + \frac{s_3(X)}{\log(3)} + \frac{s_2(X)}{\log(4)} + \frac{s_1(X)}{\log(5)}$$

Cette définition permet de répondre à la question initiale, qui est de donner une fonction de score pour un ensemble à labels discrets.

3.3 Simulations

La partie technique des simulations réside dans la mise en place des jeu de données. J'ai utilisé la base de données de Movieslens, qui fournit des ensembles de cent milles, un million, ou dix millions de notations de films, sous la forme de trois tableaux. Le premier représente les notes, c'est un tableau à trois colonnes, qui correspondent à l'identifiant du film, l'identifiant de l'utilisateur, et la note attribuée. Les deux autres, correspondent aux descriptions des utilisateurs et des films à l'aide de leur identifiant.

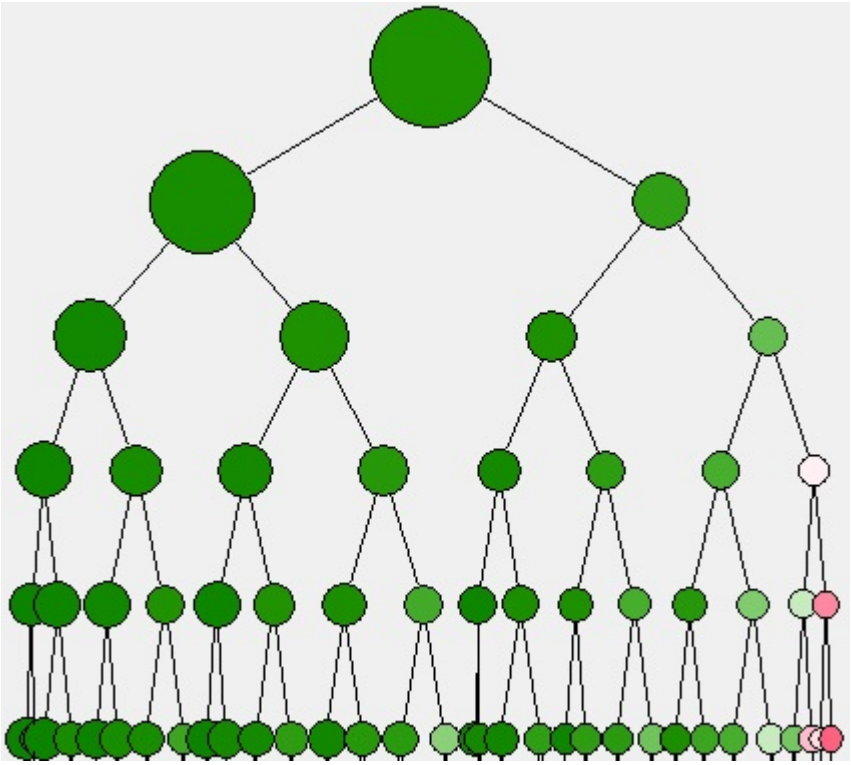

FIGURE 4 – L'arbre T_1

Chaque film est décrit comme un vecteur de booléens, chaque champ correspond à un genre de film, et la valeur du film en ce champ nous dit si le film est de tel genre. Cela permet à un film de répondre à plusieurs critères en même temps, et donne une plus grande souplesse d'utilisation. Pour les utilisateurs, je disposais de trois caractéristiques, l'âge, le genre et la profession, qu'il a fallu coder de manière cohérente par des chiffres, l'algorithme ne prenant en entrées que des nombres.

Il a fallu ensuite fusionner ces trois tableaux, le TreeRanking ne prenant qu'une seule table en entrée, et la modifier pour créer quatre tables à valeurs binaires, exploitables par le TreeRanking. Comme j'ai travaillé sur le jeu de cent milles données, il m'a fallu utiliser (et apprendre !) un langage de gestion de bases de données, le *SQL* afin de fusionner les tableaux, et de leur donner l'aspect souhaité. Une fois cela réalisé, j'ai lancé l'algorithme sur ces quatres tables, ce qui m'a donné les résultats présentés dans les figures 4 ,5, 6, 7.

Remarque. On remarque logiquement, que si j augmente, l'arbre de Ranking associé passe du vert au rose, puisqu'il y a de moins en moins de labels

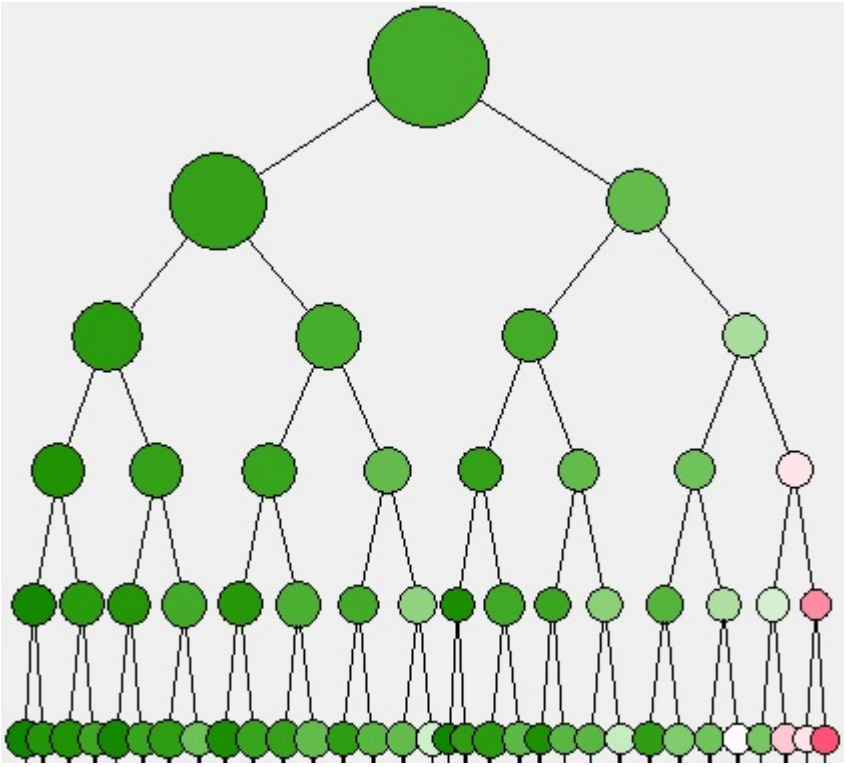

FIGURE 5 – L'arbre T_2

positifs, et de plus en plus de labels négatifs, ce qui se traduit par la couleur générale de l'arbre.

Ces simulations donnent immédiatement les fontions de scores associées s_j, en parcourant l'arbre des conditions, pour placer X dans le bon noeud, puis par l'addition décrite dans la partie précédente, à la fonction de score globale s. La dernière difficulté est de construit l'arbre global T associé à cette fonction de score s afin de trier faciler les nouvelles données. Cependant ceci reste difficile à faire.

3.4 Construire l'arbre associé à la fonction de score trouvée

On rappelle que les arbres disposent à chaque noeud d'une condition qui permet de décider dans quel ensemble de la partition on va pouvoir placer un nouveau film X. Pour construire l'arbre T, il s'agit en fait d'organiser ces conditions entres elles. Cela est assez simple si la profondeur des arbres T_i est de un. En effet, à chaque arbres correspond une condition C_i, et comme

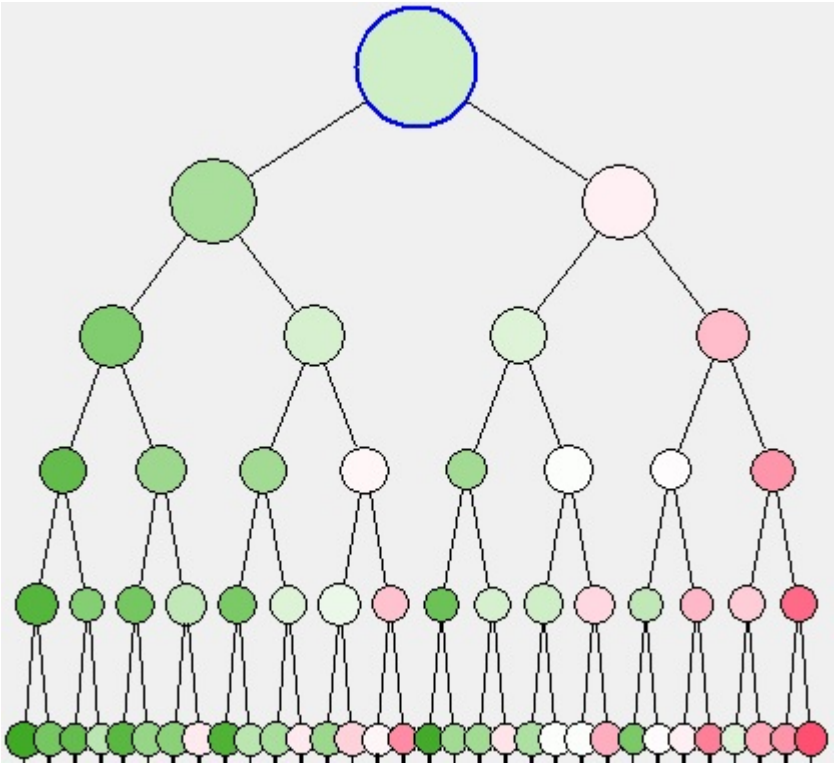

FIGURE 6 – L'arbre T_3

on a déjà décider de privilégier l'arbre T_4, on va naturellement privilégier la construction C_4 puis C_3 , etc...

Cependant, si les profondeurs sont plus grandes, il y a plus de conditions par arbre, et il est impossible de les organiser suivant la fonction de scoring s. En effet si on définit un ordre d'importance sur les conditions $C^j_{i,k}$ où j est le numéro de l'arbre, i est la profondeur du noeud en question et k sa place dans les noeuds de T_j de profondeur i, cet ordre définit certes un arbre T mais ne prendre pas en compte les places dans les arbres T_i mais le chemin parcouru, ce qui peut créer des anomalies.

4 Perspectives et Conclusion

Les perspectives de cette étude sont doubles. Premièrement, il reste du travail concernant l'élaboration d'un arbre de tri T, même si ce que j'ai présenté ci-dessus laisse penser qu'elle ne serait pas *naturelle*. Deuxièmement, on peut imaginer des routines permettant de comparer notre procédure algorithmique avec d'autres algorithmes traitant de l'ordonnançement discret.

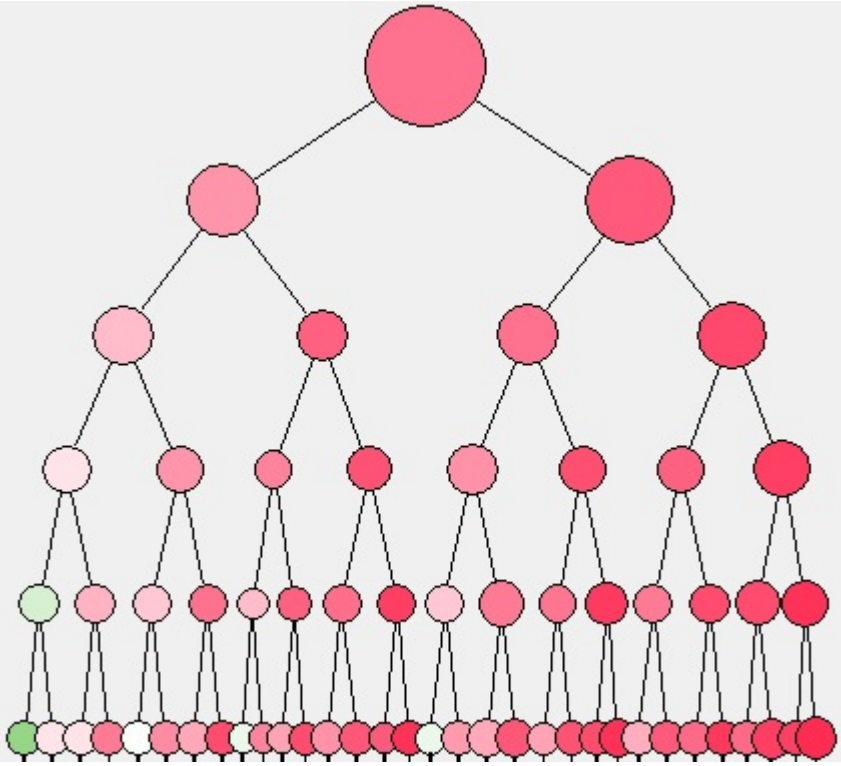

FIGURE 7 – L'arbre T_4

Mais le sujet est assez peu traité, car on lui préfère souvent l'ordonnançement binaire, qui met en oeuvre des méthodes algorithmes plus sophistiquées.

Cependant, ce travail m'a permi de découvrir un grand nombre de choses. Tout d'abord, un domaine des mathématiques que l'on n'avait qu'entraperçu en cours d'Intégration et Probabilités, qui se révèla très intéressant. Il faut prendre en compte la dimension découverte de la recherche, qui nécessita un gros travail bibliographique et de réflexion. Enfin, j'ai du aussi apprendre à utiliser R et découvrir les bases du SQL ce qui peut toujours etre utile. Tout ceci pris en compte, ce stage a développé une compréhension et une réflexion sur la recherche, tout en étant une expérience instructive.

5 bibliographie

Références

[1] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learnin algorithms. *Pattern Recognition*, 1997.

[2] Stéphan Clémençon and Nicolas Vayatis. Ranking the best instances. *J. Mach. Learn. Res.*, 8 :2671–2699, 2007.

[3] Stéphan Clémençon and Nicolas Vayatis. Tree-based ranking methods. *IEEE Trans. Inf. Theor.*, 55(9) :4316–4336, 2009.

[4] Persi Diaconis. A generalization of spectral analysis with application to ranked data. *The Annals of Stratistics*.

[5] Pinar Donmez and Jaime G. Carbonell.

[6] GroupLens. Movielens data set. http ://www.grouplens.org/node/12, Mis en ligne le5 obtobre 2006.

[7] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR '00 : Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48, New York, NY, USA, 2000. ACM.

[8] Netflix. Netflix prize. http ://www.netflixprize.com//rules, Concours commenÃ§Ã© le 2 obtobre 2006.