Modèles d'ordonnancement pour le résumé automatique et la recherche d'information

Tuong Vinh Truong, David Buffoni, Nicolas Usunier,
Massih–Réza Amini, Patrick Gallinari

25 février 2011

1 Introduction

La plupart des travaux en apprentissage se sont focalisés sur des problèmes issus des cadres de la classification et de la régression. Le but est d'apprendre une fonction de prédiction, qui va induire, pour chaque entrée, une valeur en accord avec une sortie désirée. Depuis le début des années 2000, un troisième cadre suscite un intérêt croissant dans les communautés d'apprentissage et de recherche d'information : *l'apprentissage de fonctions d'ordonnancement*. Ces fonctions considèrent plusieurs entrées, les comparent, et les renvoient sous la forme d'une liste ordonnée. L'ordre prédit doit être en accord avec une notion de préférence, spécifique au problème traité. Dans la littérature, on distingue deux types de problèmes d'ordonnancement :

- l'*ordonnancement d'alternatives* concerne les problèmes où il faut ordonner les éléments (appelés alternatives) d'une collection donnée en fonction d'un exemple d'entrée. Un exemple typique est celui du résumé automatique de texte (dans ce cas, un exemple d'entrée représente un document et les alternatives les phrases des documents). Un autre exemple est celui de la recherche d'information, où les documents d'une collection donnée doivent être ordonnés en réponse à une requête utilisateur (l'entrée est une requête, et les alternatives sont les documents de la collection). Pour ce type de problème, on fait l'hypothèse que les exemples d'entrée sont indépendamment et identiquement distribués (i.i.d.) et le but est de retourner pour chaque exemple une liste ordonnée d'alternatives, de sorte que les alternatives pertinentes par rapport à la donnée d'entrée soient ordonnées avant les alternatives non pertinentes ;
- l'*ordonnancement d'instances* concerne l'ordonnancement d'un ensemble d'exemples d'entrée. Ces exemples sont aussi supposés être i.i.d., mais il s'agit maintenant de les ordonner entre eux. Ce cadre formalise par exemple l'application du routage de documents, où un utilisateur cherche une information de façon stable, et le système dispose d'un certain nombre de documents pertinents et non pertinents par rapport à sa demande. Le but est de trier les nouveaux documents entrants et de les insérer dans la liste des documents existants, de sorte que les documents pertinents se retrouvent ordonnés au–dessus des documents non pertinents.

Dans des applications de recherche documentaire, l'ordonnancement d'alternatives est le cadre formel permettant de traiter les tâches de recherche d'in-

formation où plusieurs requêtes peuvent être envoyées au système (requête *dynamique*), mais la collection de documents est fixée (collection *statique*). Au contraire, l'ordonnancement d'instance modélise le cas où la requête est fixée (requête *statique*), et la collection de documents varie au cours du temps (collection *dynamique*). Du point de vue de l'apprentissage, la différence majeure entre ces deux cadres d'ordonnancement porte sur les entités à ordonner. Dans le cas d'ordonnancement d'instances, il s'agit des exemples d'entrée et dans le cas d'ordonnancement d'alternatives, il s'agit des alternatives associées à chaque exemple d'entrée.

Dans ce chapitre, nous allons présenter ces deux formalismes. Nous allons d'abord introduire les notions importantes sur lesquelles les principaux travaux développés en ordonnancement se sont construits. Nous éviterons néanmoins une description détaillée des méthodes : seul le cadre et les principes de résolution seront abordés. Dans un deuxième temps, nous allons présenter les tâches du résumé automatique et de la recherche d'information, qui sont deux applications emblématiques qui ont conduit au développement de ces modèles.

1.1 Ordonnancement d'instances

Nous traitons en premier l'ordonnancement d'instances. Nous introduisons le formalisme adopté par une majorité de travaux et nous montrons que ce problème peut être reformulé dans le cadre plus classique de la classification binaire. Cette formulation s'inscrit ainsi dans un raisonnement dit de *réduction* [Usu07], qui permet de traiter une tâche compliquée (l'ordonnancement) par des tâches plus simples et mieux maîtrisées (la classification).

1.1.1 Formalisme

Par analogie avec le cadre de classification, nous supposons que les observations à ordonner entre elles et leurs étiquettes souhaitées, sous la forme de scores réels, sont générées selon une distribution inconnue $\mathcal{D}$. Nous noterons $\mathcal{X}$ l'espace des observations et $\mathcal{Y} \subset \mathbb{R}$ l'espace des scores.

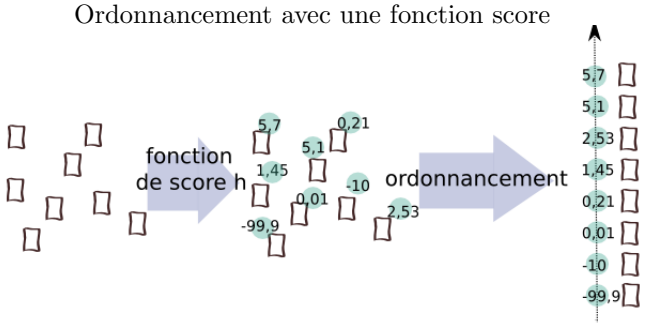
Ainsi pour un ensemble $\mathcal{S} = \{x_i, y_i\}_{i=1}^n$ d'instances étiquetées échantillonnées i.i.d. suivant $\mathcal{D}$, nous considérons un ordre $\succ$ sur les étiquettes $\mathcal{Y}$, qui permet d'exprimer une relation de préférence entre instances. Soient (x_i, y_i) et (x_j, y_j) deux instances étiquetées, $y_i \succ y_j$ signifie que x_i doit être ordonnée au–dessus de x_j.

Fonction score L'apprentissage consiste à trouver une fonction $h : \mathcal{X} \to \mathbb{R}$ qui permet d'ordonner correctement les instances. La sortie de la fonction permet d'induire un ordre sur un ensemble d'exemples. La figure 1.1.1 illustre son utilisation.

Fonction d'erreur L'apprentissage de la fonction score h passe par la définition d'une erreur d'ordonnancement. Pour un ensemble donné d'instances, cette erreur permet de comparer l'ordre induit par la fonction score et l'ordre souhaité[1]. Elle mesure ainsi à quel degré le premier ordre diffère du deuxième.

[1] c'est-à-dire l'ordre induit par la valeur des étiquettes.

Ordonnancement avec une fonction score



Comme nous l'avons souligné plus haut, en ordonnancement, la sortie de la fonction score sert uniquement à comparer les instances entre elles. La valeur absolue du score importe peu. Les fonctions d'erreur d'ordonnancement considèrent ainsi un ensemble de scores et non plus des scores individuellement.

Dans ce chapitre, nous considérons que les fonctions d'erreur sur $\mathcal{S}$, un ensemble de n entrées, pouvent s'écrire sous la forme suivante :

$$L_o :^n \times^n \to^+$$

Autrement dit, la fonction d'erreur va mesurer l'accord entre, d'une part l'ordre induit par les scores renvoyés par la fonction apprise et, d'autre part les scores désirés, nommés aussi jugements de pertinence.

Précisons un peu plus les fonctions d'erreur utilisées en ordonnancement. Nous commençons par un formalisme particulier de l'ordonnancement d'instances : la classification de paires critiques.

1.1.2 Classification de paires critiques

Idéalement, la fonction score recherchée permet d'ordonner correctement les instances de $\mathcal{X}$:

$$\forall (x_i, y_i), (x_j, y_j) \in (\mathcal{X} \times \mathcal{Y})^2, \;\; y_i \succ y_j \Rightarrow h(x_i) > h(x_j)$$

Dans ce cas, l'erreur peut se définir sur les paires d'entrées (x_i, x_j) telles que $y_i \succ y_j$, appelées *paires critiques*. On définit alors une erreur en généralisation d'ordonnancement qui mesure la proportion de paires critiques pour lesquelles l'ordre prédit par h n'est pas l'ordre souhaité [CSS97] :

$$L(h) = \mathbb{E}_{(x_i,y_i)\sim\mathcal{D},(x_j,y_j)\sim\mathcal{D}} \left\{ [\![h(x_i) \leq h(x_j)]\!] \middle| y_i \succ y_j \right\}$$

où $[\![P]\!]$ vaut 1 si le prédicat P est vrai et 0 sinon.

On peut ainsi définir une fonction d'erreur empirique d'ordonnancement notée L_{cp}, estimateur non biaisé de $L(h)$ de sorte que $L(h) = \mathbb{E}_{(X,Y)\sim\mathcal{D}^n} \left\{ L_{cp}\big(h(X), Y\big) \right\}$ par :

$$L_{cp}\big(h(X), Y\big) = \frac{1}{\sum_{i,j} [\![y_i \succ y_j]\!]} \sum_{i,j:y_i \succ y_j} [\![h(x_i) \leq h(x_j)]\!]$$

avec $h(X) = (h(x_1), ..., h(x_n))$ et $Y = (y_1, ..., y_n)$.

Remarquons que cette erreur a une relation intéressante avec la classification binaire. En considérant une paire critique (x_i, x_j), nous pouvons définir une relation binaire avec signe $(h(x_i) - h(x_j))$. Cette relation peut être interprétée comme un classifieur de paires d'entrée et l'erreur L_{cp} peut être vue comme l'erreur de classification des paires critiques[2] commise par la relation binaire.

Notons enfin que dans ce cadre, l'ensemble d'apprentissage, l'ensemble de test et les exemples ont la même forme qu'en classification ou en régression (selon la nature de l'espace $\mathcal{Y}$). La différence vient de la définition de l'erreur en généralisation, qui prend en compte les scores relatifs entre deux observations, et non plus l'accord entre la valeur prédite et la valeur souhaitée.

1.1.3 Application avec un modèle linéaire

Nous considérons maintenant la classe des fonctions linéaires. Ces fonctions sont généralement suffisantes pour traiter les problèmes de recherche d'information. La fonction score prend la forme $h(x) = w^T x$ avec w un vecteur de dimension d. La propriété du produit scalaire permet d'écrire autrement l'erreur sur une paire critique :

$$\begin{aligned} [\![h(x_i) \leq h(x_j)]\!] &= [\![w^T x_i \leq w^T x_j]\!] \\ &= [\![w^T (x_i - x_j) \leq 0]\!] \\ &= [\![h(x_i - x_j) \leq 0]\!] \end{aligned}$$

Sous cette forme, la ressemblance avec l'erreur de classification est évidente. Elle permet d'illustrer la relation binaire que nous venons d'évoquer plus haut. Dans ce cas, la paire est ainsi représentée par la différence des représentations : $x_i - x_j$ et l'ordonnancement interprété comme de la *classification de paires critiques* peut se résoudre en deux étapes :

1. Former l'ensemble des paires critiques $T(\mathcal{S}) = \{(x_i - x_j, 1) | x_i, x_j \in \mathcal{S},\ y_i \succ y_j\}$;
2. Apprendre une fonction de classification sur l'ensemble obtenu $T(\mathcal{S})$

Notons que la classification de paires critiques diffère de la classification classique sur deux points essentiels. D'une part, les paires d'instances ne sont pas indépendantes. Une instance peut en effet intervenir dans plusieurs paires critiques. D'autre part, la nouvelle base d'apprentissage est constituée uniquement d'exemples positifs.

Cette approche a l'avantage d'adapter n'importe quel algorithme de classification en méthode d'ordonnancement. L'utilisation d'un majorant convexe de la fonction indicatrice $t \mapsto [\![t \leq 0]\!]$ permet de définir des erreurs d'ordonnancement optimisables. Le lecteur peut se référer au tableau 1.1.3 et à la figure 1.1.3 pour y retrouver une liste non-exhaustive des principaux majorants convexes associés à une paire critique (x_i, x_j).

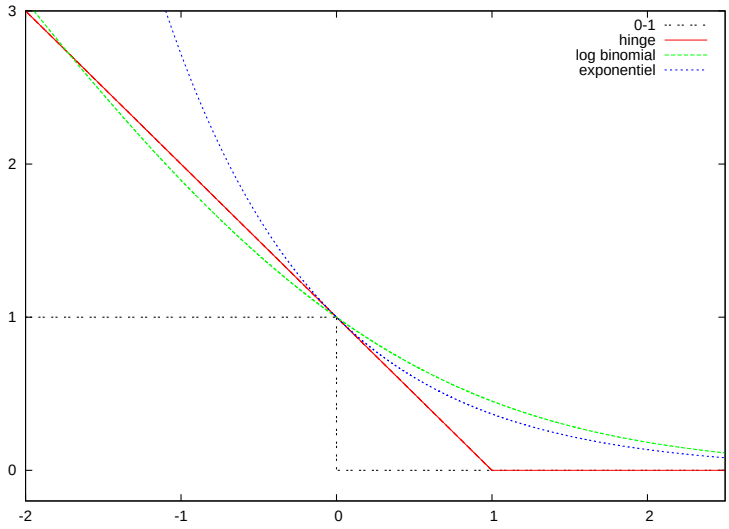
Néanmoins, en termes de tâche de classification binaire, la classification de paires critiques viole l'hypothèse *i.i.d.* des exemples d'apprentissage de $T(\mathcal{S})$ [UTAG05].

[2] En étiquetant les paires critiques avec 1.

Exemples de fonction de perte adaptées à l'ordonnancement

nom	fonctions
exponentielle	$\exp\left(w^T(x_j - x_i)\right)$
régression logistique	$\log\left(1 + exp(w^T(x_j - x_i)\right)$
hinge	$\max\left(1 + w^T(x_j - x_i), 0\right)$

Exemples de fonctions convexes majorants la fonction indicatrice avec en abscisse, la fonction h et en ordonnée l'erreur associée $L(h)$.



La justification théorique de cette méthode s'est donc naturellement posée. Des bornes sur l'erreur en apprentissage ont ainsi été proposées en premier dans un cadre plus restreint : l'ordonnancement biparti où les étiquettes ne prennent que deux valeurs [AGH+05, AN05, AR05, UAG05]. Récemment, des travaux ont concerné un cadre plus général [UTAG05, Usu07]. Notons enfin les travaux de [BBB+08, AM08] réalisés dans un cadre proche mais facilement adaptable pour l'apprentissage de fonctions score. Ainsi, ces travaux permettent de valider l'approche utilisant des paires critiques.

Complexité d'apprentissage et d'inférence La forme naïve de la classification de paires critiques consiste à former les paires critiques et à appliquer une méthode de classification. D'un point de vue algorithmique, la formation de paires est coûteuse. Dans le cas général, il peut y en avoir $O(n^2)$, avec n le nombre d'instances à ordonner. Ceci représente un inconvénient algorithmique majeur pour l'apprentissage du classifieur binaire. Dans certains cas, la complexité peut être fortement réduite en évitant de former explicitement toutes les paires critiques. Citons par exemple, dans le cadre biparti (voir section 1.1.6), l'algorithme RankBoost [FISS03].

Une fois la fonction d'ordonnancement apprise, elle peut être utilisée pour ordonner n'importe quel ensemble. L'inférence se décompose ainsi en deux étapes : calculer les scores de chaque instance et ordonner les ensembles en fonction du score. En notant m le nombre d'instances à ordonner, la première

étape nécessite $O(m.d)$ calculs et la deuxième est réalisée avec un algorithme de tri efficace comme QUICKSORT [Hoa62], qui a une complexité moyenne de $O(m.\log(m))$. La complexité totale est ainsi de :

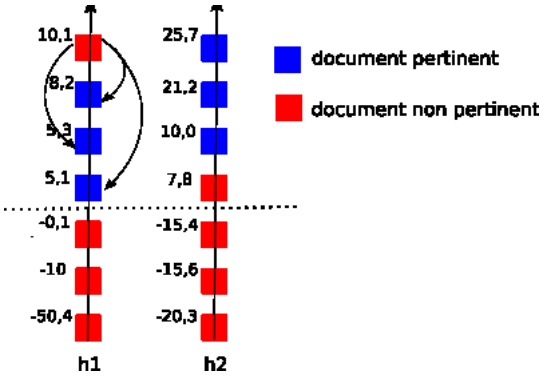
$$complexité\ d'inférence\ =\ O(d.m + m\log(m))$$

1.1.4 Ordonnancement induit par la sortie d'un classifieur

Dans le cas d'ordonnancement biparti (section 1.1.6), les instances à ordonner ont des jugements de pertinence binaires : certaines sont *pertinentes*, les autres sont *non-pertinentes*. Il est alors possible d'apprendre un classifieur binaire qui discrimine les instances pertinentes des instances non-pertinentes. Ces classifieurs d'instances sont généralement des fonctions réelles $h : \mathcal{X} \rightarrow \mathbb{R}$ dont le signe permet de classer les exemples. Ainsi, dans ce cas particulier, il est envisageable d'utiliser les scores donnés par un classifieur d'instances pour ordonner les exemples.

La figure 1.1.4 montre un exemple jouet permettant d'illustrer la différence entre les deux types d'erreur : h_1 et h_2 sont deux fonctions scores renvoyant des scores différents pour 7 instances (3 pertinentes et 4 non pertinentes). En terme de classification, elles se trompent sur la classe d'une seule instance, ce qui implique la même erreur de classification pour les deux fonctions. Or, contrairement à h_2, h_1 donne un score plus élevé à une instance non pertinente qu'aux instances pertinentes. Ainsi les deux fonctions induisent des erreurs de classification identiques mais des erreurs d'ordonnancement différentes.

Exemple jouet illustrant la différence entre l'erreur de classification et celle d'ordonnancement.



L'exemple illustre ainsi la corrélation faible entre l'erreur de classification et l'erreur d'ordonnancement. A travers une analyse théorique, les travaux de [CM03] mettent en exergue ces différences lorsque les jugements de préférence sont binaires. Les auteurs considèrent un ensemble donné d'instances ainsi que tous les ordres possibles sur cet ensemble avec une erreur de classification fixée. Ils montrent alors que la moyenne de l'erreur d'ordonnancement est une fonction croissante par rapport à l'erreur de classification. Ceci illustre le lien global entre ces deux erreurs. Cependant, ils montrent aussi que la variance peut être importante pour des bases déséquilibrées. Ainsi une même erreur

d'ordonnancement peut mener à des erreurs de classification très différentes.

On peut aussi noter les travaux de [CNM04, CKY08] qui montrent empiriquement la décorrélation entre les erreurs basées sur de la prédiction (classification, régression) et celles qui mesurent l'accord entre un ordre prédit et un ordre souhaité. Les résultats se basent sur une analyse des performances de milliers de modèles appris sur plusieurs bases.

Notons cependant, que ces conclusions se basent sur une comparaison des fonctions d'erreurs et non sur la performance des modèles. Ainsi, dans certains cas particuliers, les méthodes AdaBoost et son analogue en ordonnancement, RankBoost, renvoient des solutions identiques [RCMS05]. De plus, les résultats dans [CNM04] montrent que les algorithmes de type SVM obtiennent des bonnes performances pour les tâches d'ordonnancement. L'analyse présentée dans [CKY08] montre néanmoins que l'adaptation des SVM linéaires pour l'ordonnancement obtient des résultats significativement supérieurs aux SVM linéaires entraînées pour minimiser l'erreur de classification pour des bases de moyenne dimension.

1.1.5 Autres critères

En recherche et en filtrage d'information, il existe une panoplie de mesures de performances pour l'évaluation des systèmes. Il y a, par exemple, des mesures qui se focalisent sur les instances placées en tête de liste. Ces mesures sont généralement utilisées dans les moteurs de recherche. Pour des applications Internet, les utilisateurs ont aussi tendance à regarder les premiers liens retournés.

L'optimisation de ces mesures est difficile mais elle a fait récemment l'objet de plusieurs travaux. Les premiers sont certainement ceux de [MCM05], mais les résultats ne semblent pas probants. Il faudra attendre les travaux de [Joa05] et [TSVL07] pour trouver de nouvelles approches en apprentissage pour l'optimisation directe de ces mesures.

1.1.6 Cas particulier : l'ordonnancement biparti

L'ordonnancement biparti est un cas particulier lorsque les jugements de préférences sont binaires c'est–à–dire $\mathcal{Y} = \{-1, 1\}$. Ainsi réduite, la base d'apprentissage est similaire à celle utilisée en classification. Rappelons que les deux tâches diffèrent fondamentalement dans leur objectif. Dans notre cas, l'ordonnancement se résume à ordonner les instances positives au–dessus des instances négatives, ce qui revient à induire un ordre partiel sur les exemples.

L'ordonnancement biparti représente ainsi la forme la plus simple de l'ordonnancement d'instances. Mais il permet de traiter des applications existantes comme le routage d'information [ILS+00]. Sa similitude avec la classification aidant, il a fait l'objet de quelques études théoriques [AGH+05, AN05, AR05, UTAG05, UAG05].

Aire sous la courbe ROC et ordonnancement La courbe ROC (*Receiver Operating Characteristic*) [Faw03] a été introduite pour la première fois dans

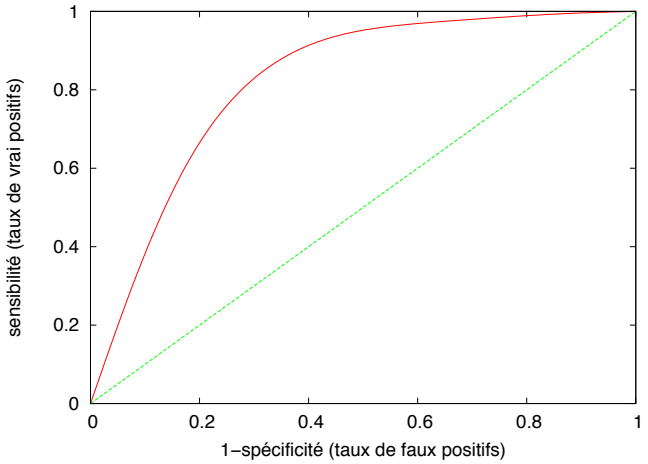
l'analyse des signaux radars. Elle est maintenant couramment utilisée en apprentissage statistique pour la sélection de modèles.

Cette courbe permet de visualiser l'évolution du taux des faux pertinents (FP) par rapport aux taux des vrais pertinents (TP). Pour une fonction score $h(x)$, la pertinence d'une instance peut être estimée en comparant son score à un seuil b. Le classifieur obtenu a ainsi la forme $f(x) = \text{signe}\,(h(x) + b)$. Pour un échantillon donné,

$$\begin{aligned} TP &= \frac{\text{pertinents bien classés}}{\text{total pertinents}} \\ FP &= \frac{\text{non pertinents mal classés}}{\text{total non pertinents}} \end{aligned}$$

La courbe est obtenue en faisant varier le seuil b. La figure 1.1.6 illustre deux courbes ROC : une courbe en pointillée dans le cas d'une décision aléatoire et en trait plein un exemple de courbe obtenue par une fonction score déterministe.

Exemple de courbe ROC



Une méthode courante pour résumer cette information est de calculer l'aire sous la courbe ROC, appelée AUC. Nous pouvons remarquer que sa valeur est comprise entre 0 et 1 et que pour une fonction score aléatoire (courbe verte) elle vaut 0.5. Une fonction score réaliste devrait donc avoir un AUC supérieur à 0.5.

D'un point de vue statistique, l'AUC a une propriété importante : elle est équivalente à la probabilité qu'une fonction score ordonne une instance pertinente choisie aléatoirement au–dessus d'une instance négative choisie aléatoirement. Ceci est équivalent au test de Wilcoxon sur les rangs. En termes d'apprentissage de fonctions d'ordonnancement dans le cas biparti, il est important de remarquer que l'erreur de classification des paires critiques est précisément égale à $1 - AUC$. La minimisation de l'erreur sur les paires cruciales est alors équivalente à l'optimisation de l'aire sous la courbe ROC.

L'ordonnancement d'instances cherche à ordonner des entrées, échantillonnées i.i.d., données au système. Cependant, on peut poser le problème d'apprendre une fonction d'ordonnancement où pour chaque

entrée, on doit ordonner un ensemble fixé d'éléments. Ce formalisme, appelé ordonnancement d'alternatives est développé par la suite.

1.2 Ordonnancement d'alternatives

1.2.1 Formalisme

L'ordonnancement d'alternatives (*label ranking*) [HPRZ02, DMS03, AS04] fait référence à un formalisme d'ordonnancement dans lequel les entrées ne sont plus les éléments à ordonner. L'ensemble à ordonner est un autre ensemble prédéfini, celui des alternatives. Pour chaque entrée, il s'agit d'inférer un ordre prédéfini sur les alternatives.

Formellement, nous notons $\mathcal{X}$ l'ensemble des entrées et $\mathcal{A}$ l'ensemble des alternatives. Chaque entrée $x \in \mathcal{X}$ est associée à un sous–ensemble d'alternatives valides $\mathcal{A}_x \subset \mathcal{A}$. Nous notons m_x, le nombre d'alternatives contenues dans $\mathcal{A}_x$. Cet ensemble est aussi muni d'un ordre $\succ_x$, qui peut être exprimé comme précédemment par des étiquettes $l_x = \{y_1, \ldots, y_{m_x}\}$. y_i exprime le degré de pertinence de la $i^{\text{ème}}$ alternative.

Nous supposons que l'apprenant dispose de n exemples avec leurs alternatives associées ainsi que leurs étiquettes. Théoriquement, ca devrait être l'ordre sur les alternatives mais nous considérons que cet ordre est induit par la valeur des étiquettes. Nous notons $\mathcal{S}_\ell$ cet ensemble d'apprentissage. Les exemples sont ainsi supposés être générés de façon i.i.d. selon une distribution fixe et inconnue sur l'ensemble $\cup_{x\in\mathcal{X}} x \times \mathcal{Y}_x$. Le but de l'ordonnancement d'alternatives est d'apprendre une fonction qui, pour un exemple donné, doit retrouver l'ordre souhaité sur l'ensemble des alternatives qui lui sont associées.

1.2.2 Modèle linéaire pour l'ordonnancement d'alternatives

Dans cette section, nous allons présenter un modèle linéaire pour l'ordonnancement d'alternatives introduit dans [AS08] qui permet de généraliser les précédents travaux dans le domaine [HPRZ02, DMS03, AS04, SBPh06]. Ce modèle permet d'apprendre une fonction $h : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ qui donne un score à chaque alternative valide pour un exemple donné. Cette fonction est très similaire à la fonction score vue pour l'ordonnancement d'instances (cf. section 1.1.1). Idéalement, elle devrait vérifier la propriété suivante :

$$\forall x \in \mathcal{X},\ \forall (a,y),(a',y') \in (\mathcal{A}_x \times \mathcal{Y}_x)^2, \quad y \succ_x y' \Leftrightarrow h(x,a) > h(x',a')$$

Représentation jointe En ordonnancement d'alternatives, il est rare de disposer d'un côté, d'une représentation des exemples d'entrée et de l'autre, d'une représentation des alternatives. L'ensemble des travaux se base plutôt sur une représentation jointe d'une entrée et d'une alternative notée $\Psi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^d$, avec $d \in \mathbb{N}$. Elle permet de représenter l'exemple et l'alternative par un vecteur de caractéristiques.

Cette représentation est liée à l'application traitée et peut en découler naturellement. Par exemple, dans le cas de la recherche documentaire, les travaux utilisent comme caractéristiques des mesures de similarité entre la requête et le document (mesure cosinus, coefficient de Jacard ou de Dice, etc.). Les bases Letor de Microsoft (décrites section 3) utilisent une telle représentation. Dans ce cas, Ψ se met sous la forme suivante :

$$\Psi(x,a) = [s_1(x,a), s_2(x,a), \ldots, s_d(x,a)]^T \text{ avec } s_i \text{ une mesure de similarité.}$$

Apprentissage et fonctions d'erreur Du point de vue apprentissage statistique, cette représentation est intéressante puisqu'elle permet de travailler avec des fonctions prenant uniquement la représentation jointe en entrée. Dans le cas linéaire, les modèles recherchés sont de la forme :

$$h(x,a) = w^T \psi(x,a)$$

Pour une entrée x fixée, cette fonction permet de donner un score à l'ensemble des alternatives. Comme pour l'ordonnancement d'instances, les scores vont induire un ordre sur $\mathcal{A}$. L'apprentissage cherche à minimiser une fonction d'erreur qui mesure l'inadéquation entre l'ordre retourné par h et l'ordre souhaité, pour l'ensemble des entrées. Dans la section 1.1.3, nous avons vu les erreurs (ou mesures de performance) utilisées pour mesurer cette différence. Nous pouvons les utiliser à nouveau en prenant la moyenne sur les entrées disponibles.

Soit une fonction d'erreur d'ordonnancement L sur un ensemble donné. Nous pouvons définir l'erreur empirique en ordonnancement d'alternatives, sur l'ensemble $\mathcal{S}_\ell$ de taille n, par :

$$R_{OA}(h, \mathcal{S}_\ell) = \frac{1}{n} \sum_{(x,l_x)\in\mathcal{S}_\ell} L\Big(\{h(x,a)\}_{a\in\mathcal{A}_x}, l_x\Big)$$

Par exemple, nous pouvons prendre l'erreur de classification de paires :

$$R_{OA}(h, \mathcal{S}_\ell) = \frac{1}{n} \sum_{(x,l_x)\in\mathcal{S}_\ell} \left[\frac{1}{\sum_{y_x, y'_x \in l_x} [\![y_x > y'_x]\!]} \sum_{y_x, y'_x \in l_x : y_x > y'_x} [\![h(x,y_x) \leq h(x,y'_x)]\!] \right]$$

L'erreur en généralisation se définit alors comme précédemment :

$$R(h) = \mathbb{E}\left[\{L(h(x,a)\}_{a\in\mathcal{A}_x}, l_x)\right] \quad (1)$$

Classification de paires critiques Avec une représentation jointe, nous pouvons former des paires critiques et appliquer une méthode de classification pour apprendre la fonction score. Notons toutefois une différence théorique majeure entre l'ordonnancement d'instances et l'ordonnancement d'alternatives. En effet, les éléments de la base d'apprentissage *initiale* de l'ordonnancement d'alternatives ne sont pas i.i.d.. Une alternative peut intervenir dans plusieurs représentations jointes $\psi(x,a)$ et $\psi(x',a)$. Dans la pratique, l'hypothèse i.i.d. est rarement vérifiée voire invalide comme ici. Les travaux de [UTAG05, Usu07] donnent des garanties théoriques pour des bases formées d'éléments qui sont dépendants entre eux.

Complexité algorithmique Cette approche peut s'avérer extrêmement coûteuse. En effet, nous retrouvons la même limite en terme de complexité algorithmique qu'en ordonnancement d'instances où un grand nombre de paires critiques est généré.

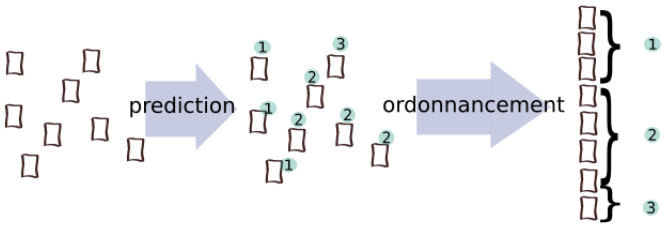
1.3 Relation avec les cadres existants

Dans cette section, nous allons présenter la régression ordinale et l'apprentissage de relations de préférence. Ces deux formalismes permettent aussi de traiter l'ordonnancement mais sous d'autres angles.

1.3.1 Régression ordinale

Dans la littérature, on rencontre un cadre très similaire à celui de l'ordonnancement d'instances : la régression ordinale. Dans les deux cas, les données sont des instances étiquetées. Les étiquettes prennent des valeurs dans un espace $\mathcal{Y}$ sur lequel il existe un ordre total. Sans perte de généralité, nous supposons $\mathcal{Y} = \{1, \ldots, K\}$. L'objectif de la régression ordinale est de prédire l'étiquette mais en cas d'erreur, l'étiquette prédite doit être la plus proche[3] possible de la vraie étiquette. Par exemple, pour une étiquette valant 5, il est préférable de prédire 4 au lieu de 3.

Ordonnancement dans le cadre de la régression ordinale.



La régression ordinale peut être ainsi vue comme de la classification multi–classes avec un ordre total sur les étiquettes. Une approche courante est d'apprendre une fonction réelle ainsi que $K-1$ seuils $\theta_1, \ldots, \theta_{K-1}$ pour déterminer l'étiquette. Cela revient à partitionner l'ensemble des réels et à associer à chaque partition une étiquette. La prédiction se fait alors en deux étapes : on attribue un score $h(x)$ à une instance x et on lui donne ensuite l'étiquette associée à la partition dans laquelle le score se trouve.

Plusieurs algorithmes de classification ont été étendus dans ce cadre. Citons par exemple le perceptron [CS01], les SVM [CK07, CK05, SL02] ou les processus gaussiens [CG05].

Lorsqu'il y a plus de deux classes, la régression ordinale permet ainsi d'ordonner indirectement les instances. Elle constitue ainsi une approche alternative

[3] Un ordre total permet en effet de définir une distance entre étiquettes par la différence entre les rangs des étiquettes

à l'approche utilisant les paires cruciales. Notons toutefois, que dans le cas binaire, la régression ordinale est équivalent à la classification alors que l'approche utilisant les paires cruciales permet d'optimiser l'AUC.

1.3.2 Apprentissage de relations de préférence

Pour l'instant, nous avons uniquement considéré le cas où l'ordre peut être induit à partir des étiquettes. Ce cadre permet d'exprimer facilement et simplement un ordre. Sa similitude avec la classification et l'ordonnancement lui confère un atout certain.

Cependant, il ne permet pas de représenter une relation qui n'est pas forcément transitive. Pour ce faire, des travaux ont considéré une relation binaire indiquant simplement qu'une entrée (ou une alternative) est préférée à une autre. Cette relation induit aussi un ordre sur un ensemble d'instances. En pratique, elle s'exprime à travers un graphe de préférence. C'est un graphe orienté. Un arc (x, x') signifie que x est préféré à x'. La base d'apprentissage est constituée d'exemples et d'un graphe de préférence.

L'approche utilisant les paires cruciale est alors adaptée pour l'apprentissage. Cependant, cette approche nécessite de former un graphe, ce qui est plus coûteux que de donner simplement des étiquettes.

2 Application au résumé automatique de textes

Dans cette section, nous allons présenter la tâche du résumé automatique et l'application des méthodes d'ordonnancement d'alternatives à ce cadre. Le résumé automatique consiste à extraire les idées pertinentes d'un document soit par rapport à un sujet donné soit par rapport à une thématique sous-jacente. Cette approche peut par exemple aider les utilisateurs à naviguer et à mieux sélectionner les documents qui peuvent les intéresser.

2.1 Présentation de l'application

Les systèmes classiques de recherche d'information retournent à l'utilisateur une liste ordonnée constituée des documents les plus pertinents par rapport à sa requête. Mais ces documents peuvent ne pas être pertinents et leur examen est coûteux en temps. Présenter à l'utilisateur des résumés de documents facilite grandement sa recherche. Un résumé peut aussi aider l'utilisateur à catégoriser des documents ou à répondre à des questions.

Les résumés similaires à ceux réalisés par un humain (résumé manuel) sont néanmoins difficiles à faire sans une compréhension poussée du contenu du texte [Man01]. Il existe beaucoup trop de variation de styles d'écriture, de constructions syntaxiques, etc. pour pouvoir construire un système de résumé générique. Un système idéal de résumé comprendrait l'information pertinente recherchée par l'utilisateur et la restituerait sous une forme cohérente et compréhensible, ce qui est difficile à faire sans un processus du langage naturel.

Pour contourner ce problème, les systèmes de résumé proposent d'extraire des passages du texte et de présenter à l'utilisateur un résumé en concaténant ces passages. Il existe deux façons d'envisager le résumé automatique de texte :

- le résumé générique qui résume le contenu par rapport à l'idée principale du texte ;

– le résumé par rapport à une requête qui résume le texte par rapport à une requête d'utilisateur.

La majorité des techniques de résumé de texte se sont intéressées au résumé par extraction d'entités textuelles. Ces entités peuvent être des groupes de mots, des phrases ou des paragraphes. Les résumés sont alors générés en concaténant des entités de base sélectionnées à partir du document original. Ainsi, le résumé automatique se fait par extraction d'entités textuelles en ordonnant celles qui convoient les idées générales (celles qui sont les plus susceptibles de faire partie du résumé) en haut de la liste. La figure 2.1 montre le texte original du discours de A. Lincoln à Gettysburg (connu sous le nom de *Gettysburg Address*) et les figures 2.1 et 2.1 donnent un exemple d'un résumé en utilisant 25 % et 15 % des phrases initiales du document (exemples pris dans [Man01]).

Le Gettysburg Address, retranscription du discours de A. Lincoln, le 19 novembre 1863

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation or any nation so conceived and so dedicated can long endure. We are met on a great battlefield of that war. We have come to dedicated a portion of that field as final resting-place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But in a larger sense, we cannot dedicate, we cannot consecrate, we cannot hallow this ground. The brave men, living and dead who struggled here have consecrated it far above our poor power to add and detract. The world will little note nor long remember what we say here, but it can never forget what they did here. It is for us the living rather to be dedicated here to be unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us – that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion – that we here highly resolve that these dead shall not have died in vain, that this nation under God shall have a new birth of freedom, and that government of the people, by the people, for the people shall not perish from the earth.

Exemple de résumé automatique du *Gettysburg Address* en utilisant 25 % du texte

Four score and seven years ago our fathers brought forth upon this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. The brave men, living and dead, who struggled here, have consecrated it fat above our poor to add or detract.

Exemple de résumé automatique du *Gettysburg Address* en utilisant 15 % du texte

Four score and seven years ago our fathers brought forth upon this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged great civil war, testing nation, nation conceived dedicated, long endure. We are met great battlefield war.

2.1.1 Forme de résumés

Outre la distinction entre abstraction et extraction, les résumés peuvent être de natures différentes selon l'objectif de l'application. Le résumé sert–il à donner une simple indication du contenu ? Ou s'agit-il d'un condensé de l'information contenue dans le texte ? Ces questions permettent d'identifier deux types de résumés : les résumés *indicateurs* et les résumés *informatifs*.

Les résumés indicateurs permettent à l'utilisateur de savoir s'il doit lire le texte source plus en détail. C'est le rôle par exemple des synopsis pour les films, qui en donnent juste un aperçu pour donner envie aux personnes d'aller les voir. A l'opposé, les résumés dits informatifs reprennent l'ensemble des informations importantes. En somme, c'est une version fidèle mais condensée du texte original. Les résumés (*abstract*) fournis au début des articles scientifiques illustrent ce genre de résumé.

Dans la littérature, une troisième forme de résumé vient s'ajouter aux deux précédentes : le résumé *critique*. Il inclut en plus des conseils, le jugement ou la critique de la personne qui l'écrit. Un exemple est donné à la figure 2.1.1. Les critiques de films ou le retour des membres du comité de programme pour l'acceptation des articles scientifiques en sont d'autres exemples.

Exemple de résumé critique du *Gettysburg Address*

The Gettsyburg address, though short, is one of the greatest American speeches. It's ending words are especially powerful –'that government of the people, by the people, for the people, shall not perish from the earth.'

2.2 Résumé automatique et apprentissage

Dans la pratique, l'extraction s'appuie sur plusieurs heuristiques que nous pouvons cataloguer dans sept grandes classes [PJ93]. Par exemple, nous pouvons considérer que si un passage contient des mots présents dans le titre alors il a des chances de faire partie du résumé. Nous pouvons aussi vérifier s'il contient des mots clés comme *en résumé*, *en conclusion* ou au contraire des mots comme *par exemple*, *peut–être* etc.. Dans la pratique, chaque heuristique se traduit par un score : plus il est élevé, plus l'heuristique donne du poids au passage pour qu'il soit dans le résumé.

Le tout est de pouvoir les combiner[4] ensemble pour obtenir un bon critère de sélection. Les premiers travaux cherchaient manuellement la meilleure

[4]Les travaux s'intéressent généralement à des combinaisons linéaires.

façon de le faire [GKMC99]. Par la suite, l'apprentissage a été utilisé pour automatiser ce traitement. Dans ce cas, le système dispose d'un ensemble de documents ainsi que de leurs résumés. Chaque passage est représenté par un vecteur (en utilisant par exemple les scores des heuristiques ou toute autre caractèristique) et forme un exemple de la base d'apprentissage. Son étiquette indique simplement si le passage fait partie ou non du résumé. L'approche classique est d'apprendre ensuite un classifieur en estimant la probabilité d'appartenance au résumé [KPC95, Ami01]. La sortie d'un classifieur est utilisée pour sélectionner les passages pour le résumé.

Résumer un nouveau document consiste ainsi à trouver la représentation vectorielle de chaque passage, à estimer leur probabilité d'appartenance au résumé, à les ordonner en fonction de ce critère. Et finalement le document est résumé par un facteur k correspondant à la proportion des passages pertinents. Dans la littérature, k est appelé le taux de compression.

L'emploi d'un classifieur pour le résumé automatique se justifie par le fait qu'une erreur de classification nulle implique un ordre correct des phrases. En effet les scores alloués aux phrases pertinentes par le classifieur sont tous supérieurs à une constante c (correspondant à l'opposé du biais du classifieur). Les scores des phrases non–pertinentes sont au contraire inférieures à c.

En pratique, l'erreur de classification n'est jamais nulle. Dans ce cas, nous pouvons considérer une phrase non–pertinente mal classée. Son score s est alors supérieur à la constante c précédente. Mais nous ne savons pas si une phrase pertinente bien classée a un score inférieur à s et combien de phrases pertinentes se retrouvent dans la même situation. L'erreur de classification ne fournit pas assez d'information pour prédire l'ordre induit par le score du classifieur des phrases d'un même document. Optimiser l'erreur de classification n'optimise donc pas les rangs des phrases pertinentes par rapport aux phrases non–pertinentes d'un même document.

Récemment, quelques travaux [AUG05, ATUL07] se sont intéressés à apprendre directement une fonction score dans le cadre de l'ordonnancement d'alternatives où les entrées représentent les documents et les alternatives les différents passages. Ces travaux ont pu mettre en évidence l'avantage du cadre d'ordonnancement par rapport au cadre de classification pour cette tâche.

3 Application à la recherche d'information

Dans cette section, nous détaillons l'application de l'apprentissage de fonctions d'ordonnancement au cas de la recherche d'information, plus spécifiquement aux moteurs de recherche sur des corpus de documents textuels. Le problème de la recherche documentaire est l'archétype de la tâche d'ordonnancement d'alternatives : à partir d'une requête utilisateur, une liste ordonnée de documents est présentée à l'utilisateur. Les enjeux économiques s'y rattachant en ont fait un domaine d'étude à part entière pour la communauté d'apprentissage.

Dans la suite de cette section, nous présentons le problème du scoring de documents à partir d'une requête. Nous détaillons ensuite la tâche d'appren-

tissage de fonctions d'ordonnancement associée, ainsi que ses spécificités par rapport au cadre général décrit en section 1. Nous illustrons ensuite l'intérêt de l'apprentissage de fonctions d'ordonnancement pour les moteurs de recherche avec des résultats expérimentaux sur le corpus Letor [LXQ+07], le corpus de référence dédié à l'évaluation des algorithmes d'apprentissage pour la recherche documentaire.

3.1 Présentation de l'application

La problématique de la recherche documentaire est bien connue : à partir d'une requête utilisateur (par exemple un ensemble de mots-clefs), le système renvoie une liste ordonnée de documents issus d'un corpus prédéfini. L'ordre de présentation des documents doit refléter la pertinence de ces-derniers par rapport au besoin d'information de l'utilisateur.

L'ordre d'apparition des documents est le point crucial du moteur, puisqu'un utilisateur va s'intéresser presqu'uniquement aux premiers documents retournés pour sa requête. Selon le principe général du fonctionnement des moteurs de recherche (voir par exemple [BYRN99]), cet ordre est donné par une fonction d'appariement qui fournit un score pour chaque document en fonction de la requête. Le problème qui en découle est de définir une fonction de score qui positionne les documents pertinents pour la requête au début de la liste de résultats.

Pour répondre à ce problème, la communauté de recherche d'information a développé un certain nombre de fonctions heuristiques, dont les principales peuvent se ranger dans trois catégories ;

- les heuristiques de similarité de contenu entre les mots-clefs de la requête et les mots apparaissant dans un document : elles ont pour but de donner des scores plus élevés aux documents qui contiennent les mots de la requête, tout en prenant en compte l'importance d'un mot pour la recherche et, éventuellement, la taille des documents. C'est dans cette catégorie que se trouve les fonctions d'appariement (requête, document) les plus connues, comme le *tf.idf*, Okapi BM25 ou encore les modèles de langage. Le lecteur intéressé peut se référer à [BYRN99] ou [MRS08] pour une description détaillée de ces heuristiques.
- les heuristiques utilisant la structure des documents ou les méta-informations : elles raffinent les heuristiques de similarité de contenu, en se restreignant à certaines parties du document particulièrement informatives pour la recherche, comme le titre du document. L'initiative INEX (*INitiative for the Evaluation of XML Retrieval*[5]) a pour objectif l'étude intensive de ce type d'heuristiques dans la recherche d'information sur des corpus XML. Ces heuristiques sont très utiles sur le Web, où les balises HTML permettent de fournir des méta-informations pertinentes, comme le titre du document.
- Les heuristiques utilisant la structure du corpus, par exemple les liens hypertextes entre documents. Les algorithmes PageRank [PBMW98] et HITS [Kle99] en sont les exemples les plus connus. Ces méthodes utilisent le principe que la valeur (pour la recherche) d'une page Web est liée au nombre de liens entrants (c'est-à-dire le nombre de liens hypertextes, situés

[5]http ://www.inex.otago.ac.nz/

sur d'autres pages Web, qui pointent vers cette page) et de liens sortants.

Ces fonctions d'appariement heuristiques sont donc nombreuses (par exemple, les versions récentes du corpus Letor [LXQ+07] comptent 65 heuristiques différentes) et complémentaires : chacune des trois catégories est fondée sur une information qui n'est pas utilisée par les autres catégories. Il est donc naturel de vouloir les combiner afin d'utiliser l'intégralité de l'information disponible. L'approche traditionnelle dans les moteurs de recherche est d'établir une fonction d'appariement globale, en affectant manuellement un poids à chaque heuristique. Cependant, avec le nombre croissant de critères à prendre en compte, cette approche est devenue difficile à mettre en œuvre. C'est ainsi que la communauté de recherche d'information a commencé à s'intéresser aux techniques d'apprentissage de fonctions d'ordonnancement.

3.2 Moteurs de recherche et apprentissage

L'apprentissage de la fonction d'appariement globale d'un moteur de recherche, prenant la forme d'une combinaison linéaire (c'est-à-dire somme pondérée) des scores renvoyés par les différentes heuristiques, entre dans le cadre de l'ordonnancement d'alternatives décrit à la section 1.2.2. En effet, la fonction de score globale peut s'écrire sous la forme $h(x, a)$, où l'instance x est la requête utilisateur, et une alternative a correspond à un document (selon le cadre défini, la tâche est donc bien d'ordonner un ensemble de documents par rapport à une requête fournie en entrée). Apprendre la fonction de score revient alors à apprendre un modèle linéaire, c'est-à-dire $h(x, a) = w^T \Psi(x, a)$, où w est le vecteur de poids, et :

$$\Psi(x, a) = [s_1(x, a), s_2(x, a), \ldots, s_d(x, a)]^T$$

où chaque $s_i(x, a)$ est le score renvoyé par la i-ième fonction heuristique, parmi celles qui sont décrites dans le paragraphe précédent.

L'application aux moteurs de recherche pose cependant un certain nombre de problèmes spécifiques. Dans les sous-sections suivantes, nous allons décrire les deux plus importants : la construction d'une base d'apprentissage, et la sélection de la fonction de coût à optimiser dans les algorithmes d'apprentissage.

3.2.1 Constitution d'une base d'apprentissage

Selon la définition de l'ordonnancement d'alternatives, une base d'apprentissage est constitué d'un ensemble $\mathcal{S}_\ell$, contenant n entrées, avec leur ensemble d'alternatives et les jugements de pertinences associées. Dans notre cas, cela signifie que pour chaque requête x, nous avons un ensemble de m_x documents $\mathcal{A}_x$, avec leurs jugements de pertinences associés $l_x = (y_1, \ldots, y_{m_x})$.

La définition de cette base d'apprentissage pose deux difficultés en recherche d'information. D'une part, il est impossible d'apprendre à ordonner l'ensemble des documents de la collection entière (qui peut contenir des milliers de milliards de pages, comme sur le Web). Il faut alors définir l'ensemble des alternatives valides $\mathcal{A}_x$ pour chaque requête x, de façon à réduire le plus possible le nombre de documents considérés pendant l'apprentissage, tout en conservant des forts taux de rappel. La seconde difficulté est l'obtention des jugements de pertinence. En effet, le nombre de documents m_x présélectionnés pour la requête x est typiquement de l'ordre du millier (voir par exemple le *benchmark* Letor). Il est illusoire d'étiqueter plusieurs milliers de documents par requêtes : même une

"petite" base d'apprentissage contenant une centaine de requêtes nécessiterait des centaines de milliers de jugements humains. Il est donc nécessaire d'avoir une méthode utilisant moins de jugements humains par requête, tout en conservant la fiabilité des mesures d'erreur optimisées.

Des biais importants peuvent apparaître lorsque la base d'apprentissage est constituée de façon incorrecte. Dans [MR08], les auteurs montrent que la première version publique du *benchmark* Letor [LXQ+07] souffrait d'un biais inhérent à l'étape de sélection de l'ensemble des documents par requête $\mathcal{A}_x$, qui rendait l'ensemble d'apprentissage inadéquat à une utilisation réelle. Une étude expérimentale poussée a ensuite été menée dans [AKP+09], mettant en lumière que les techniques de *pooling*[6] utilisées pour créer les bases d'évaluation des moteurs de recherche des conférences TREC permettent de créer des bases d'apprentissage fiables, tant pour sélectionner l'ensemble des documents à ordonner (c'est-à-dire l'ensemble des alternatives $\mathcal{A}_x$ pour la requête x) que pour récolter les jugements de pertinence.

La création d'une base d'apprentissage passe aussi par la sélection des heuristiques à combiner. Nous ne traiterons pas ce problème, car il est relié à la sélection de caractéristiques au moment de l'apprentissage, qui est hors du sujet de ce chapitre. Le lecteur intéressé pourra toutefois se référer à [LXQ+07, GLQL07] pour le choix des heuristiques pertinentes en fonction du type de collection de documents sur laquelle la recherche est effectuée (par exemple Web ou encyclopédie).

3.2.2 Favoriser le haut de la liste de documents renvoyés

L'apprentissage de fonctions d'ordonnancement pour la recherche d'information possède une autre particularité fondamentale : bien que des milliers de documents par requête doivent être ordonnés, seuls les quelques premiers documents renvoyés par le moteur vont être réellement vus par l'utilisateur. C'est pour cela que les moteurs de recherche s'évaluent généralement à l'aide de la précision moyenne ou du *(Normalized) Discounted Cumulated Gain* ((N)DCG) (voir section 1.1.5), qui donnent beaucoup de poids au premiers éléments renvoyés (par exemple les dix premiers) et qui ne dépendent pas du nombre total de documents non-pertinents dans la liste ordonnée.

Ces mesures sont à comparer avec l'aire sous la courbe ROC (AUC), qui, au contraire, va s'intéresser à la proportion de documents non-pertinents ayant un meilleur rang qu'un document pertinent pris au hasard. Ainsi, sur une liste de 1000 documents, contenant uniquement deux documents pertinents, l'AUC sera la même dans les deux cas suivants : (1) un des documents pertinent est en première position et l'autre en dernière position ; (2) les deux documents pertinents sont au milieu de la liste ordonnée (en $500^{ième}$ position). Pour un moteur de recherche, la première liste est forcément meilleure que la seconde : dans le premier cas, l'utilisateur va pouvoir accéder à un des documents pertinents, alors que dans la seconde liste, il n'aura accès à aucun des deux documents pertinents, car il ne regardera pas à cette profondeur dans la liste de résultats.

L'AUC n'est donc pas une mesure adaptée pour la recherche d'information. Or, le principe d'apprentissage d'ordonnancements basé sur la classification de paires est fortement lié au critère AUC (voir section 1.1.6). Récemment, de

[6] Le *pooling* est une technique permettant de constituer un ensemble varié de documents à partir de plusieurs modèles.

nombreux auteurs ont donc proposé de faire des algorithmes d'ordonnancement plus appropriés à la problématique des moteurs de recherches que les algorithmes basés sur la classification de paires telle que RankBoost ou RankSVM. Les approches existantes peuvent être divisées en trois catégories ;

- dans la première catégorie, les algorithmes optimisent des fonctions d'erreur qui sont des approximations dérivables des mesures d'évaluation habituelles en RI. Par exemple, dans [CZ06], les auteurs ont proposé une approximation du (N)DCG dans un cadre de régression.
- dans la seconde catégorie, les algorithmes optimisent une borne supérieure convexe de ces mêmes mesures d'évaluation. Les travaux décrits dans [LS07, XLL+08, YFRJ07] utilisent la formulation des SVM à sorties structurées de [TJHA05] pour optimiser une borne supérieure convexe de la précision moyenne ou du NDCG. Les auteurs de [XL07] proposent une formulation alternative sous forme d'un algorithme de boosting.
- Dans la dernière catégorie, les algorithmes n'ont pas pour but d'optimiser directement les mesures d'évaluations de RI, réputées extrêmement sensibles à de faibles changements de paramètres. Les auteurs définissent alors des coûts de substitutions qui ne sont pas directement liés à une mesure existante, mais conservent leurs propriétés principales tout en étant plus faciles à optimiser. Dans ce cadre, on peut citer les études de [CQL+07, XLW+08, BRL06, UBG09], qui obtiennent généralement les meilleures performances en pratique.

3.3 Résultats expérimentaux

La recherche sur l'apprentissage de fonctions d'ordonnancement en RI a été grandement facilitée par l'apparition du corpus Letor 3.0[7] [LXQ+07], qui offre une possibilité de comparaison entre différents algorithmes. Nous reproduisons dans le tableau 3.3 les résultats de différentes méthodes d'apprentissage, afin de comparer leur performances sur plusieurs corpus. Les résulats utilisent les six corpus issus des compétitions TREC dans les années 2003–2004, nommés TD03, TD04 (tâche de *topic distillation*), HP03, HP04 (tâche de *homepage finding*), et NP03, NP04 (tâche de *named page finding*). Chaque corpus possède entre 50 et 150 requêtes. Pour chaque requête, 1000 documents ont été échantillonnés. Pour chaque document et chaque requête, il y a 65 heuristiques à combiner. Les résultats affichés sont les erreurs de test, selon la mesure macro-précision moyenne[8], obtenus en respectant le protocole expérimental de [LXQ+07].

Les résultats des algorithmes de regression, RankSVM [Joa02], SVM^{map} [YFRJ07] et ListNet [CQL+07] ont été reproduits depuis le site internet du *benchmark* Letor. Les valeurs de OwaRank ont été reproduites à partir de [UBG09]. Il peut être noté que tous les algorithmes à base d'apprentissage apprennent des fonctions linéaires. Ces algorithmes ont été choisis comme des représentants performants de leur catégorie : RankSVMutilise une approche basée sur la classification de paires, SVM^{map} est un algorithme de la seconde catégorie, optimisant une borne supérieure convexe de la précision moyenne. ListNet et OwaRank font partie de la troisième catégorie, optimisant une

[7] Il est à noter que la version 3.0 de Letor, publiée en 2008, ne souffre pas du biais de sélection des documents signalé dans [MR08].

[8] La macro-précision moyenne est la moyenne, sur les différentes requêtes, de la précision moyenne (voir section 1.1.5).

Performances de test de différents algorithmes sur les 6 bases Letor. La mesure utilisée est la macro-précision moyenne. BM25 est une heuristique de RI (sans apprentissage), REGRESSION utilise de l'apprentissage, mais pas dans le cadre de l'ordonnancement. RANKSVM est un algorithme d'ordonnancement dans le cadre de la classification de paires. Les autres algorithmes favorisent la précision sur les premiers éléments renvoyés.

	TD03	TD04	HP03	HP04	NP03	NP04
BM25	0.134	0.150	0.567	0.498	0.578	0.514
REGRESSION	0.241	0.208	0.497	0.526	0.564	0.514
RANKSVM	0.263	0.224	0.741	0.668	**0.696**	0.659
SVM^{map}	0.245	0.205	0.742	0.718	0.687	0.662
LISTNET	0.275	0.223	**0.766**	0.690	0.690	0.672
OWARANK	**0.290**	**0.229**	0.757	**0.726**	0.685	**0.683**

fonction de substitution. Pour illustrer l'intérêt de l'apprentissage de fonctions d'ordonnancement, nous donnons aussi les résultats de deux *baselines* : REGRESSION, apprend une combinaison linéaire des heuristiques en faisant de la régression linéaire sur les jugements de pertinence, et BM25, qui est la plus performante des heuristiques participant à la combinaison.

Les résultats montrent une différence très significative de performances entre les algorithmes d'apprentissage d'ordonnancement et l'heuristique BM25 ou le modèle de régression. Ce-dernier est, en moyenne, légèrement meilleur que BM25, mais est quand même moins bon sur 2 des 6 corpus. Cela montre que le cadre existant de régression n'est pas adapté pour l'apprentissage de fonctions d'ordonnancement. Parmi les algorithmes d'ordonnancement, les meilleurs résultats sont, en moyenne sur les six jeux de données, obtenus par les algorithmes de la troisième catégorie LISTNET et OWARANK.

4 Conclusion

L'apprentissage de fonctions d'ordonnancement a émergé au début des années 2000, fortement poussé par les besoins dans divers problèmes de recherche ou d'accès à l'information. Ce nouveau cadre étend la classification binaire, et des fonctions de score linéaires peuvent être apprises en modifiant légèrement les algorithmes de classification existants. Les nouvelles techniques développées permettent d'obtenir des performances jusqu'alors inateignables, par exemple en résumé automatique ou en recherche documentaire.

La recherche sur l'ordonnancement s'est, pour l'instant, majoritairement focalisée sur le cadre de l'apprentissage supervisé. Cependant, de nouvelles directions commencent à émerger, avec des résultats très prometteurs. Par exemple, de nouvelles méthodes ont été étudiées pour limiter la taille des données étiquetées nécessaires sans dégrader les performances. Ainsi, de nouveaux travaux apparaissent dans le cadre de l'apprentissage semi-supervisé [DK08, ATG08, Tru09], où l'on essaye d'exploiter l'information présente dans des instances pour lesquelles les étiquettes ne sont pas disponibles, ou encore dans le cadre de l'apprentissage actif [AUL+06], où l'algorithme sélectionne lui-même les exemples qu'il est nécessaire d'étiqueter.

Références

[AGH+05] Shivani Agarwal, Thore Graepel, Ralf Herbrich, Sariel Har-Peled, and Dan Roth. Generalization bounds for the area under the ROC curve. *Journal of Machine Learning Research*, 6 :393–425, 2005.

[AKP+09] Javed A. Aslam, Evangelos Kanoulas, Virgil Pavlu, Stefan Savev, and Emine Yilmaz. Document selection methodologies for efficient and effective learning-to-rank. In *SIGIR '09 : Proceedings of the 32^{nd} international ACM SIGIR conference on Research and development in information retrieval*, pages 468–475, New York, NY, Etats-Unis, 2009. ACM.

[AM05] Peter Auer and Ron Meir, editors. *Learning Theory, 18th Annual Conference on Learning Theory, COLT 2005, Bertinoro, Italie, 27-30 juin, 2005, Proceedings*, volume 3559 of *Lecture Notes in Computer Science*. Springer, 2005.

[AM08] Nir Ailon and Mehryar Mohri. An efficient reduction of ranking to classification. In Rocco A. Servedio and Tong Zhang, editors, *COLT*, pages 87–98. Omnipress, 2008.

[Ami01] Massih-Reza Amini. *Apprentissage automatique et recherche de l'information : application à l'extraction d'information de surface et au résumé de texte*. PhD thesis, université Pierre et Marie Curie, LIP6, juillet 2001.

[AN05] Shivani Agarwal and Partha Niyogi. Stability and generalization of bipartite ranking algorithms. In Auer and Meir [AM05], pages 32–47.

[AR05] Shivani Agarwal and Dan Roth. Learnability of bipartite ranking functions. In Peter Auer and Ron Meir, editors, *COLT*, volume 3559 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2005.

[AS04] Fabio Aiolli and Alessandro Sperduti. Learning preferences for multiclass problems. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *NIPS*. MIT Press, 2004.

[AS08] Fabio Aiolli and Alessandro Sperduti. Supervised learning as preference optimization : Recent applications. In *Proceedings of the ECML/PKDD-Workshop on Preference Learning*, 2008.

[ASS07] Fabio Aiolli, Fabrizio Sebastiani, and Alessandro Sperduti. Preference learning for category-ranking based interactive text categorization. In *Proceedings of the International Joint Conference on Neural Networks, Celebrating 20 years of neural networks*, pages 2034–2039. IEEE, 2007.

[ATG08] Massih-Reza Amini, Tuong-Vinh Truong, and Cyril Goutte. A boosting algorithm for learning bipartite ranking functions with partially labeled data. In *SIGIR*, pages 99–107. ACM, 2008.

[ATUL07] Massih-Reza Amini, Anastasios Tombros, Nicolas Usunier, and Mounia Lalmas. Learning-based summarisation of XML documents. *Information Retrieval*, 10(3) :233–255, 2007.

[AUG05] Massih-Reza Amini, Nicolas Usunier, and Patrick Gallinari. Automatic text summarization based on word-clusters and ranking algorithms. In David E. Losada and Juan M. Fernández-Luna, editors, *ECIR*, volume 3408 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2005.

[AUL+06] Massih-Reza Amini, Nicolas Usunier, François Laviolette, Alexandre Lacasse, and Patrick Gallinari. A selective sampling strategy for label ranking. In *ECML*, pages 18–29, 2006.

[BBB+08] Maria-Florina Balcan, Nikhil Bansal, Alina Beygelzimer, Don Coppersmith, John Langford, and Gregory B. Sorkin. Robust reductions from ranking to classification. *Machine Learning*, 72(1-2) :139–153, 2008.

[BRL06] Christopher J. C. Burges, Robert Ragno, and Quoc Viet Le. Learning to rank with nonsmooth cost functions. In *NIPS*, pages 193–200, 2006.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, Mai 1999.

[CG05] Wei Chu and Zoubin Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6 :1019–1041, 2005.

[CK05] Wei Chu and S. Sathiya Keerthi. New approaches to support vector ordinal regression. In Raedt and Wrobel [RW05], pages 145–152.

[CK07] Wei Chu and S. Sathiya Keerthi. Support vector ordinal regression. *Neural Computation*, 19(3) :792–815, 2007.

[CKY08] Rich Caruana, Nikolaos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 96–103. ACM, 2008.

[CM03] Corinna Cortes and Mehryar Mohri. Auc optimization *versus* error rate minimization. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *NIPS*. MIT Press, 2003.

[CNM04] Rich Caruana and Alexandru Niculescu-Mizil. Data mining in metric space : an empirical analysis of supervised learning performance criteria. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *KDD*, pages 69–78. ACM, 2004.

[CQL+07] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank : from pairwise approach to listwise approach. In *ICML*, pages 129–136, 2007.

[CS01] Koby Crammer and Yoram Singer. Pranking with ranking. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *NIPS*, pages 641–647. MIT Press, 2001.

[CSS97] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *NIPS*. The MIT Press, 1997.

[CZ06] David Cossock and Tong Zhang. Subset ranking using regression. In *COLT*, pages 605–619, 2006.

[DK08] Kevin Duh and Katrin Kirchhoff. Learning to rank with partially labeled data. In *SIGIR*, pages 251–258. ACM, 2008.

[DMS03] Ofer Dekel, Christopher D. Manning, and Yoram Singer. Log-linear models for label ranking. In Thrun et al. [TSS04].

[Faw03] Tom Fawcett. ROC graphs : Notes and practical considerations for researchers. Technical report, HP Laboratories, 2003.

[FISS03] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4 :933–969, 2003.

[GKMC99] Jade Goldstein, Mark Kantrowitz, Vibhu O. Mittal, and Jaime G. Carbonell. Summarizing text documents : Sentence selection and evaluation metrics. In *SIGIR*, pages 121–128. ACM, 1999.

[GLQL07] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. Feature selection for ranking. In *SIGIR '07 : Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 407–414, New York, NY, Etats-Unis, 2007. ACM.

[Hoa62] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1) :10–16, 1962.

[HPRZ02] Sariel Har-Peled, Dan Roth, and Dav Zimak. Constraint classification : A new approach to multiclass classification and ranking. Technical report, University of Illinois at Urbana-Champaign, Champaign, IL, Etats-Unis, 2002.

[ILS+00] Raj D. Iyer, David D. Lewis, Robert E. Schapire, Yoram Singer, and Amit Singhal. Boosting for document routing. In *CIKM*, pages 70–77. ACM, 2000.

[Joa02] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[Joa05] Thorsten Joachims. A support vector method for multivariate performance measures. In Luc De Raedt and Stefan Wrobel, editors, *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 377–384. ACM, 2005.

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5) :604–632, 1999.

[KPC95] Julian Kupiec, Jan O. Pedersen, and Francine Chen. A trainable document summarizer. In Edward A. Fox, Peter Ingwersen, and Raya Fidel, editors, *SIGIR*, pages 68–73. ACM Press, 1995.

[LS07] Quoc Le and Alexander Smola. Direct optimization of ranking measures, avril 2007.

[LXQ+07] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. Letor : Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR 2007, in conjunction with SIGIR 2007*, 2007.

[Man01] Inderjeet Mani. *Automatic Summarization (Natural Language Processing, 3 (Paper))*. John Benjamins Publishing Co, June 2001.

[MCM05] Donald A. Metzler, W. Bruce Croft, and Andrew Mccallum. Direct maximization of rank-based metrics for information retrieval. Technical report, CIIR, 2005.

[MR08] Tom Minka and Stephen Robertson. Selection bias in the letor datasets. In *LR4IR 2008, in conjunction with SIGIR 2008*, 2008.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, Etats-Unis, 2008.

[PBMW98] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking : Bringing order to the web, 1998.

[PJ93] Chris D. Paice and Paul A. Jones. The identification of important concepts in highly structured technical papers. In Robert Korfhage, Edie M. Rasmussen, and Peter Willett 0002, editors, *SIGIR*, pages 69–78. ACM, 1993.

[RCMS05] Cynthia Rudin, Corinna Cortes, Mehryar Mohri, and Robert E. Schapire. Margin-based ranking meets boosting in the middle. In Peter Auer and Ron Meir, editors, *COLT*, volume 3559 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2005.

[RW05] Luc De Raedt and Stefan Wrobel, editors. *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Allemagne, 7-11 août, 2005*, volume 119 of *ACM International Conference Proceeding Series*. ACM, 2005.

[SBPh06] Yoram Singer, P. Bennett, and Emilio Parrado-hernändez. Efficient learning of label ranking by soft projections onto polyhedra. In *Journal of Machine Learning Research*, 2006.

[SL02] Amnon Shashua and Anat Levin. Ranking with large margin principle : Two approaches. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS*, pages 937–944. MIT Press, 2002.

[TJHA05] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6 :1453–1484, 2005.

[Tru09] Nguyen Tuong Vinh Truong. *Apprentissage de fonctions d'ordonnancement avec peu d'exemples étiquetés : une application au routage d'information, au résumé de textes et au filtrage collaboratif*. PhD thesis, université Pierre et Marie Curie - Paris VI, 2009.

[TSS04] Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors. *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, 8-13 décembre, 2003, Vancouver and Whistler, British Columbia, Canada]*. MIT Press, 2004.

[TSVL07] Choon Hui Teo, Alex J. Smola, S. V. N. Vishwanathan, and Quoc V. Le. A scalable modular convex solver for regularized risk minimization. In Pavel Berkhin, Rich Caruana, and Xindong Wu, editors, *KDD*, pages 727–736. ACM, 2007.

[UAG05] Nicolas Usunier, Massih-Reza Amini, and Patrick Gallinari. A data–dependent generalisation error bound for the AUC. In *ICML'05 workshop on ROC Analysis in Machine Learning*, 2005.

[UBG09] Nicolas Usunier, David Buffoni, and Patrick Gallinari. Ranking with ordered weighted pairwise classification. In *ICML '09 : Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1057–1064, New York, NY, Etats-Unis, 2009. ACM.

[Usu07] Nicolas Usunier. *Apprentissage de fonctions d'ordonnancement : une étude théorique de la réduction à la classification et deux applications à la Recherche d'Information.* PhD thesis, université Pierre et Marie Curie - Paris VI, 2007.

[UTAG05] Nicolas Usunier, Vinh Truong, Massih-Reza Amini, and Patrick Gallinari. Ranking with Unlabeled Data : A First Study. In *NIPS'05 Workshop on Learning to Rank (NIPS'05-LR)*, page 4, Whistler, Canada, dec 2005.

[XL07] Jun Xu and Hang Li. Adarank : a boosting algorithm for information retrieval. In *SIGIR*, pages 391–398, 2007.

[XLL+08] Jun Xu, Tie-Yan Liu, Min Lu, Hang Li, and Wei-Ying Ma. Directly optimizing evaluation measures in learning to rank. In *SIGIR*, pages 107–114, 2008.

[XLW+08] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank : theory and algorithm. In *ICML*, pages 1192–1199, 2008.

[YFRJ07] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *SIGIR*, pages 271–278, 2007.