Apprentissage de fonctions d'ordonnancement par classification par paires ordonnées et pondérées (OWPC)

David Buffoni — Nicolas Usunier — Patrick Gallinari

Laboratoire d'Informatique de Paris 6
104, avenue du Président Kennedy
75016 Paris, France
prénom.nom@lip6.fr

RÉSUMÉ. *Apprendre les fonctions d'ordonnancement pour les moteurs de recherche est une tâche difficile parce que les critères d'évaluations généralement utilisés sont difficilement optimisables directement. Dans ce cas, nous sommes contraints d'optimiser une fonction d'erreur d'ordonnancement qui en est proche. Dans ce papier, nous proposons de définir une fonction d'erreur d'ordonnancement en utilisant un opérateur d'agrégation convexe des erreurs de classification sur les paires appelé OWA (Yager, 1988) qui suivant son paramétrage peut donner un poids plus important aux erreurs commises au début de la liste. En utilisant des coûts de classification de type hinge loss, notre problème est similaire à une SVM à sorties structurées. Les expériences menées nous permettent de valider notre approche.*

ABSTRACT. *Learning a ranking function for Information Retrieval is a difficult task because evaluation criteria are not directly optimisable. We propose to optimize a convex loss functions for ranking, based on a convex aggregation operators of the classification losses (OWA (Yager, 1988)) where depending on their weights can be used to focus on the top ranked elements as they give more weight to the largest losses. When aggregating hinge losses, the optimization problem is similar to the SVM for interdependent output spaces. Experimental results allow us to validate our approach.*

MOTS-CLÉS : *Recherche d'Information, Apprentissage de fonctions d'ordonnancement, opérateurs OWA*

KEYWORDS: *Information Retrieval, Learning to Rank, OWA operators*

1. Introduction

Dans de nombreuses applications du domaine de la Recherche d'Information (RI), les systèmes de recherche font appel à une fonction d'ordonnancement, dont le but est d'ordonner les éléments d'une collection selon leur pertinence par rapport à une tâche spécifique. C'est le cas, par exemple, des moteurs de recherche sur le web, qui suivant une requête *adhoc*, renvoie à l'utilisateur, une liste de documents textes ordonnés par leur pertinence, souvent calculée sous la forme d'un score réel. L'ordre de cette liste est primordial puisqu'en général un utilisateur va s'intéresser davantage aux premiers documents retournés pour sa requête. Le problème qui en découle donc, est de savoir définir une fonction de score suceptible de positionner les documents les plus pertinents au début de la liste de résultats. Pour cela, diverses fonctions de calcul de score de pertinence ont été mises au point.

Pour le texte plat, ces fonctions utilisent la similarité du contenu textuel entre un document et une requête et se basent sur la fréquence des termes de la requête dans le document courant. Ainsi c'est le cas de toutes les approches historiques : TF-IDF, BM25, les modèles à langage... (Yates *et al.,* 1999). Des alternatives aux fonctions basées sur le contenu ont été proposées comme celles s'intéressant à la structure des documents au sein d'une collection. C'est notamment le cas pour la recherche d'information sur le web où l'on cherche à donner une information d'importance suivant les sites en considérant les liens hypertextes entre eux. C'est à partir de cette idée que des algorithmes comme Pagerank (Page *et al.,* 1999), HITS (Kleinberg, 1999), BrowseRank (Liu *et al.,* 2008), TrustRank (Gyöngyi *et al.,* 2004)... ont vu le jour.

Au fil du temps, afin d'affiner les performances, on a ajouté **manuellement** de nouvelles caractéristiques dans les moteurs de recherche. La fonction de score finale est une combinaison des différentes similarités dont le poids de chacune d'elles est fixée à la main. Bien évidemment, avec le nombre croissant de caractéristiques à prendre en compte, cette approche est devenue difficile à mettre en œuvre. L'intérêt s'est alors porté sur l'apprentissage automatique de cette combinaison de caractéristiques. Les premières tentatives basées sur des approches de classification ou régression on eu un succès modéré. L'approche pas l'apprentissage direct de ces fonctions d'ordonnancement a connu un succès réel et s'est rapidement développé pour devenir en deux ou trois ans, une thématique phare. Cependant, résoudre cette tâche à l'aide de la classification n'est pas adaptée dû au simple fait qu'on ne cherche pas à trouver une fonction qui ordonne les documents pertinents au début de la liste. Ce n'est que plus tard lorsque le paradigme de l'apprentissage de fonctions d'ordonnancement est apparu que la communauté de l'apprentissage et celle de la recherche d'information se sont ouvertes l'une à l'autre. Nous nous plaçons donc dans le domaine de l'apprentissage de fonction d'ordonnancement pour la RI avec la mise au point d'un algorithme efficace pour résoudre cette tâche.

Cet article présente dans un premier temps, l'état de l'art des techniques d'apprentissage de fonctions d'ordonnancement pour la RI (cf section 2). Ensuite, on détaillera

la contribution apportée au domaine en section 3 que l'on validera dans les parties 4 et 5.

2. État de l'art

Pendant plusieurs années l'apprentissage automatique s'est intéressé à la Recherche d'Information sans pour autant proposer un formalisme véritablement adéquat par rapport aux critères d'évaluation des moteurs de recherche. En effet en RI, le critère d'évaluation dépend de l'ordre sur la liste de documents renvoyée par le système. Pour ce critère ce qui importe est plus l'ordre relatif des documents renvoyés que le score de pertinence en lui même. C'est à partir de ce constat, qu'une approche apprenant à ordonner fut proposée par (Cohen *et al.,* 1997) comme base de l'apprentissage de fonctions d'ordonnancement. Ce domaine est devenu très étudié avec notamment en toîle de fond des applications comme la RI, mais aussi le filtrage collaboratif (systèmes de recommandation) et toutes autres tâches nécessitant d'inférer un ordre sur un ensemble d'éléments. L'apprentissage de fonctions d'ordonnancement doit apprendre une fonction qui permet d'ordonner les éléments suivant leur pertinence relative. Par la suite, on considerera deux classes de pertinence : les documents pertinents et les documents non pertinents, cadre que l'on appelle ordonnancement biparti et qui est adapté à d'autres problématiques en RI. L'objectif est donc d'apprendre une fonction qui donnera un score plus élevé aux documents pertinents qu'à des documents non pertinents. Á noter que l'hypothèse faite ici, est qu'à partir d'un ordre partiel sur la collection (on ne compare pas les documents de même pertinence entre eux) qu'on puisse en inférer un ordre total. Dans ce cadre d'ordonnancement, il subsiste le problème de définir un critère d'apprentissage approprié aux mesures d'évaluation. En effet, que les métriques généralement utilisées comme la Mean Average Precision (MAP) (Yates *et al.,* 1999), le Normalized Discounted Cumulated Gain (NDCG) (Järvelin *et al.,* 2000)... sont difficiles à optimiser directement. Néanmoins, diverses solutions ont été proposées suivant comment l'on aborde le problème de l'ordonnancement. Nous allons par la suite, décrire et faire un résumé des méthodes existantes avec d'abord des méthodes optimisant le rang moyen des éléments et ensuite, celles favorisant le haut de la liste.

2.1. *Optimisation du rang moyen*

Le cadre de travail considéré comme base de l'ordonnancement est l'approche de la classification par paires d'éléments. En général, à partir d'un ensemble de documents étiquetés, on construit un ensemble de couples de documents (pertinent , non pertinent). Ensuite, l'algorithme apprend une fonction d'ordonnancement sur ce nouvel ensemble (cf (Freund *et al.,* 2003) ou (Har-Peled *et al.,* 2002)).

Pour pouvoir ordonner, on doit déterminer la position ou le rang d'un élément pertinent afin d'être capable de comptabiliser les erreurs d'ordonnancement. Le rang d'un élément pertinent peut être trouvé simplement en comptant le nombre d'éléments non pertinents ayant eu un meilleur score. Ainsi, dans le cadre de la classification

par paires, l'erreur dépendra du signe de la différence des scores pour un couple de documents (pertinent, non pertinent). Ensuite, on comparera toutes les paires de la base d'apprentissage et on agrégera les erreurs de classification commises. Au final, cela nous permettra de construire des coûts convexes. Par exemple, si l'on prend la moyenne des erreurs de classification de paires comme c'est le cas dans les articles de (Burges *et al.,* 2005, Cao *et al.,* 2006, Freund *et al.,* 2003, Joachims, 2002, Tsai *et al.,* 2007) où pour chaque fonction d'erreur proposée, on en revient à optimiser le rang moyen d'un document pertinent.

Malgré tout le travail effectué, optimiser le rang moyen d'un document pertinent n'est certainement pas la manière la plus convenable pour optimiser des critères d'évaluation de RI qui dépendent du début de la liste. Pour ce faire, il faut trouver une façon d'optimiser une fonction d'ordonnancement favorisant les éléments en haut de la liste, c'est l'objet de la suite de cet état de l'art.

2.2. *Favoriser le haut de liste*

Ce n'est que récemment, que des approches favorisant les éléments pertinents en haut de la liste ont été présentées. On peut classer les différentes propositions en 3 catégories qui en général marchent de manière équivalente en pratique.

– Dans la première, les fonctions de coût sont des approximations plus lisses des mesures d'évaluation de RI. Par exemple, (Cossock *et al.,* 2006) ont proposé une approximation du (N)DCG dans un cadre de régression alors que (Taylor *et al.,* 2008) ou (Volkovs *et al.,* 2009) utilisent une distribution de probabilités sur les rangs des documents pour une requête.

– Dans la deuxième, on essaie d'optimiser une borne supérieure convexe des mesures d'évaluation comme c'est le cas de (Le *et al.,* 2007), (Xu *et al.,* 2008) ou (Yue *et al.,* 2007) qui en utilisant des SVM à sorties structurées (Tsochantaridis *et al.,* 2005) optimisent l'AP ou le DCG. Pour ce qui est d'une approche de type boosting, on peut citer le travail de (Xu *et al.,* 2007).

– Dans la troisième catégorie, on définit une fonction de coût de substitution plus facile à optimiser. Les études de (Burges *et al.,* 2006) et (Cao *et al.,* 2007) en proposent chacun une. Nous allons voir dans la section suivante, une description de notre méthode qui peut être placée dans cette catégorie. Nous proposons un modèle permettant de favoriser les éléments pertinents en haut de la liste tout en optimisant une fonction de perte de substitution.

3. Apprentissage de fonctions d'ordonnancement avec OWPC

Dans la section précédente, nous avons vu que pour se conformer aux critères d'évaluation en RI, les études se portaient sur l'optimisation de fonctions de coût privilégiant les rangs des documents pertinents en haut de la liste. Dans cette partie, nous décrivons une méthode ayant cet objectif et proposant une nouvelle fonction de

coût en extension à la classification par paires. La seule différence avec l'approche de base (optimisation du rang moyen) est que l'on agrège les erreurs commises grâce à d'autres opérateurs que la moyenne. Pour ce faire, nous utilisons les opérateurs *Ordered Weighted Averaging* (OWA) (Yager, 1988) qui peuvent être vus comme une généralisation des opérateurs moyenne et maximum (souvent utilisés en apprentissage car donnant une fonction convexe). Ainsi, en choisissant un opérateur OWA entre les deux opérateurs précédents (moyenne et maximum), nous optimisons une nouvelle fonction de coût favorisant les documents pertinents en haut de la liste.

3.1. *Définitions et Notations*

On donne, en entrée de notre système d'ordonnancement, une *observation* z, définissant une séquence, écrite en gras, de *candidats* notée $\mathbf{X}(z) \stackrel{def}{=} \big(\mathbf{X}_1(z), ..., \mathbf{X}_{[z]}(z)\big)$ (où $[z]$ est utilisé comme raccourci syntaxique de $[\mathbf{X}(z)]$ et les indices comme un élément de l'ensemble)[1]. En RI, z correspond à une requête d'un utilisateur et $\mathbf{X}_j(z)$ est la représentation dans l'espace des caractéristiques du $j$ème document.

En outre, nous disposons des degrés de pertinence des documents. Par simplicité, nous nous mettons dans le cadre biparti de l'ordonnancement qui considère seulement deux classes de pertinence (les documents pertinents et non pertinents). On peut alors définir la séquence $\mathbf{y}$ (respectivement $\bar{\mathbf{y}}$) comme une séquence qui contient les indices des candidats *pertinents* (resp. *non pertinents*).

Pour que l'on puisse ordonner, on emploie une fonction de score à valeur réelles f qui prend en paramètre la représentation sous forme de caractéristiques d'un document $(f\big(X_j(z)\big))$ et qui renvoie son score. En guise de clarté, on utilisera la notation $f_j(z)$ comme le score du $j$ème document. Enfin, le système d'ordonnancement doit, pour chaque requête, renvoyer la liste des candidats ordonnés par scores décroissants. On fera apprendre notre système au préalable sur une base d'apprentissage S qui n'est autre qu'un ensemble de documents annotés (suivant une classe de pertinence) selon une requête.

Dans la suite, nous allons définir une fonction d'erreur d'ordonnancement qui a pour objectif de favoriser le rang des documents pertinents au début de la liste.

3.2. *Erreurs d'Ordonnancement*

Avant de définir, l'erreur d'ordonnancement pour notre système, il faut avant tout fixer la fonction qui prédit le *rang* d'un élément pertinent. On a alors la formule sui-

1. les crochets [.] informent sur la taille de la séquence.

vante, étant donné une observation z, ses candidats pertinents $\mathbf{y}$ (et implicitement les non pertinents $\bar{\mathbf{y}}$), et une fonction score f :

$$\forall y \in \mathbf{y}, rank_y(f,z,\mathbf{y}) \stackrel{def}{=} \sum_{\bar{y}\in\bar{\mathbf{y}}} \mathrm{I}\left(f_y(z) \leq f_{\bar{y}}(z)\right) \quad [1]$$

où I est une fonction indicatrice. Cette définition du rang nous installe dans l'approche de *la classification par paires* comme vue précédemment (cf (Freund *et al.,* 2003) ou (Har-Peled *et al.,* 2002)).

On peut voir l'erreur du classificateur par paires comme $\mathrm{I}\left(f_y(z) \leq f_{\bar{y}}(z)\right)$ que l'on peut reécrire en $\mathrm{I}\left(\text{signe}\big(f_y(z) - f_{\bar{y}}(z)\big) \neq 1\right)$ (le signe de la différence entre les scores). D'une façon plus générale, en utilisant la définition précédente du rang, on peut considérer la fonction erreur d'ordonnancement suivante :

Définition 1. *L'*erreur d'ordonnancement *d'une fonction à valeurs réelles f sur $(z,\mathbf{y})$ est définie comme suit :*

$$\mathtt{err}(f,z,\mathbf{y}) \stackrel{def}{=} \frac{1}{[\mathbf{y}]} \sum_{y\in\mathbf{y}} \Phi_{[\bar{\mathbf{y}}]}\left(rank_y(f,z,\mathbf{y})\right)$$

Où, pour tout $n \in \mathbb{N}^*$, il existe un vecteur $\boldsymbol{\alpha}^n \in [0,1]^n$ de valeurs positives et non croissantes $\alpha_1^n \geq \alpha_2^n \geq ... \geq \alpha_n^n \geq 0$ avec $\sum_{j=1}^n \alpha_j^n = 1$ de sorte que :

$$\forall k \in \{0..n\}\, \Phi_n(k) \stackrel{def}{=} \sum_{j=1}^{k} \alpha_j^n \quad [2]$$

La fonction Φ_n donne une pénalité lorsque l'on place k éléments non pertinents avant un pertinent dans la liste. Cette fonction d'erreur n'est pas décroissante (somme cumulée de valeurs positives) et a pour valeur 0 si l'élément pertinent est avant tous les non pertinents et de 1 s'il est après. La fonction Φ_n peut donc définir des erreurs qui se focalisent plus ou moins sur le haut de la liste car la pénalité encourue suite à la perte d'un rang d'un document pertinent est $\Phi_n(k+1) - \Phi_n(k) = \alpha_{k+1}^n$ qui décroît avec k. Ainsi, si l'on fixe les premiers poids α_k^n avec des valeurs fortes cela reviendra plus cher à faire une erreur en haut de la liste qu'à la fin ce qui intuitivement est la contrainte posée par les critères d'évaluation. Enfin en faisant la moyenne sur tous les éléments pertinents de $\Phi_n(rank_y(f,z,\mathbf{y}))$, on obtient moins d'erreurs $\mathtt{err}(f,z,\mathbf{y})$ pour les listes ordonnées avec beaucoup d'éléments pertinents en haut de la liste que pour les listes avec des éléments pertinents plus bas.

On note que cette classe de fonction d'erreur généralise les coûts

– $0/1$ utilisé en classification multiclasse (si l'on fixe $\alpha_1^n = 1$ et tous les autres αs à 0),

– du nombre moyen de paires mal ordonnées, généralement considéré dans l'approche de la classification par paires, (en fixant $\alpha_j^n = 1/n$ pour tout j).

Nous allons voir dans la section suivante que pour un élément y, la fonction $\Phi_{[\bar{\mathbf{y}}]}\left(rank_y(f,z,\mathbf{y})\right)$ peut en fait s'écrire comme une agrégation convexe des valeurs binaires $\left(\mathrm{I}\left(f_y(z) \leq f_{\bar{y}}(z)\right)\right)_{\bar{y}\in\bar{\mathbf{y}}}$.

3.3. *Erreurs de classification par paires agrégées grâce aux opérateurs OWA*

Soit la définition introduite par (Yager, 1988) :

Définition 2 (Opérateur OWA)**.** *Soit $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)$ une séquence de n nombres non négatifs avec $\sum_{j=1}^{n} \alpha_j = 1$. L'opérateur* Ordered Weighted Averaging (OWA) *associé à $\boldsymbol{\alpha}$, est la fonction* $\mathrm{owa}^{\boldsymbol{\alpha}} : \mathbb{R}^n \to \mathbb{R}$ *suivante :*

$$\forall \mathbf{t} = (t_1, ..., t_n) \in \mathbb{R}^n, \mathrm{owa}^{\boldsymbol{\alpha}}(\mathbf{t}) = \sum_{j=1}^{n} \alpha_j t_{\sigma(j)}$$

où $\sigma \in \mathfrak{S}_n$ ($\mathfrak{S}$ l'ensemble des permutations) sachant que $\forall j, t_{\sigma(j)} \geq t_{\sigma(j+1)}$.

En d'autres termes, l'opérateur OWA fait une somme pondérée des t_js. Le poids donné à t_j ne dépend donc pas de la valeur de j, mais de la position t_j dans la liste $t_{\sigma(1)} \geq t_{\sigma(2)} \geq ... \geq t_{\sigma(n)}$. Par la suite, nous retirerons l'exposant $\boldsymbol{\alpha}$ lorsque le contexte sera clair. De plus, $\mathrm{owa}^{\boldsymbol{\alpha}}(\mathbf{t})$ sera noté $\underset{j\in\{1..n\}}{\mathrm{owa}}\ t_j$ quand cela sera plus commode.

Il est aisé de noter que les opérateurs moyenne et maximum (et aussi minimum) sont des cas particuliers de l'opérateur OWA. Par exemples, pour le max, nous pouvons fixer $\alpha_1 = 1$ et $\alpha_j = 0$ pour $j > 1$ alors que pour la moyenne, les poids seront de la sorte : $\alpha_j = \frac{1}{n}$ pour tout j. Ces deux opérateurs appartiennent à la classe particulière d'opérateurs OWA avec des poids *non-croissants* et ayant les propriétés suivantes :

Proposition 3. *On pose $\boldsymbol{\alpha}$ sous les contraintes $\sum_{j=1}^{n} \alpha_j = 1$ et $\alpha_1 \geq \alpha_2 \geq ... \geq \alpha_n \geq 0$.*

1) $\forall \mathbf{t} \in \mathbb{R}^n, \mathrm{owa}^{\boldsymbol{\alpha}}(\mathbf{t}) = \max_{\sigma\in\mathfrak{S}_n} \sum_{j=1}^{n} \alpha_j t_{\sigma(j)} = \max_{\sigma\in\mathfrak{S}_n} \sum_{j=1}^{n} \alpha_{\sigma(j)} t_j$

2) $\mathbf{t} \to \mathrm{owa}^{\boldsymbol{\alpha}}(\mathbf{t})$ *est* convexe

3) Si $\mathbf{t}$ *et* $\mathbf{t}'$ *sont dans* $\mathbb{R}^n$ *et* $\forall j, t_j \geq t'_j$, *alors* $\mathrm{owa}^{\boldsymbol{\alpha}}(\mathbf{t}) \geq \mathrm{owa}^{\boldsymbol{\alpha}}(\mathbf{t}')$

Démonstration. Le premier point est dûe à l'inégalité de réarrangement qui par conséquent permet de dériver les points 2 et 3. □

En utilisant l'opérateur OWA nous pouvons faire le lien entre l'agrégation des coûts de classification par paires et les erreurs d'ordonnancement décrites dans la partie précédente, ce qui donne :

Proposition 4. *Soit Φ_n de l'équation* [2] *et $\boldsymbol{\alpha} = (\alpha_j)_{j=1}^n$ la séquence de poids non croissants associée alors, pour tout $\mathbf{t} \in \mathbb{R}^n$ et pour toute fonction convexe $\ell : \mathbb{R} \to \mathbb{R}_+$ avec $\mathrm{I}(t_j \leq 0) \leq \ell(t_j)$, on a :*

1) $\mathbf{t} \mapsto \underset{j\in\{1..n\}}{\mathrm{owa}^{\boldsymbol{\alpha}}} \ell(t_j)$ *est convexe*

2) $\Phi_n\Big(\sum_{j=1}^{n} \mathrm{I}(t_j \leq 0)\Big) \leq \underset{j\in\{1..n\}}{\mathrm{owa}^{\boldsymbol{\alpha}}} \ell(t_j)$

Démonstration. Le premier point est la définition même de Φ_n. Les Points 2 et 3 sont des conséquences directes de la proposition 3. □

La première propriété nous permet de dire qu'agréger une fonction convexe à l'aide de l'opérateur OWA permettait d'obtenir une fonction elle aussi convexe. La seconde propriété quant à elle nous permet de montrer qu'agréger avec OWA une fonction convexe qui est une borne supérieure à la fonction indicatrice, nous donne une borne supérieure convexe à l'erreur définie dans la définition 1. Agréger les coûts de classification par paires à l'aide de l'opérateur OWA est adapté aux fonctions d'erreurs d'ordonnancement définies précédemment. Dans la section suivante, on va appliquer notre proposition à la problématique de l'apprentissage de fonctions d'ordonnancement.

3.4. *Application à l'ordonnancement*

La proposition 4 suggère que le coût proposé précédemment est une borne supérieure convexe du risque empirique, on a alors d'après l'équation [2] :

$$\begin{aligned} \hat{R}^{\Phi}(f,S) &\overset{def}{=} \underset{(z,\mathbf{y})\sim S}{\hat{\mathbb{E}}} \frac{1}{[\mathbf{y}]} \sum_{y\in\mathbf{y}} \Phi_{[\bar{\mathbf{y}}]}\big(rank_y(f,z,\mathbf{y})\big) \\ &\leq \underset{(z,\mathbf{y})\sim S}{\hat{\mathbb{E}}} \frac{1}{[\mathbf{y}]} \sum_{y\in\mathbf{y}} \underset{\bar{y}\in\bar{\mathbf{y}}}{\mathrm{owa}}\, \ell\big(f_y(z) - f_{\bar{y}}(z)\big) \end{aligned} \qquad [3]$$

où $\hat{\mathbb{E}}_{(z,\mathbf{y})\sim S}$ est la moyenne sur l'ensemble d'apprentissage, ℓ est une borne supérieure convexe et non croissante sur $\mathrm{I}(t \leq 0)$, l'ensemble $S = \big((z_1,\mathbf{y}_1),...,(z_m,\mathbf{y}_m)\big)$ est la base d'apprentissage, et owa traduit l'opérateur OWA associé aux poids de la fonction correspondante $\Phi_{[\bar{\mathbf{y}}]}$[2].

En général, le coût convexe va associer des poids forts aux coûts les plus forts (car ℓ ne décroît pas). Ainsi, pour un élément pertinent donné, le convexe donnera des poids plus forts aux éléments non pertinents qui sont mieux classés. Si les poids α sont strictement décroissants, alors perdre des rangs au début de la liste sera plus pénalisant

2. Pour clarifier les notations, on peut enlever l'exposant $\boldsymbol{\alpha}$ de l'opérateur owa mais on sous-entend que les poids d'un exemple donné $(z,\mathbf{y}) \in S$ dépendent de $[\bar{\mathbf{y}}]$.

(si chaque rang est associé à un fort poids) que d'en perdre plus bas dans la liste. C'est finalement, le comportement que l'on souhaiterait pour un système d'ordonnancement dédié à la RI.

Une chose intéressante à remarquer, est lorsque la fonction de coût ℓ est le hinge loss $t \mapsto [1-t]_+$, avec $[.]_+$ la partie positive, et que l'opérateur owa est fixé au $\max$ (i.e. $\alpha_1 = 1$ dans la définition de Φ), alors la borne supérieure convexe est exactement la même utilisée dans les SVMs pour de la classification multiclasses (Crammer *et al.,* 2001) ou (Bordes *et al.,* 2007). Si par contre, l'opérateur owa est la moyenne, on retrouve la classification par paires standard avec une borne similaire aux Ranking SVMs de (Cao *et al.,* 2006) ou (Joachims, 2002).

Il nous reste à présent de reformuler cette fonction de coût convexe dans un formalisme de problème à vaste marge pour pouvoir l'optimiser.

3.5. *Formulation sous la forme d'un problème à vaste marge*

Dans la section précédente, nous avons mis au point une fonction d'erreur donnant un coût plus fort aux erreurs faites au début de la liste plutôt qu'à la fin. Nous allons maintenant, formuler ce problème sous la forme d'un SVM ce qui nous permettra d'utiliser ce solveur lors de l'apprentissage. On montre à présent qu'une version régularisée du risque empirique de l'équation [3] utilisant le hinge loss $\ell : t \mapsto [1-t]_+$ peut être résolue en utilisant des algorithmes pour sorties structurées (Tsochantaridis *et al.,* 2005). Soit le problème d'optimisation suivant généralisant les SVMs multiclasses et l'approche de la classification par paires :

$$\min_w \frac{1}{2}||w||^2 + C \sum_{(z,\mathbf{y})\in S} \frac{1}{[\mathbf{y}]} \sum_{y\in\mathbf{y}} \underset{\bar{y}\in\bar{\mathbf{y}}}{\mathrm{owa}} [1 - \langle w, X_y(z) - X_{\bar{y}}(z)\rangle]_+$$

où $\langle .,. \rangle$ dénote le produit scalaire. Pour retomber sur une formulation SVM à sorties structurées, on va utiliser la proposition 4 et noter que $[1-t]_+ = \max_{b\in\{0,1\}} b(1-t)$ ce qui nous donne :

$$\begin{aligned} &\frac{1}{[\mathbf{y}]} \sum_{y\in\mathbf{y}} \underset{\bar{y}\in\bar{\mathbf{y}}}{\mathrm{owa}}\, \ell\big(\, \langle w, X_y(z) - X_{\bar{y}}(z)\rangle \,\big) \\ &\quad = \max_{\sigma,\mathbf{b}} \big\{ \Delta_{(z,\mathbf{y})}(\sigma,\mathbf{b}) - \big\langle w, \Psi_{(z,\mathbf{y})}(\sigma,\mathbf{b})\big\rangle \big\} \end{aligned} \qquad [4]$$

Où $\max$ est calculé suivant $\sigma \in \mathfrak{S}_{[\bar{\mathbf{y}}]}$, $\mathbf{b} = (b_{ij})_{i\le[\mathbf{y}], j\le[\bar{\mathbf{y}}]} \in \{0,1\}^{[\mathbf{y}]\times[\bar{\mathbf{y}}]}$ et :

$$\Delta_{(z,\mathbf{y})}(\sigma,\mathbf{b}) \overset{def}{=} \frac{1}{[\mathbf{y}]} \sum_{i=1}^{[\mathbf{y}]} \sum_{j=1}^{[\bar{\mathbf{y}}]} \alpha_j b_{i\sigma(j)} \qquad [5]$$

$$\Psi_{(z,\mathbf{y})}(\sigma,\mathbf{b}) \overset{def}{=} \frac{1}{[\mathbf{y}]} \sum_{i=1}^{[\mathbf{y}]} \sum_{j=1}^{[\bar{\mathbf{y}}]} \alpha_j b_{i\sigma(j)} \Big(X_{y_i}(z) - X_{\bar{y}_{\sigma(j)}}(z) \Big) \qquad [6]$$

	TD03	TD04	HP03	HP04	NP03	NP04
owpc $g_{=1}^{5\%}$	0,278	0,213	**0,752**	**0,690**	**0,683**	**0,689**
owpc $g_{=1}^{10\%}$	**0,298**	0,229	0,743	0,681	**0,683**	0,678
owpc $g_{=1}$	0,244	**0,232**	0,738	0,678	0,676	0,675

Tableau 1. *Performance en test de l'OWPC en termes de MAP sur Letor pour différents poids.* $g_{=1}$ *est le poids constant,* $g_{=1}^{p\%}$ *pour un poids constant sur les premiers* $p\%$ *de la liste (donc tous les autres poids valent* 0 *après le seuil* $p\%$*).*

	TD03	TD04	HP03	HP04	NP03	NP04
RSVM	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
Adarank	0.228	0.219	**0.771**	0.722	0.678	0.622
ListNet	0.275	0.223	0.766	0.690	0.690	0.672
owpc g_{All}	**0.282**	**0.227**	0.745	**0.726**	0.680	**0.690**
owpc $g_{/}$	**0.290**	**0.229**	0.757	**0.726**	0.685	**0.683**

Tableau 2. *Les performances en MAP sur la base de test de l'OWPC et d'autres baselines du site Letor.* $g_{/}$ *correspond aux poids décroissant linéairement,* g_{All} *aux poids choisis sur la base de validation.*

ce qui nous permet d'obtenir le programme final :

$$\begin{aligned} \min_{w,\boldsymbol{\xi}} \ & \frac{1}{2}||w||^2 + C \sum_{(z,\mathbf{y})\in S} \xi_{(z,\mathbf{y})} \\ \text{u.c. } & \forall (z,\mathbf{y}) \in S, \xi_{(z,\mathbf{y})} \geq 0 \\ & \forall (z,\mathbf{y}) \in S, \forall \sigma \in \mathfrak{S}_{[\bar{\mathbf{y}}]}, \forall \mathbf{b} \in \{0,1\}^{[\mathbf{y}]\times[\bar{\mathbf{y}}]} : \\ & \left\langle w, \Psi_{(z,\mathbf{y})}(\sigma,\mathbf{b}) \right\rangle \geq \Delta_{(z,\mathbf{y})}(\sigma,\mathbf{b}) - \xi_{(z,\mathbf{y})} \end{aligned} \qquad [7]$$

Optimisation

Le problème 7 est formellement équivalent à un SVM à sorties structurées, on peut donc utiliser les algorithmes d'optimisation avec des méthodes de plans séquants (Tsochantaridis *et al.,* 2005) ou bien LaRank (Bordes *et al.,* 2007) ou (Bordes *et al.,* 2008). Nous avons choisi LaRank, pour sa simplicité d'implémentation et son efficacité en classification multiclasses ou en étiquetage de séquences. Pour plus de détails, le lecteur peut se référer aux articles correspondants.

4. Expériences sur Letor

Dans cette partie, on présente les expériences menées sur le benchmark Letor 3.0[3] (Liu *et al.,* 2007). Nous avons utilisé les 6 bases de données .GOV[4], nommées TD 2003, TD 2004, HP 2003, HP 2004, NP 2003 et NP 2004. Ces données ont été collectées lors de différentes compétitions de RI entre 2003 et 2004. Chaque base de données contient entre 50 et 150 requêtes avec au maximum une liste de 1000 documents étiquetés par requête. Pour chaque requête, chaque document a une représentation sous la forme d'un vecteur d'une soixantaine de caractéristiques généralement utilisées en RI (score d'Okapi, de modèles à langage, de Pagerank...). Une description détaillée des bases de données peut être trouvée sur le site de Letor.

Chaque base de données est divisée en 5 dont chaque partie contenant spécifiquement une base d'apprentissage ($3/5$ des requêtes), une base de validation ($1/5$) et une base de test ($1/5$). Lors de nos expériences nous évaluons suivant les deux critères couramment utilisés en RI : le MAP (Yates *et al.,* 1999), et le NDCG (Järvelin *et al.,* 2000).

Définition des poids de l'OWA

Nous devons définir les poids de l'opérateur OWA pour se positionner dans une approche de classification ordonnée par paires (OWPC). On se restreint à de simples combinaisons de poids de la forme : $\alpha_j = g(j,n)/\sum_{k=1}^{n} g(k,n)$, où $g : \mathbb{N}^{*2} \rightarrow \mathbb{R}^+$ est appelée *fonction génératrice*, et n est le nombre de documents non pertinents de la requête[5]. On va considérer trois types de fonctions génératrices : **(1)** des poids constants sur les premiers $p\%$ de la liste que l'on notera $g_{=1}^{p\%} : g_{=1}^{p\%}(j,n) = 1$ si $j/n \leq p\%$ et 0 sinon. **(2)** des poids linéairement décroissants : notés $g_{/} : g_{/}(j,n) = 1/j$. **(3)** des poids exponentiellement décroissants : notés $g_{exp}^{p\%} : g_{exp}^{p\%}(j,n) = 2^{-100/p*j/n}$ (Le poids est divisé par 2 tous les $p\%$ de la liste).

Protocole expérimental

Nous avons fait 2 types d'expériences : **(1)** les poids sont fixés arbitrairement sans s'être préoccupé des données, **(2)** les poids sont considérés comme des hyperparamètres de l'algorithme et sont sélectionnés sur l'ensemble de validation. Dans tous les cas, pour une combinaison de poids donnée et pour chaque base de données, l'hyperparamètre C de l'équation [7] est choisi dans l'intervalle $\{10^{-3}, 10^{-2}, ..., 10^{3}\}$ grâce au score en MAP sur l'ensemble de validation. Lorsque les poids sont considérés comme des hyperparamètres, la valeur optimale de C est choisie en première pour chaque fonction génératrice qui elle sera choisie si elle a la meilleure valeur MAP sur l'ensemble de validation. Dans ce cas, les fonctions génératrices suivantes ont été utili-

---

3. http ://research.microsoft.com/en-us/um/beijing/projects/letor/index.html
4. Les résultats sur OHSUMED ne sont pas reportés car nous nous sommes focalisés sur un ordonnancement biparti
5. On rappelle que les poids dépendent du nombre de documents non pertinents pour une requête due aux contraintes de normalisation $\sum_{j=1}^{n} \alpha_j = 1$.

sées : les poids décroissants linéairement, ($g_{/}$), les poids constants sur les premiers $p\%$ ($g_{=1}^{p\%}$) et les poids exponentiellement décroissants ($g_{exp}^{p\%}$) avec $p \in \{5, 10, 20\}$. Nous avons aussi inclus le modèle de base $g_{=1} \stackrel{def}{=} g_{=1}^{100}$, correspondant à la combinaison standard de la classification par paires.

Influence des poids

Tout d'abord, nous montrons dans une première expérience, l'effet produit lorsque l'on se focalise sur les premiers $p\%$ de la liste (avec un poids constant, i.e. la fonction génératrice $g_{=1}^{p\%}$). Les performances de test en MAP sur les 6 ensembles de données sont dans le tableau 1. Par souci de clarté nous n'avons mis que les valeurs pour $p = 5\%$, $p = 10\%$ et les poids constants sur toute la liste (la fonction génératrice $g_{=1}$). $g_{=1}$ est donc notre modèle témoin qui correspond au nombre moyen d'erreur de paires mal ordonnées. Comme attendu, se focaliser au début de la liste donne une amélioration significative des performances en terme de MAP par rapport à $g_{=1}$ sur 5 bases de données sur 6. Enfin, se concentrer sur les premiers 5% de la liste donne les meilleurs performances sur 4 bases sur 6.

Comparaison avec d'autres algorithmes

Les tableaux 2, 3 et 4 montrent la performance de l'OWPC en termes de MAP, NDCG@1 et NDCG@3 sur les 6 bases de données. Pour synthétiser, nous avons mis les fonctions $g_{/}$, qui tendent à avoir les meilleures performances en général, et les résultats des poids considérés comme hyperparamètres g_{All} choisis sur l'ensemble de validation.

Dans ces tableaux, nous rapportons aussi les résultats d'algorithmes de l'état de l'art tels que : RankingSVM (Joachims, 2002), SVM^{MAP}(Yue *et al.,* 2007), Adarank[6] (Xu *et al.,* 2007) et ListNet (Cao *et al.,* 2007). Ces résultats sont disponibles sur le site de Letor[7]. On note, en gras les résultats de l'OWPC s'ils sont meilleurs que les autres modèles, ou l'un des modèles de base est meilleur que l'un des deux OWPC.

Le modèle avec les poids linéairement décroissants ($g_{/}$) a des performances significativement meilleurs que les modèles de base (ou est aussi bon) sur 4 bases de données sur 6 selon les trois mesures ($5/6$ pour NDCG@1). Ces résultats prouvent la pertinence de notre approche pour l'ordonnancement mais aussi que les poids linéairement décroissants peuvent être un bon choix par défaut comme fonction génératrice. Le modèle avec les poids choisis sur l'ensemble de validation (g_{All}), quant à lui améliore ou donne les mêmes performances sur 4 bases sur 6 en termes de MAP et NDCG@1. Pour NDCG@3, il a les mêmes performances que ListNet. Une fois de plus, ceci montre que l'approche OWPC donne des résultats dignes de l'état de l'art. La comparaison entre nos deux modèles, le modèle g_{All} tend à avoir un moins bon

6. Pour cet algorithme, on a mis que les résultats d'AdaRank^{map} pour le MAP, et d'adaRank^{ndcg} pour le NDCG.
7. Ainsi que les résultats d'autres algorithmes mais nous nous sommes contentés de donner les performances des 4 meilleurs en général

	TD03	TD04	HP03	HP04	NP03	NP04
RSVM	0.320	0.413	0.693	0.573	**0.580**	0.507
SVM^{map}	0.320	0.293	0.713	**0.627**	0.560	0.520
Adarank	0.360	0.427	0.713	0.587	0.560	0.507
ListNet	0.400	0.360	**0.720**	0.600	0.567	0.533
owpc g_{All}	**0.420**	0.333	**0.727**	**0.627**	0.567	**0.560**
owpc $g_{/}$	**0.440**	**0.453**	**0.720**	0.613	**0.580**	**0.560**

Tableau 3. *Ensemble de test avec NDCG@1 d'OWPC et des modèles de base de Letor.*

	TD03	TD04	HP03	HP04	NP03	NP04
RSVM	**0.344**	0.347	0.775	0.715	0.765	0.750
SVM^{map}	0.320	0.304	0.779	0.754	**0.767**	0.749
Adarank	0.291	**0.369**	0.790	0.751	0.716	0.672
ListNet	0.337	0.357	**0.813**	0.721	0.758	**0.759**
owpc g_{All}	0.342	0.349	0.782	**0.808**	0.737	**0.775**
owpc $g_{/}$	**0.361**	**0.371**	0.794	**0.800**	0.731	**0.759**

Tableau 4. *Ensemble de test avec NDCG@3 d'OWPC et des modèles de base de Letor*

MAP et NDCG@1 ce qui doit être dû à un phénomène de surapprentissage sur l'ensemble de validation car les bases de données sont petites et que le nombre possible de modèles pour g_{All} est grand.

5. Expériences sur Inex

Dans cette partie, on présente les expériences menées sur le corpus de document Wikipedia proposé dans le cadre de la compétition INEX[8]. Nous avons étiqueté manuellement une vingtaine de requêtes prises aléatoirement entre les éditions 2006 et 2008 d'INEX ce qui nous a permis de constituer une base d'apprentissage pour comparer nos modèles. La base de test quant à elle, est composée de 69 requêtes étiquetées par les participants du challenge. On représente les documents retournés pour chaque requête suivant une quinzaine de caractéristiques dites de contenu à savoir différents scores BM25, de modèles à langage.... Concernant l'évaluation de nos modèles, nous utilisons la mesure MAP (Yates *et al.,* 1999) car les documents ont un degré de pertinence binaire.

8. http ://www.inex.otago.ac.nz

	$g_{=1}$	$g_{=1}^{10\%}$	$g_{=1}^{5\%}$	$g_{log/}$	$g_{/}$
INEX'09	0.3498	0.3506	0.3492	0.3525	**0.3528**

Tableau 5. *Performance en test de l'OWPC en termes de MAP sur INEX'09 pour différents poids de l'OWA. $g_{=1}$ est le poids constant, $g_{=1}^{p\%}$ donne un poids constant sur les premiers $p\%$ de la liste (donc tous les autres poids valent 0 après le seuil $p\%$), $g_{/}$ est un poids linéairement décroissant et enfin $g_{log/}$ est un poids linéairement décroissant de façon logarithmique.*

Définition des poids de l'OWA

D'une manière similaire à la section 4 nous définissons les poids de l'opérateur OWA suivant les fonctions génératrices suivantes : les poids constants sur les premiers $p\%$ de la liste $g_{=1}^{p\%}$ avec $p \in \{5, 10, 100\}$, les poids linéairement décroissants $g_{/}(j, n) = 1/j$ et enfin les poids linéairement décroissants de façon logarithmique $g_{log/}(j, n) = 1/log(j)$.

Protocole expérimental

Nous nous sommes positionnés dans le même protocole expérimental que la section 4 où nous avons comparé les comportements des fonctions génératrices sur la base de test INEX'09. Cependant, n'ayant pas d'ensemble de validation, l'hyperparamètre C de l'équation [7] est choisi dans l'intervalle $\{10^{-3}, 10^{-2}, ..., 10^{3}\}$ grâce au score en MAP sur l'ensemble d'apprentissage.

Influence des poids

Nous cherchons à montrer, l'influence produite lorsque l'on fixe les poids de l'opérateur OWA de manière de plus en plus agressive afin de se focaliser sur le haut de la liste afin de valider les résultats de la section 4. Les performances en test sont en MAP sur la base INEX'09 et peuvent être retrouvés dans le tableau 5. Pour rappel, $g_{=1}$ est notre modèle témoin qui correspond au nombre moyen de paires mal ordonnées. Comme entrevu auparavant, plus on se focalise au début de la liste plus on améliore les performances en terme de MAP par rapport à notre modèle témoin. Enfin, il semblerait, tout comme dans la section 4 que les poids linéairement décroissants donnent des meilleures performances par rapport à des modèles moins agressif sur le haut de la liste. D'un point de vue expérimental, ceci nous conforte donc dans l'idée que choisir une fonction génératrice avec des poids linéairement décroissants est un bon choix.

6. Conclusion

Nous avons présenté une extension de la classification par paires mais en agrégeant les erreurs de classification suivant les opérateurs OWA plutôt que la moyenne. Ceci nous a permis d'avoir des fonctions de coût convexes se focalisant sur le haut de liste ordonnée qui sont plus faciles à optimiser. L'approche est en théorie proche de

la classification multiclasses ce qui peut s'implanter facilement avec des méthodes à vaste marge avec sorties structurées en guise d'algorithmes d'optimisation. Nos expériences montrent dans un premier temps qu'en fixant les poids de notre algorithme afin de favoriser le haut de la liste donnait en général de meilleures performances que des poids optimisant le rang moyen. Ensuite, dans un deuxième temps les performances de notre modèle ont de meilleures performances que les algorithmes de l'état de l'art sur un benchmark reconnu en apprentissage de fonctions d'ordonnancement.

7. Bibliographie

Bordes A., Bottou L., Gallinari P., Weston J., « Solving multiclass support vector machines with LaRank », *ICML*, p. 89-96, 2007.

Bordes A., Usunier N., Bottou L., « Sequence Labelling SVMs Trained in One Pass », *ECML PKDD '08*, p. 146-161, 2008.

Burges C. J. C., Ragno R., Le Q. V., « Learning to Rank with Nonsmooth Cost Functions », *NIPS*, p. 193-200, 2006.

Burges C. J. C., Shaked T., Renshaw E., Lazier A., Deeds M., Hamilton N., Hullender G. N., « Learning to rank using gradient descent », *ICML*, p. 89-96, 2005.

Cao Y., Xu J., Liu T.-Y., Li H., Huang Y., Hon H.-W., « Adapting ranking SVM to document retrieval », *SIGIR*, p. 186-193, 2006.

Cao Z., Qin T., Liu T.-Y., Tsai M.-F., Li H., « Learning to rank : from pairwise approach to listwise approach », *ICML*, p. 129-136, 2007.

Cohen W. W., Schapire R. E., Singer Y., « Learning to Order Things », *NIPS*, 1997.

Cossock D., Zhang T., « Subset Ranking Using Regression », *COLT*, p. 605-619, 2006.

Crammer K., Singer Y., « Pranking with Ranking », *NIPS*, p. 641-647, 2001.

Freund Y., Iyer R., Schapire R. E., Singer Y., « An efficient boosting algorithm for combining preferences », *JMLR*, vol. 4, p. 933-969, 2003.

Gyöngyi Z., Garcia-Molina H., Pedersen J. O., « Combating Web Spam with TrustRank », *in* , M. A. Nascimento, , M. T. Özsu, , D. Kossmann, , R. J. Miller, , J. A. Blakeley, , K. B. Schiefer (eds), *VLDB*, Morgan Kaufmann, p. 576-587, 2004.

Har-Peled S., Roth D., Zimak D., « Constraint Classification for Multiclass Classification and Ranking », *NIPS*, p. 785-792, 2002.

Järvelin K., Kekäläinen J., « IR evaluation methods for retrieving highly relevant documents », *SIGIR*, ACM, New York, NY, USA, p. 41-48, 2000.

Joachims T., « Optimizing search engines using clickthrough data », *KDD*, p. 133-142, 2002.

Kleinberg J. M., « Authoritative sources in a hyperlinked environment », *J. ACM*, vol. 46, n° 5, p. 604-632, 1999.

Le Q. V., Smola A. J., Direct Optimization of Ranking Measures, Technical report, NICTA, 2007.

Liu T.-Y., Xu J., Qin T., Xiong W., Li H., « LETOR : Benchmark dataset for research on learning to rank for information retrieval », *LR4IR 2007, in conjunction with SIGIR 2007*, 2007.

Liu Y.-T., Gao B., Liu T.-Y., Zhang Y., Ma Z., He S., Li H., « BrowseRank : letting web users vote for page importance », *in* , S.-H. Myaeng, , D. W. Oard, , F. Sebastiani, , T.-S. Chua, , M.-K. Leong (eds), *SIGIR*, ACM, p. 451-458, 2008.

Page L., Brin S., Motwani R., Winograd T., The PageRank Citation Ranking : Bringing Order to the Web., Technical Report n° 1999-66, Stanford InfoLab, November, 1999. Previous number = SIDL-WP-1999-0120.

Taylor M., Guiver J., Robertson S., Minka T., « SoftRank : optimizing non-smooth rank metrics », *WSDM '08*, ACM, p. 77-86, 2008.

Tsai M.-F., Liu T.-Y., Qin T., Chen H.-H., Ma W.-Y., « FRank : a ranking method with fidelity loss », *SIGIR*, p. 383-390, 2007.

Tsochantaridis I., Joachims T., Hofmann T., Altun Y., « Large Margin Methods for Structured and Interdependent Output Variables », *Journal of Machine Learning Research*, vol. 6, p. 1453-1484, 2005.

Volkovs M., Zemel R. S., « BoltzRank : learning to maximize expected ranking gain », *in* , A. P. Danyluk, , L. Bottou, , M. L. Littman (eds), *ICML*, vol. 382 of *ACM International Conference Proceeding Series*, ACM, p. 137, 2009.

Xu J., Li H., « AdaRank : a boosting algorithm for information retrieval », *SIGIR*, p. 391-398, 2007.

Xu J., Liu T.-Y., Lu M., Li H., Ma W.-Y., « Directly optimizing evaluation measures in learning to rank », *SIGIR*, p. 107-114, 2008.

Yager R. R., « On ordered weighted averaging aggregation operators in multi-criteria decision making », *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, p. 183-190, 1988.

Yates R. B., Ribeiro-Neto B., *Modern Information Retrieval*, Addison Wesley, 1999.

Yue Y., Finley T., Radlinski F., Joachims T., « A support vector method for optimizing average precision », *SIGIR*, p. 271-278, 2007.