Apprentissage de fonctions d'ordonnancement avec un flux de données non-étiquetées

Vinh Truong[1], Massih-Reza Amini[2], Patrick Gallinari[1]

[1] Université Pierre et Marie Curie
Laboratoire d'Informatique de Paris 6
104, avenue du Président Kennedy
75016 Paris, France
{Prenom.Nom}@lip6.fr

[2] National Research Council Canada
Institute for Information Technology
283, boulevard Alexandre-Taché
Gatineau, QC, J8X 3X7, Canada
{Prenom.Nom}@nrc-cnrc.gc.ca

Résumé : Dans ce papier, nous traitons de l'apprentissage de fonctions d'ordonnancement bipartite avec des données partiellement étiquetées. Contrairement aux études précédentes, nous supposons que les données non-étiquetées arrivent en grande quantité de façon séquentielle. Ce cadre évite de garder en mémoire toute la base d'apprentissage et permet de traiter les applications de routage d'information faisant intervenir des flux de données. La méthode proposée peut être vue comme une extension des modèles auto–apprenants proposés en classification semi–supervisée. Le modèle est d'abord initialisé sur les instances étiquetées puis traite à la volée les données non-étiquetées en continu. Notre algorithme se base sur une méthode d'optimisation en ligne des SVMs linéaires. Les expériences menées sur un grand nombre de collections montrent que le flux de données non-étiquetées permet d'améliorer les performances d'une fonction apprise uniquement sur les instances étiquetées.

1 Introduction

La tâche d'ordonnancement suscite depuis quelques années un grand intérêt dans la communauté d'apprentissage. Cette tâche est principalement motivée par les applications qui cherchent à ordonner les entrées entre elles (Iyer *et al.*, 2000; Christopher D. Manning & Schütz, 2008; Robertson & Soboroff, 2001). C'est le cas par exemple du routage d'information, où le système reçoit des données sous la forme de flux de documents et doit les ordonner par rapport aux préférences de l'utilisateur. Dans cet article, nous nous intéressons au cas où l'utilisateur juge les entrées comme étant PERTINENT ou NON-PERTINENT par rapport à une thématique donnée. Pour

apprendre un système d'ordonnancement, il est en général nécessaire d'avoir une base d'entraînement étiquetée. Or, pour la plupart des applications, ceci revient à examiner manuellement les données et à les évaluer, ce qui limite automatiquement la taille de la base d'apprentissage. Pour y remédier, le cadre d'apprentissage semi–supervisée a été proposé en classification où il s'agit d'apprendre un classifieur sur peu d'exemples étiquetées et beaucoup d'exemples non-étiquetés.

Récemment, quelques méthodes ont été proposées pour apprendre une fonction de score avec des exemples partiellement étiquetés (Zhou *et al.*, 2004; Agarwal, 2006; Duh & Kirchhoff, 2008; anonymes, 2008). La plupart de ces approches sont développées suivant le cadre transductif et apprennent une fonction d'ordonnancement pour ordonner les exemples non-étiquetés de la base d'apprentissage. La limitation de ce cadre est qu'il n'est pas adapté pour les applications où il y a un flux de données en continu comme la tâche de Routage, et que pour un nouvel ensemble non-étiqueté il faut réapprendre une nouvelle fonction d'ordonnancement.

Dans cet article, nous nous sommes intéressés à l'apprentissage d'une fonction score capable d'ordonner les exemples entrants. De plus, nous proposons de traiter ce problème dans un nouveau cadre, où les données non-étiquetées arrivent sous la forme d'un flux continu. Ce cadre présente un double avantage. D'un point de vue applicatif, il permet de mieux modéliser le routage d'information et d'un point de vue algorithmique, il évite de stocker en mémoire l'ensemble des données. Notre méthode s'inspire des modèles auto–apprenants proposés en classification (Abney, 2004; Joachims, 1999; Yarowsky, 1995) et des algorithmes d'optimisation en ligne (Bordes *et al.*, 2007; Crammer *et al.*, 2006). Concrètement, notre algorithme est initialisé sur la base étiquetée et traite les données non-étiquetées à la volée et en continue. Après chaque exemple non-étiqueté, il estime sa pertinence (principe auto–apprenant) et met à jour le modèle (apprentissage en ligne) en augmentant ou en diminuant son score tout en respectant l'ordre sur la base étiquetées initiale.

Le plan de ce papier est comme suit ; nous allons dans un premier temps faire un bref rappel de l'apprentissage supervisé de fonctions d'ordonnancement bipartite (section 2). Nous présenterons ensuite notre modèle (section 3) et discuterons des résultats obtenus sur différentes collections de documents réelles à la section 4. La conclusion de ce travail est donnée à la section 5.

2 Apprentissage supervisé de fonctions d'ordonnancement bipartite

Dans cet article, nous nous intéressons à l'ordonnancement bipartite, qui est un cas particulier de l'ordonnancement d'instances. Il s'agit d'apprendre à partir d'un ensemble $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ de n exemples $x_i \in \mathbb{R}^d$ muni d'un jugement de préférence y_i. Dans le cas bipartite, nous considérons un jugement de valeur binaire, qui reflète, pour un profil utilisateur donné, soit la pertinence d'un exemple ($y = 1$) soit sa non–pertinence ($y = -1$). L'apprentissage consiste à trouver une fonction $h : \mathbb{R}^d \rightarrow \mathbb{R}$

qui donne un score plus élevé aux exemples pertinents qu'aux exemples non-pertinents. Formellement, cela revient à minimiser le nombre de paires d'exemples pertinent/non–pertinent mal ordonnées :

$$\mathcal{E}(h;\mathcal{D}) = \frac{1}{|\mathcal{D}_1||\mathcal{D}_{-1}|} \sum_{x\in\mathcal{D}_1} \sum_{x'\in\mathcal{D}_{-1}} [\![h(x) < h(x')]\!]$$

où $[\![p]\!]$ est la fonction indicatrice valant 1 si le prédicat p est vrai et vaut 0 sinon. $\mathcal{D}_1$ (resp. $\mathcal{D}_{-1}$) est l'ensemble des exemples pertinents (resp. non-pertinent) de $\mathcal{D}$. Nous pouvons noter qu'il existe un lien entre l'erreur d'ordonnancement et l'aire sous la courbe ROC (AUC) puisque $\text{AUC}(h;\mathcal{D}) \geq 1 - \mathcal{E}(h;\mathcal{D})$.

Pour résoudre le problème d'optimisation, nous pouvons utiliser une fonction convexe comme majorant de $[\![x]\!]$, en prenant par exemple la fonction de perte *hinge* : $[\![x]\!]_+ = \max(0, 1-x)$. Dans le cas linéaire, l'apprentissage revient au problème d'optimisation (Herbrich *et al.*, 2000; Joachims, 2002; Rakotomamonjy, 2004; Brefeld & Scheffer, 2005) suivant :

$$\min_w \frac{1}{2}\|w\|^2 + C \sum_{x\in\mathcal{D}_1} \sum_{x'\in\mathcal{D}_{-1}} [\![\rho - w(x-x')]\!]_+ \tag{1}$$

avec $\rho \in \mathbb{R}^+$ la marge et C un coefficient permettant de pondérer l'apport de la base d'apprentissage dans la fonction objective. Le premier terme est un régulariseur utilisé pour limiter la complexité de la solution. Nous pouvons remarquer que l'ordonnancement traitée ainsi est équivalent à la classification de paires d'exemples pertinents/non–pertinents. Dans la littérature ces paires sont dites cruciales (Freund *et al.*, 2003).

3 Apprentissage semi supervisé de fonctions d'ordonnancement

En apprentissage semi–supervisé, nous considérons en plus de la base étiquetée un flux de données non-étiquetées $\{z_t\}_t$ provenant de la même source qui a permis de construire la base étiquetée. Ainsi l'apprenant possède initialement une base d'exemples étiquetés puis reçoit un à un des exemples non-étiquetés. Un cadre similaire a été récemment proposé en classification semi–supervisé (Goldberg *et al.*, 2008) : les exemples arrivent en ligne mais il y a une faible probabilité pour que l'exemple soit étiqueté.

3.1 Le principe de l'auto–apprentissage en ligne

Les méthodes d'apprentissage en ligne pour la classification suit généralement le schéma suivant : L'algorithme reçoit les exemples étiquetés un à un puis, après chaque observation, il estime sa classe et met à jour le modèle en fonction de l'erreur produite par cette prédiction. À chaque étape, la mise à jour peut être vue comme un compromis par rapport à ce qu'il a appris (état courant) et l'information contenue dans l'exemple étiqueté. Nous proposons d'adapter ce schéma pour l'apprentissage semi–supervisé et

pour la tâche d'ordonnancement.

Pour exploiter les données non-étiquetées, nous utilisons les fondements des modèles auto–apprenants (Zhu, 2005). Ces modèles proposés pour la classification semi–supervisée sont en premier initialisés sur la base étiquetée. Ils étiquettent les données qui ne le sont pas puis apprennent un modèle sur la base étiquetée et l'ensemble obtenu. Malgré sa simplicité, ce principe est appliqué avec succès pour les parseurs syntaxiques (Mcclosky *et al.*, 2006), la classification du degré de subjectivité de phrases (Wang *et al.*, 2008) ou en robotique (Wellington & Stentz, 2004).

Pour combiner ces deux principes, nous nous donnons une procédure d'étiquetage $\mathcal{A}$. Dans le cas de la classification, cette procédure est simple puisqu'elle utilise simplement la sortie du modèle. Pour l'ordonnancement, elle est moins naturelle mais nous en définissons une dans la section 3.2.2. Nous pouvons énoncer le principe général de l'auto–apprentissage en ligne : après chaque exemple non-étiqueté, la procédure d'étiquetage $\mathcal{A}$ est appliquée puis le modèle est mis à jour en fonction de l'erreur induite. Contrairement au cadre supervisé, l'étiquetage n'est malheureusement pas parfait et les erreurs commises peuvent s'amplifier pendant l'apprentissage. Pour limiter cette propagation, nous assurons que le modèle mis à jour reste cohérent avec la base étiquetée. L'erreur induite à chaque itération doit ainsi tenir compte des exemples étiquetés. Les étapes suivantes permettent de résumer le fonctionnement général :

1. Initialisation sur la base d'apprentissage
2. A l'étape t, l'algorithme reçoit un exemple non-étiqueté z_t
3. L'exemple reçoit l'étiquette par la procédure $\mathcal{A}$
4. Le modèle peut être mis à jour en tenant en compte de $(x, \mathcal{A}(x))$ suivant une erreur spécifique sur $\mathcal{D} \cup \{z_t, \phi(z_t)\}$.

– Et ainsi de suite.

Pour l'ordonnancement bipartie, l'erreur spécifique peut être l'erreur d'ordonnancement sur $\mathcal{D} \cup \{z_t, \phi(z_t)\}$ ou toute autre variante. Nous proposons par la suite une implémentation efficace du principe énoncée ci–dessus.

3.2 L'algorithme SLARANK pour l'ordonnancement semi–supervisé

3.2.1 Implémentation avec un solveur en ligne

Cette implémentation repose sur une méthode de classification en ligne. Comme dans (Grangier & Bengio, 2008) [1], l'ordonnancement en ligne est traité comme de la classification de paires en ligne : au lieu de fournir des exemples étiquetés, nous fournissons au classifieur des paires cruciales. Nous utilisons dans ce papier le solveur développé par (Bordes *et al.*, 2007), qui permet en une seule passe sur la base d'apprentissage d'obtenir une bonne approximation de la solution.

1. cas supervisé dans le papier.

Par conséquent, lorsqu'un exemple non-étiquetée z_t arrive, nous estimons le degré de préférence de z_t. Nous considérons alors l'ensemble des paires cruciales mal ordonnées et nous fournissons au solveur en ligne la paire la plus violée (cf. algorithme 3). Formellement, cela revient à sélectionner l'exemple par :

$$\Delta(z_t, w, \mathcal{D}) = \begin{cases} \underset{x \in \mathcal{D}_1}{\text{argmax}} \quad [\![\rho - w.(z_t - x)]\!] & \text{si } \widetilde{y_t} > 0 \\ \underset{x' \in \mathcal{D}_{-1}}{\text{argmax}} \, [\![\rho - w.(x' - z_t)]\!] & \text{sinon} \end{cases}$$

Après la mise à jour, nous vérifions la cohérence dans la base d'apprentissage, en fournissant les paires cruciales de la base étiquetées mal ordonnée (cf. algorithme 2). Le pseudo–code de notre méthode est décrit par l'algorithme 1. Notons que dans l'algorithme, la mise à jour n'est pas faite à toutes les étapes mais uniquement quand les conditions de mise à jour sont remplies. Cette mise à jour est faite lorsque l'estimation du degré de performance est considérée comme sûre (cf section 3.2.2).

Algorithm 1 squelette de l'algorithme SLARANK

ENTRÉES: Un ensemble d'exemples étiquetés $\mathcal{L}$ et un ensemble d'exemples non-étiquetés $\mathcal{U}$

Apprendre une fonction score sur $\mathcal{L}$:
pour t=1,2,... **faire**
  Recevoir un exemple non-étiqueté z_t
  Calculer son score $h(z_t) = w.z_t$
  Estimer son degré de pertinence : $\widetilde{y_t} = \mathcal{A}(z_t)$
  si les conditions sont réunies **alors**
    MAJ $(w, (z_t, y_t), \mathcal{L})$
  finsi
  pour $(x, y) \in \mathcal{L}$ **faire**
    MAJ $(w, (x, y), \mathcal{L})$
  fin pour
fin pour

SORTIES: le modèle

3.2.2 Estimation du degré de pertinence et conditions d'utilisation

Un point crucial de l'algorithme est l'estimation du degré de pertinence d'un exemple. Comme l'objectif de l'ordonnancement n'est pas la classification, cette estimation n'est pas naturelle. Cependant, nous supposons dans ce papier, que plus le score d'une exemple non-étiqueté est élevé par rapport aux scores des exemples non-pertinents, plus cet exemple est vraisemblablement pertinent. L'écart de scores permet alors de mesurer à quel point l'exemple peut être considéré comme pertinent.

Soit une fonction $d_h(S, S')$, qui mesure la *différence relative* des scores entre les exemples de S et de S' tel que le signe de cette fonction indique que les éléments de S

Algorithm 2 Processus de mise à jour 1

ENTRÉES: un modèle w_t, une instance étiquetée $(z_t, \widetilde{y_t})$ et un ensemble étiqueté $\mathcal{L}$

$\rho_{min} \leftarrow +\infty$
pour $(x_i, y_i) \in \mathcal{L}$ **faire**
  si $y_i \neq \widetilde{y_t}$ et $\widetilde{y_t}.w_t.(z_t - x_i) \leq \rho_{min}$ **alors**
    $i_{min} \leftarrow i$
    $\rho_{min} \leftarrow \widetilde{y_t}.w_t.(z_t - x_i)$
  finsi
fin pour
si $\rho_{min} \leq 0$ **alors**
  Ajouter $(\widetilde{y_t}.(z_t - x_{imin}, 1)$ au solveur SVM en ligne
finsi

SORTIES: w_{t+1}

Algorithm 3 Processus de mise à jour 2

ENTRÉES: un modèle w_t, une instance étiquetée $(z_t, \widetilde{y_t})$ et un ensemble étiqueté $\mathcal{L}$

pour $(x_i, y_i) \in \mathcal{L}$ **faire**
  si $y_i \neq \widetilde{y_t}$ et $y_i.w_t.(x_i - x_t) \leq \rho$ **alors**
    Former la paire cruciale p : p = sign$(y_i - \widetilde{y_t}).(x_i - z_t)$
    Ajouter $(p, 1)$ au solveur SVM en ligne
  finsi
fin pour

SORTIES: w_{t+1}

sont globalement au–dessus de S' et que sa valeur absolue mesure à quel point les scores sont distants. En particulier, nous utilisons dans ce papier simplement la moyenne de la différence des scores : $d_h(S, S') = \frac{1}{|S||S'|} \sum_{x \in S} \sum_{x' \in S'} h(x) - h(x')$. Sur cette base, nous définissons alors les critères suivants :

- $\delta_h^+(x) = d_h(\{x\}, \mathcal{S}_-)$, (mesure relative de pertinence)
- $\delta_h^-(x) = d_h(\mathcal{S}_+, \{x\})$, (mesure relative de non–pertinence)
- $\delta_h(x) = \frac{\left|\delta_h^+(x) - \delta_h^-(x))\right|}{d_h(\mathcal{S}_1, \mathcal{S}_{-1})}$, (mesure de confiance)

Une *mesure relative de pertinence* positive (rel. négative) montre que le score de l'exemple x est en moyenne au dessus (rel. en dessous) des scores des exemples non–pertinents de la base d'apprentissage. Ainsi en se basant sur l'hypothèse de départ, plus cette valeur est grande plus cet exemple sera considéré comme pertinent. Par conséquence, x est supposé pertinent lorsque $\delta_h^+(x) > \delta_h^-(x)$. Dans le cas contraire, l'exemple est supposé être non-pertinent.

$$\mathcal{A}(x) = \begin{cases} 1 & \text{si } \delta_h^+(x) > \delta_h^-(x) \\ -1 & \text{sinon} \end{cases}$$

Finalement, $\delta_h(x)$ permet de résumer ces deux valeurs et peut être utilisée en tant que mesure de confiance dans l'estimation du degré de pertinence.

En classification, les SVM transductives peuvent trouver une solution dégradée en étiquetant tous les exemples dans la même classe (Chapelle *et al.*, 2008). Pour éviter ce problème, ces méthodes s'assurent que le taux de positives trouvés dans la base non-étiquetées est la même que le taux de positive dans la base étiquetée. Comme nous avons observé un comportement similaire, nous adaptons cette approche pour l'optimisation en-ligne. Après chaque donnée non-étiquetée, nous tirons aléatoirement le degré de pertinence à attribuer avec une probabilité égale à ce taux. Si la décision concorde, alors la mise à jour est effective. Le pseudo–code 4 permet de résumer les conditions nécessaires pour assigner un jugement de préférence. Il est paramétré par une valeur réelle positive $s \in [0, 1]$.

Algorithm 4 Conditions d'étiquetage

ENTRÉES: le taux des données pertinentes p, un exemple non-étiqueté z_t, la fonction $\mathcal{A}$, un seuil s

$r = rand(0, 1)$
si $r < p$ **alors**
  $y_r \leftarrow$ PERTINENT
sinon
  $y_r \leftarrow$NON-PERTINENT
finsi
si y_r est égal à $\mathcal{A}(z_t)$ et $\delta_h(x) < s$ **alors**
  retourne VRAI
sinon
  retourne FAUX
finsi

3.2.3 Complexité

Dans la version linéaire, le solveur en ligne ne nécessite que le stockage du modèle initial, des exemples de la base d'apprentissage étiquetés (ou des paires cruciales en résultant) et d'un exemple non-étiqueté. La complexité spatiale de l'algorithme est donc de $O(n_+n_-)$ avec n_+ et n_- le nombre de données pertinentes et non pertinentes dans la base étiquetée.

À chaque itération, l'algorithme a besoin de calculer les scores des données étiquetées, de former la paire cruciale et de mettre à jour le modèle. Le solveur en ligne nécessite un nombre d'opérations proportionnel à d à chaque itération. Nous en déduisons que la complexité total de SLARANK est de $O\left(d.(n+m)\right)$, alors que les méthodes semi-supervisé en ordonnancement ont habituellement une complexité d'au moins $O\left((n+m)^2\right)$ (Zhou *et al.*, 2004).

4 Expériences

Nous avons évalué notre modèle sur des bases de références en apprentissage semi–supervisé. Pour chacune des bases, nous avons dérivé un problème de routage d'information en supposant que chaque classe correspond à un profil utilisateur. Les exemples appartenant à une classe donnée sont supposée être pertinents par rapport au profil donné. Dans le cas contraire, les exemples sont considérés comme non–pertinent.

Les bases USPS et COIL [2] sont des bases images utilisées initialement en classification multi-classes. Nous avons sélectionné les cinq premières classes pour nos profils utilisateurs. Les bases AUT-AVN et REAL-SIM proviennent de la collection USENET ARTICLES7 [3] [4] en isolant 4 groupes de discussions : simulation de course de voitures, simulation de pilotage d'avions, voitures réelles et aviation réelle.

Nous avons aussi utilisé la version originale de la base RCV1 (Lewis *et al.*, 2004). Chacune des données est représentée par un vecteur sac-de-mots en utilisant la technique TF–IDF. Chaque composante est calculée par rapport à l'ensemble d'apprentissage ou l'ensemble de test. Notons que dans nos expériences nous avons inversé le rôle de la base d'apprentissage et de la base de test initiale pour obtenir une base plus grande. Nous avons ainsi isolé $8\ 000$ exemples de l'ensemble d'apprentissage pour former aléatoirement les bases étiquetées et le reste constitue l'ensemble des données non-étiquetées. Nous avons pris les classes CCAT, ECAT et GCAT comme profil utilisateur. Les caractéristiques de l'ensemble des bases sont détaillées dans la table 1.

dataset	c	d	$n+m$	test set size	ratio d'ex. pertinents (r)
USPS	5	256	7,291	2,007	10%
COIL	5	1,024	1,440	1,000	5%
AUT–AVN	2	20,707	68,175	2,000	65%
REAL–SIM	2	20,958	69,201	2,000	31%
CCAT	2	47,152	701,275	23,149	47%
GCAT	2	47,152	701,275	23,149	30%
ECAT	2	47,152	701,275	23,149	15%

TABLE 1 – Propriétés des bases : c représente le nombre de profils utilisateurs générés, $n+m$ la taille de la base d'apprentissage (étiquetée et non-étiquetée), d la dimension du problème et r la proportion d'exemples pertinents dans la base

Les résultats obtenus dans la partie expérimentale sont des moyennes obtenues sur 10 partitions différentes de la base d'apprentissage comprenant les exemples étiquetés

2. `http ://www.kyb.tuebingen.mpg.de/bs/people/chapelle/lds/`
3. La version originale :`http ://www.cs.umass.edu/ mccallum/code-data.html`
4. La version prétraitée : `http ://people.cs.uchicago.edu/∼vikass/datasets/lskm/svml/`

et non-étiquetés. Les partitions sont formées aléatoirement en s'assurant que la base étiquetée contient des exemples des deux classes.

Pour évaluer la contribution des données non-étiquetées, nous avons utilisé comme référence le modèle supervisé SVM^{sup}, qui optimise l'AUC sur la base étiquetée avec une fonction de perte hinge (cf section 2). Une implémentation efficace est donnée dans (Joachims, 2005) et dans (Teo *et al.*, 2007).

Pour la méthode supervisé, le paramètre C est choisi dans l'ensemble $[10^{-4}, 10^{-2}, 1, 10^2, 10^4]$. Nous avons reporté le meilleur résultat pouvant être obtenu (en moyenne) sur l'ensemble de test. SLARANK possède deux hyperparamètres : le coefficient de régularisation C et le seuil s. Nous avons fixé le paramètre C à 1 et fait varier le seuil s dans l'ensemble $[0.1, 0.2, 0.3, 0.4, 0.5]$.

4.1 Stratégies pour la sélection de modèles

La sélection des modèles en apprentissage semi–supervisé reste encore un champ ouvert pour la recherche, surtout lorsqu'il y a très peu de données étiquetées. Dans ce papier, nous avons envisagé quatre stratégies différentes :

1. SLARANK* : En suivant le protocole utilisé dans (Chapelle *et al.*, 2006), nous gardons le modèle qui, pour des valeurs de paramètres fixés, obtient en moyenne le meilleur AUC sur les 10 partitions. Cette sélection n'est pas réaliste mais elle permet de voir le potentiel de l'algorithme.

2. SLARANK^{cv} : Nous utilisons dans ce cas une variation de la validation croisée. La base d'apprentissage étiqueté $\mathcal{D}$ est divisée en K sous–ensembles $\{\mathcal{D}_j^{cv}\}_{j=1}^{K}$. L'algorithme d'apprentissage est déroulé K fois en utilisant les même données non-étiquetées. À la $k^{\text{ième}}$ expérience, $\cup_{j\neq k} D_j^{cv}$ regroupe les exemples étiquetés et le modèle linéaire est évalué sur la $k^{\text{ième}}$ partition, $\mathcal{D}_k^{cv}$. Cependant, comme il y a peu d'exemples étiquetés, nous sélectionnons les paramètres du modèle en fonction de l'erreur d'ordonnancement grâce à la fonction de perte *hinge* (cf. équation 1).

3. SLARANK^{lno} : Pour des petites bases déséquilibrées, la validation croisée ne peut pas être appliquée, en particulier lorsqu'il n'y a qu'un seul exemple pertinent. Dans ce cas, nous adaptons le protocole *leave-one-out* en retirant un exemple négatif. La sélection se fait en utilisant l'écart moyen entre cet exemple et le reste de la base d'apprentissage étiquetée.

4. SLARANK$^{val}_{1000}$: Comme (Collobert *et al.*, 2006), nous faisons une sélection de modèle en maximisant l'AUC sur une base de validation contenant 1000 exemples. Quand cette stratégie est utilisée, nous avons constitué cette base avec des exemples de la base non-étiquetée pris d'une manière aléatoire.

4.2 Résultats expérimentaux

Nous avons faire une première série d'expériences pour comparer une méthode supervisée SVM^{sup} et de notre méthode semi–supervisé SLARANK. Nous avons utilisé un nombre restreint de données étiquetées : 10 pour USPS, 20 pour COIL et 100 pour AUT-AVN, REAL-SIM, CCAT, ECAT et GCAT. Nous avons aussi comparer les différentes stratégies de sélection de modèles pour SLARANK. Comme les bases images contiennent peu de données étiquetées, nous avons uniquement testé SLARANK^{lno}. Pour le reste, nous avons utilisé la validation croisée et une base de validation.

Les résultats sont montrés dans les tableaux 2,3 et 4. Pour l'ensemble des bases, notre méthode est plus performante que la méthode supervisée montrant qu'une exploitation efficace des données non étiquetées peut améliorer significativement les performances du modèle supervisé. Par contre, nous pouvons constater que les performances de SLARANK^{lno} restent bien en deçà de SLARANK* mais elles restent en moyenne meilleures que celles obtenues dans le cadre supervisé. Pour les bases de grande taille, nous pouvons constater que la validation croisée pour 100 exemples étiquetées permet d'avoir des résultats proches de celles obtenues par l'utilisation d'une base de validation.

Dataset	SVM^{sup}	SLARANK*	SLARANK^{lno}
USPS–1	88.7	**94.1**	93.2
USPS–2	**99.7**	**99.7**	99.5
USPS–3	90.5	**93.5**	91.6
USPS–4	89.5	**91.9**	90.5
USPS–5	87.1	**90.7**	90.3

TABLE 2 – Performance AUC des différents modèles sur la base USPS.

Dataset	SVM^{sup}	SLARANK*	SLARANK^{lno}
COIL–1	92.2	94.5	**95.5**
COIL–2	64.6	**70.6**	68.6
COIL–3	87.5	**87.9**	87.3
COIL–4	96.2	**98.2**	97.6
COIL–5	74.8	**77.5**	75.8

TABLE 3 – Performance AUC des différents modèles sur la base COIL.

Enfin dans le tableau 5, nous donnons un ordre de grandeur du temps consacré à l'apprentissage par notre algorithme. Les chiffres données sont une moyenne sur l'ensemble des expériences. Elles peuvent être comparer au temps d'optimisation de la fonction ojective SVM pour de la classification. Sur la base CCAT, une descente de gradient stochastique mais un peu moins de deux secondes [5].

5. `http ://leon.bottou.org/projects/sgd`

	SVM^{sup}	SLARANK*	SLARANK$^{val}_{1000}$	SARANK^{cv}
REAL–SIM	93.4	94.1	94.3	93.8
AUT–AVN	93.6	**95.6**	**95.6**	95.2
CCAT	89.5	91.5	**91.7**	91.5
GCAT	96.2	97.2	**97.4**	97.1
ECAT	83.7	**85.4**	85.3	**85.4**

TABLE 4 – Performance AUC des différents modèles sur les bases restantes.

base	temps
REAL–SIM	24.6
AUT–AVN	10.5
CCAT	41.4
GCAT	63.1
ECAT	34.3

TABLE 5 – Temps consacré à l'apprentissage (en secondes).

4.3 Évolution de l'AUC et la précision moyenne en fonction du nombre d'exemples étiquetés

Dans cette section, nous avons tracé l'évolution de l'AUC et de la précision moyenne sur l'ensemble de test en fonction des données étiquetés pour les collections RCV1, AUT–AVN and REAL–SIM. La précision moyenne est la moyenne des précisions sur l'ensemble des listes tronquées à partir d'un exemple pertinent.

La taille de la base étiquetée varie de 10 à 500. Les paramètres sont choisies avec une base de validation. Dans les graphes, notre méthode est représenté en rouge alors que le SVM^{sup} en bleu. L'ensemble des résultats montrent que SLARANK exploite efficacement les données non-étiquetées et obtient clairement de meilleures performances que la méthode supervisée.

FIGURE 1 – Évolution des mesures de performances AUC (à gauche) et précision moyenne (à droite) sur la base CCAT

5 Conclusion et perspectives

Dans cet article, nous avons proposé un nouvel algorithme semi–supervisé pour l'ordonnancement bipartite. Nous avons défini un nouveau cadre dans lequel, les données non étiquetées arrivent sous la forme d'un flux. L'approche proposée est basée sur le principe des modèles auto–apprenants et de l'optimisation en ligne. Dans la pratique, elle demande peu de ressources, seule la base étiquetée est stockée en mémoire et sa complexité est de l'ordre de $O\left(d.(n+m)\right)$.

FIGURE 2 – Évolution des mesures de performances AUC (à gauche) et précision moyenne (à droite) sur la base ECAT

FIGURE 3 – Évolution des mesures de performances AUC (à gauche) et précision moyenne (à droite) sur la base GCAT

FIGURE 4 – Évolution des mesures de performances AUC (à gauche) et précision moyenne (à droite) sur la base AUT-AVN

FIGURE 5 – Évolution des mesures de performances AUC (à gauche) et précision moyenne (à droite) sur la base REAL-SIM

Plusieurs directions pour un travail futur sont envisagées. En premier, une perspective intéressante serait de tester l'algorithme sur des données non étiquetées dont la distribution varie au cours du temps. Dans l'article (Goldberg *et al.*, 2008), les auteurs ont montré que l'approche en ligne était efficace pour ce genre de données. De plus, l'extension au cas non linéaire, qui reste coûteux en ordonnancement supervisé, et au co–training sont des pistes intéressantes.

Références

ABNEY S. (2004). Understanding the yarowsky algorithm. *Computational Linguistics*, **30**(3), 365–395.

AGARWAL S. (2006). Ranking on graph data. In *Proceedings of the 23rd International Conference on Machine Learning*.

ANONYMES A. (2008). Titre absent pour des raisons d'anonymat. In *SIGIR '08 : Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*.

BORDES A., BOTTOU L., GALLINARI P. & WESTON J. (2007). Solving multiclass support vector machines with larank. In Z. GHAHRAMANI, Ed., *Proceedings of the 24th International Machine Learning Conference*, p. 89–96, Corvallis, Oregon : OmniPress.

BREFELD U. & SCHEFFER T. (2005). Auc maximizing support vector learning. In *In Proc. ICML workshop on ROC Analysis in Machine Learning*.

O. CHAPELLE, B. SCHÖLKOPF & A. ZIEN, Eds. (2006). *Semi-Supervised Learning*. Cambridge, MA : MIT Press.

CHAPELLE O., SINDHWANI V. & KEERTHI S. S. (2008). Optimization techniques for semi-supervised support vector machines. *J. Mach. Learn. Res.*, **9**, 203–233.

CHRISTOPHER D. MANNING P. R. & SCHÜTZ H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

COLLOBERT R., SINZ F., WESTON J., BOTTOU L. & JOACHIMS T. (2006). Large scale transductive svms. *Journal of Machine Learning Research*, **7**, 2006.

CRAMMER K., DEKEL O., SHALEV-SHWARTZ S. & SINGER Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*.

DUH K. & KIRCHHOFF K. (2008). Learning to rank with partially-labeled data. In *SIGIR '08 : Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, p. 251–258, New York, NY, USA : ACM.

FREUND Y., IYER R., SCHAPIRE R. E. & SINGER Y. (2003). An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, **4**, 933–969.

GOLDBERG A. B., LI M. & ZHU X. (2008). Online manifold regularization : A new learning setting and empirical study. In *ECML PKDD '08 : Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, p. 393–407.

GRANGIER D. & BENGIO S. (2008). A discriminative kernel-based approach to rank images from text queries. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **30**(8), 1371–1384.

HERBRICH R., GRAEPEL T. & OBERMAYER K. (2000). *Large margin rank boundaries for ordinal regression*, In *Advances in Large Margin Classifiers*, p. 115–132. MIT Press.

IYER R. D., LEWIS D. D., SCHAPIRE R. E., SINGER Y. & SINGHAL A. (2000). Boosting for document routing. In *CIKM '00 : Proceedings of the ninth international conference on Information and knowledge management*, p. 70–77.

JOACHIMS T. (1999). Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning (ICML)*, p. 200–209, Bled, Slowenien.

JOACHIMS T. (2002). Optimizing search engines using clickthrough data. In *KDD '02 : Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, p. 133–142, New York, NY, USA : ACM Press.

JOACHIMS T. (2005). A support vector method for multivariate performance measures. In *ICML '05 : Proceedings of the 22nd international conference on Machine learning*, p. 377–384, New York, NY, USA : ACM.

LEWIS D. D., YANG Y., ROSE T. G. & LI F. (2004). Rcv1 : A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, **5**, 361–397.

MCCLOSKY D., CHARNIAK E. & JOHNSON M. (2006). Effective self-training for parsing. In *In Proc. N. American ACL (NAACL*, p. 152–159.

RAKOTOMAMONJY A. (2004). Optimizing auc with svms. In *Proceedings of the Workshop on ROC Curves and AI*.

ROBERTSON S. E. & SOBOROFF I. (2001). The TREC 2001 filtering track report. In *Text REtrieval Conference*.

TEO C. H., SMOLA A., VISHWANATHAN S. V. & LE Q. V. (2007). A scalable modular convex solver for regularized risk minimization. In *KDD '07 : Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*.

WANG B., SPENCER B., LING C. X. & ZHANG H. (2008). Semi-supervised self-training for sentence subjectivity classification. In *Canadian Conference on AI*, p. 344–355.

WELLINGTON C. & STENTZ A. T. (2004). Online adaptive rough-terrain navigation in vegetation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, p. 96 – 101.

YAROWSKY D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, p. 189–196.

ZHOU D., WESTON J., GRETTON A., BOUSQUET O. & SCHÃ¶LKOPF B. (2004). Ranking on data manifolds. In *Advances in Neural Information Processing System 16* : MIT Press.

ZHU X. (2005). *Semi-Supervised Learning Literature Survey*. Rapport interne 1530, Computer Sciences, University of Wisconsin-Madison.