TME 9 : Aprentissage d'ordonnancement Évaluation de l'apprentissage structuré

Travaux sur Machines Encadrés

Dans ce TME, on souhaite instancier un modèle d'apprentissage structuré afin de résoudre un problème d'ordonnancement (Ranking). Pour chaque exemple d'apprentissage, l'entrée $\mathcal{X}$ et la sortie $\mathcal{Y}$ seront définies de la manière suivante :

— $\mathcal{X} = \{x_0, ..., x_{N-1}\}$: l'entrée est une liste de N images, où chaque image x_i est représentée par un vecteur de description $\phi(x_i) \in \mathbb{R}^d$ (*e.g.* boW).

— $\mathcal{Y} = \{y_0, ..., y_{N-1}\}$: la sortie structurée est une liste contenant la position de chacune des images par rapport à la requête. Cette liste peut servir pour générer une matrice de ranking $\mathbf{Y}$ tq :

$$y_{ij} = \begin{cases} +1 & \text{si } x_i \prec_y x_j \ (x_i \text{ est classé avant } x_j \text{ dans la liste ordonnée}) \\ -1 & \text{si } x_i \succ_y x_j \ (x_i \text{ est classé après } x_j) \end{cases}$$

On considérera des données issues d'un étiquetage $\oplus/\ominus$ pour de la classification binaire. On disposera donc lors de l'apprentissage d'un seul exemple $(\mathbf{x}, \mathbf{y})$, $\mathbf{x}$ étant une liste des représentations image, et $\mathbf{y}$ un ordonancement pour lequel l'ensemble des images $\oplus$ est placé avant l'ensemble des images $\ominus$.

On rappelle la définition des fonctions $\Psi(x, y)$ et $\Delta(y_i, y)$ dans ce cas de ranking :

$$\Psi(x, y) = \sum_{i \in \oplus} \sum_{j \in \ominus} y_{ij} \left(\phi(x_i) - \phi(x_j)\right) \tag{1}$$

$$\Delta(y_i, y) = 1 - AP(y) \tag{2}$$

$AP(y)$ est la precision moyenne (Average Precision), calculée comme l'aire sous la courbe rappel/précision. Cette métrique est intéressante pour des problèmes de recherche d'information, car elle pénalise plus fortement les erreurs d'étiquetage commises en somment de liste par rapport à celles commises en queue de liste.

On utilisera la classe `RankingOutput` fournie pour représenter la sortie structurée. Cette classe contient en particulier :

— Une variable d'instance `ranking` pour représenter un ordonancement $\mathbf{y}$ particulier, contenant une liste d'entiers indiquant les indices des éléments triés par ordre décroissant. Par exemple, si on considère une liste de 4 éléments $\{x_0, x_1, x_2, , x_3\}$, `ranking` $= [2; 0; 1; 3]$ signifie que l'ordre des éléments est $[x_2; x_0; x_1; x_3]$.

  — Il sera parfois pratique d'utiliser une représentation "duale" du `ranking`, qu'on note ici `positioning`, et qui contient une liste d'entiers indiquant la position de chaque exemple x_i dans la liste. Dans l'exemple précédent, le `positioning` de la liste est `positioning`$= [1; 2; 0; 3]$, ce qui signifie que x_0 est positionné au rang 1, x_1 au rang

2, x_2 au rang 0, et x_3 au rang 3. On pourra convertir `positioning` $\leftrightarrow$ `ranking` en utilisant la méthode `getPositionningFromRanking()` de la classe `RankingOutput` [1].

— Une variable d'instance `labelsGT` qui contient les labels ($\oplus/\ominus$) pour chaque exemple de la liste **x**. Dans l'exemple précédent, si on suppose que `labelsGT`=$[-1;-1;+1;-1]$, ceci signifie que seul l'exemple x_2 est $\oplus$, les autres sont $\ominus$. Dans ce cas, le `ranking` $= [2;0;1;3]$ est optimal, puisqu'il classe tous les éléments $\oplus$ avant les éléments $\ominus$ (et l'AP de ce `ranking` sera donc de 1.0). `labelsGT` sera utilisée :
  — Pour le calcul de la feature map de ranking $\Psi(x,y)$, voir Eq (1).
  — Pour le calcul de la fonction de coût $\Delta(y_i,y)$, voir Eq (2).

Exercice 1 Ranking Strcuturé

Pour pouvoir apprendre un modèle de prédiction en ranking, on demande de mettre en place :

1. Une classe `RankingInstantiation` implémentant l'interface `IStructInstantiation<List<double[]>, RankingOutput>`. Donner en particulier le code des méthodes $\Psi(x,y)$ et $\Delta(y_i,y)$:
   — Pour le calcul de $\Delta(y_i,y)$, utiliser la méthode static `averagePrecision`, fournie dans la classe `RankingFunctions` du package `upmc.ri.struct.ranking`.
   — Pour $\Psi(x,y)$, on pourra chercher à accélérer le calcul de l'Eq (1) en comptant le nombre de fois que chaque éléments est compté (positivement ou négativement)

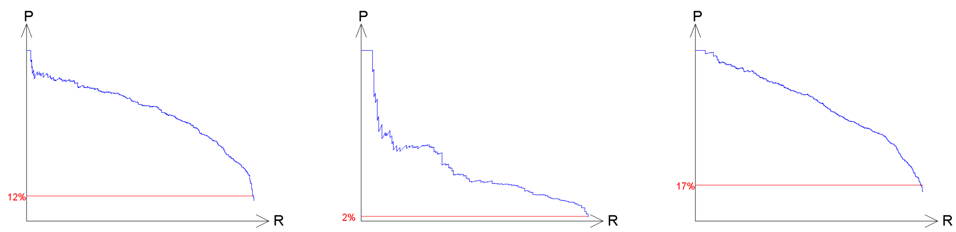
   <u>N.B.</u> : La méthode `enumerateY()` ne sera pas utilisée ici (renvoyer `null`), car on ne pourra pas explorer exhaustivement $\mathcal{Y}$ (voir ci-après).
2. Une classe `RankingStructModel` dérivant de `LinearStructModel<List<double[]>,RankingOutput>`. Il faudra pour cela redéfinir les méthodes pour la résolution des des problèmes d'inférence et de "loss-augmented", qui ne pourront être résolu de manière directe dans le cas du ranking, car l'exploration exhaustive de l'espace $\mathcal{Y}$ n'est pas praticable vu sa taille.. On demande donc de :
   — Surcharger la méthode `predict`. On rappelle que l'inférence, *i.e.* la prédiction $\hat{y}(x,w) = \arg\max_{y \in \mathcal{Y}} \langle w, \Psi(x,y) \rangle$, revient dans le cas du ranking à trier la liste par ordre décroissant de $\langle w; \phi(x_i) \rangle$ - Eq (1). Une fois ce tri effectué, on créera un objet de la classe `RankingOutput`, en lui passant les paramètres `nbPlus` et `labelsGT` du `STrainingSample` passé à la méthode `predict`.
   — Surcharger la méthode `loss_augmented_inference`. On utilisera pour cela l'algorithme greedy optimal introduit par Yue *et.al.* [YFRJ07], dont le code est fourni dans la méthode `loss_augmented_inference` de la classe `RankingFunctions`.
   — <u>Bonus :</u> pour accélérer l'algorithme d'apprentissage, vous pouvez modifier le code de la la méthode `train` de la classe `SGDTrainer`, afin de pré-calculer la valeur de $\Psi(x,y_i)$, fixe au cours de l'apprentissage.

1. `positioning` et `ranking` contiennent la même quantité d'information, mais l'un ou l'autre sera plus facile à manipuler au cours de lapprentissage.

Exercice 2 Évaluation de l'ordonnancement

On souhaite appliquer le modèle d'ordonancement structuré mis en place pour apprendre à trier des images. On utilisera les données des TME 7-8, où un label de classe (parmi 9) est associé à chaque image. Chacune des 9 classes sera considérée comme une requête pour apprendre un modèle structuré sur des données d'apprentissage (chaque exemple sera étiquetée $\oplus$ s'il a le label de la requête, $\ominus$ sinon), puis de tester les performances sur un ensemble de test.

1. Dans le package upmc.ri.bin, mettre en place une classe Ranking avec une méthode principale, pour apprendre un modèle d'apprentissage structuré instancié en ranking. Les opérations suivantes seront effectuées :
   - Charger l'ensemble des données générées au TME 7 (au format DataSet<double[], String>). Utiliser la méthode convertClassif2Ranking founire dans la classe RankingFunctions du package upmc.ri.struct.ranking, afin de convertir, pour une requête donnée (*i.e.* une des 9 classes), l'ensemble des données au format DataSet<List<double[]>,RankingOutput>.
   - Créer un objet de la classe RankingInstantiation, un RankingStructModel dont on fixera le type à l'objet RankingInstantiation précédemment créé.
   - OPT : création d'un évaluateur Evaluator<List<double[]>,RankingOutput> auquel on passe les données de train, de test, et le modèle.
   - Instantiation d'un objet de type SGDTrainer<List<double[]>,RankingOutput> pour apprendre le modèle (TME 7), et appel de la fonction train. N.B. : avec les données du TME, l'ordre de grandeur des paramètres est par exemple $\lambda = 10^-6$, $\gamma = 10$ et ~ 50 époques.
2. Évaluation de résultats : en plus cu calcul de la récision moyenne en apprentissage et en test à l'issue de l'apprentissage, utiliser la fonction traceRecallPrecisionCurve fournie dans la classe Drawing du package upmc.ri.utils pour visualiser la courbe Rappel-Precision. La figure 1 présente des résultats en test sur les classes ambulance, wood-frog et harp. Visualiser l'ensemble des résultats en apprentissage et en test sur les 9 classes, et calculer la précision moyenne, Mean Average Precision (MAP).
3. Comparer les performances (AP) issues du classifier appris en ranking à un classifieur appris pour résoudre un problème de classification bi-classes (TME 7). Conclure.



Ambulance, AP = 65.3% Wood-frog, AP = 31.8% Harp, AP = 69.5%

FIGURE 1 – Évaluation des peformances : Average Precicion (AP) en test. La droite rouge illustre la performance d'un classifieur aléatoire, qui renvoie un exemple au hasard : sa performance dépend donc du ratio entre le nombre d'images positives et le nombre total d'images.

Références

[YFRJ07] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims, *A support vector method for optimizing average precision*, Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (New York, NY, USA), SIGIR '07, ACM, 2007, pp. 271–278.