Rapport Recherche d'Informations Visuelles par Apprentissage Structuré

Loïc HERBELOT & Sébastien PEREIRA

January 1, 2018

Abstract

Dans ces TME nous allons nous intéresser au paradigme d'apprentissage structuré appliqué à différentes problématiques de recherche d'informations visuelles. On instanciera dans un premier temps ce problème pour résoudre des tâches de classification d'images puis dans un second temps pour résoudre un problème d'ordonnancement renvoyant une liste d'images qui correspondent à la recherche d'un utilisateur.

1 Introduction

Contrairement à la régression où notre espace de sortie est $Y = \Re$ et à la classification où $Y = 1, ..., C$ (où C est le nombre de classes), l'apprentissage structuré définit un espace de sortie Y où les $y \in Y$ sont des structures comme des arbres, des graphes, des ordonnancements etc.

1.1 Formalisme

Soit X l'espace d'entrée et Y l'espace de sortie structuré. On définit une *joint feature map* $\Psi(x, y)$ qui décrit le lien entre l'entrée $x \in X$ et la sortie $y \in Y$ et une fonction de coût $\Delta(y_i, \hat{y})$ qui mesure la dissimilarité entre notre prédiction $\hat{y}$ et la vérité terrain y_i. Le problème d'apprentissage revient à trouver un modèle linéaire qui donne un score à chaque paire (x, y), le score est défini par $< w, \Psi(x, y) >$, notre paramètre est $w \in R^d$ La prédiction du modèle est de la forme :

$$\hat{y}(x, w) = \arg\max_{y \in Y} < w, \Psi(x, y) > \tag{1}$$

De plus on souhaite avoir une régularisation sur le paramètre w, le problème d'apprentissage s'écrit :

$$\min_w \frac{1}{2}||w||^2 + \frac{C}{n}\sum_{i=1}^{n} \Delta(y_i, \hat{y}(x, w)) \tag{2}$$

Ce problème d'optimisation étant NP difficile, on optimise une borne supérieure convexe et sous différentiable de $\Delta(y_i, \hat{y})$ définie par :

$$\max_{y \in Y}[\Delta(y_i, \hat{y}) + < \Psi(x_i, y), w >] - < \Psi(x_i, y_i), w > \tag{3}$$

Le problème d'optimisation devient donc :

$$\min_w \frac{1}{2}||w||^2 + \frac{C}{n}\sum_{i=1}^{n}[\max_{y \in Y}[\Delta(y_i, \hat{y}(x, w)) + < \Psi(x_i, y), w >] - < \Psi(x_i, y_i), w >] \tag{4}$$

1.2 Algorithme d'apprentissage

L'objectif des ces TME est de développer un *framework* d'apprentissage structuré générique capable de résoudre divers problèmes. Aussi pour instancier des modèles capables de faire cela, notre code s'articulera toujours de la même manière. Nous allons dans le premier TME travailler sur les aspects génériques et réutilisables pour n'importe quel modèle.
Dans la première partie nous allons développer une classe *VisualIndexes* qui générera nos ensembles de données à partir desquels on fera nos apprentissages. La génération de ces ensembles de données nécessite tout d'abord le calcul des *Bag of Words* à partir des *Bag of Features* pour chaque image. Ensuite nous effectuerons une réduction de dimension de type linéaire (analyse en composantes principales) afin de débruiter nos données et d'accélérer l'apprentissage.
Nous allons ensuite développer une classe *SGDTrainer* qui implémente une méthode *train* qui réalise une descente de gradient stochastique pour trouver le vecteur w, solution de notre problème de minimisation. Cette méthode *train* sera la même utilisée quel que soit le problème, il suffira pour l'utiliser de spécifier la fonction de coût ainsi que le type de modèle.

2 Apprentissage structuré multi-classes et hiérarchique

Dans la première partie on s'intéresse à la classification d'images à partir d'un apprentissage structuré, on observera l'influence du choix de la fonction de coût sur les résultats obtenus.

2.1 Classification multi-classes

Dans le cas d'un problème d'apprentissage structuré instancié pour les problèmes de classification multi-classes on définit la *joint feature map* et la fonction de coût de la façon suivante :

- $\Psi(x, y) = \begin{bmatrix} 0^d \\ 0^d \\ ... \\ \phi(x) \\ ... \\ 0^d \\ 0^d \end{bmatrix}$
- $\Delta(y, y') = \begin{cases} 1 & \text{si } y \neq y' \\ 0 & \text{sinon.} \end{cases}$

Où $\phi(x)$ est la représentation des images sous forme de *Bag of Words*
Dans cette partie nous allons écrire plusieurs classes génériques qui permettent d'instancier les problèmes de classification multi-classes. On commencera par écrire une classe *LinearStructModel_Ex* qui hérite de la classe *LinearStructModel* utilisée pour les modèles linéaires comme l'extension des SVM aux *Structured Support Vectors Machines*. Cette classe *LinearStructModel_Ex* permet de définir la fonction de prédiction du modèle ainsi que la *Loss Augmented inference* nécessaire au calcul du gradient pour l'apprentissage. Ensuite nous définirons une classe *MultiClass* implémentant l'interface *IStructInstantiation* et permettant d'instancier notre modèle pour un problème classification multi-classes. Cette classe permettra de calculer la *joint feature map* $\Psi(x, y)$ et la fonction de coût $\Delta(y, y')$ définies précedemment. De plus elle implémente une méthode qui permet d'évaluer notre modèle grâce au calcul d'une matrice de confusion.

2.1.1 Résultats

Courbes d'erreur en apprentissage et en évaluation

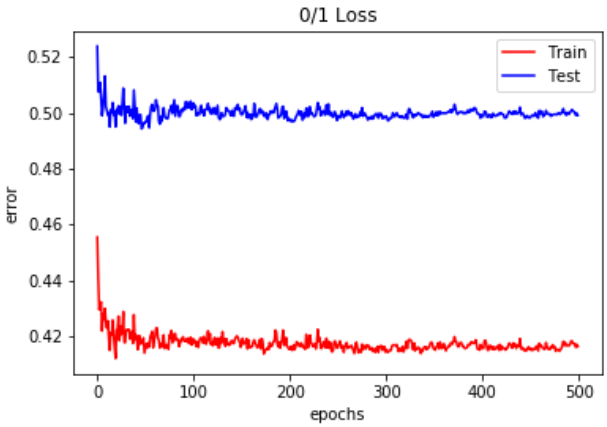


Figure 1: Évaluation de la 0/1 loss en train et en test sur 100 epochs.

Matrice de confusion normalisée en évaluation

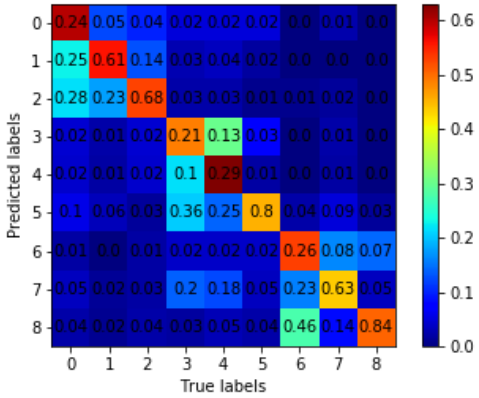


Figure 2: Matrice de confusion normalisée pour un modèle entrainé avec une 0/1 loss sur 100 epochs. 0 = taxi, 1 = ambulance, 2 = minivan, 3 = acoustic guitar, 4 = electric guitar, 5 = harp, 6 = wood frog, 7 = tree frog, 8 = european fire salamander

2.1.2 Interprétations

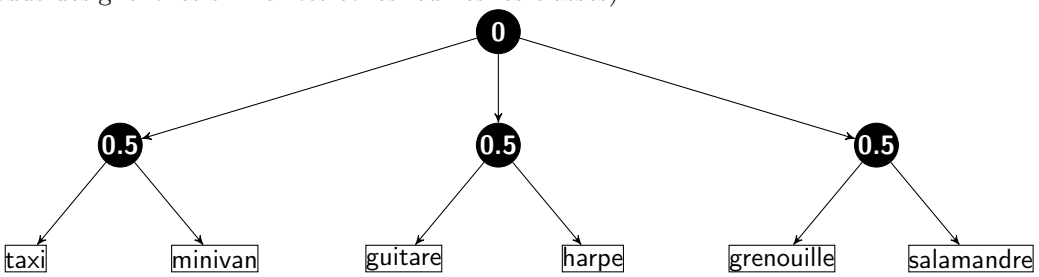
- Première catégorie : véhicules
  - la classe taxi (0) est bien classée dans **24%** des cas. Dans 28% des cas on classe ces taxis comme des minivan (1) et dans 25% des cas comme des ambulances (2).
  - la classe ambulance (1) est bien classée dans **61%** des cas. Dans 23% des cas on les classe comme des minivan et dans seulement 5% des cas comme des taxis.
  - la classe minivan (2) est bien classée dans **68%** des cas. Dans seulement 14% des cas on l'inteprète comme la classe ambulance.
- Deuxième catégorie : instruments de musique
  - la classe acoustic guitar (3) est bien classée dans **21%** des cas. Dans 36% des cas on la confond avec la classe harp et dans seulement 10% des cas on la confond avec la classe electric guitar.
  - la classe electric guitar (4) est bien classée dans **29%** des cas. Dans 25% des cas on la confond avec la classe harp et dans seulement 13% des cas on la confond avec la classe acoustic guitar.
  - la classe harp (5) est bien classée dans **80%** des cas.
- Troisième catégorie : amphibiens
  - la classe wood frog (6) est bien classée dans **26%** des cas. Dans 46% des cas on la confond avec la classe european fire salamander et dans 23% des cas on la confond avec la classe tree frog.
  - la classe tree frog (7) est bien classée dans **63%** des cas. Dans 14% des cas on la confond avec la classe european fire salamander et dans seulement 8% des cas on la confond avec la classe wood frog.
  - la classe european fire salamander (8) est bien classée dans **84%** des cas.

2.2 Classification hiérarchique

Dans le cas d'un problème d'apprentissage structuré instancié pour les problèmes de classification hiérarchique on garde la *joint feature map* précédente et on définit la fonction de coût de la façon suivante :

- $\Delta(y, y') = \begin{cases} 1 - \textit{Similarité}(y, y') & \text{si } y \neq y' \\ 0 & \text{sinon.} \end{cases}$

Où la similarité est définie par une distance dans un arbre par exemple de la façon suivante (les noeuds désignent les similarités et les feuilles les classes) :



Dans cette partie nous utiliserons de nouveau la classe générique *LinearStructModel* utilisée pour les modèles linéaires. Ensuite nous définirons une classe *MultiClassHier* héritant de la classe *MultiClass* et permettant d'instancier notre modèle pour un problème classification hiérarchique. Cette classe utilise la même *joint feature map* $\Psi(x, y)$ que précédemment et calcule la nouvelle fonction de coût $\Delta(y, y')$ (Cf arbre de similarité sémantique).

2.2.1 Résultats

Courbes d'erreur en apprentissage et en évaluation

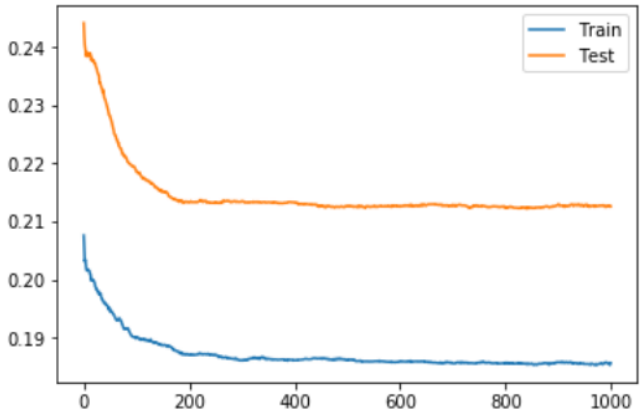


Figure 3: Évaluation de la Hierarchic loss en train et en test sur 1000 epochs.

2.3 Conclusion

On remarque qu'il existe 3 catégories sémantiques dans nos 9 classes, la première catégorie sémantique rassemble des véhicules (taxi, ambulance et minivan) la deuxième catégorie sémantique rassemble des instruments à cordes (guitare acoustique, guitare electrique et harpe) enfin la troisième catégorie sémantique rassemble des amphibiens (grenouille des bois, grenouille arboricole et salamandre).
Les classes **taxi et wood frog semblent être particulièrement difficiles à classer correctement** (Cf colonnes 0 et 6). Les classes **harp et european fire salamander sont quant à elles les classes présentant les meilleurs résultats** (Cf colonnes 5 et 8).
Par ailleurs on constate que **la grande majorité des erreurs de notre modèle provient de confusions au sein d'une même catégorie sémantique**, on peut donc en conclure que notre modèle n'arrive pas à capter les différences entre les données provenant d'une même catégorie sémantique.
Ainsi si on utilise un évaluateur qui utilise une fonction de coût de type hiérarchique, le taux d'erreur diminue car la plupart des erreurs restent dans la même catégorie sémantique.
Néanmoins on peut également considérer que les fautes commises au sein d'une même catégorie sémantique peuvent être moins "graves" pour nombre d'applications. On remarque également qu'en dehors de ces remarques **les performances du modèle sont satisfaisantes avec une *accuracy* multi-classe de 50,6% (en test) contre 11,1% pour un estimateur aléatoire**.

3 Application aux problèmes d'ordonnancement

Dans cette seconde partie on s'intéresse à la résolution d'un problème d'ordonnancement grâce à l'apprentissage structuré.

3.1 Problème d'ordonnancement

On définit cette fois-ci encore notre espace d'entrée X comme étant l'ensemble des représentations des images de notre ensemble de données sous forme de vecteurs de descriptions (*Bag of Words*). On définit l'espace de sortie structuré Y comme étant une liste d'ordonnancement des images par rapport à une requête.
Le but de ce problème est de pouvoir renvoyer à un utilisateur des *items* correspondant à sa requête. Ici les requêtes utilisateurs sont une classe d'image par exemple 'taxi', la recherche d'information se fait sur les contenus visuels des *items* (*Bag of Words*) et l'objectif est de renvoyer en tête de liste les *items* correspondant à la classe de la requête. Pour ce problème de *ranking*, on considère donc un étiquetage binaire des données. Dans ce cas on définit la *joint feature map* et la fonction de coût de la façon suivante :

$$\Psi(x, y) = \sum_{i \in \oplus} \sum_{j \in \ominus} y_{ij}(\phi(x_i) - \phi(x_j)) \tag{5}$$

$$\Delta(y_i, y) = 1 - AP(y) \tag{6}$$

Où

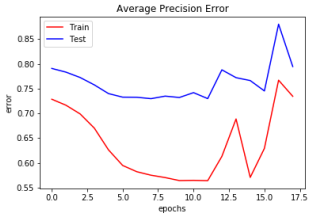
- $\oplus$ représente l'ensemble des images placées avant l'ensemble $\ominus$ dans l'ordonnancement y
- $y_{ij} = \begin{cases} +1 & \text{si } x_i \prec x_j \\ -1 & \text{si } x_i \succ x_j \end{cases}$
- *Average Precision AP* (Précision Moyenne) représente l'aire sous la courbe de précision/rappel

Dans cette partie nous allons écrire plusieurs classes génériques qui permettent d'instancier les problèmes d'ordonnancement. On commencera par écrire une classe *RankingStructModel* qui hérite de la classe *LinearStructModel*. Cette classe *RankingStructModel* permet de définir la fonction de prédiction du modèle. On rappelle qu'avec notre *framework* l'inférence $\hat{y}(x, w) = \arg\max_{y \in Y} < w, \Psi(x, y) >$ revient dans le cas du problème d'ordonnancement à renvoyer la liste triée par ordre croissant des $< w, \phi(x_i) >$.
Ensuite nous définirons une classe *RankingInstanciation* implémentant l'interface *IStructInstantiation* et permettant d'instancier notre modèle pour un problème d'ordonnancement. Cette classe permettra de calculer la *joint feature map* $\Psi(x, y)$ et la fonction de coût $\Delta(y, y')$ redéfinies précédemment.

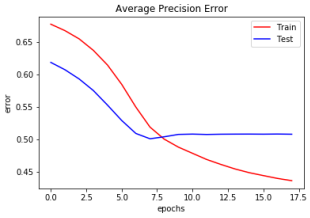
3.2 Résultats

3.2.1 Première catégorie sémantique

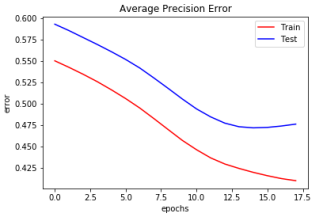
Erreur de précision moyenne en apprentissage et en test



(a) taxi

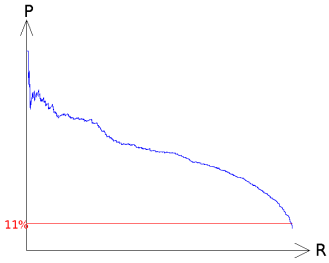


(b) ambulance

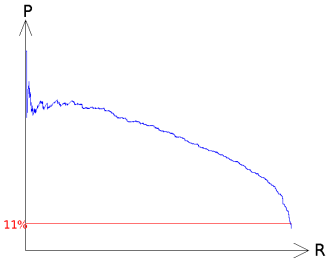


(c) minivan

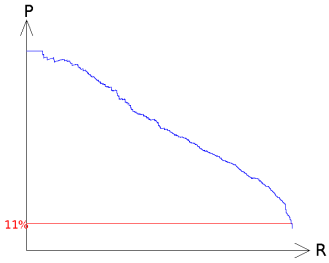
Courbes de précision moyenne en apprentissage



(a) taxi AP = 0.50

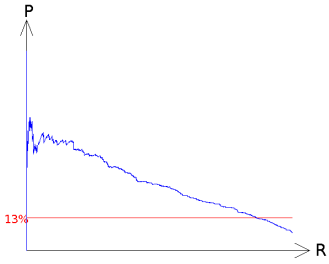


(b) ambulance AP = 0.58

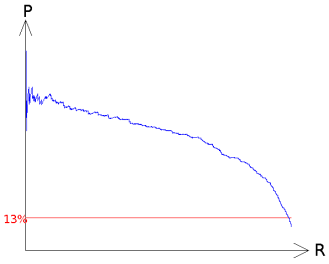


(c) minivan AP = 0.64

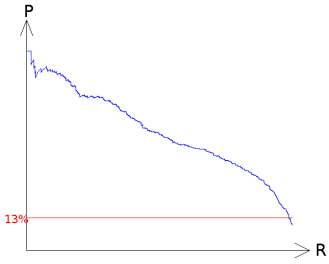
Courbes de précision moyenne en évaluation



(a) taxi AP = 0.34



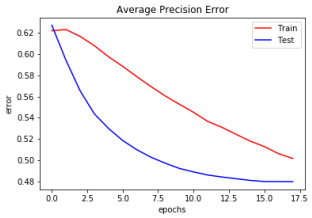
(b) ambulance AP = 0.58



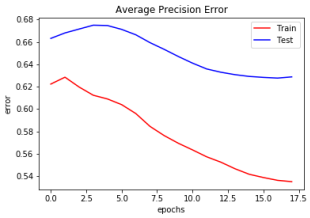
(c) minivan AP = 0.60

3.2.2 Deuxième catégorie sémantique

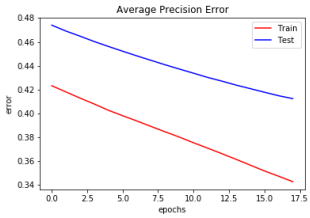
Erreur de précision moyenne en apprentissage et en test



(a) acoustic_guitar

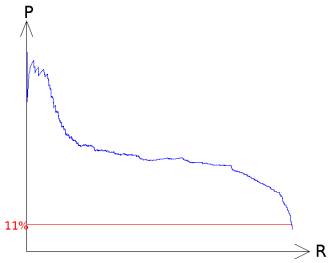


(b) electric_guitar

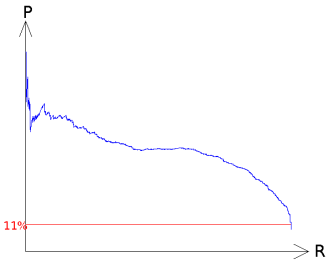


(c) harp

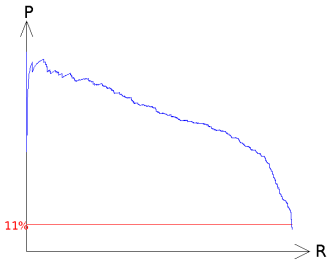
Courbes de précision moyenne en apprentissage



(a) acoustic_guitar AP = 0.49

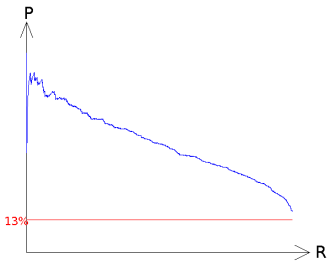


(b) electric_guitar AP = 0.51

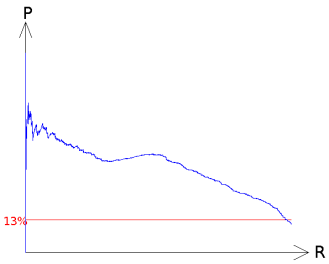


(c) harp AP = 0.70

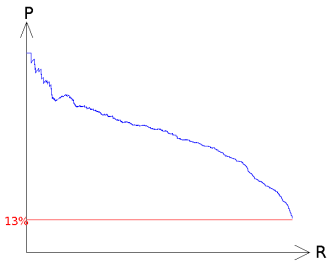
Courbes de précision moyenne en évaluation



(a) acoustic_guitar AP = 0.55



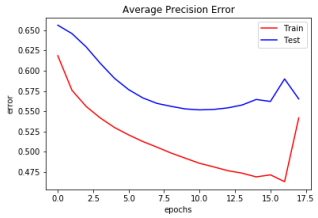
(b) electric_guitar AP = 0.43



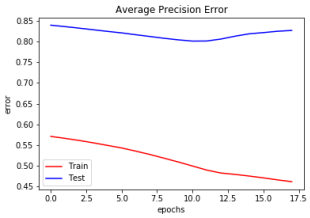
(c) harp AP = 0.60

3.2.3 Troisième catégorie sémantique

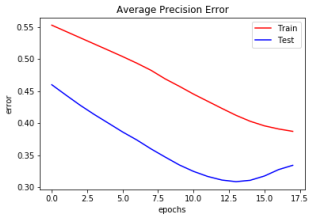
Erreur de précision moyenne en apprentissage et en test



(a) tree-frog

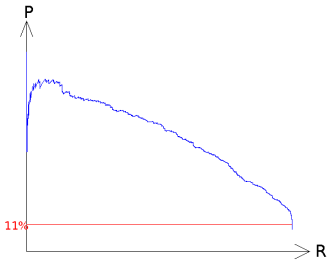


(b) wood-frog

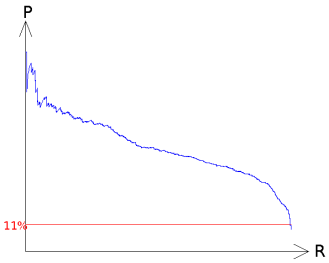


(c) salamander

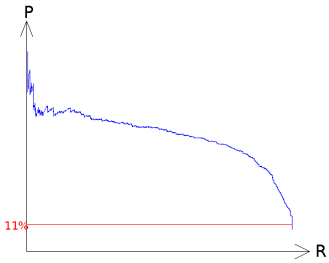
Courbes de précision moyenne en apprentissage



(a) tree-frog AP = 0.59

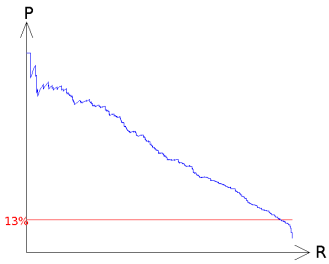


(b) wood-frog AP = 0.53

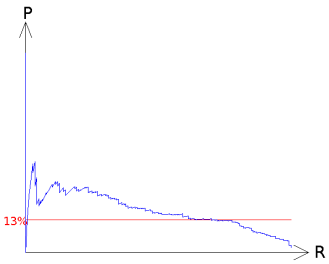


(c) salamander AP = 0.59

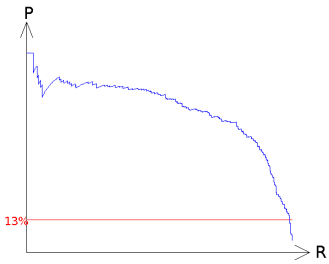
Courbes de précision moyenne en évaluation



(a) tree-frog AP = 0.52



(b) wood-frog AP = 0.21



(c) salamander AP = 0.72

3.3 Inteprétations

On voit sur les courbes d'erreurs en précision moyenne que les apprentissages fonctionnent correctement car on a une diminution à la fois en apprentissage mais également en évaluation pour éviter le sur-apprentissage.
Il est également intéressant de constater que les résultats sont analogues à la classification multi-classes. En effet on constate en comparant les valeurs de précision moyenne que les classes qui posent problème sont les mêmes qu'en classification. Nous avons vu en classification multi-classes que les classes taxi et wood frog étaient paticulièrement difficiles à classer. Ces deux classes sont également celles qui ont les moins bonnes performances dans le cas du *ranking* avec les **précisions moyennes de 0.34 et 0.21 en évaluation**.
À l'inverse nous avons constaté que les classes harp et european fire salamander étaient celles qui présentent les meilleures performances en classification De plus ce sont également les classes qui ont les meilleures précisions moyennes dans le problème d'ordonnancement (**harp 0.60 et salamander 0.72 en évaluation**).

Finalement on obtient les résultats suivants en moyenne sur les 9 classes :

- La précision moyenne en apprentissage est de **0.57**
- La précision moyenne en évaluation est de **0.51**

Malgré les difficultés à ordonner correctement les classes taxi et wood frog, le modèle de *ranking* présente néanmoins des **performances satisfaisantes avec une précision moyenne de 0.51 (en test) contre 0.13 pour un modèle aléatoire**.

3.4 Conclusion

Dans cette dernière partie de TME nous nous sommes intéressés à un problème d'ordonnancement de documents par leurs contenus visuels. Notre *framework* d'apprentissage structuré nous à permis d'instancier des *joint feature map* et fonction de coût adaptées afin de résoudre ce problème.
Les résultats en précision moyenne sur l'ensemble des 9 classes sont de **0.57** en apprentissage et **0.51** en évaluation (*learning rate* = 10, *penalty coefficient* = 10^-6).