TME 7 - Bagging, Boosting

Forêts aléatoires

En utilisant les fonctions du module `sklearn.ensemble`, expérimentez les forêts aléatoires sur les données artificielles 2D habituelles. Vous étudierez comment évoluent les frontières de décision, l'erreur en apprentissage et en test en fonction de la profondeur des arbres, du nombre d'arbres et des autres paramètres disponibles. Expérimentez également sur les chiffres manuscrits.

Boosting : AdaBoost

Rappel de l'algorithme pour un ensemble d'apprentissage $E = \{\mathbf{x}^i, y^i\}_{i=0}^N$

- Initialiser la distribution D_0 sur les exemples en considérant des poids uniformes : $w_0^i = \frac{1}{N}$
- Répéter :
  1. Apprendre h_t sur la distribution D_t
  2. Calculer l'erreur $\epsilon_t = \sum_i w_t^i \mathbf{1}_{h_t(\mathbf{x}^i) \neq y^i}$
  3. Fixer $\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  4. Fixer $w_{t+1}^i = \frac{1}{Z_t} w_t^i e^{-\alpha_t y_i h_t(x^i)}$, avec Z_t normalisation de la distribution.
- Le classifieur est $F(x) = \sum_t \alpha_t h_t(x)$

Programmez cet algorithme de boosting. Utilisez comme classifieurs faibles les arbres de décision de `sklearn` (en faisant varier les hyper-paramètres). Expérimentez sur les données artificielles et réelles. Regardez en particulier l'évolution des poids (en utilisant `plt.imshow` par exemple) et l'évolution de $Z = \prod Z_t = \frac{1}{N} \sum_{i=1}^N e^{-y^i F(x^i)}$ en fonction du nombre d'itérations.

Détection de visage dans une photo - Bonus pour ceux qui ont envie

Le boosting a en particulier été popularisé par l'article de Viola et Jones [1] qui l'applique à un problème de détection de visage dans une image.
Le principe de la méthode est de construire un ensemble très grand de features très simple sur une petite sous-partie de l'image. Le boosting est ensuite utilisé pour sélectionner les features les plus intéressants pour la classification et ne considérer que ceux-la lors de l'inférence. Cela permet de rendre l'inférence très rapide car uniquement les features considérés seront à calculer lors de cette phase et non pas tous les features qui ont servi de base à l'apprentissage.
Dans leur version la plus simple, ces features correspondent à la différence entre la somme des valeurs des pixels dans différentes régions rectangulaires de l'image. Deux types de features seront considérés dans la suite, ceux indiqués dans la figure suivante (les autres types de features consistant principalement à considérer 3 et 4 zones rectangulaires adjacentes) : Les deux régions rectangulaires considérées sont

1. Paul Viola et Michael Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, IEEE CVPR, 2001

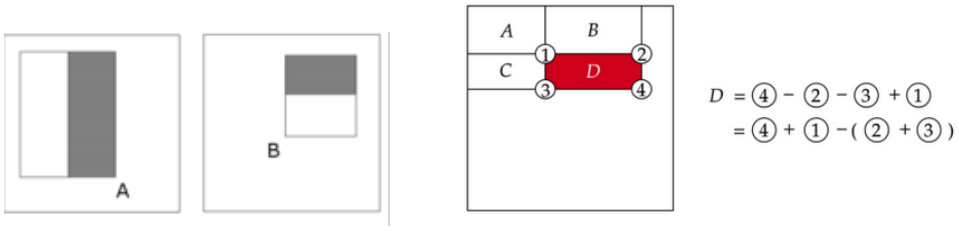

FIGURE 1 – Fig. gauche : deux types de features, Fig. droite : calcul efficient des features

adjacentes et de même taille. La valeur du feature résultant correspond à la différence entre la somme des pixels dans la région grisée et la somme des pixels dans la région blanche. Chaque feature indique ainsi une sorte de contraste entre deux régions adjacentes. Pour une taille de sous-image donnée (24 par 24 dans leur article), tous les features possibles correspondant à ce schéma sont calculés : on fait varier la taille de la région rectangulaire en largeur et en hauteur, ainsi que sa position. Cela donne plus de 72000 features pour cette taille de fenêtre.

Afin de rendre plus rapide les calculs, une représentation intermédiaire de l'image est calculée : l'image *intégrale*. Il s'agit de remplacer la valeur $p(x, y)$ de chaque pixel en (x, y) par $ii(x, y) = \sum_{i \leq x, j \leq y} p(i, j)$ la somme cumulée des pixels jusqu'en (x, y). De cette manière, le calcul d'un feature peut être fait en temps constant à partir de l'image intégrale. Sur la figure de droite, la somme des valeurs de la région rouge D est obtenue en calculant $ii(4) - ii(2) - ii(3) + ii(1)$.

Une fois l'ensemble des features calculé pour chaque image, un algorithme de boosting utilisant un arbre de profondeur 1 est utilisé afin d'obtenir un classifieur. L'apprentissage s'effectue uniquement sur des images de 24 par 24 représentant des visages (exemples positifs) et des images 24 par 24 quelconques (exemples négatifs). Les exemples de visages sont mis à l'échelle pour rentrer dans la taille impartie. Des astuces d'implémentations permettent de rendre le calcul très efficace (trier une fois pour toutes par valeur de features pour chaque feature les exemples ce qui permet de rendre quadratique en nombre de feature et en nombre d'exemples la recherche du meilleur arbre de profondeur 1 à chaque itération). Pour l'inférence, il suffit d'extraire de l'image à classer que les quelques features (de l'ordre d'une centainte) qui ont servis à la construction du classifieur et non pas l'ensemble de features. Cependant, la recherche de visage dans une image reste coûteuse : il faut scanner toute l'image par petite fenêtre de 24 par 24 et ceci pour plusieurs mises à l'échelle de l'image pour repérer les réponses positives du classifieur. Pour optimiser ce temps de calcul, une dernière technique d'optimisation est utilisée appelée *cascade* de classifieurs (cf l'article ou la page wikipedia).

Un jeu de données de photos avec visages vous est fourni où les positions des visages dans les photos sont annotées. Le code vous permet de charger le jeu de données. Codez une fonction qui va permettre de lister tous les features considérés dans une sous image de taille fixe (par exemple 24 comme dans l'article). Vous pouvez pour cela rendre une liste de 6-uplet correspondant aux 6 points à connaître pour effectuer les opérations de calcul du feature. Codez ensuite une fonction qui à partir de cette liste vous renvoie la valeur associée à chaque features (la fonction `integral_image` permet de calculer l'image dite intégrale).

Il faut ensuite préparer votre jeu de données : pour chaque photo, il faut extraire les zones qui correspondent à des visages, les mettre à l'échelle de 24 par 24 puis de calculer les features à l'aide des fonctions codées précedement. Comme exemples négatifs, vous pouvez soit prendre d'autres photos sans visage, soit considéré des sous-images de ces photos qui ne correspondent pas à des visages en tirant aléatoirement des coordonnées dans l'image et en extrayant la sous-image associée (en vérifiant qu'elle n'est pas incluse en partie dans un visage).

Entraînez ensuite un algorithme de boosting (préferez GradientBoostingClassifier dans sklearn, plus rapide qu'Adaboost) ou mieux codez le à la main afin d'optimiser la phase d'apprentissage.

Vous pouvez évaluer votre classifieur avec un jeu de test habituel. Vous pouvez également l'utiliser pour détecter les régions dans une image correspondant à un visage.