Classification de photographies prises par un drone



Un drone est un aéronef sans personne à bord, télécommandé ou autonome, qui peut éventuellement emporter une charge utile, destinée à des missions (ex. : de surveillance, de renseignement, d'exploration, de combat, de transport, etc.). Les drones ont d'abord été utilisés au profit des forces armées ou de sécurité (police, douane, etc.) d'un État, mais ont aussi des applications civiles (Cinéma, télévision, agriculture, environnement) ou cinématographiques [1].
Sa taille et masse (de quelques grammes à plusieurs tonnes) dépendent des capacités recherchées. Le pilotage automatique ou à partir du sol permet des vols longs de plusieurs dizaines d'heures (à comparer aux deux heures typiques d'autonomie d'un chasseur).
Un drone peut être en pilotage automatique : il se dirige seul, en analysant en temps réel les données observées de son environnement. Dans notre cas, la mission du drone est de se promener, et de signaler régulièrement s'il se trouve à la mer ou non, sur la base des photographies qu'il prend avec ses caméras (une caméra avant, et une caméra "ventrale"). Pour cela, il embarque un classifieur qui prend en entrée une photographie, et qui renvoit en sortie si cette photographie correspond à un lieu de mer.
Votre mission est de construire ce classifieur, à l'aide d'un perceptron à noyau. Après avoir programmé ce perceptron et l'avoir testé sur des données jouets, il s'agit de l'entraîner sur des images photographiques transformées de ce drone. Cela permettra alors de créer un modèle de classification, donc un classifieur, qui sera finalement testé sur des photographies 'test'. Si le classifieur a de bonne performances, nous pourrons l'embarquer sur Qarmen, le drone de l'équipe Qarma du LIF !
Pour cette dernière séance de TP, un travail en deux temps est proposé :

1. Lors du TP, l'objectif est de coder le perceptron à noyau, et de le tester avec les données du TP3.
2. Tout ce qui est fait lors de ce TP sera utilisé pour le projet : le projet est l'application du perceptron à noyau à un problème de classification de photographies prises par un drone. Serez-vous capables d'apprendre un modèle qui sait reconnaître au mieux des photographies de lieu de mer ou d'autres lieux ?
3. A l'issue du TP, un rapport vous est demandé (cf. section 3.1.).

1 Programmation de fonctions noyaux

1. Programmer la fonction `noyauGaussien(v1,v2,sigma)` qui calcule et retourne, pour deux vecteurs de réels de même taille, et pour un réel positif, la valeur :

$$K_\sigma(v_1, v_2) = \exp\left(-\frac{\|v_1 - v_2\|^2}{\sigma^2}\right)$$

2. Programmer la fonction `noyauPolynomial(v1,v2,k)` qui calcule et retourne, pour deux vecteurs de réels de même taille, et pour un entier positif, la valeur :

$$K_k(v_1, v_2) = (1 + \langle v_1, v_2 \rangle)^k$$

1. Source : wikipedia, 12 décembre 2014

3. Pour chacune de ces fonctions noyaux, créer la fonction `computeGram(data)` qui calcule et retourne la matrice de Gram obtenue par la fonction noyau appliquée à chaque couple de vecteurs de l'échantillon `data`.

2 Programmation du perceptron à noyau

1. Programmer la fonction `learnKernelPerceptron(data, target, kernel, h)` qui retourne un perceptron à noyau appris sur les données `data` de cibles `target`, avec le noyau indiqué par `kernel` et `h` [2].
2. Programmer la fonction `predictKernelPerceptron(kp, x, kernel, h)` qui prédit la classe d'un nouvel exemple `x` avec le perceptron à noyau `kp` (lui-même obtenu avec la fonction noyau indiquée par `kernel` et `h`).
3. Tester rapidement ces fonctions sur les données du TP3.

3 Projet : application à la classification d'images

3.1 Objectifs

3.1.1 Des images à classer

Dans ce projet, nous cherchons à apprendre le meilleur modèle possible pour classer des images qui se répartissent en deux classes : images de lieux de mer, et images d'autres lieux (montagne, savane, ville, etc.).
De ce fait, une première partie est consacrée à la production des représentations vectorielles des images, et une seconde partie s'attache à apprendre des modèles de classification sur chacune des représentations. Au final, il faudra préciser :

1. quelle représentation des images a été utilisée pour produire le meilleur modèle,
2. et quel est l'algorithme qui a mené à ce modèle. Si cet algorithme se base sur une fonction noyau [3] : laquelle a été choisie, et avec quel(s) hyperparamètre(s).

3.1.2 Logistique du projet

— Projet individuel.
— Un rapport devra être remis au plus tard le lundi 5 janvier 2015 à 23h59. Ce rapport fera maximum 8 pages, et présentera une synthèse des algorithmes produits, un guide de lecture de votre code python (avec éventuellement une architecture fonctionnelle), et les résultats obtenus.
— Ce rapport devra être rendu au format pdf, et éventuellement annexé par le code python. Les correcteurs doivent être capables d'exécuter le code python. Rapport + annexes constitueront une archive à envoyer par email à votre chargé(e) de TP.

3.2 Choix des hyperparamètres

Programmer la fonction `bestHyperparameter(data, target, kernel)` qui retourne la valeur de l'hyperparamètre de la fonction noyau indiqué par `kernel`, qui minimise l'erreur estimée d'un perceptron à noyau appris sur les données `data` de cibles `target`. On réalisera cette fonction via une succession de validations croisées, chacune estimant l'erreur pour une valeur donnée de l'hyperparamètre [4].

2. si `kernel` vaut 1, on utilise le noyau gaussien avec $\sigma =$`h`, sinon on utilise le noyau polynomial avec $k =$`h`.
3. Nous conseillons un perceptron à noyau !
4. Pour chaque noyau, on testera ainsi une dizaine de valeurs de l'hyperparamètre.

3.3 Représentations des images

3.3.1 Les données

Le répertoire *Data* contient deux sous-répertoires : *Mer* et *Ailleurs*. Chaque sous-répertoire contient une série d'images représentant l'une ou l'autre des classes. Les images n'ont pas forcément la même taille, le même ratio, la même résolution, etc. **Les données et programmes utiles sont ici (fichier `projetM1-2014.zip`)** :

`http://pageperso.lif.univ-mrs.fr/~francois.denis/IAAM1/index.html`

3.3.2 Chargement et affichage d'une image avec Python

Pour ouvrir et afficher une image avec Python, le code suivant est suffisant :

```
$ from PIL import Image
$ im = Image.open('mon_image.jpg')
$ im.show()
```

Testez ce code avec une image de votre choix.

3.3.3 Numpy et les images

La fonction `open()` renvoie un objet de type `Image`. Pour transformer cet objet en un tableau comme nous avons pu en manipuler pendant les TP précédents, on utilisera la fonction `numpy.array()`.

1. Recopiez et testez le code suivant. Notez les liens et les différences entre un objet de type `Image` et un objet de type `ndarray`.

   ```
   $ im = Image.open('mon_image.jpg')
   $ print im.size, im
   $ mat = array(im)
   $ print mat.shape, mat
   ```

Comme vous pouvez le constater votre image n'est autre qu'une matrice de valeurs entières comprises entre 0 et 255, par triplet (R,G,B). Au sein d'un triplet – qui correspond à un pixel –, chaque valeur représentant le niveau de la couleur correspondante du pixel (rouge, vert et bleu). Nous allons à présent étudier deux représentations de nos images.

3.3.4 Une image comme un vecteur

Afin de pouvoir travailler avec des vecteurs de même dimension (comme évoqué précédemment, les images n'ont pas toutes la même taille) nous allons devoir opérer en deux étapes :

1. dans un premier temps, on utilisera la méthode `resize()` pour transformer chaque image en une matrice de 32×32 pixels. Cette étape permettra aussi de travailler avec des données de plus petite dimension (et accélèrera ainsi les calculs, au dépens parfois de la précision).
2. Ensuite il faudra vectoriser nos matrices de pixels vers des vecteurs de taille $3 \times 32 \times 32$, puisque chaque pixel est représenté par 3 couleurs. Cette étape est effectuée grâce à la fonction `reshape`.

Voici un code complet pour effectuer le traitement sur une image :

```
$ im = Image.open('mon_image.jpg').resize((32,32))
$ mat = array(im)
$ vect = reshape(mat, (1,32*32*3))
```

Programmez une fonction `vectMatrixFromDir(directory)` qui renvoie une matrice contenant les représentations vectorielles (une par ligne) de toutes les images présentes dans le répertoire `directory`. Utilisez la fonction `os.listdir(path)` de Python pour obtenir la liste des fichiers d'un répertoire.

3.3.5 Une image comme un histogramme

Un histogramme est un tableau contenant, pour chaque couleur R – G – B, et pour chaque valeur d'intensité de couleur (de 0 à 255), le nombre de fois où un pixel de l'image possède cette valeur. Quelle que soit la taille de l'image, un histogramme sera donc toujours un vecteur $\mathbf{v} \in \mathbb{N}^{3\times 256}$. Pour obtenir un histogramme à partir d'un objet de type `Image` représentant une image en couleur, utilisez la méthode `histogram()` de celui-ci. Le code suivant permet d'obtenir un histogramme normalisé :

```
$ im = Image.open('Data/logo.png')
$ width, height = im.size
$ normalized_histogram = array(im_gray.histogram()) / (1.0 * width * height)
```

Programmez une fonction `histMatrixFromDir(directory)` qui renvoie une matrice contenant les histogrammes normalisés (un par ligne) de toutes les images présentes dans le répertoire `directory`.

3.4 Apprentissages et évaluations de fonctions de classification

A partir des deux types de représentations d'images, deux échantillons peuvent être produits :

1. S_g qui comporte les images représentées par des vecteurs de pixels (pour chaque pixel et chaque couleur parmi R,G,B : la valeur est la valeur de l'intensité de ce pixel pour cette couleur : un entier entre 0 et 255) ;
2. S_H qui comporte les images représentées par des vecteurs de couleurs, d'une taille 3*256=768 : un vecteur de taille 256 par couleur, la valeur d'un élément représente le nombre de pixels de l'image qui possèdent cette intensité de couleur.

Pour chacun de ces échantillons, un perceptron à noyau doit être appris : ce perceptron dépend du noyau choisi (Gaussien ou polynomial), et de l'hyperparamètre de ce noyau. On cherchera à apprendre le meilleur modèle pour chaque échantillon et pour chaque noyau, et donc à déterminer la meilleure valeur des hyperparamètres. Au final, pour chaque échantillon, et pour chaque noyau, on apprendra le meilleur modèle.
On remplira alors le tableau suivant (en expliquant comment ces valeurs ont été obtenues) :

	meilleur hyperparamètre	erreur réelle estimée
Noyau gaussien		
Noyau poly.		

A votre avis, quel est le meilleur modèle obtenu ? Justifier.
En guise de *baseline*[5], une validation croisée 10 folds, testant un SVM à noyau linéaire sur les histogrammes des images, avec comme paramètre de régularisation C=15, mène à un taux de bonne classification de $0.817(+/-0.187)$. Faites vous mieux ?

3.5 Prédictions sur de nouvelles images

On veut prédire ici la classe de nouvelles images qui ne font pas partie des données d'apprentissage. Pour ces images, vous n'aurez accès qu'à leur représentation vectorielle (de taille 3*1024), et à leur histogramme normalisé (de taille 768) ; vous ne verrez ni les images originales ni leurs classes réelles). Pour charger les données, qui sont sous format binaire dans les fichiers `histograms.npy` et `images.npy`, vous utiliserez les fonctions `importImageVect` et `importImageHist` du fichier `projetM1.py`. En utilisant la représentation qui vous paraît la plus pertinente (image vectorisée ou histogramme), vous produirez un vecteur des classes prédites par votre perceptron.

Ces prédictions doivent faire partie du rapport, éventuellement commentées. Elles serviront aux correcteurs à évaluer la cohérence de votre approche avec les estimations fournies dans la section précédente.

5. performance à améliorer !

3.6 Pour aller plus loin (Bonus)

Seriez-vous capable de proposer un protocole permettant d'utiliser à la fois la représentation vectorielle et l'histogramme d'une image ? Vous pouvez tenter de répondre à cette question à la fin de votre rapport et rien ne vous empêche de tester votre méthode sur les données proposées.

Bon courage !