Etude Pratique (3) – Master Informatique – IAA – 2019
Université d'Aix-Marseille

Cécile Capponi – QARMA, LIS – AMU
Cecile.Capponi@lis-lab.fr

23 janvier 2019

Objectifs de la séance :
- kernéliser un algorithme de classification linéaire (perceptron)
- recherche d'hyper-paramètres
- Familiarisation avec données images
- comparaison avec d'autres algorithmes connus

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.
Ses envergure 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 un excellent classifieur, à l'aide d'un *perceptron à noyau* à programmer, ou à l'aide de tout autre algorithme d'apprentissage que vous connaissez. 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 de nouvelles photographies à venir. Il sera comparé aux classifieurs vus en cours, et disponibles sous `sk-learn`.

1 Perceptron à noyau

L'algorithme du perceptron permet l'apprentissage rapide d'un séparateur linéaire correct permettant la classification binaire. Cependant, un écueil de taille concerne l'incapacité de fournir une bonne solution dans le cas où les

données ne sont justement pas linéairement séparables, ce qui est le cas de la plupart des jeux de données réelles. Par ailleurs, les approches à base de noyaux permettent d'apprendre un séparateur linéaire dans un espace de plongement aux dimensions très grandes, via le *kernel trick*.
Nous proposons ici d'étudier l'algorithme du perceptron à noyaux : similaire à l'algorithme classique, l'idée consiste à apprendre un hyperplan α dans l'espace des caractéristiques (sans connaître cet espace) et à effectuer les produits scalaires dans cet espace via la fonction noyau qui reflète un produit scalaire, justement.
Nous partons d'un échantillon $S = \{(x_i, y_i)\}_{i=1..n}, x_i \in \mathbb{R}^d, y_i \in \{-1, +1\}$.

Explication du perceptron à noyau L'algorithme classique du perceptron permet d'apprendre un hyperplan séparateur w qui permet de prédire la classe de nouvelles données $x \in \mathbb{R}^d$ avec la fonction

$$f(x) = \text{sgn}(\langle w, x \rangle)$$

Supposons que nous avons une fonction ϕ de plongement de chaque donnée $x \in \mathbb{R}^d$ dans un espace plus grand : $\phi(x) \in \mathbb{R}^D$, $D >> d$, dans lequel nous avons plus de chance d'avoir un séparateur linéaire.
L'algorithme du perceptron nous permet alors d'apprendre un hyperplan séparateur w qui permet de prédire la classe de nouvelles données x avec la fonction :

$$f(x) = \text{sgn}(\langle w, \phi(x) \rangle)$$

et dans ce cas, la règle de mise à jour du vecteur w dans la phase d'apprentissage passe de $w \leftarrow w + y_i x_i$ à $w \leftarrow w + y_i \phi(x_i)$.
Par ailleurs, et indépendamment de toute méthode d'apprentissage, un hyperplan séparateur est toujours exprimé comme une combinaison linéaire des données d'échantillon, ce qui nous permet d'écrire l'équation de w comme suit (où la classe des exemples apparaît) :

$$w = \sum_i \alpha_i y_i x_i$$

Donc en considérant notre fonction ϕ de plongement :

$$w = \sum_i \alpha_i y_i \phi(x_i)$$

Sous cette forme, un algorithme d'apprentissage consiste à déterminer le vecteur α puisque tout le reste est connu. En injectant cette définition de w dans la fonction de prédiction f, nous obtenons une expression magnifique de la fonction de prédiction :

$$f(x) = \text{sgn}\big(\sum_i \alpha_i y_i \phi(x_i).\phi(x)\big) = \text{sgn}\big(\sum_i \alpha_i y_i k(x_i, x)\big)$$

avec la fonction noyau de Mercer $k(x_i, x) = \phi(x_i).\phi(x)$!
L'algorithme d'apprentissage par perceptron à noyau est donc le même que l'algorithme simple du perceptron, sauf que l'on apprend les α_i à la place des w_i (forme duale), et donc la formule de mise à jour en cas d'erreur de $f(x_i)$ dans l'itération est simplement :

$$\alpha_i \leftarrow \alpha_i + 1$$

Autrement dit, α_i compte finalement le nombre d'erreurs faites sur x_i lors de la phase d'apprentissage !

Algorithme d'apprentissage : le perceptron à noyau

Entrées : $S = \{(x_i, y_i)\}_{i=1..n}, x_i \in \mathbb{R}^d, y_i \in \{-1, +1\}$, fonction noyau $k_p(.,.)$
Initialisations : $\forall i : \alpha_i \leftarrow 0$.
Tantque Condition d'arrêt non remplie Faire
    Pour tous les exemples i de S Faire
        Calcul de $\hat{y_i} = f(x_i)$ avec $f(x_i) = \text{sgn}\big(\sum_j \alpha_j y_j k_p(x_j, x_i)\big)$

Si $\hat{y}_i \neq y_i$ Alors $\alpha_i \leftarrow \alpha_i + 1$
Sortie : α

où k_p est une fonction noyau avec ses paramètres p. La condition d'arrêt de la boucle externe peut varier : ne pas dépasser un certain nombre d'itérations proportionnel à $n * d$, continuer jusqu'à ce que le nombre d'erreurs de l'itération interne soit sous un certain seuil, etc. N'oublions pas qu'afin d'apprendre un hyper-plan qui ne passerait pas par l'origine, il est nécessaire d'ajouter une composante aux vecteurs x, de valeur constante à 1.

2 Programmation de fonctions noyaux

1. Programmer la fonction `noyauGaussien(x1,x2,sigma)` qui calcule et retourne, pour deux vecteurs de réels de même taille, et pour un réel positif, la valeur :

$$K_\sigma(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{\sigma^2}\right)$$

2. Programmer la fonction `noyauPolynomial(x1,x2,k)` qui calcule et retourne, pour deux vecteurs de réels de même taille, et pour un entier positif, la valeur :

$$K_k(x_1, x_2) = (1 + \langle x_1, x_2 \rangle)^k$$

3 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` [1].
2. Programmer la fonction `predictKernelPerceptron(kp, x, data, kernel, h)` qui prédit la classe d'un nouvel exemple `x` avec le perceptron à noyau `kp` (lui-même précédemment appris sur `data` avec la fonction noyau indiquée par `kernel` et `h`).
3. Via une validation croisée, tester ces fonctions sur un jeu de données binaire que vous connaissez (avec les classes -1 et +1 à la place de False et True).

4 Application à la classification d'images

4.1 Objectifs

L'objectif est de produire et fournir l'évaluation d'une série de modèles pour la classification d'images issues de drones. L'évaluation portera sur :

1. La qualité du résumé de comparaison des modèles, et le choix argumenté du modèle (avec ses hyper-paramètres) que vous estimez le plus performant.
2. Le score en généralisation du modèle précédemment choisi, score obtenu sur un jeu de tests conservé par les enseignants.

Dans cette partie du TP, 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.). Il faut donc se consacrer à deux phases :

1. Travail sur la représentation vectorielle des images,
2. Travail sur l'apprentissage d'un bon modèle à partir de la représentation vectorielle.

1. si `kernel` vaut 1, on utilise le noyau gaussien avec `h`=σ, sinon on utilise le noyau polynomial avec `h`=k.

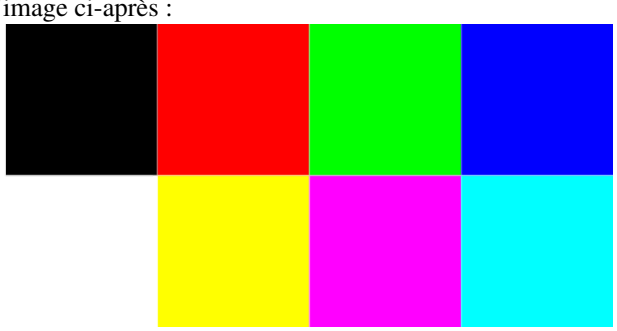
4.2 Représentations des images

Les données d'apprentissage sont fournies comme images brutes dans le répertoire `Data`, qui comporte deux sous-répertoires : des photos avec plans sur la `Mer`, et des photos sans mer dans `Ailleurs` ; ainsi, la classe de chaque photographie est le nom du répertoire dans lequel elle est stockée. Le fichier `tp5utils.py` contient une série de fonctions utiles pour manipuler ces images (les charger, les vectoriser, les organiser en datasets, etc.).
Les images n'ont pas forcément la même taille, le même ratio, la même résolution, etc. **Les données et programmes utiles sont ici (fichier `tp3-M1info2019.zip`)** :

`http://pageperso.lis-lab/cecile.capponi`, onglet Enseignement

4.2.1 Représentation sous forme de tableau de pixels

Une image numérique peut se représenter sous la forme d'un tableau de couleurs de pixels. Une couleur est attribuée à chaque pixel, sous la forme d'un triplet RGB indiquant l'intensité de rouge (R), de vert (G), et de bleu (B), pour ce pixel. Nous avons $\langle R, G, B\rangle \in [0..255] \times [0..255] \times [0..255]$.
Soit une image de taille $h * l = 2 * 4$. Sa représentation vectorielle RGB nécessite un vecteur de taille $3 * h * l$. Par exemple, le vecteur $[0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 255, 255, 255, 255, 255, 0, 255, 0, 255, 0, 255, 255]$ de taille $3 * 2 * 4*$ représente l'image ci-après :



(le premier triplet $0, 0, 0$ représente le noir, puis le triplet $255, 0, 0$ code le rouge, etc., le dernier triplet $0, 255, 255$ code le cyan).
La bibliothèque python PIL propose la classe `Image`, avec un grand nombre de fonctionnalités. Par exemple, pour vous persuader que, par défaut, une image numérique est un tableau de couleurs de pixels, vous pouvez comprendre et tester le code ci-après, sur une image de votre choix :

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

Toutes les images dans ce TP ne sont pas de même dimension : par conséquent, leurs vecteurs de pixels ne sont pas de même taille, ce qui empêche un apprentissage sur données vectorielles. Une solution pour pallier ce problème est de procéder à un pré-traitement des images avant de les considérer dans un échantillon d'apprentissage. Ce pré-traitement consiste à les **recadrer**, donc les transformer, vers des images qui sont toutes de même taille, à l'aide de la fonction `resize(h,l)` de PIL.
Dans ce TP, nous considèrerons des transformations vers des `images carrées`.

- La fonction `imageToVecteurPixels(filename, npix)` permet de charger l'image contenue dans le fichier de nom `filename`, de la transformer vers une image carrée de taille `npix*npix`, et retourne le vecteur de pixels colorés de cette image.
- La fonction `chargementVecteursImages(rep1, rep2, t1, t2, sizex)` permet de charger les images contenues dans les répertoires `rep1` et `rep2`, de les transformer et de les vectoriser avec la fonction précédente, puis de constituer un dataset de type `sklearn` (`data` et `target`). Les paramètres `t1` et `t2` sont des entiers correspondants aux classes : nous vous conseillons d'utiliser 1 et -1 commes valeurs. Le paramètre `npix` indique la taille du côté de l'image transformée (à passer à la fonction précédente).

Cette fonction retourne un quadruplet (le nombre total d'exemples chargés, le tableau de données, le tableau de classes, et la taille de la représentation vectorielle des images).

4.2.2 Représentation sous forme d'histogramme de couleurs

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 pixels de l'image possédant cette valeur. Quelle que soit la taille de l'image, un histogramme sera donc toujours un vecteur $\mathbf{v} \in \mathbb{N}^{3\times 256}$. Il est donc inutile de pré-traiter les images avant de calculer l'histogramme (comme dans le cas des vecteurs de pixels).
Pour obtenir un histogramme à partir d'un objet de type `Image` représentant une image en couleurs, on utilise la fonction `histogram()` de PIL. 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)
```

- La fonction `imageToHistogrammeCouleurs(filename)` permet de charger l'image contenue dans le fichier de nom `filename`, et retourne l'histogramme normalisé (un vecteur !) pour cette image.
- La fonction `chargementHistogrammesImages(rep1, rep2, t1, t2)` permet de charger les images contenues dans les répertoires `rep1` et `rep2`, de les vectoriser avec la fonction précédente, puis de constituer un dataset de type `sklearn` (`data` et `target`). Les paramètres `t1` et `t2` sont des entiers correspondants aux classes : nous vous conseillons d'utiliser 1 et -1 commes valeurs.
  Cette fonction retourne un quadruplet (le nombre total d'exemples chargés, le tableau de données, le tableau de classes, et la taille de la représentation vectorielle des images).

4.3 Apprentissages et évaluations de fonctions de classification

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

1. S_{V_q} qui comporte les images transformées vers une taille $q * q$ puis représentées par des vecteurs de pixels (on a donc autant d'échantillons que de valeurs raisonnables de q),
2. S_H qui comporte les images représentées par des vecteurs de couleurs (histogrammes de couleurs), d'une taille 3*256=768.

Pour chacun de ces échantillons, plusieurs algorithmes d'apprentissage doivent être utilisés pour produire différentes fonctions de classification : l'objectif est de garder le meilleur !
Parmi ces algorithmes, nous en imposons 4 parmi 6 [2] :

- Le perceptron
- Le perceptron à noyau
- Un SVM linéaire (SVC)
- Les Kppv
- Un arbre de décision
- Le classifieur naïf de Bayes

Chacun de ces algorithmes vient avec des *hyper-paramètres*. Par exemple, le perceptron dépend du noyau choisi (Gaussien ou polynomial), et des hyper-paramètres de cette fonction noyau. De surcroît, si le perceptron à noyau est appris sur les vecteurs de pixels, la taille du vecteur peut influer le résultat : la donnée `npix` est aussi un hyper-paramètre, qui conditionne la représentation des données.
On cherchera à apprendre le meilleur modèle pour chaque échantillon, pour chaque algorithme d'apprentissage, et le cas échéant et pour chaque noyau. In fine, pour chaque algo, il s'agit de déterminer la meilleure valeur des hyper-paramètres.
Donc, pour chaque échantillon, et pour chaque ensemble d'hyper-paramètres : on apprendra le meilleur modèle et on évaluera ses performances. On justifiera ce qu'est "meilleur", et comment cela est mesuré (validation croisée, testsplit, etc.) [3].

2. Libre à vous d'en proposer d'autres
3. Il peut parfois être opportun de considérer le temps d'apprentissage du modèle, et le temps de classification d'un exemple.

	meilleur(s) hp vecteur	meilleur(s) hp algo	err. réelle estimée	err. apparente
KP Noyau gaussien	V.Pix 10	$\sigma = 0.1$		
KP Noyau gaussien	V.Pix 10	$\sigma = 0.5$		
KP Noyau gaussien	V.Pix 10	$\sigma = 1.0$		
KP Noyau gaussien	V.Pix 16	$\sigma = ...$		
KP Noyau gaussien	V.Pix 32	$\sigma = ...$		
KP Noyau gaussien	V.Pix XX...	$\sigma = ...$		
KP Noyau gaussien	V.Histo	$\sigma = ...$		
KP Noyau poly	V.Pix $(10 - 16 - 32 - 64 - ...)$	$d = ...$		
KP Noyau poly	V.Histo	$d = ...$		
AD	V.Pix $(10 - 16 - 32 - 64 - ...)$	$p = ..., f = ...$		
...	...	...		

TABLE 1 – Tableau de résultats. hp signifie hyper-paramètres ; V.Pix indique un vecteur de pixels colorés ; V.Histo indique une représentation par histogramme ; KP indique Kernel Perceptron ; AD signifie Arbre de Décision avec p la profondeur, ou f nombre de feuilles. Ajoutez autant de lignes que nécessaires au regard de vos expérimentations.

On remplira alors un résumé en s'inspirant du tableau 1 (en expliquant comment ces valeurs ont été obtenues) : beaucoup d'algos et d'hyper-paramètres varient, il faut tout reporter.

A votre avis, quel est le meilleur modèle obtenu ? Justifier.
En guise de *baseline*[4], une validation croisée 10 folds, testant un SVM marges douces à noyau linéaire sur les histogrammes des images, avec comme paramètre de souplesse C=15, mène à un taux de bonne classification de $0.817(+/-0.187)$. Faites vous mieux ?

4.4 Prédictions sur de nouvelles images

L'archive `tp3M1-2019.zip` contient plusieurs fichiers de représentations d'images test, sous format binaire. Chaque fichier donne ainsi la représentation de 39 images, certaines de Mer, certaines d'autres paysages.
Le nom de chaque fichier indique sous quelle représentation les images sont données :

- `tPixel-10.npy` contient, sur chaque ligne, une image représentée par un vecteur de pixels colorés, l'image étant recadrée vers une image 10*10.
- Plus généralement, `tPixel-nnn.npy` contient, sur chaque ligne, une image représentée par un vecteur de pixels colorés, l'image étant recadrée vers une image n*n.
- `tHisto.npy` contient, sur chaque ligne, une image représentée par un histogramme de 256 couleurs.

Vous devez rendre la prédiction de la classe pour chacune de ces images, dans l'ordre du fichier d'entrée, avec le meilleur algorithme + hyper-paramètres que vous avez identifiés. Si la taille de recadrage que vous désirez manque, merci d'en faire la demande à vos enseignants.

4.5 Pour aller plus loin (Bonus)

Seriez-vous capable de proposer un protocole permettant d'utiliser à la fois la représentation vectorielle en pixel 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.

4. performance à améliorer !