Projet Apprentissage Statistique

Super-résolution d'images de température

November 22, 2016

Note : ce projet est composé d'une partie commune (mise en place de modèles de régression) et de 9 parties spécifiques à se répartir en groupe de deux.

Il faudra rendre le code sous forme matlab bien commenté. Une présentation sera également demandée pour la dernière séance. Pour le code matlab, l'entrée devra être de la forme :

```
images_superresolues= super_resolution(images_learning_basse_resolution,...
...  images_learning_haute_resolution,images_test,param)
```

où `param` est une structure de données avec au moins le champ :

- `param.method` un entier valant désignant le numéro de la méthode (0 pour la méthode SVR présentée en section 1.1, -1 pour la méthode par plus proches voisins présentée en 1.2 et le numéro de la méthode correspondant au projet associé, cf deuxième partie).

Dans cette structure vous pouvez mettre n'importe quel champ, par exemple

- `param.gamma` (ou `param.sigma`) le paramètre correspondant au coefficient du noyau gaussien :

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{y}\|^2\right) \tag{1}$$

- ...

Partie commune

1 Problème de la super-résolution

Le but de la super résolution est d'améliorer la précision spatiale des données. Il existe un grand nombre de méthodes permettant de le faire et ici, nous souhaitons nous appuyer sur de l'apprentissage statistique. Le cadre d'application concerne l'amélioration de cartes de températures dans un vignoble bordelais. Une illustration est visible en figure 1.

1.1 Formalisation comme un problème de régression

Le problème de super-résolution peut être vu comme un problème de régression. Pour trouver les liens entre résolutions, on travaille avec le *voisinage* d'un point de l'image à basse résolution (pour capter l'information de contexte) que l'on cherche à mettre en relation avec un nombre r^2 de points correspondant à l'image à haute résolution, r étant de *coefficient de super-résolution*. Pour cela, on va utiliser des couples d'images à basse et haute résolution sur lesquelles on extraira nos données d'apprentissage. En pratique ces données sont mises dans des vecteurs colonnes. Ainsi, il s'agit de trouver, en chaque pixel de l'image à basse résolution, le lien entre un vecteur multi-varié (de dimension $(2r_{lr} + 1) \times 1$) vers un autre vecteur multivarié de dimension $(r^2 \times 1)$ où r_{lr} est le *rayon* du voisinage utilisé. Une illustration est visible dans la figure 2. Il existe plusieurs techniques pour manipuler le cas où les sorties sont multi-variées mais ici, on va à chaque fois supposer que

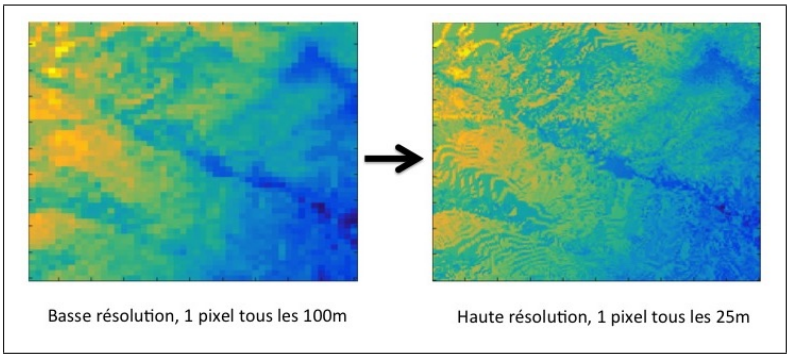
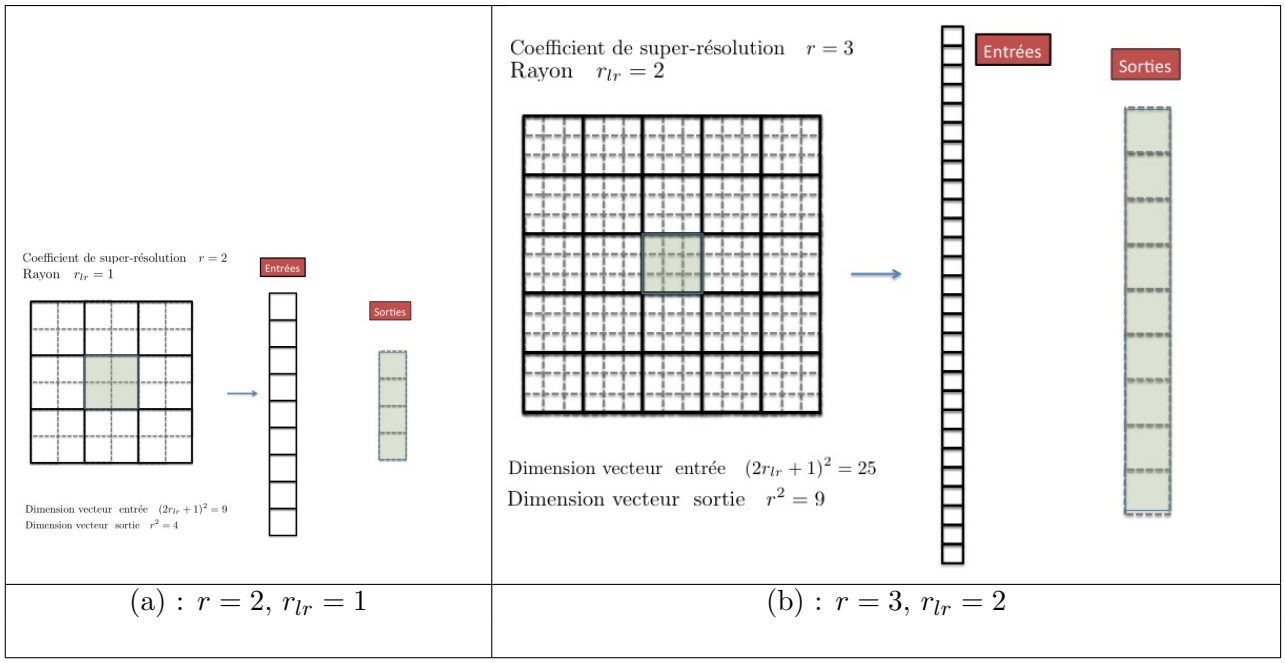


Figure 1: **Illustration de la super-résolution**



(a) : $r = 2$, $r_{lr} = 1$ (b) : $r = 3$, $r_{lr} = 2$

Figure 2: **Problème de régression associé à chaque pixel** en fonction du rayon r_{lr} du voisinage de haute résolution choisi et du coefficient r de super-résolution

l'on a r^2 problèmes de régression indépendants. Chaque problème va être abordé par une technique de SUPPORT VECTOR REGRESSION (SVR) vue en cours. Pour les détails sur la méthode, se référer au cours. En pratique, pour appliquer une technique svr, on utilise la librairie LIBSVM. Un exemple est donné dans le répertoire `svr`, dans la fonction `test_regress.m` (avec de la régression en uni ou multi-varié). En parcourant le fichier `svm_regression.m`, vous verrez qu'on fait toutes les étapes nécessaires au bon fonctionnement (on centre et réduit les données, on fait l'apprentissage et on applique ensuite au test).

Nous utiliserons cette fonction par la suite, avec différents noyaux.

1.2 Formalisation comme un problème de plus proches voisins

Le problème de super résolution peut également être vu comme un problème de PLUS PROCHES VOISINS (KNN). Dans ce cas, on dispose d'une base de couples *(basse,haute résolution)*. A chaque nouvelle données à basse résolution, on peut associer les données à haute résolution issues de la base d'apprentissage qui correspondent au vecteur à basse résolution le plus proche. On peut aussi choisir une combinaison de linéaire des k valeurs les plus proches, en pondérant de manière inversement proportionnelle à la distance à ces k voisins.

1.3 Validation-croisée

Que l'on utilise la méthode SVR avec un noyau gaussien ou les KNN, un *hyper-paramètre* (le paramètre γ lié au noyau gaussien dans le relaiton (1) ou le nombre de voisins) est à fixer. En pratique, cela se fait par *validation-croisée*. La méthode vue en cours consiste, pour une valeur donnée, à faire n découpages (la valeur de n étant à fixer par l'utilisateur, souvent on prend $n = 10$) de l'échantillon d'apprentissage et à faire l'apprentissage sur $n-1$ échantillons et à tester sur l'échantillon restant. On répète cela n fois afin que tous les échantillons aient servis une fois de test. La performance moyenne est gardée pour la valeur du paramètre donnée. On répète ce processus pour un ensemble de valeurs possibles du paramètre à fixer et on garde la valeur conduisant au meilleur résultat.

Pour découper un jeu de données, vous avez la fonction `decoupe_data.m` qui fonctionne de la manière suivante :
`[label_t,label_v,data_t,data_v]=decoupe_data(label,data,nb_folds)`
où les sorties sont des cellules contenant `nb_folds` tableaux (accessibles via `label_t{i}`, `data_t{i}`, ...) correspondant aux découpages des données de tests et de validation.

Pour évaluer chaque régression, le critère d'erreur retenu entre les données réelles $\mathcal{X}$ et les données estimées $\tilde{\mathcal{X}}$ sera l'erreur quadratique moyenne:

$$EQM(\mathcal{X}, \tilde{\mathcal{X}}) = \sqrt{\frac{1}{|\mathcal{X}|}\|\mathcal{X} - \tilde{\mathcal{X}}\|^2}, \tag{2}$$

où $|\mathcal{X}|$ est le cardinal de $\mathcal{X}$ (nombre de points).

1.4 Données disponibles

Dans le fichier `data_projet.mat`, vous avez un série de 130 images à haute résolution (variable `images_apprentissage_hr` où chaque image est de taille 200×200. En pratique cela correspond aux températures mesurées entre février et novembre 2016) et des séries correspondantes à basse résolution qui sont 4 fois plus petites (variable `images_apprentissage_lr` de chaque image est taille 100×100, dans ce cas le coefficient de super-résolution est $r = 2$) et 16 fois plus petites (variable `images_apprentissage_lr2` de chaque image est taille 50×50, dans ce cas le coefficient de super-résolution est $r = 4$). Pour accéder à une image donnée au temps `t`, on fait la commande `images_apprentissage_hr(:,:,t)`. Par exemple, pour visualiser la 45ème image haute résolution, vous pouvez faire `imagesc(images_apprentissage_hr(:,:,t))`. C'est sur ces images d'apprentissage que vous allez apprendre les modèles de régression.

Dans une seconde partie, vous appliquerez ces modèles aux images de test (`images_test_lr` et/ou `images_test_lr2`) pour générer des estimations d'images à haute résolution qui seront à comparer avec les vrais données `images_test_hr`.

1.5 Fonction utiles matlab

Ces fonctions peuvent s'avérer utiles pour la suite :

- reshape qui permet de transformer les données. Par exemple, pour changer une image en vecteur et inversement, vous pouvez faire :

  ```
  # [nb_lig,nb_col]=size(im) % On récupère la taille de l'image
  # vec_im=reshape(im,[nb_lig*nb_col,1]) % L'image est dans un vecteur de taille (nb_lig*nb_col×1)
  # % Pour retransformer le vecteur en image :
  # im=reshape(vec_im,[nb_lig,nb_col])
  ```

- Pour accéder au voisinage d'un rayon R pixel de coordonnées (i, j) dans l'image :

  ```
  # im(i-R:i+R,j-R:j+R)
  ```

1.6 Travail commun demandé dans la première partie

1.6.1 Familiarisation avec les SVR

Faire tourner le programme `test_regress` en modifiant le paramètre en ligne 1 (`unidimension`. Ce paramètre vaut 1 pour une régression avec un vecteur de sortie en dimension 1 ou 0 pour un vecteur de sortie en dimension 2) et commenter brièvement les fichiers `test_regress` et `svm_regression`.

1.6.2 Super-résolution avec les SVR

Créer une fonction :
`images_superresolues= super_resolution(images_learning_basse_resolution,...`
`... images_learning_haute_resolution,images_test,param)`
avec `param.method=0` qui effectue une super-résolution avec la technique SVR. On utilisera un noyau gaussien dont le paramètre `param.gamma` pourra être fixé manuellement ou automatiquement (si sa valeur est `param.gamma=0`) par validation croisée. Pour cela, on fera varier γ dans la gamme $[0.1, 0.5, 1, 2, 5, 10, 15, 20]$.

1.6.3 Super-résolution avec les plus proches voisins

Créer une fonction :
`images_superresolues= super_resolution(images_learning_basse_resolution,...`
`... images_learning_haute_resolution,images_test,param)`
avec `param.method=-1` qui effectue une super-résolution avec la technique KNN. Le nombre de voisins pourra être fixé manuellement dans la variable `param.k` ou automatiquement (si sa valeur est `param.k=0`) par validation croisée. Pour cela, on fera varier k dans la gamme $[1, 2, 3, 4, 5]$. La valeur du voisinage sera définie par le paramètre `param.rayon`. Si cette valeur est nulle, elle sera aussi définie par validation croisée (dans ce cas il faut faire une double boucle sur `param.rayon` et `param.k`, en prenant des rayons de $1, 2$ et 3).

1.6.4 Tests

Testez ces fonctions (avec les meilleurs paramètres obtenus par validation croisée, en n'oubliant pas de les indiquer) sur les images fournies et analysez les résultats.

Parties en binôme

Arrivé ici, chacun a implémenté les deux méthodes de base (SVR et KNN). On va à présent modifier les données en entrée et/ou les méthodes et nous analyserons les bénéfices ou non de ces changements (en comparant leurs performances à celles des deux méthodes initiales).

Projet # 1: sous modèles SVR par régions

Concernant la régression, nous avons r^2 modèles que l'on applique sur toute l'image. Etant donné qu'il y a un certain relief, il est raisonnable de penser qu'un modèle valable en plaine le sera moins en altitude, et inversement. Pour cela, on propose donc de faire un ensemble de D modèles qui correspondent à D zones distinctes de l'image. Ces D zones peuvent être obtenues par clustering de l'image initiale. On fournit pour cela la fonction `kmeans_image.m`. Expliquez brièvement la manière dont fonctionne cette clusterisation (en fonction du code). Effectuez ensuite un apprentissage pour chacune des D zones (testez avec $D = 4$ et si

vous avez le temps fixez D par validation croisée. Ce paramètre sera fixé dans `param.D`, sachant qu'ici, on fixera `param.method=1`). Comparez les résultats par rapport à la méthode SVR initiale.

Projet # 2: SVR et enrichissement des données d'entrées par les gradients

Le but de prendre un voisinage du pixel initial est d'ajouter des informations de *contexte* : la manière dont l'augmentation de résolution s'effectue ne dépend pas uniquement de la valeur en la position étudiée mais aussi de son voisinage. Ainsi, on propose d'enrichir les données et d'ajouter, en plus des valeurs de température, des gradients calculés à une certaine échelle, ce qui permet d'introduire en un pixel donné des informations sur son voisinage. Pour cela, on dispose de la fonction
`[im_x,im_y]=gradient_images(im,sigma)` qui génère les images de gradients dans les directions x et y pour une échelle donnée `sigma`. Enrichissez le vecteur d'entrée par les vecteurs gradients à différentes échelles (prenez $\sigma = [2, 3, 5, 10, 20]$) et ré-appliquez la régression SVR. Les valeurs de sigma seront stockées dans un tableau dans le paramètre `param.val_sigma`, sachant qu'ici, on fixera `param.method=2`. Prévoyez aussi un paramètre `param.acp` dont la valeur est 0 (méthode que vous venez d'effectuer) ou 1. Dans ce dernier cas, effectuez une ACP afin de réduire la dimension du vecteur d'entrée et analysez ou non les intérêts.

Projet # 3: SVR et enrichissement des données par des spectres

Le but de prendre un voisinage du pixel initial est d'ajouter des informations de *contexte* : la manière dont l'augmentation de résolution s'effectue ne dépend pas uniquement de la valeur en la position étudiée mais aussi de son voisinage. Ainsi, on propose d'enrichir les données et de prendre, en plus des valeurs de température, les spectres associés, ce qui permet d'introduire en un pixel donné des informations sur son voisinage. L'opération consiste juste à prendre, pour chaque pixel, une série dont chaque valeur correspond au pixel de l'image ayant subit un filtre d'une valeur croissante.

Pour cela, on dispose de la fonction
`im_spectres= spectres_images(im,sigma)` qui génère les images des spectres en chaque pixel (par exemple, essayez avec `sigma=0:0.5:15`, 0 correspondant à l'image initiale). Ré-appliquez la régression SVR. Les valeurs de sigma seront stockées dans un tableau dans le paramètre `param.val_sigma`, sachant qu'ici, on fixera `param.method=3`. Prévoyez aussi un paramètre `param.acp` dont la valeur est 0 (méthode que vous venez d'effectuer) ou 1. Dans ce dernier cas, effectuez une ACP afin de réduire la dimension du vecteur d'entrée et analysez ou non les intérêts.

Projet # 4: sous modèles KNN par régions

Nous avons dans la méthode KNN une base de données qui sert pour toute l'image. Etant donné qu'il y a un certain relief, il est raisonnable de penser qu'un motif en plaine sera différent d'un motif en altitude, et inversement. Pour éviter de remplir l'image à haute résolution avec des motifs correspondant à des zones trop différentes, on propose donc de faire un ensemble de D méthodes KNN qui correspondent à D zones distinctes de l'image. Ces D zones peuvent être obtenues par clustering de l'image initiale. On fournit pour cela la fonction `kmeans_image.m`. Expliquez brièvement la manière dont fonctionne cette clusterisation (en fonction du code). Effectuez ensuite un apprentissage pour chacune des D zones (testez avec $D = 4$ et si vous avez le temps fixez D par validation croisée. Ce paramètre sera fixé dans `param.D`, sachant qu'ici, on fixera `param.method=4`). Comparez les résultats par rapport à la méthode KNN initiale.

Projet # 5: KNN et enrichissement des données d'entrées par les gradients

Le but de prendre un voisinage du pixel initial est d'ajouter des informations de *contexte* : la manière dont l'augmentation de résolution s'effectue ne dépend pas uniquement de la valeur en la position étudiée mais aussi de son voisinage. Ainsi, on propose d'enrichir les données et d'ajouter, en plus des valeurs de température, des gradients calculés à une certaine échelle, ce qui permet d'introduire en un pixel donné des informations sur son voisinage. Pour cela, on dispose de la fonction
`[im_x,im_y]=gradient_images(im,sigma)` qui génère les images de gradients dans les directions x et y pour une échelle donnée `sigma`. Enrichissez le vecteur d'entrée par les vecteurs gradients à différentes échelles (prenez $\sigma = [2, 3, 5, 10, 20]$) et ré-appliquez la technique KNN. Les valeurs de sigma seront stockées dans un tableau dans le paramètre `param.val_sigma`, sachant qu'ici, on fixera `param.method=5`. Prévoyez aussi un paramètre `param.acp` dont la valeur est 0 (méthode que vous venez d'effectuer) ou 1. Dans ce dernier cas, effectuez une ACP afin de réduire la dimension du vecteur d'entrée et analysez ou non les intérêts.

Projet # 6: KNN et enrichissement des données par des spectres (1/3)

Le but de prendre un voisinage du pixel initial est d'ajouter des informations de *contexte* : la manière dont l'augmentation de résolution s'effectue ne dépend pas uniquement de la valeur en la position étudiée mais aussi de son voisinage. Ainsi, on propose d'enrichir les données et de prendre, en plus des valeurs de température, les spectres associés, ce qui permet d'introduire en un pixel donné des informations sur son voisinage. L'opération consiste juste à prendre, pour chaque pixel, une série dont chaque valeur correspond au pixel de l'image ayant subit un filtre d'une valeur croissante.

Pour cela, on dispose de la fonction
`im_spectres= spectres_images(im,sigma)` qui génère les images des spectres en chaque pixel (par exemple, essayez avec `sigma=0:0.5:15`, 0 correspondant à l'image initiale). Ré-appliquez la technique KNN. Les valeurs de sigma seront stockées dans un tableau dans le paramètre `param.val_sigma`, sachant qu'ici, on fixera `param.method=6`. Prévoyez aussi un paramètre `param.dtw` dont la valeur est 0 (méthode que vous venez d'effectuer) ou 1. Dans ce dernier cas, utilisez la distance DTW au lieu de la distance euclidienne pour effectuer le KNN. On dispose de la fonction `dist=dtw(t_1,t_2)` qui permet de calculer une distance au sens de DTW. Comparez les résultats par rapport à la méthode KNN initiale (avec ou sans DTW).

Projet # 7: KNN et enrichissement des données par des spectres (2/3)

Le but de prendre un voisinage du pixel initial est d'ajouter des informations de *contexte* : la manière dont l'augmentation de résolution s'effectue ne dépend pas uniquement de la valeur en la position étudiée mais aussi de son voisinage. Ainsi, on propose d'enrichir les données et de prendre, en plus des valeurs de température, les spectres associés, ce qui permet d'introduire en un pixel donné des informations sur son voisinage. L'opération consiste juste à prendre, pour chaque pixel, une série dont chaque valeur correspond au pixel de l'image ayant subit un filtre d'une valeur croissante.

Pour cela, on dispose de la fonction
`im_spectres= spectres_images(im,sigma)` qui génère les images des spectres en chaque pixel (par exemple, essayez avec `sigma=0:0.5:15`, 0 correspondant à l'image initiale). Ré-appliquez la technique KNN. Les valeurs de sigma seront stockées dans un tableau dans le paramètre `param.val_sigma`, sachant qu'ici, on fixera `param.method=7`. Prévoyez aussi un paramètre `param.acp` dont la valeur est 0 (méthode que vous venez d'effectuer) ou 1. Dans ce dernier cas, effectuez une ACP afin de réduire la dimension du vecteur d'entrée et analysez ou non les intérêts.

Projet # 8: KNN et enrichissement des données par des spectres (3/3)

Le but de prendre un voisinage du pixel initial est d'ajouter des informations de *contexte* : la manière dont l'augmentation de résolution s'effectue ne dépend pas uniquement de la valeur en la position étudiée mais aussi de son voisinage. Ainsi, on propose d'enrichir les données et de prendre, en plus des valeurs de température, les spectres associés, ce qui permet d'introduire en un pixel donné des informations sur son voisinage. L'opération consiste juste à prendre, pour chaque pixel, une série dont chaque valeur correspond au pixel de l'image ayant subit un filtre d'une valeur croissante.

Pour cela, on dispose de la fonction
`im_spectres= spectres_images(im,sigma)` qui génère les images des spectres en chaque pixel (par exemple, essayez avec `sigma=0:0.5:15`, 0 correspondant à l'image initiale). Ré-appliquez la technique KNN. Les valeurs de sigma seront stockées dans un tableau dans le paramètre `param.val_sigma`, sachant qu'ici, on fixera `param.method=8`. Prévoyez aussi un paramètre `param.ot` dont la valeur est 0 (méthode que vous venez d'effectuer) ou 1. Dans ce dernier cas, utilisez la distance du transport optimal au lieu de la distance euclidienne pour effectuer le KNN. On dispose de la fonction `cout = optimal_transport_1D(serie1,serie2)` qui permet de calculer une distance au sens du transport optimal. Comparez les résultats par rapport à la méthode KNN initiale.

Projet # 9: KNN et enrichissement des données d'entrées par les gradients

Le but de prendre un voisinage du pixel initial est d'ajouter des informations de *contexte* : la manière dont l'augmentation de résolution s'effectue ne dépend pas uniquement de la valeur en la position étudiée mais aussi de son voisinage. Ainsi, on propose d'enrichir les données et d'ajouter, en plus des valeurs de température, des gradients calculés à une certaine échelle, ce qui permet d'introduire en un pixel donné des informations sur son voisinage. Pour cela, on dispose de la fonction
`[im_x,im_y]=gradient_images(im,sigma)` qui génère les images de gradients dans les directions x et y pour une échelle donnée `sigma`. Enrichissez le vecteur d'entrée par les vecteurs gradients à différentes échelles (prenez $\sigma = [2, 3, 5, 10, 20]$) et ré-appliquez la technique KNN. Les valeurs de sigma seront stockées dans un tableau dans le paramètre `param.val_sigma`, sachant qu'ici, on fixera `param.method=9`. Prévoyez aussi un paramètre `param.knn_gamma` dont la valeur est 0 (méthode que vous venez d'effectuer) ou > 0. Dans ce dernier cas, effectuez une affectation par plus proche voisins dans un espace transformé par un noyau gaussien (cf relation (1)) de paramètre $\gamma =$ `param.knn_gamma`. Evaluez les intérêts de chaque méthode.