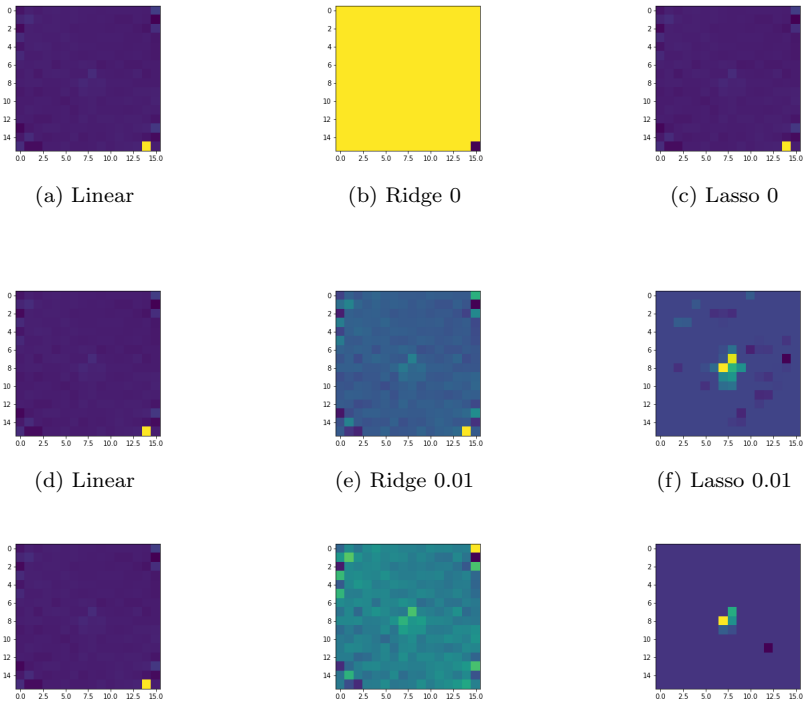
Rapport du projet ARF : Inpainting

Sylvain Sénéchal, Jean Lapostolle

Mai 2019

1 Régression linéaire, régression ridge et LASSO

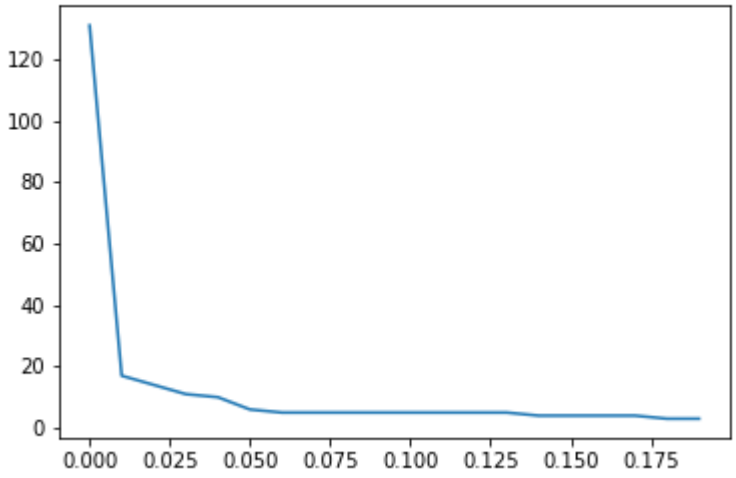


(a) Linear (b) Ridge 0 (c) Lasso 0

(d) Linear (e) Ridge 0.01 (f) Lasso 0.01

(g) Linear (h) Ridge 0.1 (i) Lasso 0.1

Figure 1: Coefficients selon la regression et alpha, nombre 0 et 8



(a) Coefficient non nuls vs Alpha

Figure 2: Nombre de coefficients non nuls en fonction de alpha dans le Lasso

On a réalisé une analyse des coefficients pour chaque regression, et pour le Lasso une analyse du nombre de coefficient non égaux à 0. La comparaison est faite entre les nombres 0 et 8 des données USPS.
On constate que les régressions ridge et Lasso vont avoir un nombre de coefficients à 0 de plus en plus importants quand alpha est croissant.
C'est-à-dire que ces 2 méthodes ne se servent que d'une petite partie des features pour faire la classification. Ce qui est intéressant avec les nombre 0 et 8 est qu'on voit bien que la classification se fait sur les pixels du milieu, puisque dans les 0 les pixels du milieu ne sont pas colorés ce qui est tout le contraire du chiffre 8.
Pour la classification des nombres USPS ces méthodes ont toutes de bonnes performances mais on va voir que pour la suite du travail le Lasso présente certaines particularités intéressantes.

2 Inpainting

2.0.1 Fonctions implémentées

- **readImage(path)** : Lecture de l'image
- **noise(img, percent, h)** : Création de bruit avec probabilité percent sur l'image (sauf les bords de largeur h)

- **deleteRectangle(img, col, row, height, width)** : Suppression d'un rectangle dans l'image
- **getPatch(img, row, col, h)** : Récupère le patch centré en (row, col) de largeur h
- **patchToVector(patch)** : Met le tenseur sous forme de vecteur
- **vectorToParch(vector, h)** : Met le vecteur sous forme de tenseur de largeur h
- **getDictionnary(img, h)** : Renvoie le dictionnaire de patchs de largeur h
- **getOnePatch(img, h, nextRow)** : Renvoie le premier patch trouvé qui a un pixel ou plus à reconstruire. (nextRow est un paramètre sauvegardé pour ne pas faire la recherche depuis le début de l'image à chaque fois)
- **lassoInpainting(dictionnary, patch, h, alpha)** : Renvoie le patch reconstruit qui approxime au mieux le patch de l'image originale
- **reconstructNoisyImage(noisyImage, h, alpha)** : Reconstruit l'image entière en executant plusieurs fois lassoInpainting
- **qualityReconstruction(sourcreImg, reconstructedImg)** : Evalue la quality de la reconstruction de l'image
- **benchmarkAlpha(image, noisyImage)** : Exécute plusieurs fois reconstructionImage avec des valeurs de Alpha différentes pour tracer le graphe de la qualité en fonction de alpha
- **benchmarkCoeff(image, noisyImage)** : Exécute plusieurs fois le lassoImpainting sur un même patch, avec différentes valeurs de Alpha pour comparer les coefficients des poids obtenus.

2.1 Travail sur l'inpainting d'images bruitées

NB :

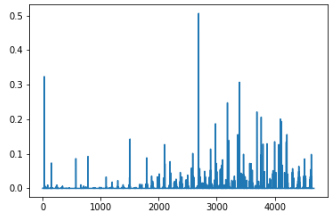
- Dans toute la suite du projet, on calcule l'erreur de reconstruction de l'image comme la somme pour chaque pixel de l'écart entre la couleur originale du pixel, et la couleur du pixel reconstruit.
- Pour les images bruitées, on est obligé pour des raisons de performances de ne bruiter qu'une quantité très faible de pixels (moins de 0.5 pourcent), si l'on veut pouvoir faire tourner plusieurs fois l'algorithme pour faire des comparaisons en un temps raisonable.
  La différence visuelle entre l'image de base et l'image reconstruite est alors peu perceptibles mais les résultats seront plus intéressant pour le travail sur images avec des gros blocs de pixels manquants.

Ici on va s'intéresser à déterminer le meilleur paramètre Alpha pour le Lasso.

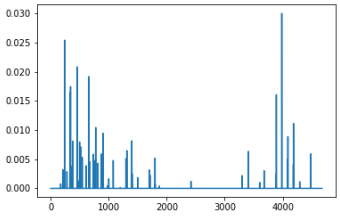
2.1.1 Dictionnaire incomplet, composé uniquement des patchs sans pixels à reconstruire

On va visualiser les coefficients de chaque patch du dictionnaire en fonction de la valeur d'alpha.

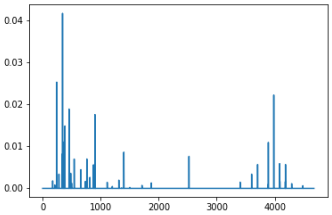
On voit que plus la valeur d'alpha est élevée, moins il y a de patch qui sont sélectionnés fortement. Pour alpha = 0, tous les patchs sont au moins un peu sélectionnés, c'est presque un simple floutage / moyennage.



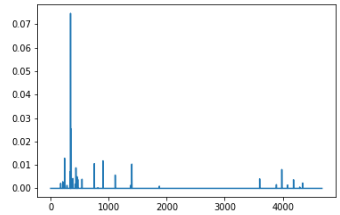
(a) Alpha = 0



(b) Alpha = 0.1

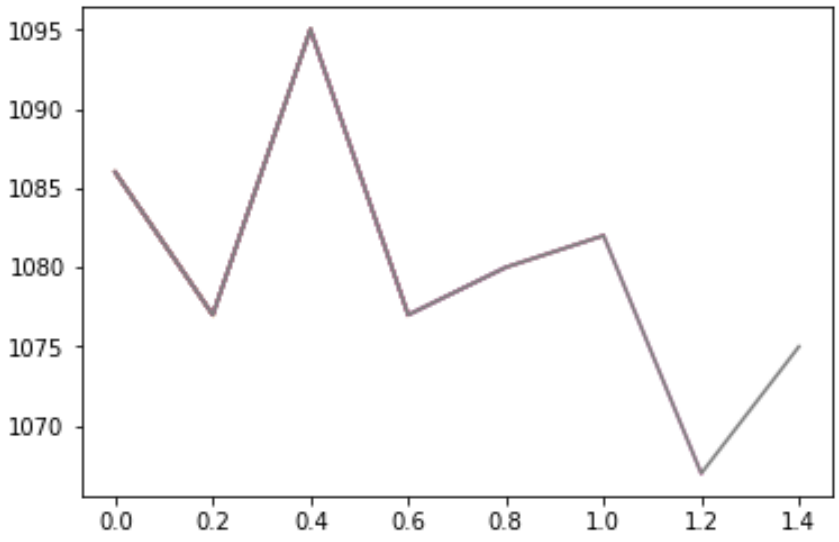


(c) Alpha = 0.3



(d) Alpha = 0.9

Figure 3: Lasso weights vs Alpha, dictionnaire incomplet



(a) Alpha = 0

Figure 4: Erreur de reconstruction pour le dictionnaire incomplet

(a) Image bruitée

(b) Image reconstruite

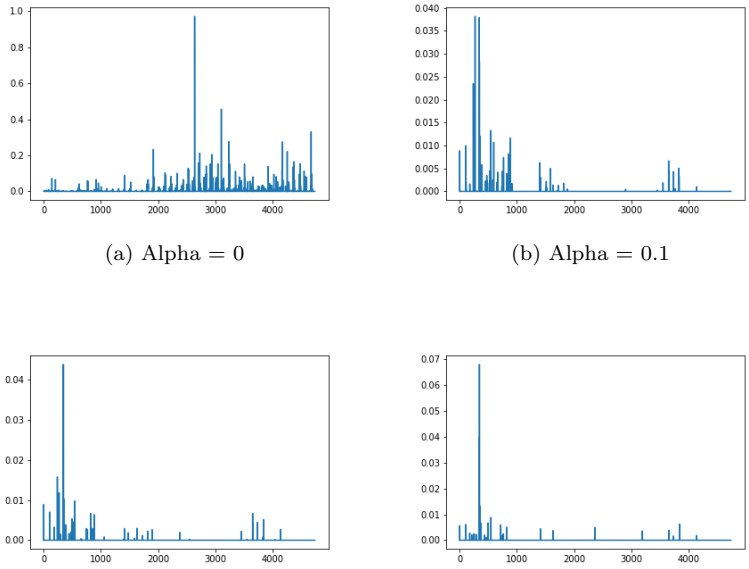
Figure 5: Reconstruction de l'image avec 2 pourcent des pixels bruités, dictionnaire incomplet

Dans une image bruitée, puisqu'on a toujours qu'un seul pixel à reconstruire par patch (sauf si a un bruit très important ce qui n'est pas le cas ici), le problème de régression est assez simple quand il ne manque qu'un pixel, d'autant

plus que le dictionnaire est presque complet avec seulement 0.05 pourcent de pixels bruités. La reconstruction est donc visuellement bonnes, et la faire avec le dictionnaire complet n'apportera pas forcément une meilleure qualité (même si théoriquement on doit avoir une nulle nulle sur le dictionnaire complet).

2.1.2 Dictionnaire issu de l'image complète

Pour le dictionnaire issu de l'image complète les résultats sont similaires. Un patch en particulier semble permettre de bien reconstruire le pixel manquant.



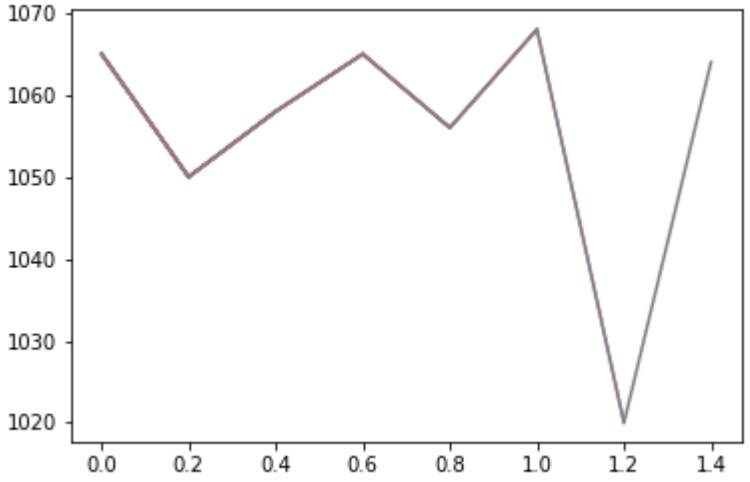
(a) Alpha = 0

(b) Alpha = 0.1

(c) Alpha = 0.3

(d) Alpha = 0.9

Figure 6: Lasso weights vs Alpha, dictionnaire complet



(a) Alpha = 0

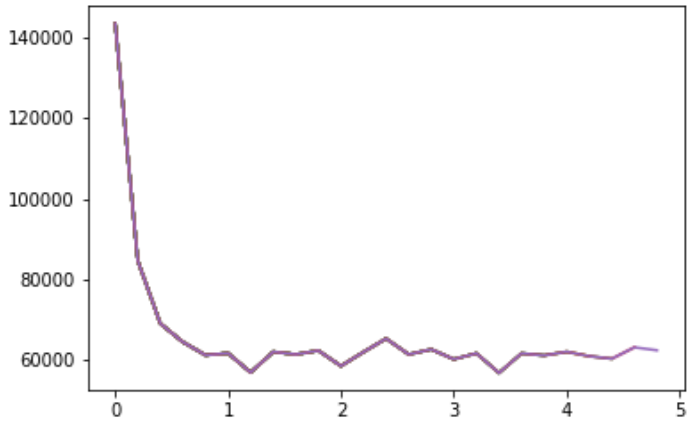
Figure 7: Erreur de reconstruction sur le dictionnaire complet

L'erreur de reconstruction que ce soit avec le dictionnaire issu de toute l'image, ou seulement des patchs non bruités ne varie pas énormément pour les images bruitées. Dans les deux cas puisqu'il n'y souvent qu'un seul pixel bruité par patch, la reconstruction est bonne.

2.2 Travail sur l'inpainting d'images avec parties rectangulaires manquantes

Le travail sur des images bruitées n'est pas forcément le plus intéressant puisque pour reconstruire un unique pixel isolé, faire un moyennage des pixels environnants est déjà souvent une approche satisfaisante.
Pour des parties entières de l'image manquantes, l'inpainting devient plus compliqué et la méthode du Lasso propose alors des résultats assez intéressants.

On détermine la valeur de alpha optimale en faisant des tests pour différentes valeurs de alpha :



(a) Erreur en fonction de alpha

Figure 8: Erreur de reconstruction vs alpha

La valeur d'alpha optimale semble être autours de 1.
En pratique, sur les images on obtient les reconstructions suivantes :

(a) Partie manquante

(b) Alpha = 0

(c) Alpha = 1

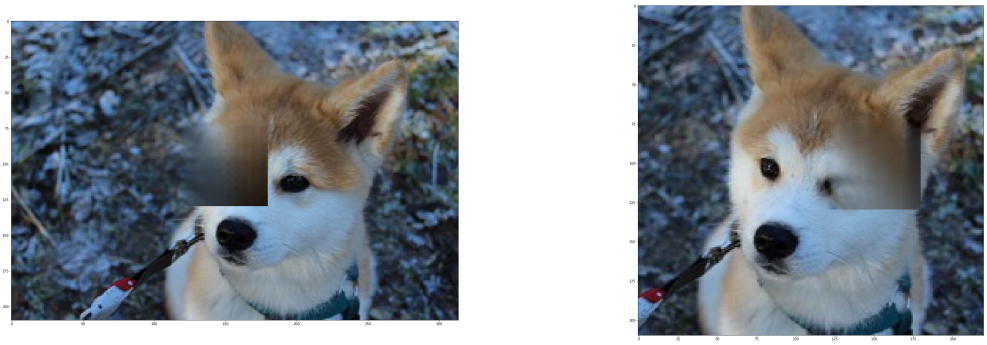
(d) Alpha = 5

Figure 9: Suppression d'une voiture sur l'autoroute pour différentes valeurs de alpha

Alpha est un paramètre qui permet en fait d'éviter que le Lasso ne fasse qu'un simple moyennage de l'image en sélectionnant tous les patchs avec un poids très faible, et en preferrant sélectionner quelques patchs avec un poids élevé. Cela donne un meilleur contraste entre pixels, plus naturelle, là où un moyennage simple a tendance a se voir directement à l'oeil dans une image.

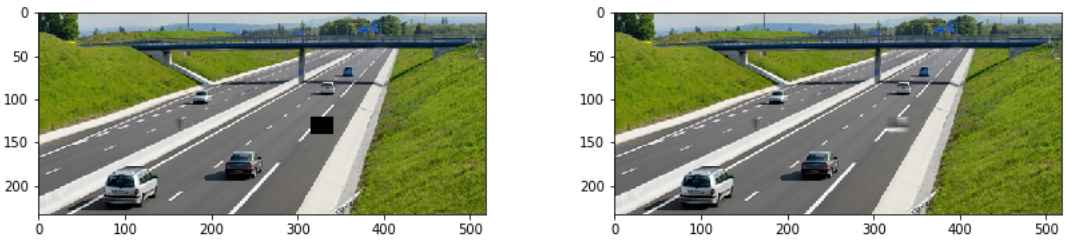
2.3 Améliorations de l'algorithme

Les résultats sont plutôt encourageants mais on remarque lors de la reconstruction que l'ordre dans lequel on prend les patchs a une importante. Lorsqu'on les prends de gauche à droite et de haut en bas, l'erreur de reconstruction a tendance à se propager à chaque fois, on le voit sur ces exemples :



(a) Exemple 1 (b) Exemple 2

Figure 10: Reconstruction de 2 images, le flou devient significatif vers la droite de la partie à reconstruire



(a) Partie noire retirée (b) Image reconstruite

Figure 11: Reconstruction d'une ligne blanche sur l'autoroute

Sur ces exemples, la partie reconstruite vers la droite tend à être flou et de mauvaise qualité après avoir accumulée tous les défaults de reconstruction de la

partie déjà reconstruite plus à gauche.
Sur la reconstruction de la ligne blanche, la ligne blanche a donc tendance à s'étirer sur la droite, la reconstruction ne tient pas assez compte de la structure à reconstruire.
Dans la papier [3] donné dans l'énoncé, à chaque itération on doit chercher à reconstruire le patch de priorité la plus élevée, c'est-à-dire le patch qui se trouve sur un contour et souvent à l'intersection d'un changement important de texture.
On peut aussi plus simplement remplir les contours en premiers et se rapprocher du centre, pour diminuer ce moitié la longueur de l'effet de flou.
On peut tenter de reconstruire des patchs de 1 pixels manquant plutôt que des patchs avec un pixel manquant centré qui a toujours au moins la moitié de ses pixels qui sont manquant, cette technique est cependant plus longue à mettre en place.
On pourrait aussi calculer la variance de la couleur des pixels de chaque patch à reconstruire, et reconstruire ceux qui ont la variance la plus élevée en premier.