Faculté des Sciences et Ingénierie
Sorbonne Université

Projet Inpainting : Projet de l'UE Apprentissage et Reconnaissance de Formes

Julien Denes, Michaël Trazzi

Master 1 Informatique, spécialité ANDROIDE
Année 2017 – 2018, Semestre 2

Ressources

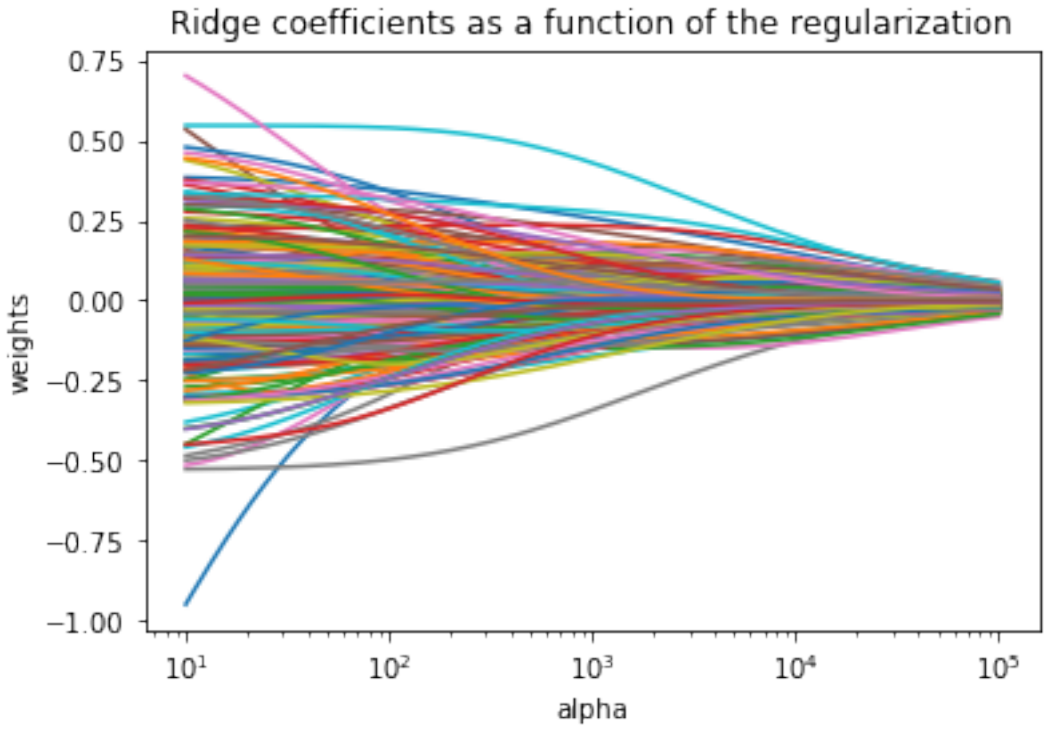
Le présent rapport, les figures qu'ils contient, les sources, ainsi que le code de ce projet, sont disponibles sur Github à l'adresse : `https://github.com/mtrazzi/Inpainting`. En particulier, nous avons réalisé des notebook jupyter pour chaque question, disponibles dans le dossier `src` (`https://github.com/mtrazzi/Inpainting/tree/master/src`).

Q1.1 - Régression linéaire, régression ridge et LASSO

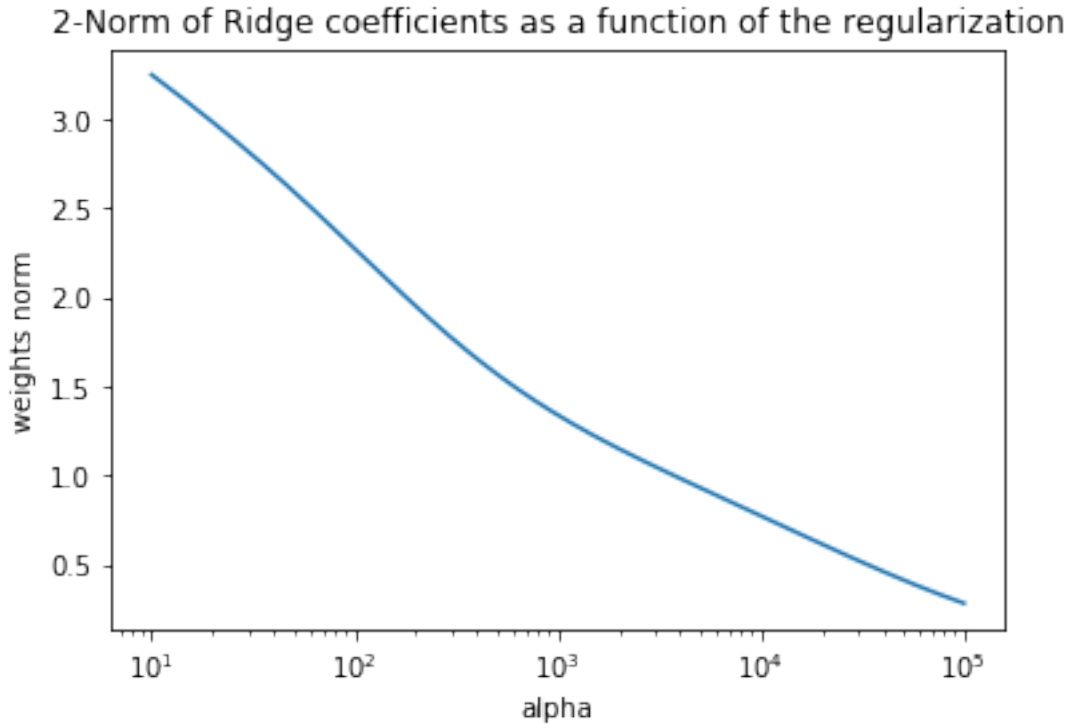
Nous avons ici comparé les algorithmes de regression linéaire, regression ridge et regression lasso en utilisant sklearn.

Ridge

Afin de déterminer quel était le coefficient de régularisation optimal, nous avons cherché dans le logspace `10**1` à `10**5`. On voit que les coefficients se rapprochent de plus en plus de zéros quand le coefficient de régularisation tend vers l'infini. L'écart-type entre les poids diminue de façon parcimonieuse.

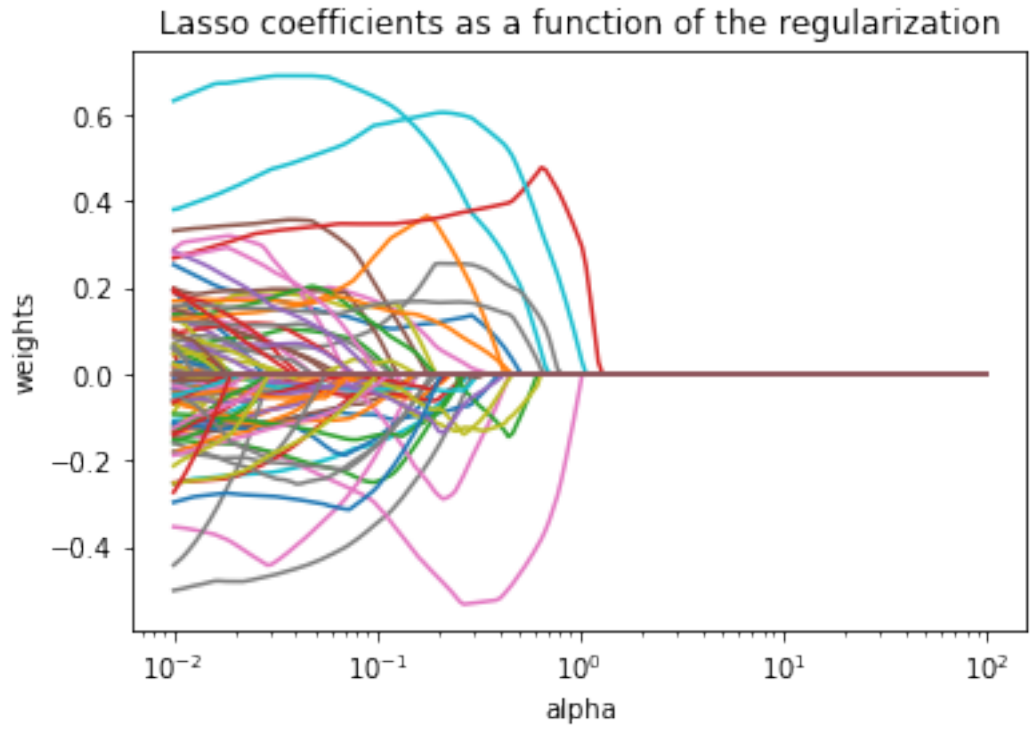


La norme du vecteur de poids diminue de façon linéaire (sur l'échelle logarithmique).

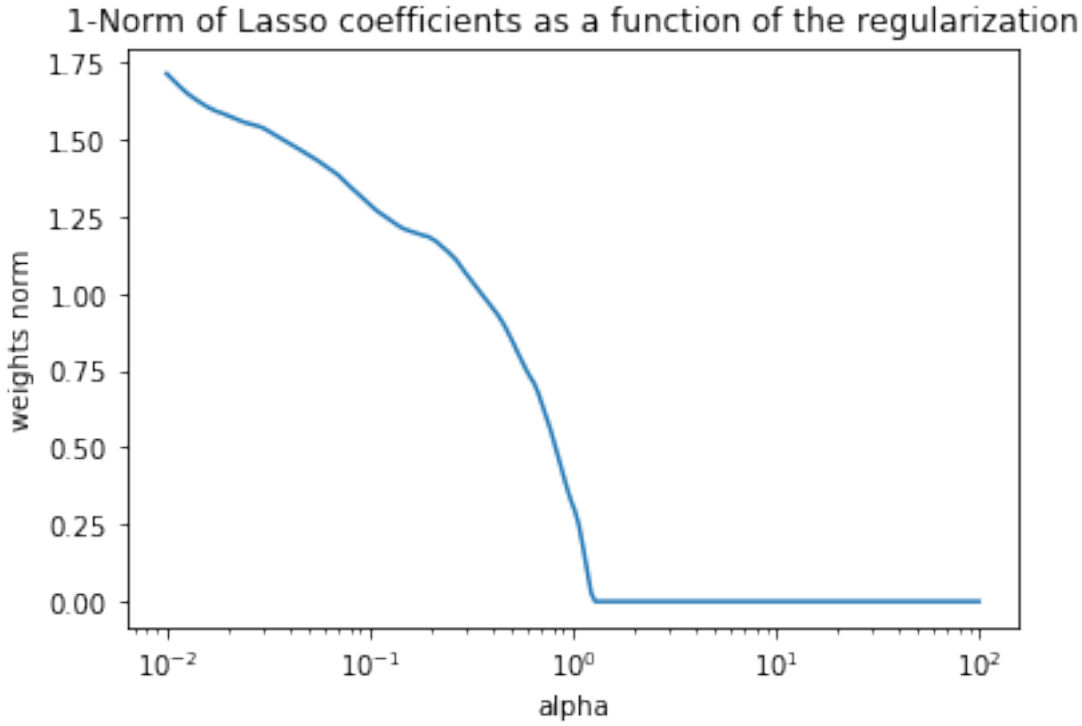


Lasso

Afin de déterminer quel était le coefficient de régularisation optimal, nous avons cherché cette fois-ci dans le logspace `10**-2` à `10**2`. Pour LASSO, les coefficients deviennent tous nuls quand alpha dépasse 1. On voit également que l'écart-type entre poids ne semble pas diminuer. Tout le long, des poids prennent des valeurs extrêmes, pour enfin chuter brusquement vers 0.



La norme du vecteur de poids diminue bien plus brusquement (sur l'échelle logarithmique) pour enfin devenir nul (à partir de alpha = 1).



Comparaison des algorithmes

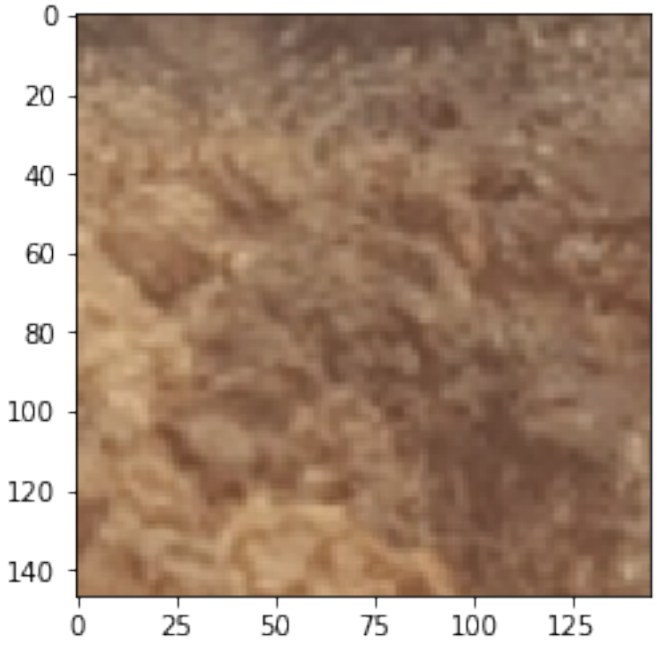
Voici l'écart obtenu entre la prédiction et les labels, sur la base de test des données USPS (entraînement sur la base de données d'entraînement avec les fonctions `linear_model.RidgeCV` et `linear_model.LassoCV` afin d'obtenir automatiquement le meilleur coefficient de régularisation (on obtient respectivement $\alpha = 142$ et $\alpha = 0.005$ pour ridge et lasso)).

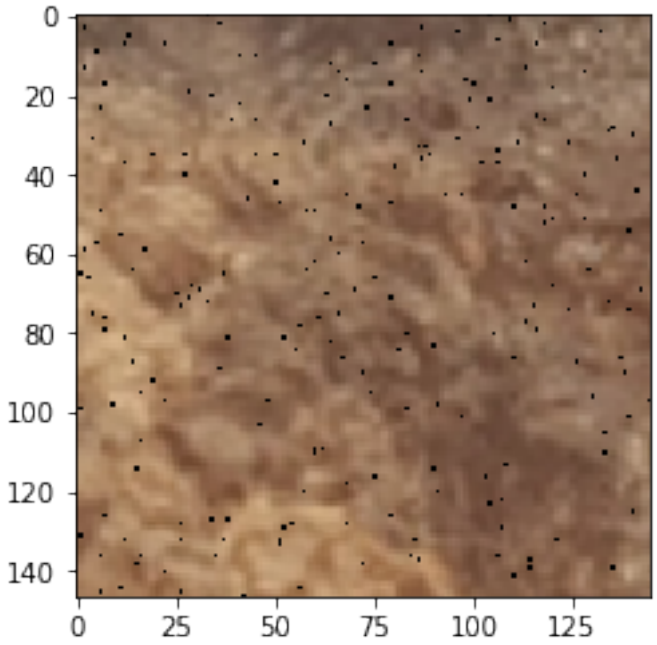
— Régression linéaire : `MSE` = 3.7549053135627135
— Ridge : `MSE` = 3.6942365444755985
— Lasso : `MSE` = 3.709303352040891

On voit donc qu'utiliser Lasso ou Ridge n'améliore pas sensiblement la `MSE`. L'intérêt dans ce cas d'utilier Lasso est d'éviter le suraprentissage, en obtenant rapidement un grand nombre de coefficients à 0, à partir de $\alpha = 1$, comme nous avons pu le voir dans les courbes ci-dessus.

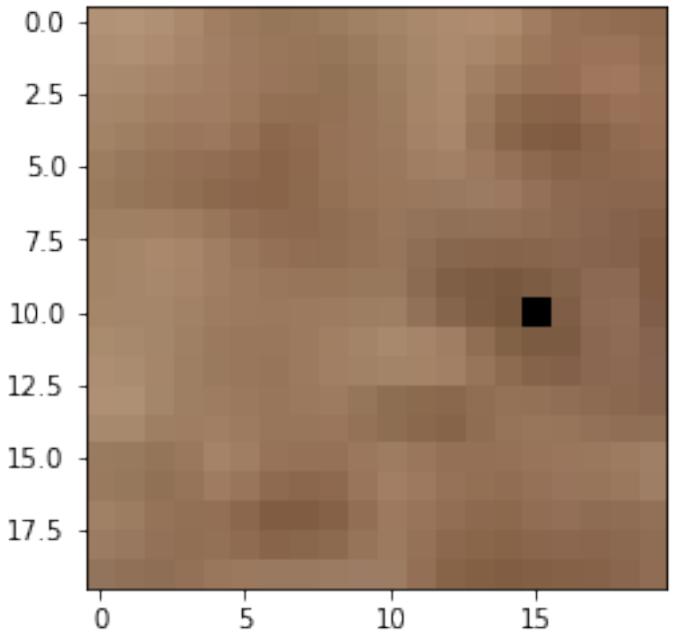
LASSO et inpainting

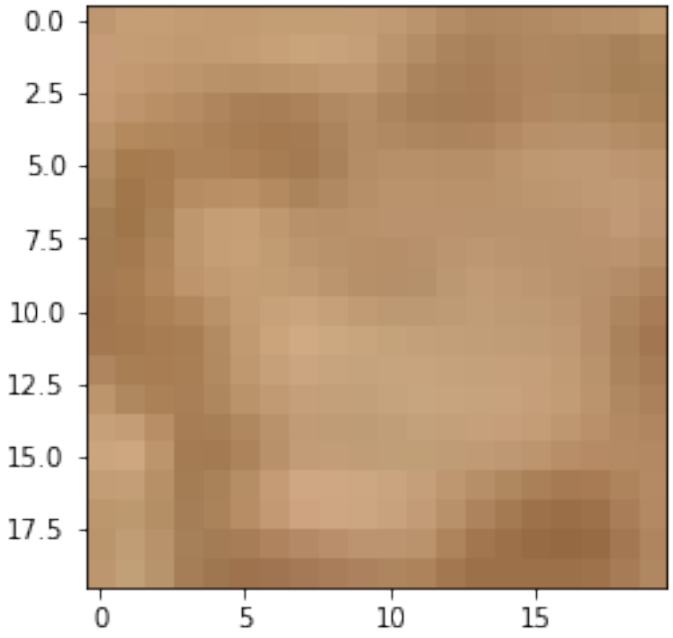
On commence pour cette partie par implémentée les fonctions suggérées. En particulier, on peut affichier les images et appliquer un bruitage sur une proportion des pixels :





On dispose également de fonctions `get_all_patches`, textttget_bad_patches et `get_dictionnary` qui permettent respectivement d'obtenir tous les patches possibles, ceux qui comportent du bruitage et le dictionnaire composé de ceux sans bruitage. On trouve par exemple ces éléments respectivement dans chacuns des deux derniers groupes :





Nous avons par la suite souhaité implémenter la méthode d'inpaiting avec Lasso, en utilisant Sklearn. On utilise pour cela le patch à prédire comme label (sous forme de vecteur, auquel on a retiré les pixels noirs) et le dictionnaire comme attributs (sous forme d'un ensemble de m vecteurs, où m est le nombre de patches dans le dictionnaire). On a une unique observation (un seul patch à prédire), puisqu'on ne souhaite pas concevoir un modèle généraliste mais seulement un modèle qui colle parfaitement au patch à prédire. Nos variables d'entrainement sont donc de taille $(1, k)$ pour Y, avec k le triple du nombre de pixels dans un patch, et $(1, m, k)$ pour les attributs X. En lieu d'attribut, on utilise donc un vecteur du dictionnaire. Le vecteur w que l'on souhaite obtenir devrait donc être de taille m.

Nous avons cependant rencontré un blocage lors de la prédiction avec sklearn, puisque ce dernier ne supporte pas les attributs multidimensionnels comme nos vecteurs du dictionnaire. Nous avons donc décidé de transformer X en une matrice de taille $(1, m * k)$. De la sorte, on associe

à chaque pixel un poids de chaque pixel de chaque vecteur du dictionnaire. Nous avons ensuite moyenné les poids sur chacun des vecteurs pour obtenir un poids w de taille correcte. Dans les faits, cette méthode se révèle peu efficace puisque tous les poids déterminés par le modèle sont égaux à 0.

La reconstitution des images, possible grâce à la fonction `recompose`, ne montre donc pas de grands résultats en l'absence de w correct.

Partie de l'image manquante

Lorsqu'on remplit l'image au fur et à mesure des bords vers le centre, l'ordre a une importance. En effet, il suffit de considérer l'exemple d'une photographie d'immeubles sur fond de ciel :

En parcourant d'abord du haut vers le centre puis du bas vers le centre, l'algorithme va commencer par remplir le haut en ciel car il ne verra pas la base de l'immeuble. En revanche, si on avait commencé en bas, on aurait déjà une base d'immeuble, donc on aurait pu mieux reconstruire le haut de l'immeuble

Plus généralement, on pourra appliquer la technique d'Harrison (P. Harrison. A non-hierarchical procedure for re-synthesis of complex texture. In Proc. Int. Conf. Central Europe Comp. Graphics, Visua. and Comp. Vision, Plzen, Czech Republic, February 2001.). C'est la technique utilisée dans « Region Filling and Object Removal by Exemplar-Based Image Inpainting » :

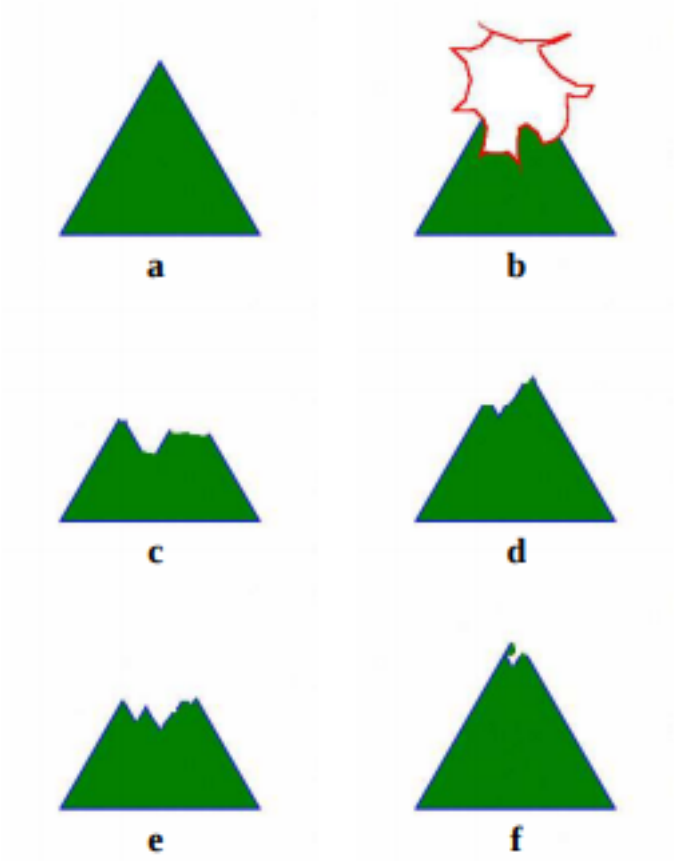


Fig. 8. **Effect of filling order on a synthetic image.** (a) The original image; (b) The target region has been selected and marked with a red boundary; (c) Filling the target region in raster-scan order; (d) Filling by concentric layers; (e) The result of applying Harrison's technique which took $2\,'\,45\,''$; (f) Filling with our algorithm which took $5\,''$. Notice that even though the triangle upper vertex is not complete our technique performs better than the others.