

Rapport de projet ARF Inpainting

Etudiants :

BAPTISTE JARRY
LUCAS BECIRSPAHIC

1 Préambule : régression linéaire, régression ridge et LASSO

Dans un premier temps, on cherche à tester les caractéristiques de la régression linéaire, la régression ridge et l'algorithme du LASSO. Pour ce faire, on utilise la base USPS qui est constituée de représentations de chiffres sous la forme d'images de 16x16 pixels. La base de données possède 10 types de label qu'on cherche à transformer en classification binaire. On effectue donc des tests en 1-versus-all, c'est à dire que pour le chiffre i tous les exemples représentant un i valent 1 et les autres valent -1.

On note tout de même que l'erreur aux moindres carrés n'est pas adaptée pour faire de la classification, par conséquent on testera plus les comportement des algorithmes en fonction de leurs paramètres que leurs résultats à proprement parler.

1.1 Protocole expérimental

On sépare nos données en une base de test et une base d'apprentissage. On apprend nos modèles sur la base d'apprentissage et on les évalue sur la base de test. Les régressions ridge et LASSO disposent d'un hyper-paramètre α dont on testera l'utilité. Pour chaque test, on regarde l'erreur au carré, la valeur minimale des poids, la valeur moyenne des poids, et le nombre de poids nuls.

model	error	nb0	lowerW	meanW
LinearRegression	161.83353591342077	0	-0.0827239468713741	-0.0009738344559072142
Lasso-alpha=1e-05	161.72284862809317	2	-0.07902895111699114	-0.0009766375184113937
Lasso-alpha=0.001	159.32110487457746	120	-0.051720076537509145	-0.0005198040063159515
Lasso-alpha=0.1	375.8377848909438	226	-0.04640298575457643	0.0012102055877842954
Lasso-alpha=0	161.83353245876967	0	-0.0827244184592763	-0.0009738363146636789
Lasso-alpha=1	917.4129598136857	256	-0.0	0.0
Lasso-alpha=10	917.4129598136857	256	-0.0	0.0
Lasso-alpha=100	917.4129598136857	256	-0.0	0.0
Lasso-alpha=1e16	917.4129598136857	256	-0.0	0.0
Ridge-alpha=1e-05	161.83353527307327	0	-0.08272391095920417	-0.0009738345685446765
Ridge-alpha=0.001	161.83347188294968	0	-0.08272035581441921	-0.0009738457204455762
Ridge-alpha=0.1	161.82715563685437	0	-0.08236649663051795	-0.0009749672314500567
Ridge-alpha=0	161.8335359134008	0	-0.08272394687144098	-0.0009738344559062094
Ridge-alpha=1	161.77163545558082	0	-0.0792914725255098	-0.0009853224518229387
Ridge-alpha=10	161.3376478223347	0	-0.05775798558606536	-0.0010459097359519014
Ridge-alpha=100	159.83816599354262	0	-0.050864279778332015	-0.0010293529718606882
Ridge-alpha=1e16	917.4129596659507	0	-1.914612789221231e-12	-4.270375584120613e-13

FIGURE 1 – Résultats expérimentaux sur l'apprentissage du "1"

On observe que seule la régression Lasso a des paramètres à 0 dans son vecteur w. De plus, on peut observer que le nombre de 0 augmente avec la valeur de alpha. En revanche la qualité de la prédiction se détériore jusqu'à une valeur maximale de 917 quand tous les paramètres sont à 0. Les résultats de Ridge et de Lasso sont identiques à la régression linéaire quand $\alpha = 0$, ce qui est parfaitement logique. Étonnamment, la valeur moyenne des poids est identique pour la plupart des valeurs de α , ce qui semble correspondre à un poids minimal qui garantit une bonne solution. En revanche pour un alpha vraiment très grand, on obtient des poids minuscules mais on a une erreur bien plus grande car il privilégie la réduction des poids à la prédiction de l'image. De plus on remarque que pour la

régression ridge, la valeur minimale de w est relativement proche de la moyenne, ce qui est logique car si un terme était beaucoup plus grand la pénalisation au carré de w le pénaliserait grandement. Donc l'algorithme va essayer de répartir la décision sur les différents paramètres.

1.2 Conclusion

régression ridge : La régression ridge introduit un terme $\alpha|w|^2$ dans la fonction objectif. Cela a pour effet de réduire considérablement la valeur de tous les termes de la matrice w . Cela permet également d'avoir une décision qui prend en compte tous les critères de manière plus équitable que la régression classique. De plus :

- si $\alpha = 0$, on obtient une régression linéaire classique
- si $\alpha = \infty$, on a $w =$ vecteur de taille minuscule mais non nul
- sinon les coefficients ont une valeur absolue entre 0 et celle obtenue par la régression classique.

régression lasso : La régression lasso (ou Least Absolute Shrinkage and Selection Operator) introduit un terme $\alpha|w|$ dans la fonction objectif. On observe qu'en plus de réduire la valeur des termes du vecteur w , on a un grand nombre de termes valant 0 pour une valeur de α suffisamment grande. Cela s'explique par le fait qu'on optimise la régression RIDGE avec un algorithme de descente du gradient, alors que l'algorithme du lasso utilise le coordinate descent. En effet, lors d'une descente de gradient on calcule le vecteur de poids de la manière suivante : $w \leftarrow w - \epsilon \nabla_w$. On effectue donc des opérations sur les flottants, donc on n'obtiendra jamais exactement 0. Le fait d'obtenir un grand nombre de poids nuls est une des grandes forces de l'algorithme du LASSO, ce qui permet d'extraire les paramètres réellement importants pour la décision.

2 Inpainting

Dans cette partie on traite le problème de l'inpainting qui s'attache à la reconstruction d'images détériorées ou au remplissage de parties manquantes (éliminer une personne ou un objet d'une image par exemple).

2.1 Reconstruction d'images détériorées

Dans cette partie, on bruite artificiellement des images, et on cherche à reconstruire l'image. Pour ce faire, on procède de la manière suivante :

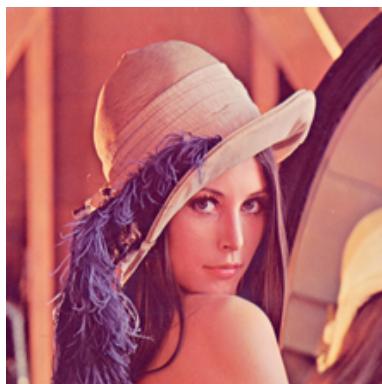
1. Construction d'un dictionnaire de patch non bruité nommé d
2. Pour chaque pixel bruité en position (i,j) , on construit un patch p^i autour de taille h
3. On construit une base de données de la manière suivante : chaque ligne correspond à un pixel et chaque colonne à un patch valide du dictionnaire d
4. On construit une base d'apprentissage. Pour ce faire, on considère que $Y = l'ensemble des pixels valides de $p^i$$ et $X = Les pixels correspondant sur chaque patch$. Puis, on utilise l'algorithme du LASSO pour trouver la combinaison linéaire qui approxime le mieux le patch (apprentissage de w)
5. On construit la base de test constituée des pixels manquant de p^i et on fait un *predict* avec les poids trouvés précédemment ; cela nous permet de trouver la nouvelle valeur du pixel.

Notre modèle dispose de 2 paramètres : la hauteur h d'un patch et le step qui correspond au pas utilisé lors de la construction de notre dictionnaire.

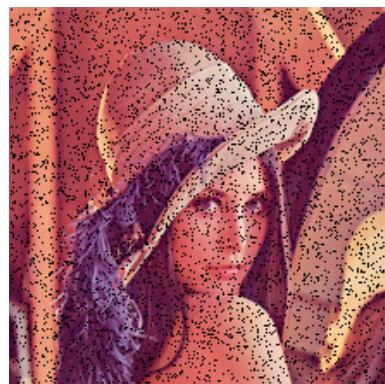
Intuitivement, si la hauteur du patch est trop grande, alors pour une image fortement bruitée le patch comportera très probablement un pixel mort donc on aura un dictionnaire d comportant peu de patchs. À l'inverse, si la hauteur est trop petite, on risque de ne pas avoir assez d'exemples du voisinage pour obtenir une prédiction fiable du patch avec le LASSO.

Un step petit permet de construire plus de patchs et donc d'avoir plus de valeurs dans notre dictionnaire d . C'est particulièrement utile quand on a un bruit important. De plus en prenant un step $< h$, on est certain d'avoir tous les pixels dans au moins un patch (même si celui-ci peut être bruité).

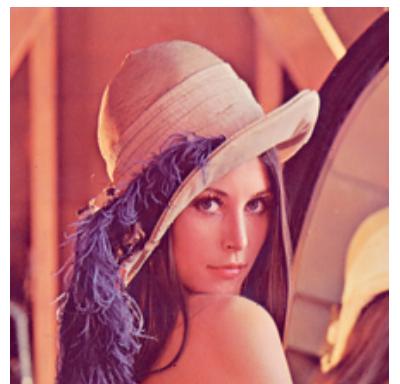
L'image utilisée pour les exemples suivants est de résolution 220 x 220, et on utilise une hauteur $h = 5$ et step = 10.



(a) original picture

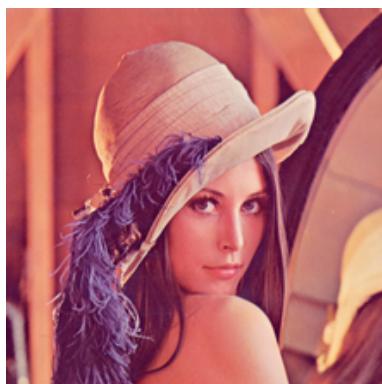


(b) noised picture



(c) corrected picture

FIGURE 2 – Inpainting de l'image lena avec 10 % de bruit



(a) original picture



(b) noised picture



(c) corrected picture

FIGURE 3 – Inpainting de l'image lena avec 30 % de bruit

Remarques : On observe que la qualité de l'image est bonne pour 10 % de bruit, et que la qualité de l'image se détériore en augmentant le bruit. En réalité, le format de notre image est relativement petit (220 x 220) afin d'accélérer les calculs. Par conséquent, si on augmente le bruit on risque de ne pas obtenir de patch non bruité pour former notre dictionnaire d'apprentissage. Donc

il est impossible d'apprendre dans cette situation. On peut en revanche résoudre ce problème en augmentant la résolution de l'image, mais cela augmente le temps de calcul.

Par conséquent nous travaillons désormais sur une image de résolution plus grande (512 x 512), et nous utilisons des patchs de taille 3 et un step de taille 6.



(a) original picture



(b) noised picture



(c) corrected picture

FIGURE 4 – Inpainting de l'image lena avec 30 % de bruit



(a) original picture



(b) noised picture

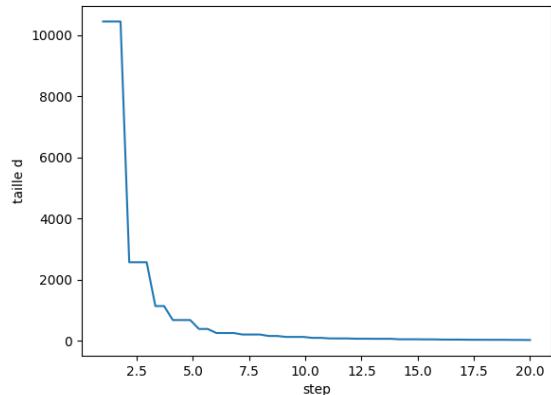


(c) corrected picture

FIGURE 5 – Inpainting de l'image lena avec 50 % de bruit

remarque : Dans ce cadre expérimentale nous avons un step de 6 et une hauteur de 3. Par conséquent nous n'avons pas tous les pixels de l'image qui sont couvert par un patch. Mais nous avons tout de même de très bon résultats. Cela illustre la force du lasso car la plupart des patchs sont redondant car non utilisés pour la correction et donc tant que nous avons un patch pour les différents motifs(mur, chapeau, ect) cela n'est pas gênant.

On cherche à évaluer la relation entre le step et le nombre de patchs dans d.



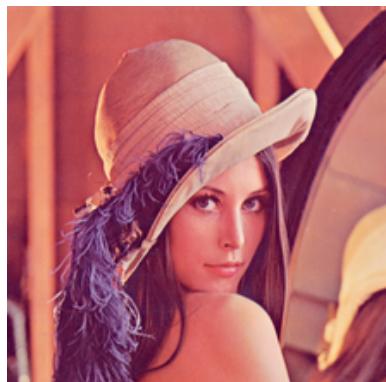
Nombre de patch dans d en fonction du step

remarque : Il n'est pas nécessaire d'avoir beaucoup de patchs dans le dictionnaire d car de toute façon le LASSO en prend seulement un petit nombre en compte

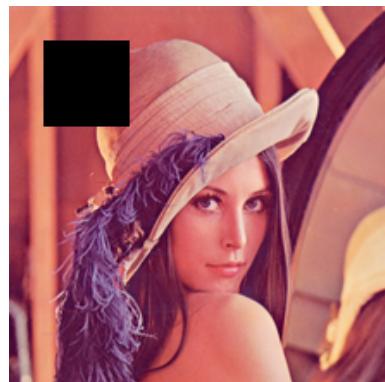
2.2 Reconstruction d'une partie manquante de l'image

On cherche maintenant à réparer une image dont une partie entière est manquante.

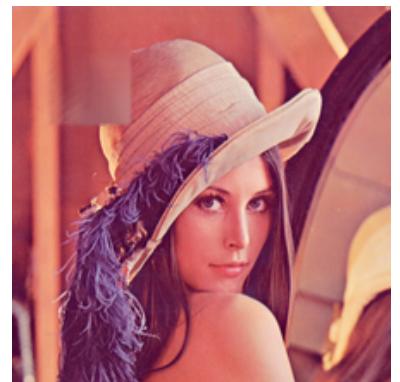
Dans un premier temps, nous étudions une heuristique naïve qui consiste à prendre en priorité les pixels qui ont le plus de voisin non bruités. L'hypothèse est que l'on dispose de plus d'information pour la correction, donc on aura une correction de meilleure qualité.



(a) original picture



(b) noised picture



(c) corrected picture

FIGURE 6 – Inpainting avec un trou de taille 50 x 50

Remarque : La bordure du carré est bien redessinée, mais le reste de la zone corrigée est flou. Le problème vient du fait qu'en cas d'*ex aequo* l'algorithme choisit en priorité les pixels les plus en haut à gauche de l'image. Ainsi quand on corrige un morceau du carré noir les pixels de référence sont ceux qui ont déjà été corrigés. Par conséquent, il y a une propagation du bruit de haut en bas et de gauche à droite, ce qui explique les mauvais résultats.

Pour éviter ce comportement, on reconstruit à chaque étape l'ensemble des pixels avec le plus de voisins non bruités, mais sans en ajouter de nouveaux avant de les avoir tous traités. C'est à dire qu'on corrige tous les ex *aequo* avant de mettre à jour les valeurs des pixels.



FIGURE 7 – Quelques étapes du nouvel algorithme



FIGURE 8 – Inpainting avec un trou de taille 50 x 50

Sur ce type d'algorithme, le pas et la taille des patchs utilisés jouent un rôle important dans la qualité du résultat. En effet, plus il y a de pixels exploitables autour d'un point, plus l'algorithme sera capable d'en faire une estimation précise.



FIGURE 9 – Test de différentes tailles de patchs

Dans le cas $h=5$, le mur de gauche est plus lumineux (clair) et propage une erreur dans le carré. C'est du au fait qu'il y a pas de patch couvrant une partie suffisante de la zone de transition. Donc quand la hauteur des patchs est petite, on remarque que les erreurs se propagent plus dans l'image, ce qui donne une impression de flou plus forte.

2.2.1 Approche en spirale

Nous avons implémenté une approche qui parcourt les bords du carré comme une spirale, mais elle donne des résultats moins bons que l'approche précédente.



De plus, on retrouve les problèmes de propagation de bruit évoqués précédemment.

2.3 Approche heuristique de confiance

On peut également suivre une approche d'heuristique à base de confiance. On considère que plus un patch contient de pixels originaux de l'image, plus il est fiable. L'algorithme fonctionne de la manière suivante :

1. Initialiser la matrice de confiance avec 1 sur tous les pixels valides et 0 sur les pixels morts
2. Classer les pixels morts par confiance (on prend la moyenne de confiance sur le patch autour du pixel) et prendre le plus fiable de valeur f_{min} .
3. Corriger le patch autour de ce pixel, et on attribue f_{min} a tous les pixels corrigés du patch
4. Réitérer tant qu'il reste des pixels morts.

Un des avantages de cet algorithme est qu'il va corriger d'abord les bords de la zone à corriger puis l'intérieur car les pixels intérieurs ont une valeur de confiance plus faible.



(c) noised picture

(d) corrected picture

FIGURE 10 – Approche basée sur la confiance

remarque : Le rendu graphique est assez satisfaisant mais on peut remarquer que la forme du chapeau n'est pas très bien restituée. Cela s'explique par le fait que notre algorithme ne prend pas en compte la structure graphique de l'image et en particulier les contours de l'objet à restituer. On pourrait donc améliorer cet algorithme en ayant une heuristique de forme et en calculant le meilleur pixel comme : $confiance(p_i) + forme(p_i)$