M1 Informatique – UE ML
Projet : Inpainting

Noms, prénoms:

Jouve Vincent
Cadiou Antoine

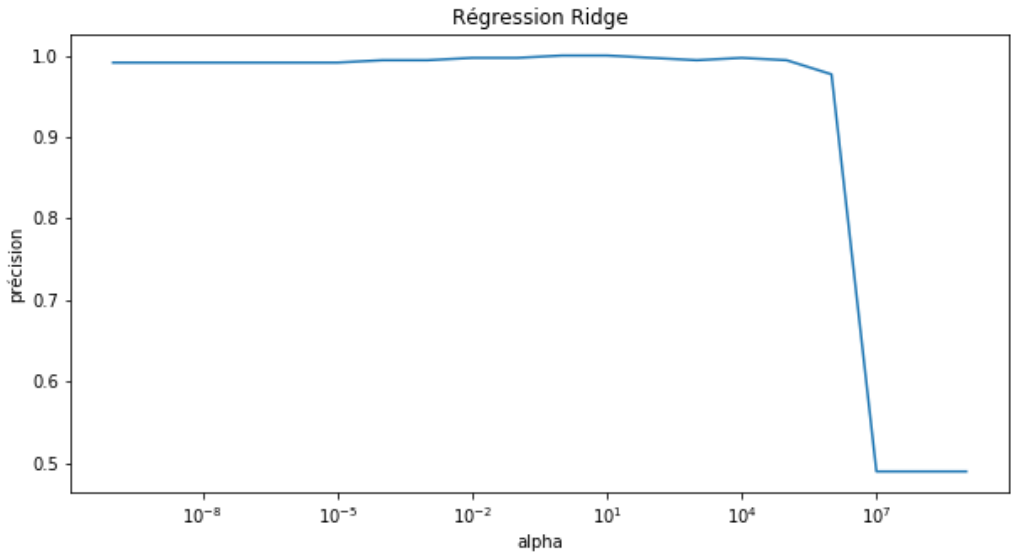
I. Préambule

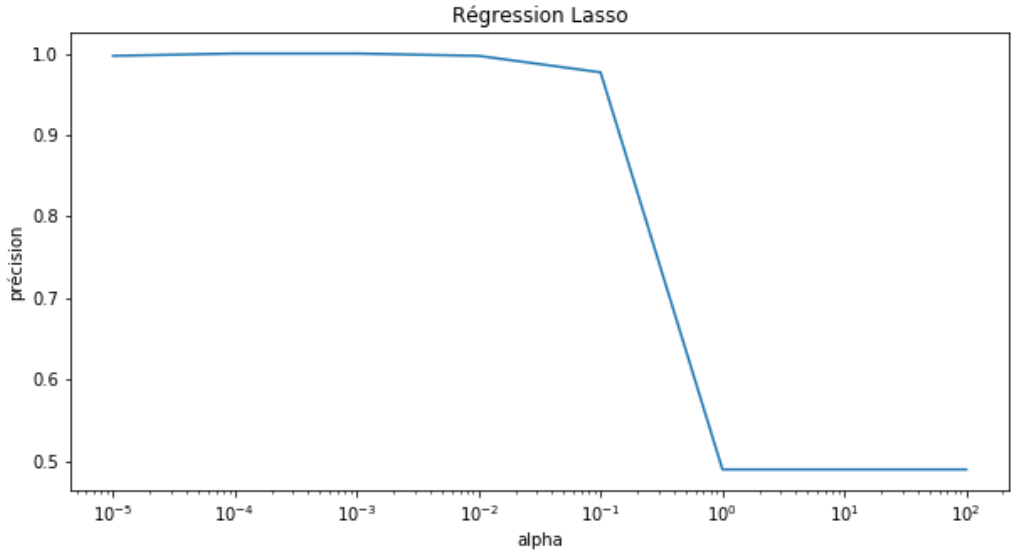
Dans un premier temps, sur les données USPS nous avons extrait les 6 et les 9 pour faire de la classification binaire, à partir de la méthode de régression linéaire et de la méthode plug-in. Ainsi nous pouvons comparer les différentes méthodes en fonction de leur précision.

Pour commencer, les 6 contre les 9 en prenant en compte seulement l'erreur mse sans régularisation, nous avons un taux de bonnes classifications de 99,1% sur la base de test.

Avec les régularisations ridge et lasso, on doit se soucier du paramètre alpha.

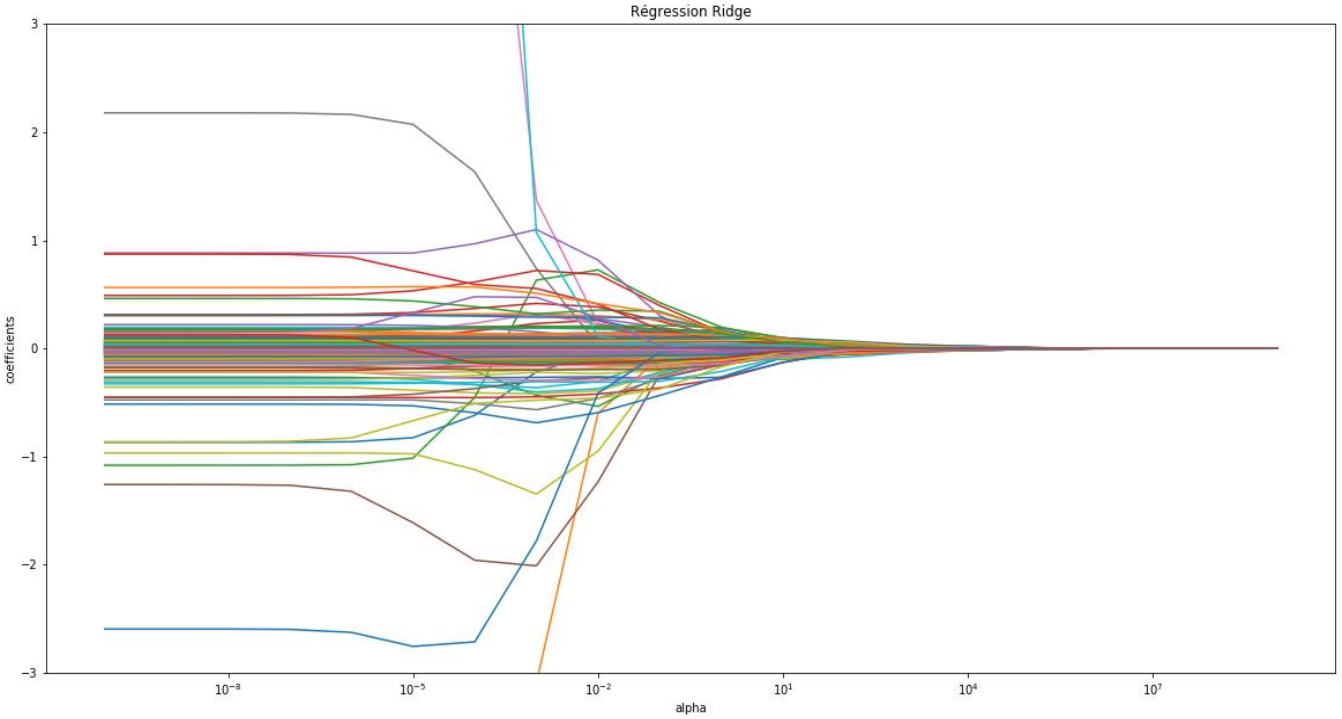
Toujours pour les 6 contre les 9, on obtient ces courbes de précision en fonction de alpha.

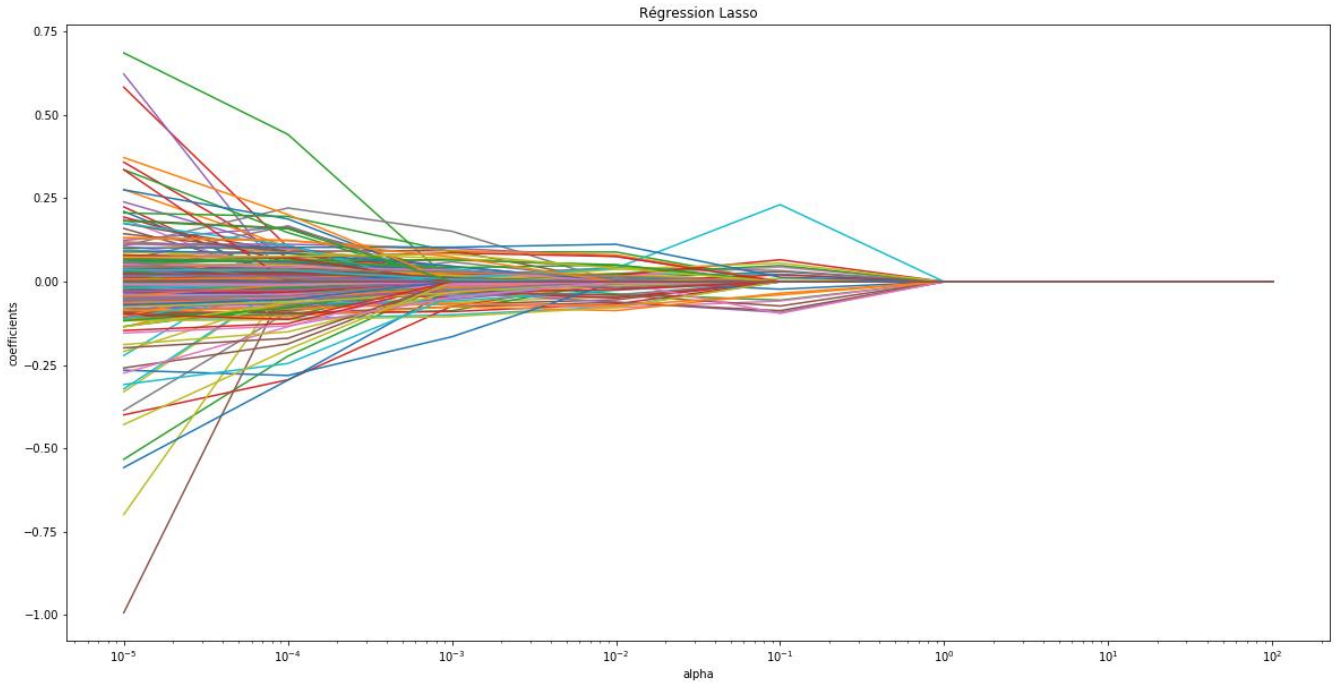




On observe que pour un alpha très petit la précision est égale à la précision avec seulement le mse. Ce qui est normal car si alpha est très petit seul le mse compte, dans la formule de ridge et lasso. Pour ridge on atteint même 100% de précision pour alpha=1 et pour lasso on atteint 100% pour alpha=10^{-4}. Donc on a une légère augmentation de la précision avec ridge et lasso, mais peut-être pas assez pour être significatif.

Ce qu'il est intéressant d'observer se sont les valeurs des composantes du vecteur de poids en fonction d'alpha.

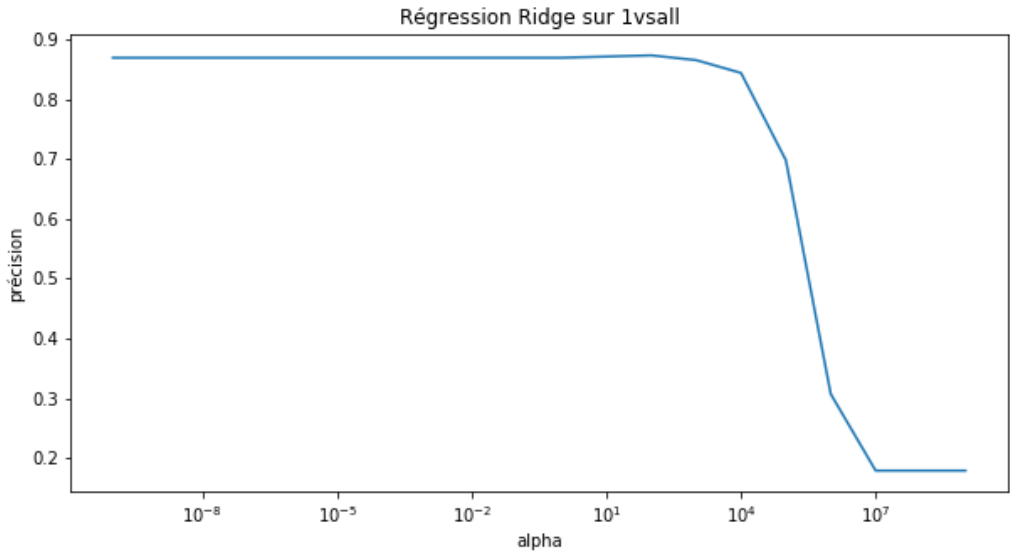


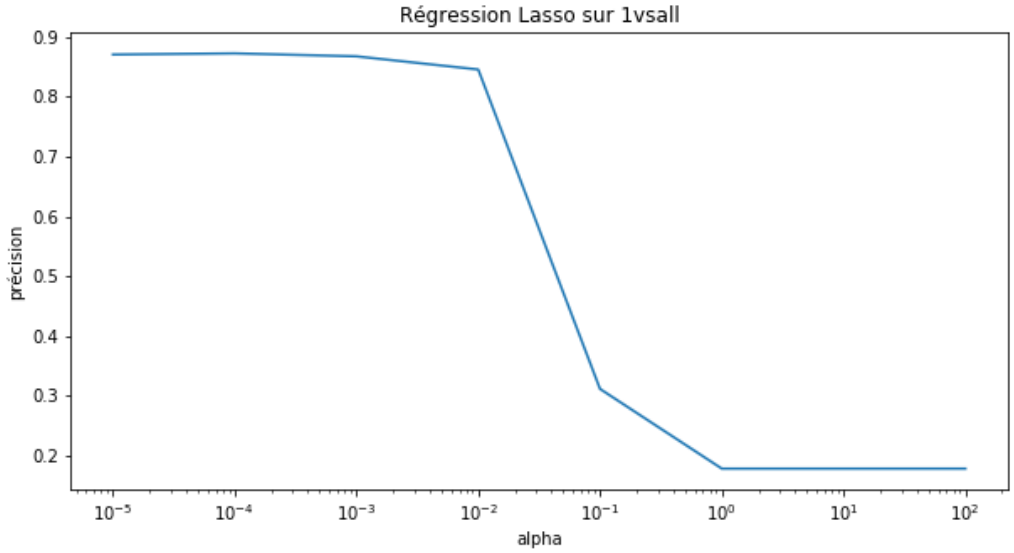


On voit que les coefficients finissent tous par converger en 0 quand alpha devient grand. Plus alpha est grand plus dans la formule qu'on doit minimiser c'est la norme l1 ou l2 qui prend le dessus sur le mse, et donc pour minimiser la formule, seul minimiser la norme est important, d'où le fait d'avoir tous les coefficients à zéro. On peut voir aussi que les coefficients ne tendent pas vers zéro à la même vitesse. La régularisation sert justement à faire un tri sur les coefficients en ne gardant que les plus important. Dernière remarque, on voit que pour la régularisation ridge, les coefficients tendent moins vite vers zéro qu'avec le lasso, car quand les coefficients du vecteur de poids sont petits (< 1) la norme l2 est plus petite que la norme l1.

Pour finir nous avons fait un classifier multi-class de type un contre tous, avec seulement mse nous avons 87% de précision.

Pour ridge et lasso nous obtenons ces courbes en fonction de alpha.





Pour ridge la meilleure précision obtenue est 87,3% pour alpha=100 et pour lasso 87,2% pour alpha=10^{-4}. L'écart est trop petit pour être significatif, nous aurions pu faire de la cross-validation pour s'en assurer. On peut en déduire que ni avec ridge ni avec lasso nous n'avons eu de meilleurs résultats, ils sont même beaucoup moins bons lorsque alpha est grand. On peut donc dire que lasso et ridge ne sont pas adaptés à ce type de problèmes.

II. Inpainting

Pour l'inpainting comme décrit dans le sujet on reconstruit une partie de l'image avec d'autres parties de l'image, ces parties de l'image sont appelé patch. On se doute que seulement certains doivent servir pour la reconstruction d'un patch bruité et non tous les patchs de l'image. C'est pour cela qu'adopter la régularisation lasso va être utile, c'est pour automatiquement garder seulement quelques patchs qui vont le plus servir à la reconstruction d'un patch bruité. Ce qui se traduit par avoir beaucoup de composantes du vecteur de poids à zéro.

Nous allons principalement travailler avec cette image.

Pour des raisons de performance de l'algorithme, nous allons travailler dans un premier temps sur une plus petite partie de l'image qui ne fera que 200*200 pixels.

Nous avons bruité cette image avec 5% de bruit pour commencer.

Pour reconstruire l'image nous procédons de la façon suivante, dans un premier temps nous calculons tous les patchs de taille h*h non bruités de l'image, ce qui constitue le dictionnaire. Puis pour chaque pixel manquant nous calculons le patch qui a pour centre ce pixel. On apprend avec le lasso puis on prédit tout le patch avec les coefficients appris et avec le dictionnaire de patchs. Enfin on prend le pixel au centre du patch prédit, à la place du pixel bruité. On pourrait également récuperer tous les patchs dans lesquels le pixel apparait et faire une moyenne, le développement est un peu plus compliqué, nous avons codé uniquement la première méthode.

Pour la taille des patchs: il faut des patchs pas trop grands sinon, dans un même patch, nous aurons trop d'informations différentes car il y aura plusieurs régions de l'image qui n'auront rien à voir ensemble et donc l'apprentissage sera impossible. Si le patchs et trop petit nous n'aurons pas assez de pixel non bruités sur lesquels s'appuyer lors de l'apprentissage et de même nous aurons une mauvaise prédiction. Il faut aussi prendre en compte le pourcentage de bruit de l'image: plus il y a de bruit moins nous aurons de patchs de taille moyenne, il faudra donc faire avec des patchs assez petits.

Ici dans l'exemple le dictionnaire est constituer de 531 patchs de 9*9 entierement connus.

Après avoir appliqué l'algorithme avec h=9 alpha=0.1, le nombre moyen de composantes non nulles du vecteur de poids est de 0.0, autrement dit alpha est trop grand et l'algorithme n'a rien appris. Cependant les pixels bruités sont devenus gris, nous pensons que c'est dû au biais uniquement.

Après avoir appliqué l'algorithme avec h=9, alpha=0.01, le nombre moyen de composantes non nulles du vecteur de poids est de 4.37, là il y a bien eu apprentissage cependant on peut constater quelques imperfections. Donc alpha est encore trop grand, on ne garde que 4.37 patchs en moyenne pour la prédiction d'un patch, ce n'est probablement pas assez.

Après avoir appliqué l'algorithme avec h=9, alpha=0.001, le nombre moyen de composantes non nulles du vecteur de poids est de 23.56, c'est mieux mais il y a encore des imperfections.

Après avoir appliqué l'algorithme avec h=9, alpha=0.0001, le nombre moyen de composantes non nulles du vecteur de poids est de 70.84, c'est bien mieux, donc c'est probablement le meilleur alpha.

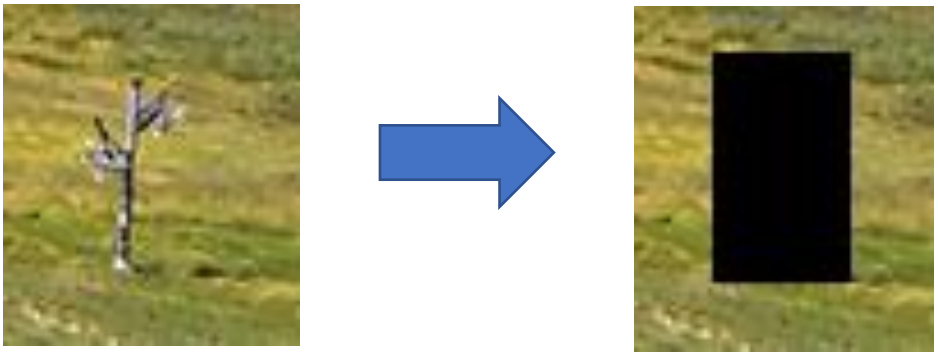
Pour le plaisir nous allons tester sur image beaucoup plus bruité comme par exemple à 40%.

Pour avoir un dictionnaire suffisamment conséquent nous avons exécuté l'algorithme avec h=3 et alpha=0.0001

Après une itération il reste quelque pixel bruité, ce sont les pixels dont le patch était complètement bruité et donc sur lesquels on n'a rien pu apprendre. On a réexécuté une fois l'algorithme pour obtenir ce résultat. On itère jusqu'à ce qu'il n'y ait plus de pixel bruité.

Maintenant essayons de faire disparaitre ce qui ressemble à un panneau.

Pour cela on va zoomer sur le panneau et le recouvrir d'un rectangle de pixels bruités. Le fait de zoomer est un moyen simple de ne s'occuper que des patchs voisins, toujours pour la performance de l'algorithme.



Ici plusieurs questions se posent. Quelles valeurs pour les paramètres h et alpha ? Dans quel ordre reconstruire l'image ? Est-ce que l'ordre a de l'importance ? Faut-il recalculer les patchs entièrement connus au fur et à mesure que l'on reconstruit l'image ?

On va garder le même alpha que l'on a trouvé auparavant càd 0.0001. Pour h on va faire pareil que précédemment on veut qu'il soit assez grand tout en ayant un nombre de patchs suffisant dans le dictionnaire. Concernant l'ordre de reconstruction, on peut faire depuis les bords au fur et à mesure en itérant pour avoir un rectangle de plus en plus petit, si on ne fait pas ça les patchs qui seront plus au centre n'auront que peu ou pas de pixels non bruités et il sera difficile ou impossible d'apprendre dessus et de faire une bonne prédiction. Donc l'ordre a forcément un impact sur l'image reconstruite. Pour chaque itération on a choisi de corriger seulement les pixels les plus au bord et itérer jusqu'à ce qu'il n'y ait plus de pixel bruité. Et pour qu'il y ait une meilleure cohérence de l'image entre chaque itération on recalcule les patchs qui forment le dictionnaire de patchs non bruités.

Premier essai avec h=5 le résultat n'est pas vraiment satisfaisant mais comme nous l'avons dit il faut un h plus grand.

On a donc pris h=17 et l'on obtient ce résultat qui est bien plus satisfaisant.

Si on remplace la partie de l'image de base avec le panneau par ce morceau sans panneau on obtient cette image. Ni vu ni connu le panneau !