Rapport projet ML

Master 1 ANDROIDE : Jérôme Arjonilla, Arthur Esquerre-Pourtère

2020

Préambule : régression linéaire, régression ridge et LASSO

Afin de comparer les trois algorithmes de classifications, nous avons comparé les régressions pour chaque combinaisons de classes possibles. Plus précisement, nous avons effectué une stratégie dite "One-vs-One", c'est à dire prédire une classe de chiffre face à une autre classe. Dans notre cas, il y a en tout 45 comparaisons (n(n-1)/2).

De plus, pour chaque comparaison, nous avons lancé 10 runs indépendants, afin d'obtenir des résultats fiables. Et enfin, pour les régressions de types Lasso et Ridge, nous avons effectué un Grid Search pour le paramètre alpha avec comme valeur possible : [10000,1000,100,10,1,0.1,0.01,0.001]

Grâce à notre protocole, nous obtenons les résultats suivants :

- Linear Regression :
  - Probabilité de réussite sur les données d'apprentissage : 0.9972
  - Probabilité de réussite sur les données d'apprentissage : 0.973
  - Norme des poids : 1.028e12
  - Nombre de poids valant 0 : 0.377
- Ridge Regression :
  - Probabilité de réussite sur les données d'apprentissage : 0.9945
  - Probabilité de réussite sur les données d'apprentissage : 0.980
  - Norme des poids : 10.43
  - Nombre de poids valant 0 : 1.55
- Lasso Regression :
  - Probabilité de réussite sur les données d'apprentissage : 0.9943
  - Probabilité de réussite sur les données d'apprentissage : 0.978
  - Norme des poids : 7.58
  - Nombre de poids valant 0 : 150

On peut voir les poids attribués à chaque pixel sur un graphe, on observe les poids suivants pour le cas 3 vs 8 :

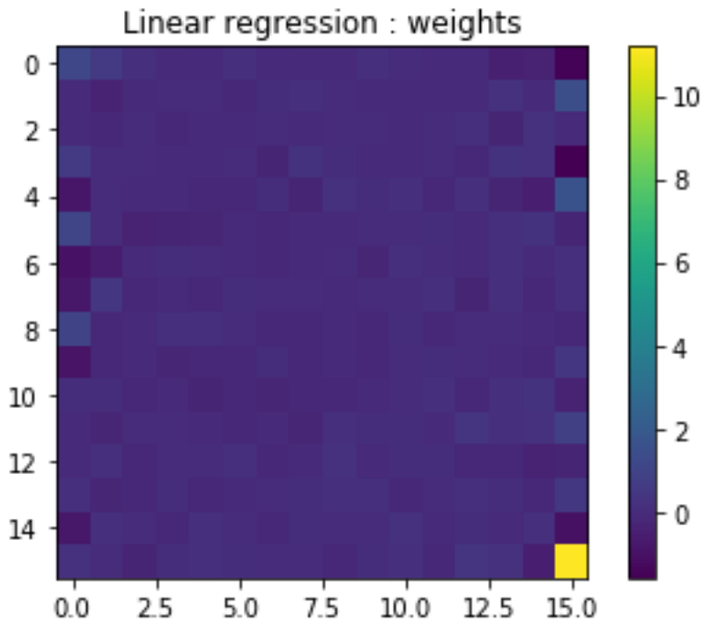

Figure 1: Poids pour la régression linéaire

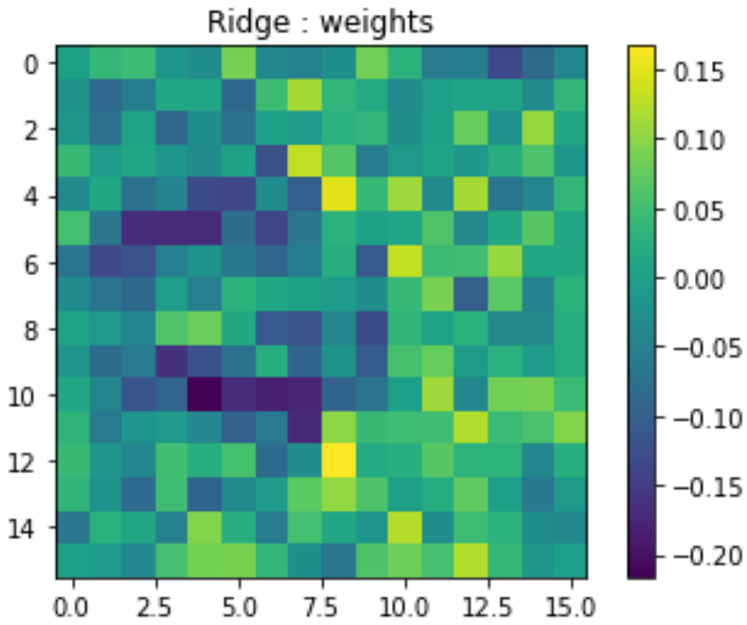

Figure 2: Poids pour la régression Ridge

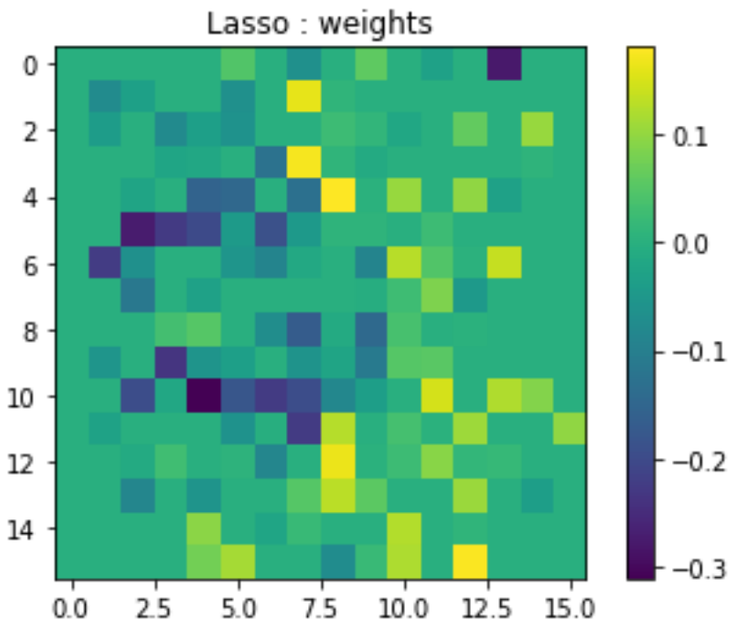

Figure 3: Poids pour la régression Lasso

Voici un autre exemple, pour le cas 0 vs 2 on observe notamment des valeurs extrèmes pour les poids de la régression linéaire :

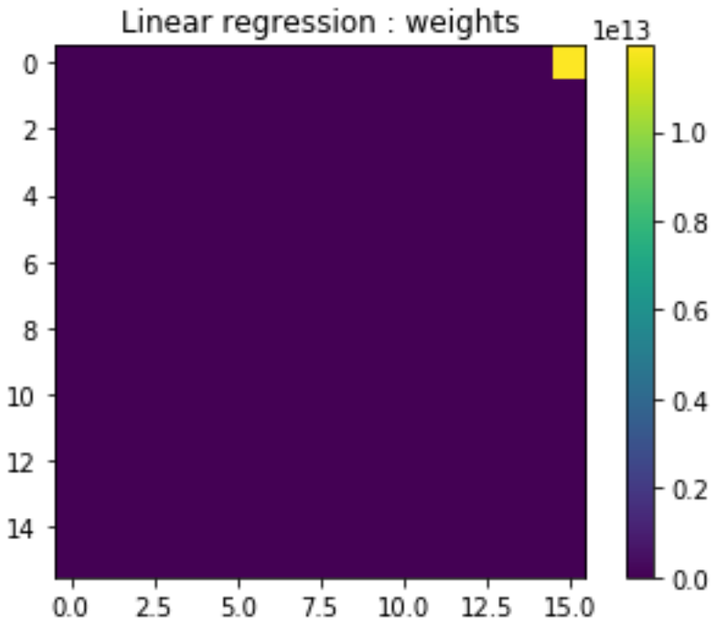

Figure 4: Poids pour la régression linéaire

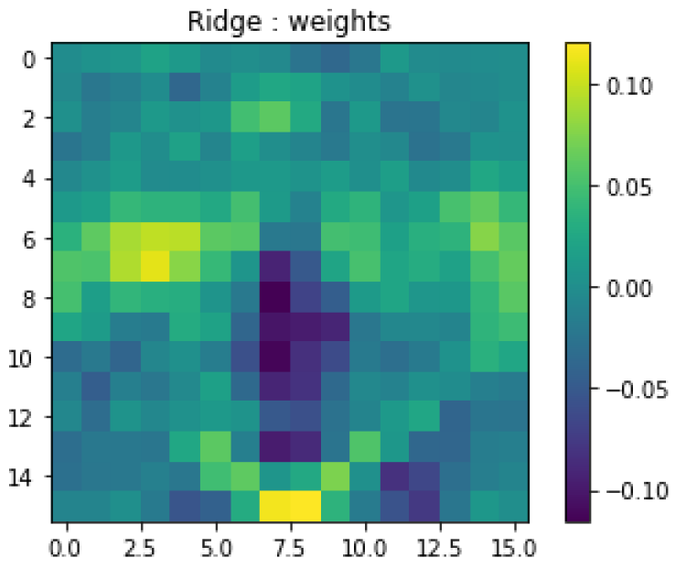

Figure 5: Poids pour la régression Ridge

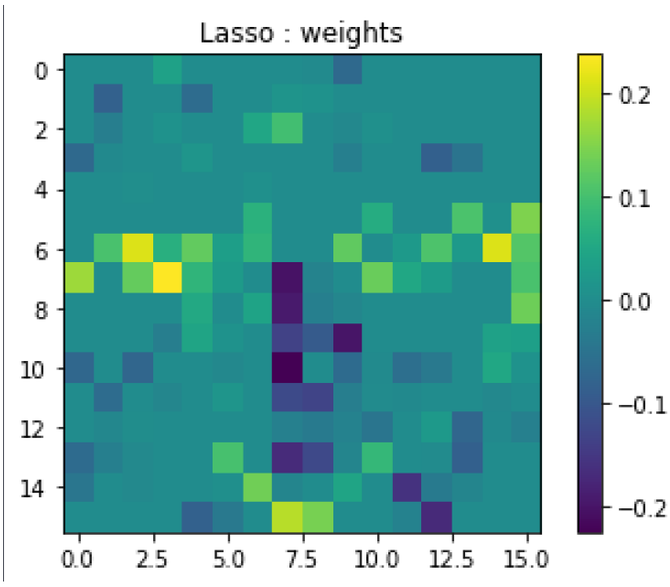

Figure 6: Poids pour la régression Lasso

Grâce à toute ces informations, on peut en déduire, que les 3 modèles fonctionnent très bien pour de la classification de type binaire, en effet, les 3 modéles ont une probabilité de succès d'environ 98%. En revanche, ce qu'il les différencie fortement c'est la valeur des poids.

En effet, pour la régression linéaire, la somme des valeurs absolues des poids possède une valeur très importante de 1.028e12, alors que les 2 autres régression ont des valeurs beaucoup plus petites, proches de 10. Cette somme est même inférieur dans le cas de la régression lasso. L'effet de la régularisation a permis de contrôler fortement la valeurs des poids. De plus, on peut observer que le nombre de poids valant 0 est beaucoup plus important dans le cas de la régression Lasso.

On peut en déduire, que la régression régularisé va favoriser les valeurs de poids faibles, et dans le cas de la régression Lasso, la régularisation va favoriser les poids de valeurs 0.

On peut supposer que dans le cas de la régression linéaire, notre modèle donne parfois des poids extremement grands à des pixels qui n'influent pas sur la prédiction car dans le jeu d'entrainement, ils valent toujours 0. Dans le cas de la régression ridge, on évite ce problème, et seuls les pixels apportants de l'information ont des poids dont la valeur absolue est élevée. Dans le cas du Lasso, l'algorithme donne surtout des poids aux pixels les plus significatifs, et donne des valeurs proches voir égales à zéro aux autres.

LASSO et Inpainting

Pour notre projet, nous avons utilisé plusieurs images, notamment l'image ci dessous.

Figure 7: Image de Base

Nous avons créer plusieurs fonctions, notamment une fonction permettant de bruiter une image, on peut le voir ci-dessous :

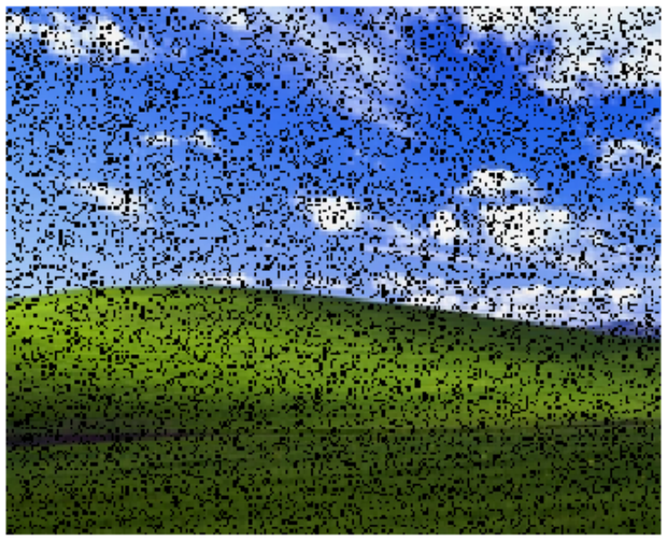

Figure 8: Image avec 30% de bruit

On peut choisir de façon plus précise de seulement bruiter un zone :

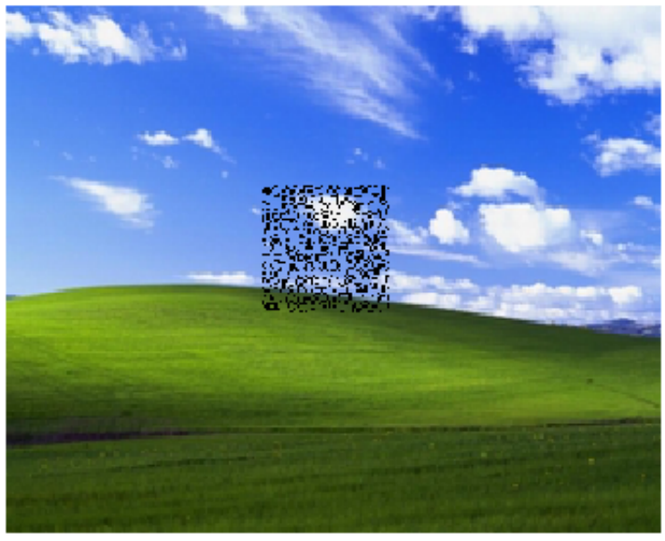

Figure 9: Image avec un rectangle bruité

Il est aussi possible d'enlever une partie de notre image, comme ci dessous :

Figure 10: Image avec un rectangle en moins

Reconstruction d'une image bruitée

Nous avons crée une fonction permettant de déterminer les pixels manquants d'un patch, cette fonction est "approxime_Patch" dans notre code.

Elle prend un patch en entrée et le dictionnaires de bons patch, et elle retourne le patch en ayant approximé les pixels manquants. Pour ce faire notre fonction va, dans un premier temps, déterminer l'hyperparamètre alpha et s'entrainer sur les pixels non manquants, et dans un second temps, notre fonction va prédire les pixels manquants.

On peut voir le résultat dans les image ci dessous :

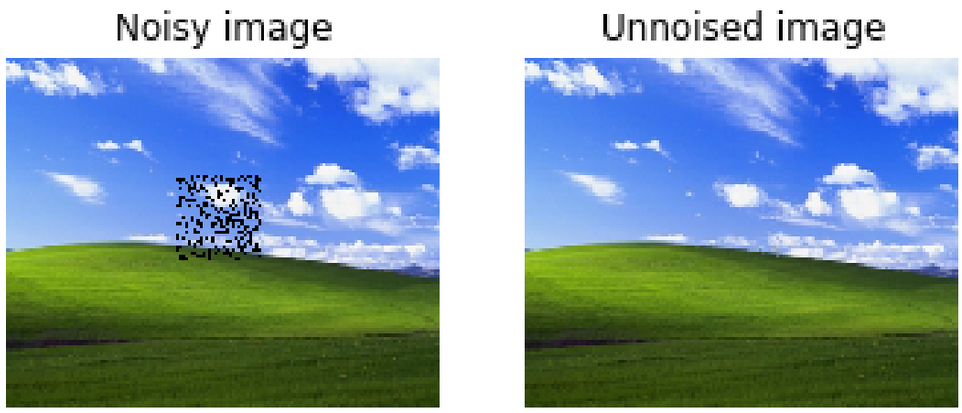

Figure 11: Reconstruction de l'image

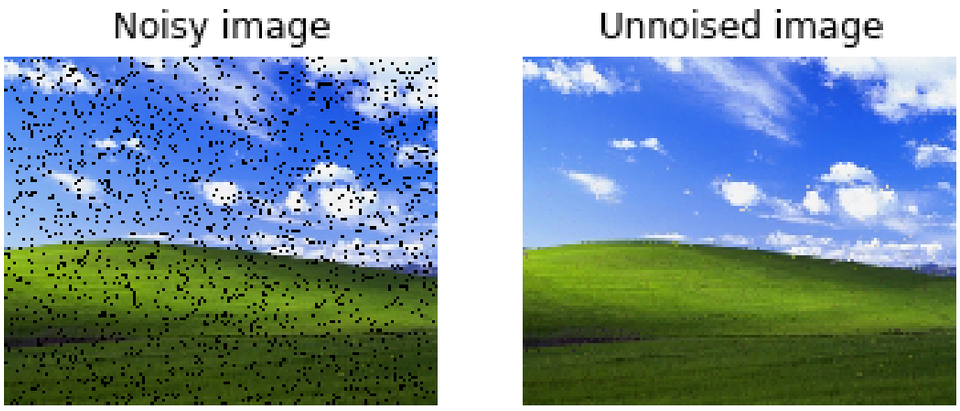

Figure 12: Reconstruction de l'image

L'algorithme fonctionne mieux dans le cas où une grande partie de l'image n'est pas bruitée car dans ce cas l'algorithme a plus de patch non bruités qu'il peut utiliser pour approximer les pixels manquants.

De plus afin de gérer le fait que la largeur ou la hauteur de l'image n'est pas forcément divisible par la taille du patch, ce qui pourrait avoir pour effet que l'on ne prédit pas les pixels manquants sur les bords de l'image, on rajoute des pixels manquants en bas et sur la droite de l'image afin d'avoir des patchs complets.

Reconstruction d'une image dont une zone est manquante

Dans la situation où tout une partie de l'image est manquante, l'ordre de remplissage est très important, en effet on prédit les pixels manquants en entrainant notre algorithme lasso sur les pixels environnants. Dans le cas où tout une partie l'image est manquante, on commence donc par prédire les pixels sur les bords de la zone manquante en se basant sur les valeurs des pixels non manquants. Mais lorsqu'on se rapproche du centre de la zone manquante, les pixels environnants sont tous des pixels manquants. On doit donc utiliser les valeurs prédites des pixels manquants pour entrainer notre algorithme. Cela rend donc bien sûr l'ordre des prédictions très important.

Nous avons utilisé une première heuristique, naïve, pour laquelle on prédit en priorité les patchs pour lesquels on a le moins de pixels ayant une valeur non prédite, en donnant une importance égale aux pixels déjà présents au départ et aux pixels dont la valeur a été prédite, on obtient ce résultat, en utilisant des patch de taille 13x13 :

Figure 13: Reconstruction de l'image, en utilisant la méthode naïve

Nous avons aussi utilisé une deuxième heuristique, plus maligne, chaque pixel a un score de confiance. Au début les pixels manquants valent 0, les pixels non manquants valent 1. Lorsqu'on prédit les pixels manquants d'un patch, on met à jour la valeur de confiance de ces pixels, en leur donnant une valeur correspondant à la moyenne des score des pixels du patch. De plus, pour choisir le patch à prédire, on prend le patch pour lequel la moyenne des scores de confiance des pixels contenus dans ce patch est la plus élevée.

Ainsi, l'algorithme prédit les pixels des bords de la zone manquante vers le centre, en ce concentrant toujours sur les pixels qu'il peut prédire avec la plus grande certitude. Ce qui, en théorie, devrait donner de meilleur résultats.

Figure 14: Reconstruction de l'image, en utilisant la méthode utilisant une valeur de confiance

On observe, en effet, que notre deuxième heuristique donne de meilleurs résultats, en particulier sur les bords de la zone manquante, au centre de de la zone manquante, le résultat est aussi légérement meilleur. Toutefois, ces résultats dépendent beaucoup de l'image utilisée, de la taille de la zone manquante et de la taille des patch. Généralement la deuxième heuristique semble être surtout plus performante que la première lorsque la taille de la zone manquante est grande.

Un autre point intéréssant lorsqu'on utilise les scores de confiance est qu'on peut observer ces scores à la fin de l'algorithme, comme on peut le voir ci-dessous :

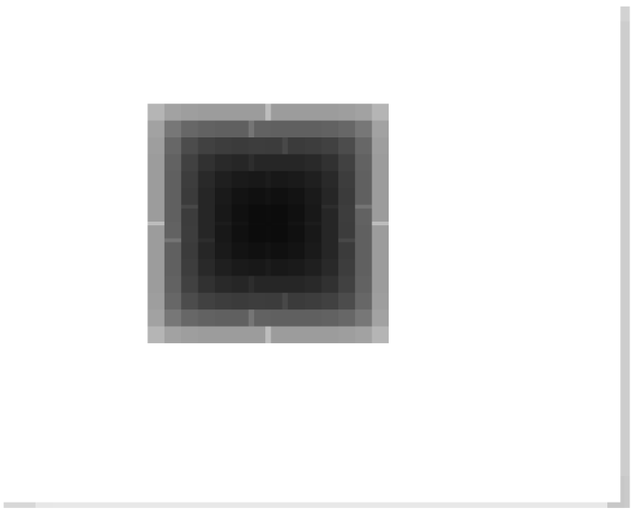

Figure 15: Score de confiance des pixels de l'image reconstituée, un score de confiance de 1 est représenté par un pixel blanc, un score de confiance plus faible est plus grisé

On observe en effet que notre algorithme est plus confiant pour les pixels prédits proches des bords de la zone manquante que pour ceux prédits qui sont proches du centre.