ML : Rapport du Projet

Ivan Kachaikin

1 Introduction

Ce rapport présente le travail effectué dans le cadre du projet de l'UE ML.

2 Régression

Tout d'abord, avant de passer vers la question **Q 1.1**, où l'on doit mettre en place un protocole de comparaison des algorithmes différents de régression, l'on a besoin d'avoir ces algorithmes implémenté. Bien que l'on ait eu une possibilité de se servir du module `scikit-learn`, l'on a décidé de ne pas l'utiliser pour ces modèles de régression. Au lieu de cela, l'on a adopté réécrivant certains bouts du code une classe `Lineaire` du TME 3 pour obtenir plus de flexibilité en réalisation de fonctionnalités diverses. À la fin, l'on a reçu une nouvelle classe `RegressionClassifier` qui représente un classificateur par la régression (*plug-in*). Une telle classe permet à l'utilisateur d'entraîner des modèles différents, plus précisément, *régression simple* (sans aucune régularisation), *régression ridge* (avec la régularisation de Tikhonov), *LASSO* (avec la régularisation L_1) ou bien n'importe quel modèle "personnalisé" défini extérieurement. Par ailleurs, le type de descente de gradient est également modifiable : l'on peut utiliser *batch*, *SGD*, *mini-batch* ou un autre l'algorithme d'optimisation multidimensionnelle fourni par l'utilisateur qui a le même modèle. Cette classe concernée se trouve dans le module `RegressionClassifier.py` ainsi que certaines méthodes auxiliaires et utiles.

Q 1.1. Pour le protocole de comparaison des modèles différents l'on propose un critère suivant. Étant donné un modèle de classification M entraîné sur les données $X \in \mathbb{R}^{n \times d}$ et $y \in \mathbb{R}^n$ l'on défini une fonction $S : M, X, y \longmapsto s$ de manière suivante :

$$S(M,X,y) = \frac{1}{2}\Big(w_{train} \cdot s_{train}(M,X,y) + w_{test} \cdot s_{test}(M,X,y) - w_{diff} \cdot |\, s_{train}(M,X,y) - s_{test}(M,X,y)\,|\Big) \tag{1}$$

où $s_{train}(M,X,y)$, $s_{test}(M,X,y)$ sont des scores de bonne classification en apprentissage et en test respectivement tandis que $w_{train}, w_{test}, w_{diff} \geq 0$ sont des poids d'"importance" de chaque critère considéré. En outre, l'on considère par défaut que $w_{train} = w_{test} = w_{diff} = 1$, et alors $S(M,X,y) \in [0,1]$.

Un tel protocole est inspiré de l'hypothèse assez essentielle : pour quel que soit classificateur l'on voudrait maximiser son score obtenu en apprentissage, en même temps, l'on voudrait également maximiser son score obtenu en test. De plus, l'on voudrait aussi minimiser une différence entre ces deux valeurs pour éviter le sous-apprentissage ainsi que le sur-apprentissage. Par conséquent, l'on a en général le problème d'optimisation multicritère. Ainsi, la métrique proposée ne donne que la somme pondérée des critères que l'on cherche à optimiser. Alors, selon ce protocole proposé ici, étant donné les données X et les étiquettes y les meilleurs modèles de classification correspondent aux valeurs de $S(M,X,y)$ les plus élevées. Et réciproquement, moins est le $S(M,X,y)$, pire est le modèle.

Afin de comparer les trois algorithmes de régression (*linéaire*, *ridge* et *LASSO*), l'on a lancé l'apprentissage de ces trois modèles en données de `USPS` (soit X_{usps} et y_{usps}) pour savoir séparer deux chiffres entre elles (6 vs. 9). Le script exécutant cet entraînement est fourni dans le fichier

`classification_test.py`. Après l'avoir exécuté utilisant *SGD* pour la descente de gradient, l'on a obtenu des résultats suivants :

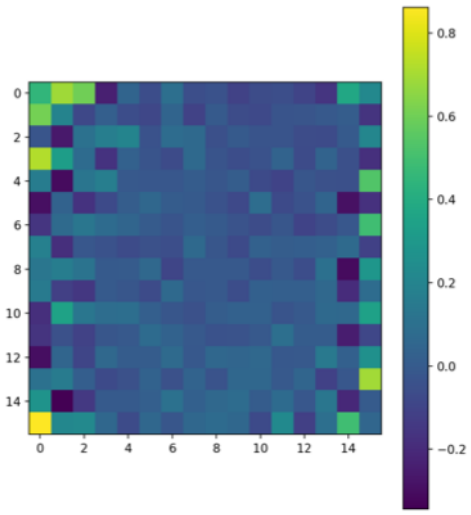
- $S(\text{linéaire}, X_{\text{usps}}, y_{\text{usps}}) = 0.991$, avec $s_{train}(\text{linéaire}, X_{\text{usps}}, y_{\text{usps}}) = 0.998$, $s_{test}(\text{linéaire}, X_{\text{usps}}, y_{\text{usps}}) = 0.991$ et $\left\|\mathbf{w}_{fin}\right\|_2 = 2.653$;
- $S(\text{ridge}, X_{\text{usps}}, y_{\text{usps}}) = 0.994$, avec $s_{train}(\text{ridge}, X_{\text{usps}}, y_{\text{usps}}) = 1.0$, $s_{test}(\text{ridge}, X_{\text{usps}}, y_{\text{usps}}) = 0.994$ et $\left\|\mathbf{w}_{fin}\right\|_2 = 0.209$;
- $S(\text{LASSO}, X_{\text{usps}}, y_{\text{usps}}) = 0.994$, avec $s_{train}(\text{LASSO}, X_{\text{usps}}, y_{\text{usps}}) = 0.999$, $s_{test}(\text{LASSO}, X_{\text{usps}}, y_{\text{usps}}) = 0.994$ et $\left\|\mathbf{w}_{fin}\right\|_2 = 0.413$.

En plus, dans la Figure 1 l'on fournit les dernières matrices des poids obtenues par les approches différentes.

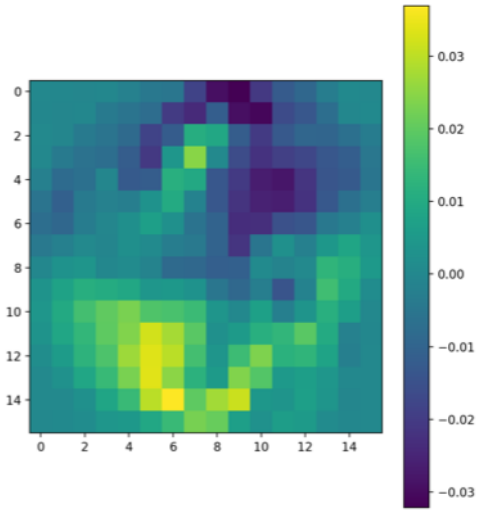
Analysant tous ces résultats, l'on constate que la régularisation améliore tous les deux scores de bonne classification, et alors l'on obtient une valeur plus élevée de $S(M, X_{\text{usps}}, y_{\text{usps}})$ pour les modèles régularisé. Par ailleurs, cela produit telles matrices finales de poids dont les normes euclidiennes (et alors toutes les autres) ont des plus petites valeurs, et qui contient beaucoup moins de bruit. En outre, une matrice correspondante à *LASSO* a le plus petit nombre d'éléments non nulles alors que ce modèle reste efficace selon le critère établi. Cela veut dire que *LASSO* ne prend en considération que les composants les plus principales de X ignorant ceux qui n'influence pas beaucoup la différence entre chiffres prises en compte. De manière générale, cette propriété de *LASSO* peut être très utile car l'on pourrait exclure de la considération certaines caractéristiques de X ce qui nous permettrait de réduire la taille du problème, et enfin, de diminuer le temps d'exécution de l'algorithme. Même si pour le problème de classification observé il n'y avait pas de difficultés temporelles, pour les bases de données de très grande taille cela peut être une qualité cruciale.

3 LASSO et Inpainting

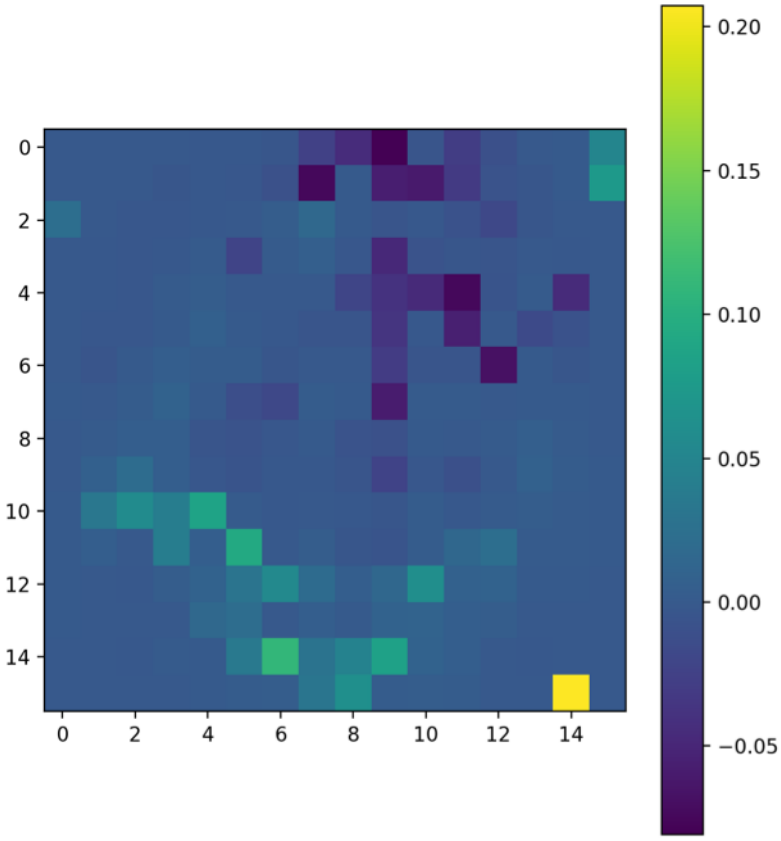
Cette partie n'a pas été finie. Cependant, l'on a réalisé une classe `Image` qui donne une réponse à la question **Q 2.1**. Sa source se trouve dans le fichier `Inpainting.py`.



(a)



(b)



(c)

FIGURE 1 – Dernières matrices de poids obtenues par les algorithmes différents : 1(a) – Régression *linéaire*, 1(b) – Régression *ridge*, 1(c) – *LASSO*