# Régularisation et optimisation des modèles

Michel Deudon, Maximilien Servajean

Université Paul Valéry, Département MIAp
prenom.nom@univ-montp3.fr

**Abstract.** De nombreux problèmes de Machine Learning (ML) peuvent être considérés comme des problèmes d'optimisation (en santé, en logistique, dans les moteurs de recommendation, etc). Ce cours est une introduction à l'optimisation en ML. L'objectif est d'apprendre quelques algorithmes d'optimisation et comment les appliquer en pratique à la régression linéaire et logistique. Cette introduction ne se veut pas exhaustive. Nous n'étudierons pas l'optimisation combinatoire, évolutive, les algorithmes de recherche opérationnel, les méthodes heuristiques ou l'optimisation de boites noires.

**Keywords:** Optimisation, régularisation, régréssion linéaire, régression logistique, descente de gradient, coordinate descent, descente de gradient stochastique, algorithme de Newton, Ridge regression, Lasso

## 1 Définitions et notations

### 1.1 Rappel sur les fonctions

Définitions, propritiés d'une fonction continue, dérivable, convexe, k-Lipschitzienne.

### 1.2 Dérivés

**Gradient** Soit $\theta \in \mathbb{R}^d$ un vecteur d-dimensionnel, appelé paramètres, et $L(\theta)$ une fonction scalaire de $\mathbb{R}^d$ dans $\mathbb{R}$, appelée fonction de coût. Le gradient de $L$ en $\theta$ est le vecteur $\nabla L(\theta) = [\frac{\partial L(\theta)}{\partial \theta_1}, \frac{\partial L(\theta)}{\partial \theta_2}, \dots, \frac{\partial L(\theta)}{\partial \theta_d}] \in \mathbb{R}^d$

**Hessien** La matrice hessienne de $L$ par rapport à $\theta$, notée $\nabla^2 L(\theta)$ ou simplement $H$, est la matrice des dérivées partielles

$$\nabla^2 L(\theta) = \begin{bmatrix} \frac{\partial^2 L}{\partial \theta_1^2} & \cdots & \frac{\partial^2 L}{\partial \theta_1 \partial \theta_d} \\ \vdots & & \vdots \\ \frac{\partial^2 L}{\partial \theta_d \partial \theta_1} & \cdots & \frac{\partial^2 L}{\partial \theta_d^2} \end{bmatrix} \in \mathbb{R}^{dd}$$

## 2 Apprentissage en ML

Soit n points $(x_i, y_i)_{i=1}^n$. On cherche à trouver une fonction $f$, parmi un ensemble de fonctions $C$, qui minimise une certaine fonction de coût $L$ sur un jeu de données $(x_i, y_i)_{i=1}^n$, par exemple l'erreur absolue ou quadratique entre les prédictions $f(x_i)$ et les vrai valeurs $y_i$ dans le cas d'un apprentissage supervisé. Dans ce cours nous nous interesserons à des modèles dits paramétriques, c'est à dire avec des paramètres $\theta$, comme illustré dans la figure 1 ci-dessous. Pour des polynomes du 2e degré, $\theta \in \mathbb{R}^3$ (paramètres a,b,c) et pour des polynomes du 3e degré (courbe bleue), $\theta \in \mathbb{R}^4$. Dans ce cas, nous noterons plus simplement $L(\theta)$ la fonction de coût associée au modèle $f_\theta$ et au jeu de données $(x_i, y_i)_{i=1}^n$.

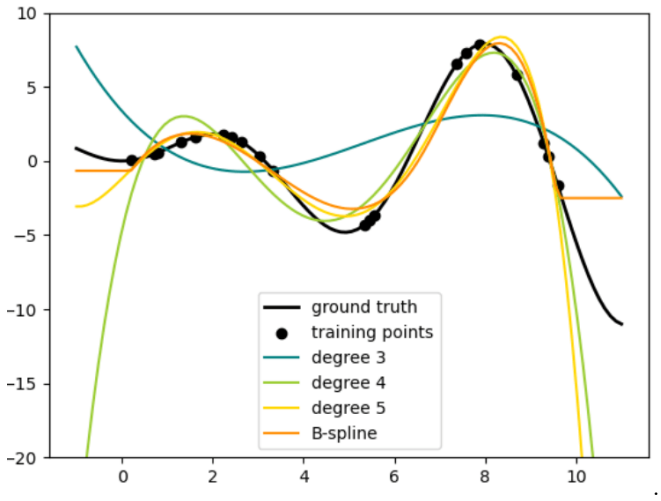


**Fig. 1.** Illustration du problème général d'apprentissage en ML, ici avec des polynômes. La courbe noir (ground truth) correspond à la fonction $f^*$ qu'on cherche à approcher pour prédire sa valeur en de nouveaux points (l'inconnue). C'est ce que nous souhaitons apprendre et généraliser à partir d'exemples donnés sous la forme de points de coordonnées $(x_i, y_i)$. Notons que tous les modèles ici sont faux. Crédit image [11].

## 3 Optimisation en ML

Optimiser un modèle en ML revient souvent à trouver le minimum d'une fonction de coût $L(\theta)$ comme illustré dans la figure 2. Les méthodes d'optimisations dépendent de la fonction de coût et de la nature du problème. Lorsque $L(\theta)$ est dérivable, on peut chercher un minimum local qui annule le gradient de la fonction de coût $\nabla L(\theta)$ et on peut vérifier que ce minimum est un minimum global si $L(\theta)$ est convexe (ex: régression linéaire).

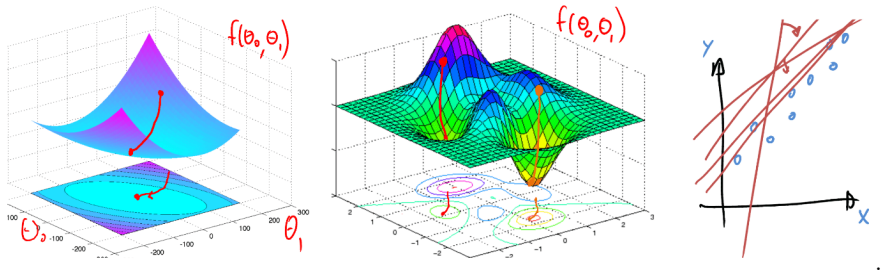


**Fig. 2.** Illustration du problème général d'optimisation avec une fonction de coût $L(\theta)$ continue, dérivable, convexe (à gauche) et non convexe (au milieu). À chaque valeur de $\theta$, correspond un modèle $f_\theta$(image de droite), qui permet plus ou moins bien d'expliquer les observations. Crédit image [12].

### 3.1 Hypothèse iid

Dans ce cours, nous ferons l'hypothèse que les points $(x_i, y_i)_{i=1}^n$ sont indépendants et identiquement distribués (iid). Dans ce cas, nous optimisons des fonctions de coût de la forme

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(\theta, x_i, y_i) \tag{1}$$

Lorsque la fonction est dérivable, le gradient correspondant s'écrit

$$\nabla L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla l(\theta, x_i, y_i) \tag{2}$$

### 3.2 Algorithme de descente de gradient

Un des algorithmes d'optimisation le plus simple est la descente de gradient et peut s'écrire sous la forme:

$$\theta_{k+1} = \theta_k - \eta_k \nabla L(\theta_k) \tag{3}$$

où k indexe les itérations de l'algorithme, $\nabla L(\theta_k)$ est le gradient à l'itération k et $\eta_k > 0$ la taille des pas (learning rate ou step size en anglais).

**Variante** L'algorithme Coordinate descent [10] minimise $\theta$ une coordonnée à la fois, en gardant les autres fixées. Cette variante est utilisée dans de nombreux algorithmes dont scikit-learn Lasso et Elastic-net [11]. Le choix des coordonnés est aléatoires (permutation uniforme) et l'algorithme a la même complexité que la descente de gradient $O(nd)$ pour itérer sur l'ensemble des coordonnées.

### 3.3 Algorithme de Newton

L'algorithme de Newton [1] permet d'adapter la taille des pas d'apprentissage à chaque itération lorsque L est deux fois dérivable. Une itération peut s'écrire

$$\theta_{k+1} = \theta_k - H_k^{-1} \nabla L(\theta_k) \tag{4}$$

où $H_k$ est le Hessien de L en $\theta_k$. Intuitivement, si la fonction varie lentement, de grands pas sont nécessaires et à l'inverse si la fonction varie rapidement, il faut faire de petits pas (voir figure 3).

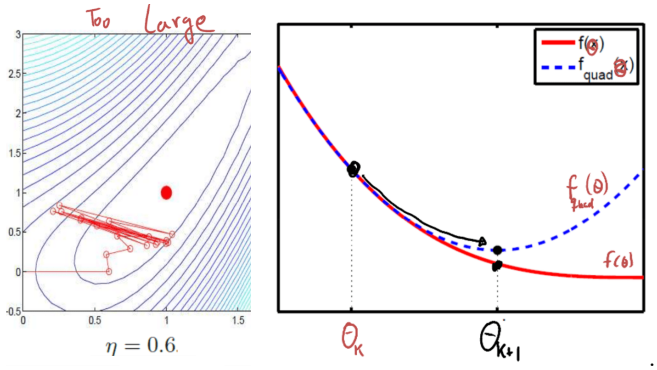


**Fig. 3.** Comment choisir la taille des pas $\eta_k$ dans l'algorithme de descente de gradient? Gauche: Pour $\eta = 0.6$ l'algorithme de descente de gradient oscille et ne parvient pas à converger efficacement. Droite: Approximation de Taylor (en bleu) de $L(\theta)$ autour de $\theta_k$: $L_{quad}(\theta) = L(\theta_k) + \nabla L(\theta_k)^T(\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T H_k(\theta - \theta_k)$. L'agorithme de Newton est obtenue en annulant $\nabla L_{quad}(\theta) = \nabla L(\theta_k) + H_k(\theta - \theta_k)$ en $\theta_{k+1}$. Crédit image [12].

**Data:** $(x_i, y_i)_{i=1}^n$
**Result:** Paramètres appris $\theta$
Initialiser $\theta$ aléatoirement;
**for** *k=1,2... jusqu'à convergence* **do**
  Evaluer $g = \nabla L(\theta)$;
  Evaluer $H = \nabla^2 L(\theta)$;
  $\theta = \theta - H^{-1} g$;
**end**
**return** last $\theta$;

**Algorithm 1:** Pseudo code de l'algorithme de Newton

Remarque: Plutôt que d'inverser la matrice $H$ à l'itération k, on peut résoudre le système linéaire d'équations $Hd = -\nabla L(\theta)$ pour trouver $d$. Lorsque la matrice $H$ est creuse, la méthode dite du gradient conjugué [4, 6, 7] est efficace pour résoudre le système d'équations linéaires.

### 3.4 Cas de la régression linéaire

Notons $X = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^{nd}$ et $y = [y_1, y_2, \ldots, y_n]^T \in \mathbb{R}^n$.

$$f_\theta(x_i) = x_i^T \theta = (X\theta)_i \tag{5}$$

$$L(\theta) = \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 = ||y - X\theta||_2^2 \tag{6}$$

où $||y - X\theta||_2^2 = (y - X\theta)^T(y - X\theta)$ est l'erreur quadratique entre les prédictions du modèle et la vérité sur le jeu de données. On peut vérifier que $L(\theta)$ est continue, deux fois dérivables. Son gradient et l'algorithme de descente de gradient sont donnés par

$$\nabla L(\theta) = 2X^T(X\theta - y)$$

$$\theta_{k+1} = \theta_k - 2\eta_k X^T(X\theta_k - y)$$

Son Hessien H est positive donc $L(\theta)$ convexe et la fonction admet un minimum global donné par l'algorithme de Newton (à démontrer en exercice sous l'hypothèse que $X^T X$ est pas inversible)

$$H = \nabla^2 L(\theta) = 2X^T X$$

$$\theta^* = (X^T X)^{-1} X^T y \tag{7}$$

### 3.5 Descente de Gradient Stochastique (SGD)

L'algorithme de descente de gradient est basé sur le calcul du gradient complet. À chaque itération, il faut calculer $\nabla L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla L_i(\theta)$ sur l'ensemble du jeu de donné. Que se passe-t-il si $n$ devient grand? Calculer $\nabla L(\theta)$ devient long.

**Stochastic gradients** Soit $s \in [1 \ldots n]$ choisi aléatoirement, uniformement,

$$\mathbb{E}[\nabla L_s(\theta)] = \nabla L(\theta)$$

$\nabla L_s(\theta)$ est un estimateur non biaisé mais très bruyant (variance élevée) du gradient complet $\nabla L(\theta)$. Intuitivement, on peut obtenir une assez bonne estimation du gradient en regardant juste quelques exemples. Il est souvent préférable d'avoir une estimation bruyante du gradient et se déplacer rapidement dans l'espace des paramètres (voir figure 4). L'algorithme de descente de gradient stochastique (SGD) [5] est souvent moins enclin à rester coincé dans des minima locaux peu profonds, car il ajoute une certaine quantité de "bruit". L'algorithme est populaire pour entrainer des modèles tels que les réseaux de neurones ou pour optimiser des fonctions non convexes.

**Data:** $(x_i, y_i)_{i=1}^n$, pas d'apprentissage $\eta_k$
**Result:** Paramètres appris $\theta$
Initialiser $\theta$ aléatoirement;
**for** *k=1,2... jusqu'à convergence* **do**
  Choisir aléatoirement, uniformément $s$ in [1... n];
  Calculer $\theta = \theta - \eta_k \nabla L_s(\theta)$ ;
**end**
**return** $\theta$;

**Algorithm 2:** Pseudo code de l'algorithme SGD

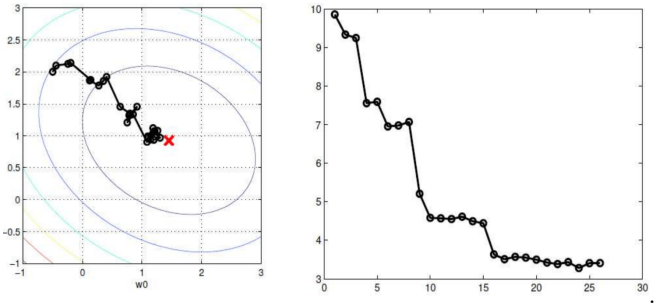


**Fig. 4.** Exemple de descente de gradient stochastique (SGD). Avec SGD, la suite des $\theta_k$ devient aléatoire et dépend du choix des indices $s_1 ... s_t$. Il convient d'utiliser une graine (seed en anglais) pour s'assurer de la reproductibilité des expériences. Crédit image [16, 17].

En pratique SGD calcule le gradient d'un mini-lot de données b (mini-batch en anglais). Si $b = n$, il s'agit de la descente de gradient classique. Typiquement, b est une puissance de 2 (par exemple 16 ou 32). SGD peut être utilisé en faisant défiler l'ensemble des données plusieurs fois; chacune de ces passes sur l'ensemble des données est appelée une époque (epoch en anglais). Chaque itération a une complexité de O(bd) au lieu de O(nd).

**Méthodes de réduction de variance et variantes** SGD est très rapide dans les premières itérations de l'algorithme mais peut échouer a converger précisement au minimum en raison du bruit. Des variantes permettent de réduire la variance de $\nabla L_s(\theta)$. L'idée générale consiste à utiliser une autre variable aléatoire fortement corrélée à $\nabla L_s(\theta)$ pour réduire la variance. L'algorithme SAGA [14] et Stochastic Variance Reduced Gradient [13] remplacent, dans les itérations, $\nabla L_s(\theta_k)$ par $\nabla L_s(\theta_k) - \nabla L_s(\tilde{\theta}) + \nabla L(\tilde{\theta})$ où $\tilde{\theta}$ est une ancienne valeur des paramètres. La descente de gradient s'effectue ensuite comme d'habitude. On peut vérifier qu'il s'agit d'un estimateur non biaisé de $\nabla L(\theta)$. Ces méthodes ont une complexité de $O(bd)$, comme pour l'algorithme classique SGD, et sont

plus rapide que la descente de gradient complet en pratique. De nombreuses autres variantes de SGD existent. L'idée derrière Adam [15] par exemple est d'introduire le concept d'inertie dans les itérations (momentum en anglais) pour échapper les minimum locaux.

## 4 Régression Logistique

La régression logistique [3] est l'algorithme de classification le plus populaire.

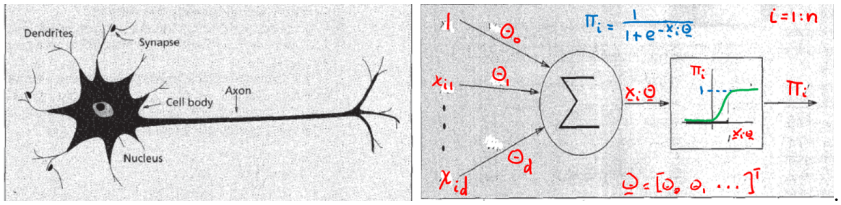


**Fig. 5.** Gauche: McCulloch-Pitts modèle d'un neurone [2]. Droite: La regression logistique comme un réseau de neurone à un seul neurone. Crédit image [12].

Nous souhaitons prédire un signal $y$ (ex: la présence d'un danger) en fonction d'un stimulus $X$ (ex: la température, la couleur) et utilisons pour celà un modèle de $y_i|x_i$, illustré dans la figure 5. Pour $y_i \in \{0, 1\}$, on modélise avec la fonction sigmoid $\sigma$ (de $\mathbb{R}$ dans [0,1]) la probabilité de $y_i$ étant donné $x_i$ et les paramètres $\theta$, aussi appelé poids du modèle:

$$\mathbb{P}(y_i = 1|x_i, \theta) = \sigma(x_i^T\theta) = \frac{1}{1 + e^{-x_i^T\theta}} \tag{8}$$

$\mathbb{P}(y_i = 1|x_i) \geq \mathbb{P}(y_i = 0|x_i)$ ssi $x_i^T\theta \geq 0$. Il s'agit d'un classifieur linéaire, respectivement aux caractéristiques de x. $x^T\theta = 0$ définit l'equation d'un hyperplan comme frontière de décision (voir figure 6). On souhaite calculer $\theta$ en maximisant la vraisemblance $\mathbb{P}(y|X, \theta)$ ou de manière équivalent en minimisant la log vraisemblance négative $-log(\mathbb{P}(y|X, \theta))$ (NLL en anglais)[1] sur $(x_i, y_i)_{i=1}^n$. Sous l'hypothèse $(x_i, y_i)$ iid et en notant $B_p$ une variable de Bernoulli($p$),

$$\mathbb{P}(y|X, \theta) = \Pi_{i=1}^n B_{\sigma(x_i\theta)}(y_i) = \Pi_{i=1}^n \sigma(x_i\theta)^{y_i}(1 - \sigma(x_i\theta))^{1-y_i}$$

$$L(\theta) = -log(\mathbb{P}(y|X, \theta)) = \sum_{i=1}^{n} log(1 + e^{-y_i x_i^T\theta}) \tag{9}$$

[1] Le log offre une certaine stabilité numérique (addition vs multiplication dans [0,1])

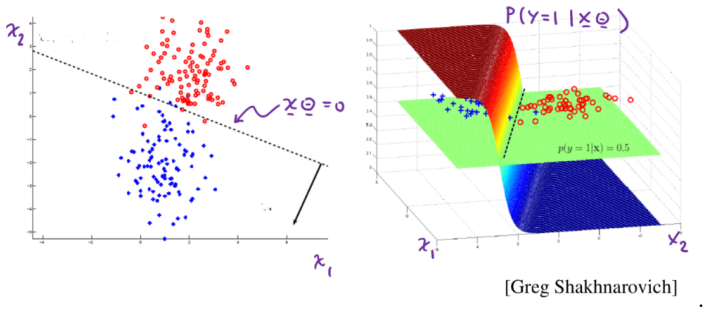


**Fig. 6.** Frontière de décision, représentée sous la forme d'un hyperplan. Crédit image [12].

### 4.1 Iteratively Reweighted Least Squares (IRLS)

$L(\theta)$ est continu, convexe. On peut trouver un minimiseur par l'algorithme de descente de gradient dont une itération est donnée par

$$\theta_{k+1} = \theta_k - \eta_k X^T(p_k - y)$$

Une itération de l'algorithme de Newton est donnée par (à démontrer)

$$\theta_{k+1} = (X^T S_k X)^{-1} X^T [S_k X \theta_k + y - p_k]$$

où $(p_k)_i = \sigma(x_i \theta_k)$, $H_k = X^T S_k X$ et $S_k = diag((p_k)_i(1 - (p_k)_i))$.

Le code python pour une itération de l'algorithme IRLS, avec la libraire numpy, est donné ci-dessous:

```
def irls(X, y, theta):
    a = np.dot(X, theta)
    p = sigmoid(a)
    SX = X * (p - p*p).reshape(-1,1)
    XSX = np.dot(X.T, SX)
    SXtheta = np.dot(SX, theta)
    theta = np.linalg.solve(SXS, np.dot(X.T, SXtheta+y-p))
    return theta
```

## 5 Régularisation en ML

### 5.1 Pourquoi régulariser?

Tous les modèles sont faux, certains sont utiles. En ML, la régularisation fait référence à un processus consistant à ajouter de l'information à un problème, s'il est mal posé. Cette information prend généralement la forme d'une pénalité envers la complexité du modèle. Par exemple 2, 4, 6, 8... une suite arithmétique.

### 5.2 Disgression sur l'évaluation en ML

Un algorithme d'apprentissage doit être capable de généraliser, c'est-à-dire fonctionner sur de nouveaux exemples, qui n'ont pas été montrés pendant la phase d'entraînement. La cross-validation permet d'évaluer la généralisation d'un modèle et d'optimiser les hyperparamètres (ex: $b$ et $\eta$ pour SGD). k-Fold cross-validation est la technique la plus populaire et illustrée dans la figure 7. L'ensemble des données est partitionné aléatoirement en k groupes. Un modèle est entraîné et testé sur chaque partition. Typiquement k = 5 ou 10.

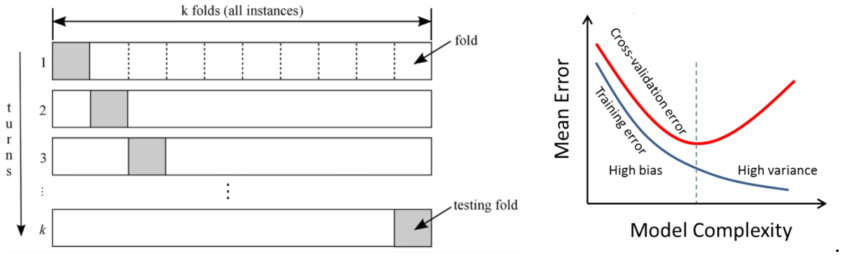


**Fig. 7.** Gauche: Exemple de partionnement pour entrainer et évaluer un modèle en ML. Chaque partition du jeu de donnée (train/test split en anglais) donne lieu à un modèle et un score dont la distribution sur l'ensemble des partitions permet de tester la généralisation des modèles entrainés. Droite: Exemple de training et cross validation error pour détecter l'underfitting (sous-apprentissage) et overfitting (sur-apprentissage). On dira qu'un modèle généralise bien s'il n'overfit pas. Crédit image [16, 17].

Les modèles complexes ont tendance à overfitter (erreur de test >> erreur d'apprentissage). Trop peu de données, par rapport à la complexité du modèle, peut également entraîner de l'overfitting. Une solution simple consiste à pénaliser la complexité des modèles, notamment les valeurs extrêmes des paramètres, qui correspondent souvent à un surapprentissage. Pour cela, on utilise une norme sur ces paramètres, que l'on va ajouter à la fonction qu'on cherche à minimiser. Les normes les plus couramment employées sont la norme L1 et L2. Ces deux méthodes réduisent l'amplitude des coefficients (coefficient shrinkage en anglais) et la complexité du modèle pour éviter le surappentrissage (overfitting).

### 5.3 Reformulation du problème d'optimisation

On souhaite trouver $\theta \in \mathbb{R}^d$ qui minimise

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, f_\theta(x_i)) + \lambda pen(\theta) \tag{10}$$

où $l$ est une fonction de coût, pen une pénalité pour empecher $\theta$ d'être trop complexe et $\lambda$ est un hyper-paramètre qui balance la qualité de l'ajustement d'une part et la pénalisation d'autre part (voir figure 8).

| | $l(y, y') = (y - y')^2$ | $l(y, y') = log(1 + e^{-yy'})$ |
|---|---|---|
| $pen(\theta) = \|\|\theta\|\|_2^2$ | Ridge regression | l2-penalised logistic regression |
| $pen(\theta) = \|\|\theta\|\|_1$ | Lasso | l1-penalised logistic regression |

**Table 1.** Combinaison de fonctions de coût et de pénalités classiques.

**Choix de la fonction de coût** Le choix de $l$ influence la solution apprise $f_\theta$ et dépend du problème considéré. Parmi les fonctions de coût classique, l'erreur moyenne quadratique (mse) est utilisée pour les problèmes de régréssion avec des variables continues. Elle est sensible aux outliers. L'erreur moyenne absolue l'est moins mais n'est pas dérivable en 0. Pour des problèmes de classification (variables discrètes), la fonction de coût Negative Log-Likelihood (nll) est populaire et utilisée pour la régression logistique. Minimiser $l_{nll}$ revient à maximiser la vraisemblance des données $y|X, \theta$.

$$l_{mse}(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2, y_i \in \mathbb{R} \tag{11}$$

$$l_{logistic}(\hat{y}_i, y_i) = log(1 + e^{-y_i \hat{y}_i}), y_i \in \{0, 1\} \tag{12}$$

Il existe de nombreuses autres fonctions de coût: max margin, Fisher discriminant, mutual information... pour l'apprentissage supervisé ou non.

**Choix de la fonction de pénalisation** Les normes L1 et L2 sont utilisées pour réduire la complexité des modèles et éviter le surappentrissage (overfitting).

**Régularisation L2 et régression de Ridge**

$$pen(\theta) = \frac{1}{2}||\theta||_2^2 = \frac{1}{2}\sum_{j=1}^{d} \theta_j^2 \tag{13}$$

Avec la méthode des moindres carrés, en annulant $\nabla L(\theta)$ on obtient

$$\theta_{Ridge} = (X^T X + \lambda I)^{-1} X^T y \tag{14}$$

Pour des jeux de données volumineux, une itération de SGD s'écrit

$$\theta_{k+1} = (1 - 2\lambda\eta)\theta_k + 2\eta X^T (y - X^T \theta) \tag{15}$$

La norme L2 est continue, dérivable, convexe. La régression de Ridge [8] rend le problème d'optimisation plus simple à résoudre s'il est mal posé et si $X^T X$ n'est pas inversible. Remarquons que les valeurs des paramètres $\theta$ sont petites mais non nulles dans le cas de caractéristiques (features) corrélées.

**Régularisation L1 et Lasso**

$$pen(\theta) = |\theta|_1 \tag{16}$$

Paradigme d'apprentissage: Seules quelques caractéristiques sont statistiquement pertinentes et comptent pour les prédictions. L'hypothèse $\theta^*$ contient beaucoup de 0 (sparse en anglais) conduit à un modèle plus simple, de dimension "réduite", idéal pour les problèmes inverses, le remote sensing, etc. Une séléction des paramètres (feature selection) permet d'expliquer le modèle.

L'inconvénient principal de la régularisation L1 ou Lasso [9] est dans le cas de caractéristiques fortement corrélées, une seule sera séléctionnée. Par ailleurs, la norme L1 n'est pas différentiable en 0. Avec la méthode des moindres carrés, calculer $\theta \in argmin||y - X\theta||^2$ sous la contrainte $|\theta|_1 \leq t$ est un problème de programmation quadratique, sans solution explicite comme pour la régression Ridge. La régression au moindre angles exploite la structure du problème de Lasso et fournit un moyen efficace pour calculer les solutions.

**Data:** $(x_i, y_i)_{i=1}^n$
**Result:** Paramètres appris $\theta$
Commencer avec tous les coefficients $\theta_j$ égaux à zéro;
Poser $r = y$;
**for** *k=1,2... d* **do**
  Trouvez la variable prédictive $j \in [1...d]$ la plus corrélée avec $r$;
  Ajoutez-la dans le modèle $\theta$;
  Calculer les résidus $r = y - \hat{y}$;
**end**
**return** $\theta$;

**Algorithm 3:** Pseudo code de la régression au moindre angles

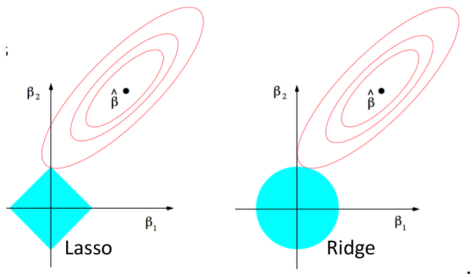


**Fig. 8.** Lasso vs Ridge. Contours des fonctions de coût en rouge (moindres carrés) et contraintes (régularisation) en bleu $||\theta||_1 \leq t$ and $||\theta||_2^2 \leq t^2$. Crédit image [16, 17].

### 5.4 Autre forme de régularisation: Ensembling

## 6 Optimisation sous contraintes

## 7 Méthodes bayésiennes

## References


1. J. Wallis. A treatise of algebra, both historical and practical. London. 1685.
2. W. S. McCulloch & W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics. 1943.
3. Berkson, J. Application of the logistic function to bio-assay. Journal of the American statistical association. 1944.
4. C. Lanczos. An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators, Journal of Research of the National Bureau of Standards. 1950.
5. H. Robbins & S. Monro. A stochastic approximation method. The annals of mathematical statistics. 1951.
6. C. Lanczos. Solution of Systems of Linear Equations by Minimized Iterations. Journal of Research of the National Bureau of Standards. 1952.
7. M. R. Hestenes & E. L. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems, Journal of Research of the National Bureau of Standards. 1952.
8. A. E. Hoerl & R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics. 1970.
9. R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. 1996.
10. W. J. Fu. Penalized regressions: the bridge versus the lasso. Journal of computational and graphical statistics. 1998.
11. F. Pedregosa, G. Varoquaux, A. Gramfort et al. Scikit-learn: Machine learning in Python. The Journal of Machine Learning Research. 2011.
12. N. Freitas. CPSC540 lecture notes. University of British Columbia. 2012.
13. R. Johnson & T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. Advances in neural information processing systems. 2013.
14. A. Defazio, F. Bach, & S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. Advances in neural information processing systems. 2014.
15. D. P. Kingma & J. Ba. Adam: A method for stochastic optimization. International Conference for Learning Representations. 2015.
16. M. Vazirgiannis. INF554 lecture notes. Machine Learning 1. Polytechnique. 2016.
17. S. Gaiffas. MAP569 lecture notes. Machine Learning 2. Polytechnique. 2017.