Étude expérimentale des méthodes d'optimisation du premier ordre en apprentissage

Hugo RAMILISON et Mohamed REGAIEG

Rapport de TP - Cours OMA, M2RO, Toulouse, France

Février 2026

Abstract

L'optimisation numérique est au cœur de nombreux problèmes en apprentissage automatique, par exemple lors de l'entraînement de grands modèles de langage (LLM) ou dans les systèmes de recommandation de livres et de films, où l'on cherche à ajuster des paramètres en minimisant une fonction objectif composée d'une **perte** et d'un terme de **régularisation**. Dans le cadre d'une fonction objectif convexe et différentiable, ou plus généralement non différentiable, les méthodes de gradient constituent des approches classiques de résolution. Dans cet article, nous présentons une étude expérimentale sur différents algorithmes de descente de gradient appliqués à un problème de régression linéaire multivariable.

1 Introduction

Les méthodes d'optimisation numérique sont au centre de nombreuses applications scientifiques, telles que la résolution de problèmes inverses ou l'apprentissage automatique [1,4]. Les problèmes rencontrés consistent le plus souvent à minimiser une fonction objectif sous des hypothèses classiques de convexité ou de régularité. Dans ce contexte, les méthodes de gradient sont largement employées. Lorsque la fonction est différentiable, la descente de gradient constitue la méthode de référence [1].

Cependant, l'algorithme de gradient ne peut plus être appliqué lorsque la fonction à minimiser n'est pas différentiable. Le cas le plus fréquent correspond à l'ajout de termes de régularisation, afin d'obtenir des solutions soit parcimonieuses, comme avec la régularisation Lasso, soit plus robustes, comme dans le cas de la régression Ridge [4].

C'est notamment grâce à la méthode du sous-gradient, qui étend l'algorithme de descente de gradient, qu'il est possible d'optimiser des fonctions convexes non différentiables [3]. Dans le cas particulier où la fonction est non différentiable mais s'écrit comme la somme d'un terme différentiable et d'un terme convexe non différentiable, une alternative efficace consiste à utiliser les méthodes proximales [2]. Ces méthodes exploitent la structure du problème via l'opérateur proximal, qui permet de prendre en compte l'effet du terme non différentiable. C'est en particulier dans les problèmes de régularisation que ces méthodes se révèlent les plus adaptées [4].

Enfin, dans le cadre de l'apprentissage automatique sur de très grands ensembles de données, et afin d'accélérer les calculs tout en assurant la convergence vers une solution minimisant la fonction objective, on utilise la descente de gradient stochastique [1,3]. Comme son nom l'indique,

cette méthode estime le gradient de manière statistique à partir d'un sous-échantillon aléatoire des données, sans calculer exactement le gradient global de la fonction [3].

Références

1. L. Bottou, F. Curtis and J. Nocedal (2016) *Optimization Methods for Large Scale Machine Learning*, preprint arXiv:1606.04838.
2. S. Boyd, N. Parikh, E. Chu, B. Peleato and J. Eckstein (2011) *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*, Foundations and Trends in Machine Learning, Vol. 3, No. 1 (1–122).
3. J. C. Duchi (2018) *Introductory Lectures on Stochastic Optimization*, IAS/Park City Mathematics Series.
4. Hastie, T., Tibshirani, R. and Friedman, J. (2009) *The Elements of Statistical Learning – Data Mining, Inference, and Prediction*, Second Edition, Springer Series in Statistics, Springer-Verlag, New York.

2 Le problème de régression linéaire en grande taille

Le problème étudié dans ce travail est celui des moindres carrés linéaires. On cherche à résoudre le système linéaire

$$Ax = b,$$

où $A \in \mathbb{R}^{n\times d}$ et $b \in \mathbb{R}^n$, en déterminant $x \in \mathbb{R}^d$. Cette recherche revient à faire tendre l'erreur $Ax - b$ vers 0. L'idée des moindres carrés est alors de raisonner sur le carré de l'erreur, ce qui permet de pénaliser plus fortement les grandes valeurs de l'écart.

On obtient ainsi le problème d'optimisation suivant :

$$\min_x f(x) = \frac{1}{2n}\|Ax - b\|^2 = \frac{1}{2n}\sum_{i=1}^{n}\left(a_i^\top x - b_i\right)^2.$$

On peut voir que f est μ-fortement convexe avec

$$\mu = \lambda_{\min}(\nabla^2 f(x)) = \frac{1}{n}\lambda_{\min}(A^\top A),$$

et que son gradient est L-Lipschitz avec

$$L = \lambda_{\max}(\nabla^2 f(x)) = \frac{1}{n}\lambda_{\max}(A^\top A),$$

puisque la matrice Hessienne est constante et ne dépend pas de x.

Lorsque $A^\top A$ est inversible, ce qui est le cas ici, on dispose d'une solution analytique

$$x^\star = \arg\min_{x\in\mathbb{R}^d} f(x) = (A^\top A)^{-1}A^\top b,$$

obtenue en résolvant $\nabla f(x^\star) = 0$. Cependant, lorsque n et d sont très grands, l'utilisation de cette formule devient trop coûteuse en termes de calcul à cause de l'inversion de matrice, d'où la nécessité d'utiliser des algorithmes d'optimisation de premier ordre.

De plus, afin de prendre en compte certains enjeux liés aux algorithmes d'apprentissage, comme le compromis biais-variance dans notre cas, nous introduisons dans la fonction objectif un terme de régularisation en norme $L1$.

3 Les algorithmes du premier ordre utilisés dans cette étude

Dans cette section, nous présentons un aperçu des différents algorithmes utilisés dans cette étude. Ces méthodes sont principalement itératives et reposent sur un principe d'amélioration progressive. Elles sont fondées sur l'utilisation du gradient.

3.1 L'algorithme du gradient

Considérons le problème d'optimisation suivant :

$$\min_{x\in\mathbb{R}^d} f(x), \qquad f \text{ différentiable.}$$

Le principe des méthodes de descente consiste à partir d'un point initial x_0 et à mettre à jour la solution de manière itérative en se déplaçant selon une direction de descente. On peut écrire la mise à jour sous la forme générale

$$x_{k+1} = x_k + \alpha_k d_k,$$

où α_k est le pas (ou taux d'apprentissage) et d_k une direction de descente. Un choix classique est de prendre

$$d_k = -\nabla f(x_k),$$

car l'opposé du gradient correspond à la direction de plus forte diminution locale de f. Le choix de α_k est un point important : un pas trop faible peut ralentir la convergence, tandis qu'un pas trop grand peut conduire à une divergence.

Dans le cas de la régression linéaire, on cherche à minimiser la fonction de coût

$$f(x) = \frac{1}{2n}\|Ax - b\|^2,$$

où A est la matrice des caractéristiques, b le vecteur des observations et x le vecteur de coefficients à estimer. Son gradient est donné par

$$\nabla f(x) = \frac{1}{n}A^\top(Ax - b).$$

La mise à jour par descente de gradient s'écrit alors

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k).$$

Pour cette étude, le critère d'arrêt retenu est le nombre d'itérations maximal. D'autres critères peuvent également être considérés, par exemple :

$$\|\nabla f(x_k)\| < \varepsilon, \qquad |f(x_{k+1}) - f(x_k)| < \varepsilon, \qquad \|x_{k+1} - x_k\| < \varepsilon.$$

3.2 Accélération du gradient

Le principe d'accélération du gradient consiste à ajouter un terme d'inertie dans l'expression de l'itéré courant. L'idée est de prolonger, à chaque itération, la descente dans la direction précédente. Il existe plusieurs manières d'introduire cette notion d'inertie (ou terme de friction).

Dans cette étude, nous utilisons l'accélération de Polyak, aussi appelée méthode de la boucle pesante, qui s'écrit sous la forme d'un schéma en deux étapes :

$$\begin{cases} p_k = -\nabla f(x_k) + \gamma_k p_{k-1}, \\ x_{k+1} = x_k + \alpha_k p_k. \end{cases}$$

Cette écriture peut également se réexprimer sous la forme :

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1}),$$

avec

$$\beta_k = \frac{\alpha_k \, \gamma_k}{\alpha_{k-1}}.$$

On identifie alors le terme supplémentaire par rapport à la descente de gradient classique,

$$\beta_k (x_k - x_{k-1}),$$

qui correspond au terme d'inertie (ou de friction).

L'algorithme dépend de deux paramètres qui contrôlent la descente, et leur choix influence le comportement de convergence. En particulier, en s'appuyant sur une propriété de type Lyapunov, on peut obtenir une convergence linéaire en prenant ces paramètres constants :

$$\alpha_k = \alpha = \frac{4}{\left(\sqrt{L} + \sqrt{\mu}\right)^2} \qquad \text{et} \qquad \beta_k = \beta = \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}.$$

3.3 L'agorithme du sous-gradient

L'algorithme du sous-gradient est utilisé lorsqu'on cherche à résoudre un problème dont la fonction objectif est convexe mais non différentiable. Le principe consiste, par rapport à la descente de gradient classique, à remplacer le gradient par un sous-gradient. On dit qu'un vecteur g est un sous-gradient d'une fonction convexe f en un point x si, pour tout y,

$$f(y) \geq f(x) + g^\top (y - x).$$

Dans notre problème initial, la fonction f est différentiable, et il n'y a donc pas d'intérêt à appliquer directement le sous-gradient. On ajoute alors un terme de régularisation en norme $L1$, ce qui conduit à un problème classique connu sous l'acronyme LASSO (*Least Absolute Shrinkage and Selection Operator*). On obtient ainsi :

$$f(x) = \frac{1}{2n} \|Ax - b\|^2 + \lambda \|x\|_1,$$

avec $\lambda > 0$.

On peut aussi écrire :

$$f(x) = \begin{cases} \frac{1}{2n} \|Ax - b\|^2 + \lambda \sum_i x_i & \text{si } x > 0, \\ \frac{1}{2n} \|Ax - b\|^2 & \text{si } x = 0, \\ \frac{1}{2n} \|Ax - b\|^2 - \lambda \sum_i x_i & \text{si } x < 0. \end{cases}$$

Le sous-gradient de f, noté g, est alors donné par :

$$g(x) = \begin{cases} \frac{1}{n}A^\top(Ax-b)+\lambda & \text{si } x>0, \\ \frac{1}{n}A^\top(Ax-b) & \text{si } x=0, \\ \frac{1}{n}A^\top(Ax-b)-\lambda & \text{si } x<0, \end{cases}$$

ou encore :

$$g(x) = \frac{1}{n}A^\top(Ax-b)+\lambda\,\mathrm{sign}(x).$$

3.4 L'algorithme du gradient proximal

Cet algorithme s'applique lorsque la fonction objectif se décompose en une partie différentiable et une autre non différentiable, et en particulier lorsque l'opérateur proximal associé à la partie non différentiable est facilement calculable. Dans le cas du LASSO, on considère le problème

$$\min_x f(x) = g(x) + h(x),$$

avec

$$g(x) = \|Ax-b\|^2 \quad \text{la partie différentiable,} \qquad h(x) = \lambda\|x\|_1 \quad \text{la partie non différentiable.}$$

L'idée consiste à approximer la fonction $g(x)$ par son approximation linéaire au point courant x_k, tout en conservant $h(x)$ inchangée et en ajoutant un terme de proximité. On obtient alors l'itération suivante :

$$x_{k+1} = \arg\min_x \left\{ h(x) + g(x_k) + \langle \nabla g(x_k), x-x_k\rangle + \frac{1}{2\alpha_k}\|x-x_k\|^2 \right\}.$$

Le terme $g(x_k)+\langle\nabla g(x_k), x-x_k\rangle$ correspond à l'approximation linéaire de $g(x)$ et $\frac{1}{2\alpha_k}\|x-x_k\|^2$ est le terme de proximité.

On définit l'opérateur proximal par :

$$\mathrm{prox}_h(x) = \arg\min_u \left\{ h(u) + \frac{1}{2}\|x-u\|^2 \right\}.$$

Ainsi, comme par la définition du proximal, l'algorithme de gradient proximal s'écrit :

$$x_{k+1} = \mathrm{prox}_{\alpha_k h}\big(x_k - \alpha_k \nabla g(x_k)\big).$$

Dans le cas du LASSO, avec $h(x) = \lambda\|x\|_1$, l'opérateur proximal s'écrit composante par composante :

$$\Big(\mathrm{prox}_{\alpha\lambda\|\cdot\|_1}(x)\Big)_i = \begin{cases} x_i - \alpha\lambda, & \text{si } x_i > \alpha\lambda, \\ 0, & \text{si } x_i \in [-\alpha\lambda, \alpha\lambda], \\ x_i + \alpha\lambda, & \text{si } x_i < -\alpha\lambda, \end{cases} \qquad \forall i = 1,\dots,d.$$

4 Variantes stochastiques des algorithmes

Considérons le problème suivant :

$$F : \mathbb{R}^d \to \mathbb{R}, \qquad \min_x F(x) = \mathbb{E}_\xi[f(x,\xi)],$$

avec

$$F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

et ξ un tirage aléatoire dans $\{1, \ldots, n\}$.

Le principe des algorithmes stochastiques repose sur l'idée que le gradient évalué sur un échantillon est suffisamment représentatif du gradient complet. Cela correspond bien à notre contexte de régression linéaire de grande taille, pour lequel le calcul du gradient total devient coûteux. À chaque itération, on effectue donc un tirage aléatoire ξ_k, on échantillonne $f(x, \xi_k)$ et on calcule une direction de descente à partir de cet échantillon. La taille de l'échantillon aléatoire est appelée minibatch.

L'algorithme du gradient stochastique s'écrit alors :

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k, \xi_k).$$

Son bon fonctionnement repose sur certaines hypothèses, notamment que la direction $-\nabla f(x_k, \xi_k)$ soit une direction de descente comparable à $-\nabla F(x_k)$ et que la variance du gradient stochastique reste bornée.

Dans le cadre des méthodes pour l'apprentissage à grande échelle, on peut considérer deux familles de pas de déplacement, comme dans [1].

- Une première possibilité consiste à utiliser un pas constant $\bar{\alpha}$ vérifiant

  $$0 < \bar{\alpha} < \frac{\mu}{L\, M_G}, \qquad \text{avec } M_G = M_V + \mu_G^2.$$

  Dans notre cas, on prend $\mu = \mu_G = 1$ car, comme indiqué dans [1], les inégalités de l'Assumption 4.3(b) sont vérifiées « immédiatement » avec $\mu_G = \mu = 1$ dès lors que $g(w_k, \xi_k)$ est un estimateur non biaisé de $\nabla F(w_k)$.

  $$\bar{\alpha} < \frac{1}{L(1 + M_V)}.$$

- Une seconde possibilité consiste à utiliser un pas décroissant défini par

  $$\alpha_k = \frac{\beta}{\gamma + k},$$

  avec les conditions

  $$\beta > \frac{1}{c\mu}, \qquad \gamma > 0, \qquad \alpha_1 \leq \frac{\mu}{L\, M_G},$$

  où c désigne la constante de forte convexité de F.

Dans notre problème de régression linéaire de grande taille, nous avons d'abord défini une méthode d'évaluation du gradient sur un minibatch, puis appliqué l'algorithme général présenté précédemment. Les paramètres comme la taille du minibatch et le pas de déplacement sont choisis expérimentalement, en tenant compte des conditions précédentes.

5 Expériences numériques

Au regard des méthodes de descente décrites ci-dessus, nous avons implémenté chacun des schémas, puis les avons testés sur un jeu de données adapté à de la régression généré grâce à la bibliothèque `sklearn.datasets`. Pour les méthodes déterministes, nous avons généré 1000 échantillons

de données en dimension 100. Pour la méthode du gradient stochastique, nous avons augmenté le nombre d'échantillons à 10 000 et réduit leur dimensionnalité à 10. Notre objectif était de jouer sur les différents paramètres de chacune des méthodes afin d'étudier leur influence. Comme nous manipulions des problèmes jouets, la solution optimale pouvait être caluclée et ainsi servir de référence pour l'évaluation de chacune des méthodes.

5.1 L'algorithme du gradient

5.1.1 Vérification de la convergence

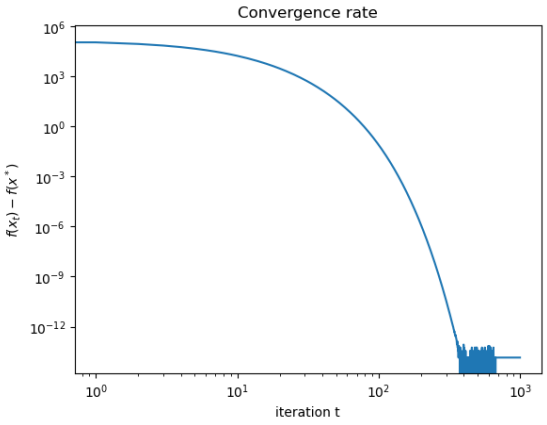


Figure 1: $f(x_t) - f(x^*)$ en fonction du nombre d'itérations t

Il s'agissait tout d'abord de vérifier la convergence de la méthode du gradient. Nous avons choisi un taux d'apprentissage α arbitraire à 0.1 pour cette première expérience. La courbe présente la différence entre la valeur de l'objectif à l'itération t et la valeur optimale. On observe bien qu'à partir de l'itération 300 nous passons en dessous des valeurs de 0 numérique. Ce qui confirme la convergence de la méthode. Désormais afin d'affiner la méthode, nous avons choisi d'étudier l'effet du taux d'apprentissage α.

5.1.2 Effet du taux d'apprentissage

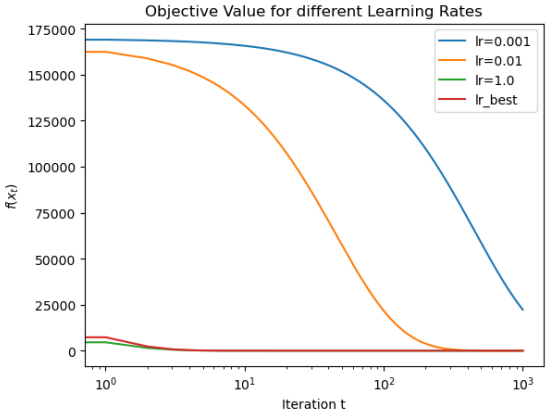


Figure 2: $f(x_t)$ pour différents α

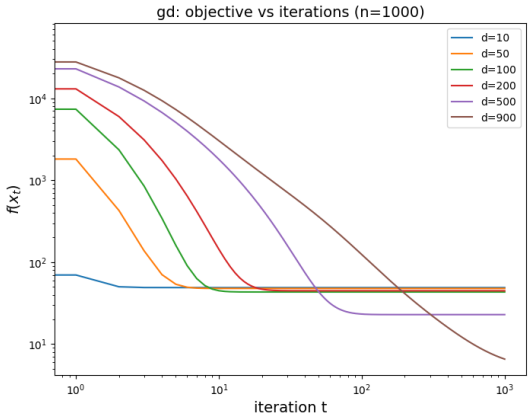
Nous pouvons analytiquement établir que la valeur optimale de α pour la descente de gradient pour une fonction à gradient L-Lipschitz est:

$$\alpha_{opti} = \frac{1}{L}$$

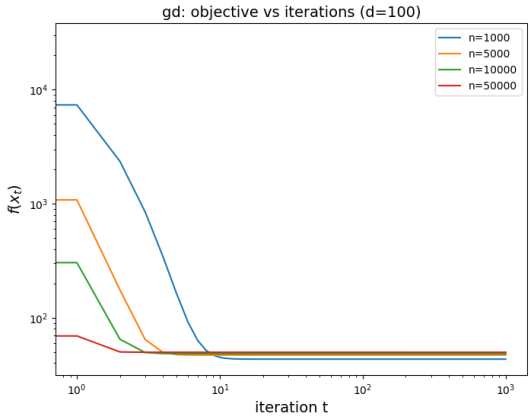
Pour notre problème, nous avons donc tracé l'évolution de l'objectif pour différentes valeurs de α. Nous pouvons alors remarquer sur la Figure 2 qui pour des trop petites valeurs de α la convergence n'est pas atteinte au bout de 1000 itérations. On a dans ce cas, un algorithme trop "prudent" qui finalement avance trop lentement. Alors que pour des taux d'apprentissages proche du pas optimal nous arrivons très rapidement à l'objectif. Cela peut s'expliquer par le fait que dans le cas d'une fonction à gradient L-Lipschitz, en prenant le pas optimal, nous maximisons la minimisation de l'objectif à chaque fois. Cependant, il reste important de ne pas prendre un pas trop grand, sinon il peut arriver que l'algorithme diverge car il saute autour de la solution sans jamais l'atteindre.

5.1.3 Impact de la dimension d et du nombre d'observations n

On applique l'algorithme de descente de gradient (GD) sur le problème des moindres carrés, en prenant un pas constant $\alpha = 1/L$ (avec L la constante de Lipschitz du gradient), ce qui est le choix le plus performant ici puisque la fonction est fortement convexe et à gradient L-Lipschitz. On réalise deux séries de tests : (i) on fixe $n = 1000$ et on fait varier d ; (ii) on fixe $d = 100$ et on fait varier n. À chaque exécution, on trace $f(x_t)$ en fonction des itérations en échelle log-log.



(a) GD : $f(x_t)$ pour $n = 1000$ et différentes dimensions d.



(b) GD : $f(x_t)$ pour $d = 100$ et différentes tailles n.

Figure 3: Descente de gradient avec $\alpha = 1/L$: influence de d et de n.

- On observe globalement une pente proche de -1 en échelle log-log, ce qui correspond à une convergence linéaire, comme attendu pour une fonction fortement convexe avec le choix $\alpha = 1/L$. Lorsque d dépasse environ 500, $f(x_t)$ semble se rapprocher davantage de la solution finale, ce qui n'était pas forcément attendu uniquement à cause de l'augmentation de la dimension.
- Là aussi, la pente quasi constante en log-log confirme la convergence linéaire du gradient. Quand n augmente, la descente devient plus stable et atteint rapidement un plateau, ce qui reste cohérent avec un problème mieux conditionné.

5.2 Accélération du gradient

Nous avons ensuite implémenté la méthode d'accélération de la Boule pesante. La première chose à vérifier, c'est que celle-ci augmente bien la vitesse de convergence.

5.2.1 Comparaison avec l'algorithme du gradient

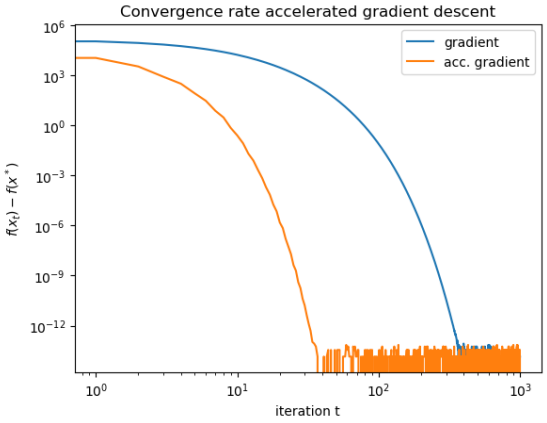


Figure 4: Comparaison de la convergence entre gradient standard et accéléré

Nous pouvons remarquer que le gradient accéléré converge dès l'itération 20 contre 300 pour le gradient standard. Ce qui confirme bien l'intérêt de cette méthode. Son efficacité peut s'expliquer par le fait qu'elle respecte une certaine réalité physique. En effet, l'algorithme de gradient standard respecte une approche "cupide" (ou *greedy*), elle ne regarde que ce qui améliore sa situation à l'itération t sans se soucier des choix précédents. Alors que l'algorithme accéléré conserve en partie l'inertie de sa descente précédente pour poursuivre. Si le gradient classique est un randonneur qui s'arrête à chaque pas pour décider où aller, la boule pesante est comme un skieur qui continue de glisser tout en mettant à jour sa direction.

5.2.2 Effet du paramètre d'inertie

L'inertie conservée dans cette méthode est capturée par le paramètre β qui peut s'interpréter comme la proportion du pas précédent que l'on conserve dans le pas suivant. Nous avons également une formule analytique pour calculer sa valeur optimale.

$$\beta_{opti} = \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}$$

Nous avons donc étudié l'évolution de la valeur objectif selon différentes valeurs de β

Cette fois-ci, avoir un β trop faible n'est pas trop problématique car cela revient seulement à utiliser une descente de gradient classique c'est pourquoi on a toujours la convergence tant que le pas α est choisi convenablement. En revanche nous pouvons bien voir sur le graphe que l'algorithme devient instable si β est trop grand, en l'occurence, pour un $\beta = 1$ on perd même la convergence à partir de l'itération 20 et on obtient de grandes instabilités. C'est ce qui est prévisible car une trop grande inertie peut être cause d'instabilités au fur et à mesure que l'on se rapproche de la valeur objectif où nous avons besoin de nous arrêter ce qui est impossible avec trop d'inertie.

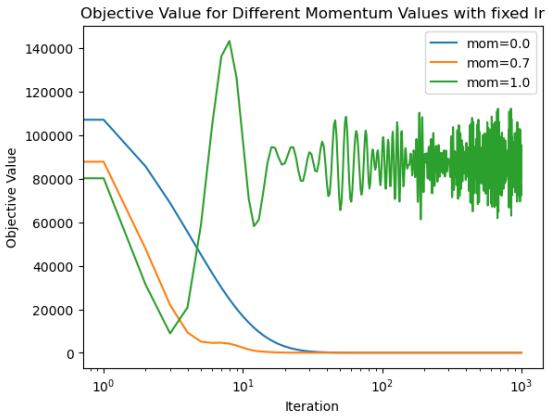


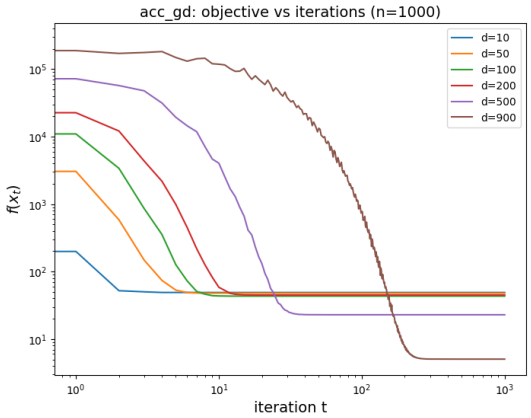
Figure 5: Valeurs de l'objectif en fonction de l'itération pour différents β

5.2.3 Impact de la dimension d et du nombre d'observations n

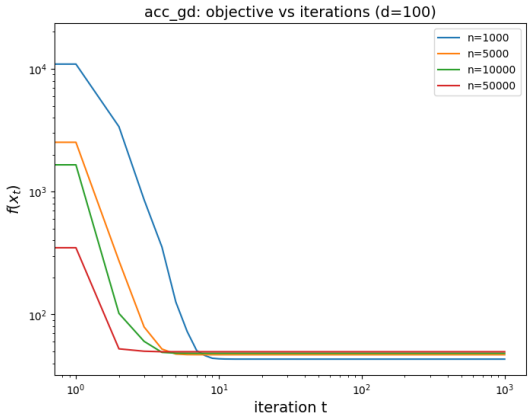
On applique l'algorithme du gradient accéléré de Polyak (méthode de la boule pesante) sur le problème des moindres carrés. On utilise les paramètres théoriques

$$\alpha = \frac{4}{(\sqrt{L}+\sqrt{\mu})^2} \qquad \text{et} \qquad \beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}},$$

où $\mu = \frac{1}{n}\lambda_{\min}(A^\top A)$ et $L = \frac{1}{n}\lambda_{\max}(A^\top A)$. Comme précédemment, on fait deux séries de tests : (i) on fixe $n = 1000$ et on fait varier d ; (ii) on fixe $d = 100$ et on fait varier n. On trace ensuite $f(x_t)$ en fonction des itérations en échelle log-log.



(a) Acc-GD : $f(x_t)$ pour $n = 1000$ et différentes dimensions d.



(b) Acc-GD : $f(x_t)$ pour $d = 100$ et différentes tailles n.

Figure 6: Gradient accéléré (boule pesante) : influence de d et de n.

- À n fixé, on retrouve globalement le même effet qu'avec GD : quand d augmente, la convergence ralentit, et à partir de $d \gtrsim 500$ la valeur finale de $f(x_t)$ semble se rapprocher davantage de la solution.

- À d fixé, quand n augmente, la descente devient plus rapide et plus stable, avec une pente globalement régulière en log-log, ce qui reste cohérent avec une convergence linéaire sur un problème fortement convexe.

5.3 L'algorithme du sous-gradient

5.3.1 Effet du paramètre de pénalisation

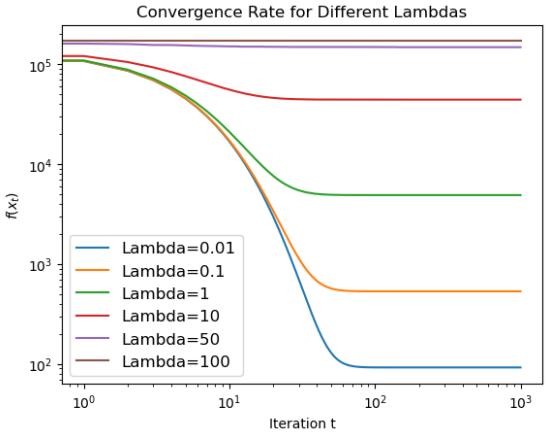
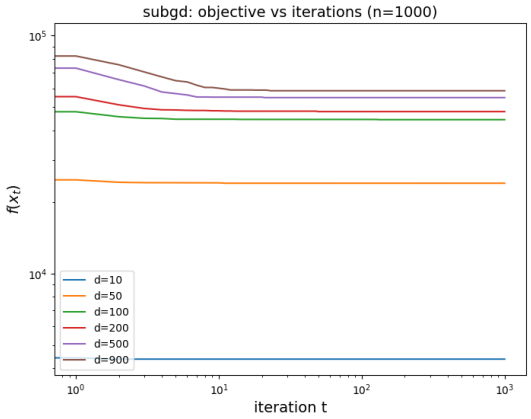


Figure 7: SubGD: $f(x_t)$ pour différents λ

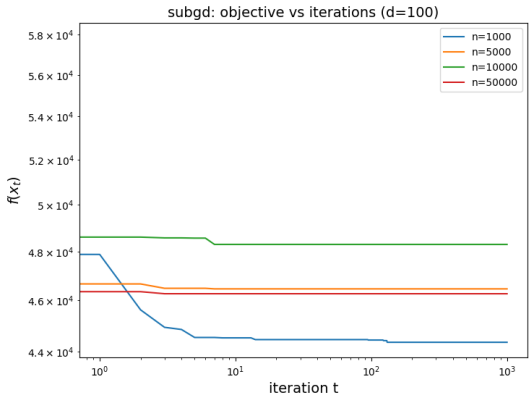
Dans le sous gradient nous faisons intervenir un paramètre de pénalité λ pour assurer la parcimonie de la solution. Nous pouvons le voir sur la figure 7 plus le paramètre de pénalisation est grand, plus nous convergeons loin de la solution optimale.

5.3.2 Impact de la dimension d et du nombre d'observations n

On applique l'algorithme du sous-gradient sur le problème LASSO en prenant un pas constant $\alpha = 1/L$, comme pour la descente de gradient. On étudie l'influence de la dimension d à n fixé puis l'influence de la taille de l'échantillon n à d fixé, en traçant $f(x_t)$ en fonction des itérations.



(a) SubGD : $f(x_t)$ pour $n = 1000$ et différentes dimensions d.



(b) SubGD : $f(x_t)$ pour $d = 100$ et différentes tailles n.

Figure 8: Sous-gradient : influence de d et de n.

- À n fixé, quand d augmente, la valeur de l'objectif reste élevée et la décroissance est très limitée, ce qui est cohérent avec le fait que la fonction n'est plus différentiable et que le sous-gradient converge plus lentement.
- À d fixé, augmenter n ne change pas beaucoup la forme des courbes : on observe surtout un plateau rapide, ce qui confirme que le sous-gradient est moins efficace que GD ou Acc-GD dans ce cadre.

5.3.3 Parcimonie de la solution

Afin de vérifier que l'algorithme du sous-gradient offre une solution plus parcimonieuse, nous avons affiché les valeurs obtenues de la solution par une descente de gradient classique contre celles obtenues avec un algorithme de sous-gradient sur la figure 9.

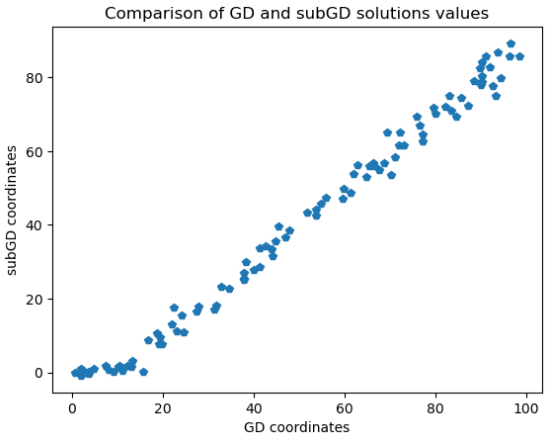


Figure 9: Solutions obtenues par le sous-gradient contre celles du gradient classique

Nous pouvons ainsi voir que pour les mêmes valeurs de solution obtenues par un gradient classique, environ 20 % de celles du sous gradient sont au zéro numérique. En effet en pénalisant les solutions en norme $L1$ nous forçons une meilleure parcimonie de solution.

5.4 L'algorithme du gradient proximal

5.4.1 Comparaison avec la méthode du sous gradient

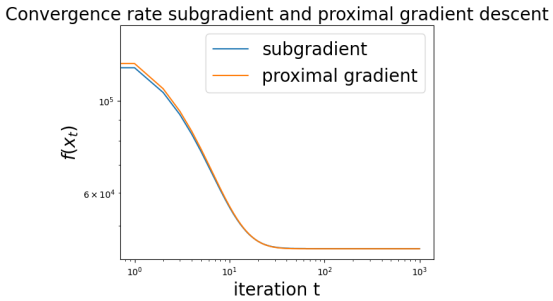
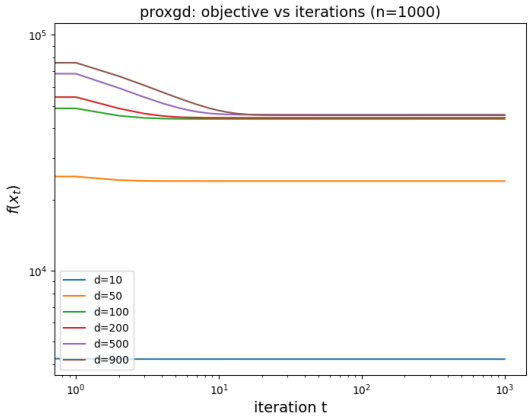


Figure 10: subGD vs proxGD

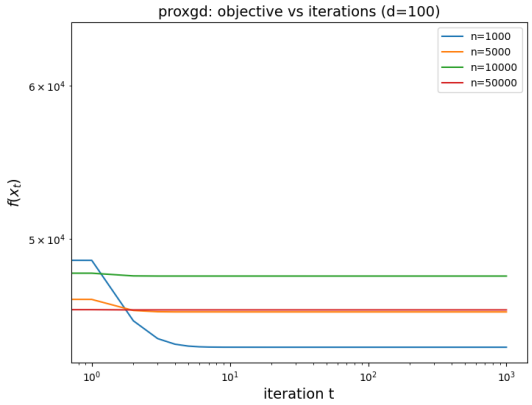
La figure 10 présente une comparaison entre le sous gradient et le gradient proximal. Il est pertinent de les comparer car ces méthodes s'appliquent à des types de fonction construites similairement comme somme d'une partie différentiable et non différentiable. Les deux méthodes convergent à un rythme similaire mais tout de même moins rapidement que le gradient classique, ce qui est attendu car ces deux méthodes convergent sous linéairement en $O(\frac{1}{\sqrt{t}})$ contre une convergence sous linéaire en $O(\frac{1}{t})$ pour le gradient classique.

5.4.2 Impact de la dimension d et du nombre d'observations n

On applique l'algorithme du gradient proximal sur le problème LASSO avec un pas constant $\alpha = 1/L$, afin de comparer directement avec l'algorithme du sous-gradient dans le cadre non différentiable. Comme précédemment, on étudie l'influence de la dimension d à n fixé puis l'influence de la taille de l'échantillon n à d fixé, en observant l'évolution de $f(x_t)$.



(a) ProxGD : $f(x_t)$ pour $n = 1000$ et différentes dimensions d.



(b) ProxGD : $f(x_t)$ pour $d = 100$ et différentes tailles n.

Figure 11: Gradient proximal : influence de d et de n.

- À n fixé, quand d augmente, on observe une décroissance plus régulière que pour le sous-gradient, ce qui est attendu puisque l'opérateur proximal permet de mieux gérer le terme non différentiable.
- À d fixé, l'effet de n reste assez limité sur la forme des courbes, mais le proximal atteint plus rapidement un plateau, ce qui montre une meilleure stabilité par rapport au sous-gradient avec le même choix $\alpha = 1/L$.

5.5 Variantes stochastiques des algorithmes

Dans cette partie, nous étudions le comportement de la descente de gradient stochastique en fonction du choix du pas et de la taille du minibatch.

5.5.1 Pas constant.

Nous commençons par considérer une famille de pas constants de la forme $\alpha = 1/(Lk)$ avec $k \in \{6, 8, 10, 12, 16\}$. La Figure 12 montre l'évolution de $f(x_t) - f(x^\star)$ pour ces valeurs. On peut observer que pour $\alpha = 1/(L \cdot 6)$ la courbe diverge, ce qui suggère que la condition sur le pas n'est pas respectée en pratique. Pour des valeurs plus petites du pas, la décroissance de la fonction objectif devient plus régulière. Plus le pas est petit, plus la courbe est lisse et présente moins d'oscillations, mais la convergence devient aussi plus lente.

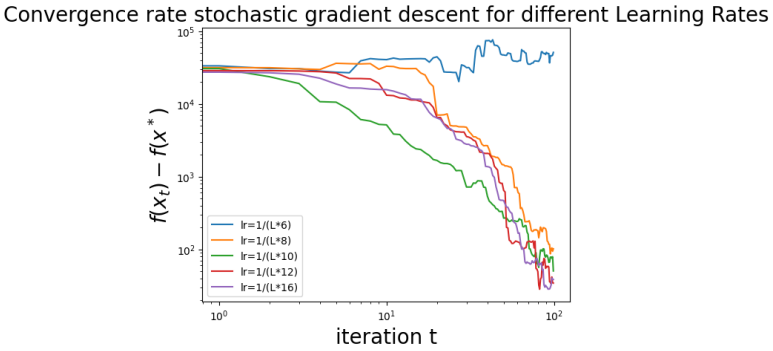


Figure 12: Convergence du SGD pour différents pas constants.

5.5.2 Pas décroissant.

Nous considérons ensuite une famille de pas décroissants

$$\alpha_k = \frac{\beta}{\gamma + k}.$$

Dans nos expériences, on fixe $\beta = 2/c$, ce qui vérifie $\beta > \mu/c$ avec $\mu = 1$ comme indiqué précédemment. La condition $\alpha_1 < \mu/(LM_G)$ s'écrit alors

$$\frac{\beta}{\gamma + 1} < \frac{\mu}{LM_G} \quad \Rightarrow \quad \gamma > LM_G\beta - 1,$$

soit ici $\gamma > \frac{L}{c}(1 + M_V) - 1$. La Figure 13 illustre l'influence de γ : pour de petites valeurs de γ (par exemple $\gamma = 1/c$ ou $\gamma = 2/c$), on observe une phase initiale avec de fortes oscillations et même une augmentation de $f(x_t) - f(x^\star)$ avant que la courbe ne redescende, ce qui est cohérent avec un pas initial trop grand. À l'inverse, lorsque γ augmente (par exemple $\gamma = 8/c$, $\gamma = 10/c$ ou $\gamma = 16/c$), les trajectoires deviennent plus régulières et la décroissance est globalement plus stable sur l'ensemble des itérations.

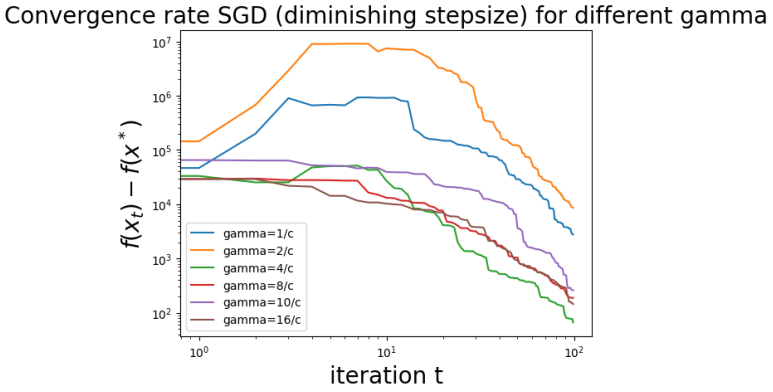


Figure 13: Convergence du SGD avec pas décroissants pour différentes valeurs de γ.

5.5.3 Influence de la taille du minibatch.

Enfin, nous comparons différentes tailles de minibatch. La Figure 14 montre que des minibatchs très petits introduisent plus de variabilité dans la trajectoire de l'algorithme, alors que des tailles plus grandes donnent des courbes plus régulières, proches du gradient plein. Toutefois, augmenter la taille du minibatch augmente aussi le coût de calcul par itération, ce qui introduit un compromis entre stabilité et coût computationnel.

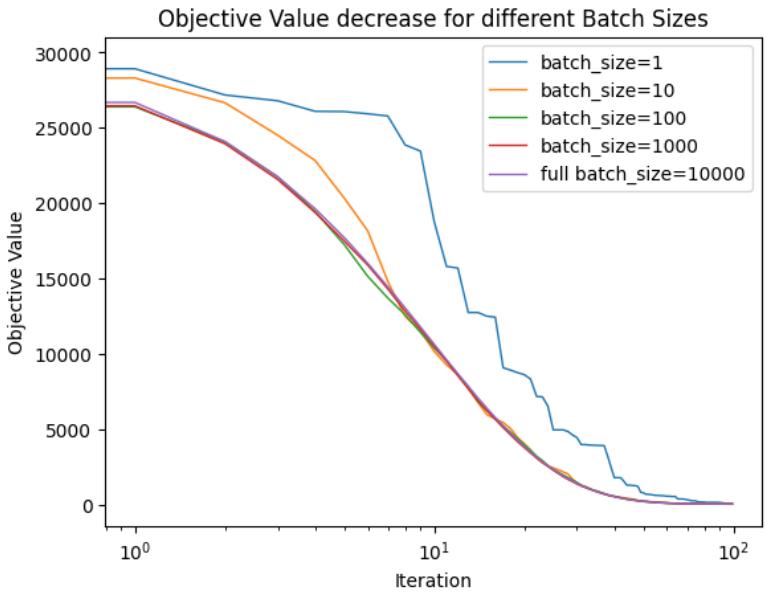


Figure 14: Influence de la taille du minibatch sur la convergence du SGD.

6 Extension au cas de la classification par sup- port vector machines

6.1 L'agorithme du sous-gradient stochastique

Dans cette partie, nous adaptons l'algorithme du gradient stochastique afin de résoudre le problème des SVM. Ce problème se formule de la manière suivante :

$$\min_{w,b} f(w,b) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max\{0,\, 1 - y_i(w^\top x_i + b)\},$$

où C est un paramètre. Dans la suite, on note $\theta = (w, b)$ avec $w = (\theta_1, \ldots, \theta_d)$ et $b = \theta_{d+1}$.

La particularité de ce problème est la présence d'un terme non différentiable

$$C\sum_{i=1}^{n} \max\{0,\, 1 - y_i(w^\top x_i + b)\}.$$

Ainsi, au lieu d'évaluer le gradient sur un minibatch comme dans le SGD classique, on utilise un sous-gradient, ce qui conduit à l'algorithme du sous-gradient stochastique.

Le sous-gradient du SVM se calcule comme suit :

$$\nabla_w\big(\tfrac{1}{2}\|w\|^2\big) = w, \qquad \nabla_b\big(\tfrac{1}{2}\|w\|^2\big) = 0.$$

Pour

$$h(w,b) = C\sum_{i=1}^{n} \max\{0,\, 1 - y_i(w^\top x_i + b)\},$$

on a :

- si $y_i(w^\top x_i + b) > 1$, le terme ne contribue pas (sous-gradient nul) ;
- si $y_i(w^\top x_i + b) < 1$, il contribue dans la fonction objectif.

On obtient alors

$$\nabla_w h = -y_i x_i, \qquad \nabla_b h = -y_i.$$

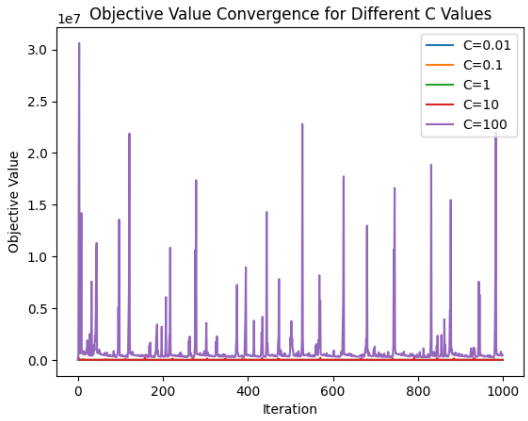
Ainsi, le sous-gradient de f, noté $g(\theta)$ ou $g(w,b)$, s'écrit :

$$g_w(\theta) = w - C\sum_{i=1}^{n} \mathbf{1}_{\,y_i(w^\top x_i+b)\le 1}\, y_i x_i, \qquad g_b(\theta) = -C\sum_{i=1}^{n} \mathbf{1}_{\,y_i(w^\top x_i+b)\le 1}\, y_i,$$

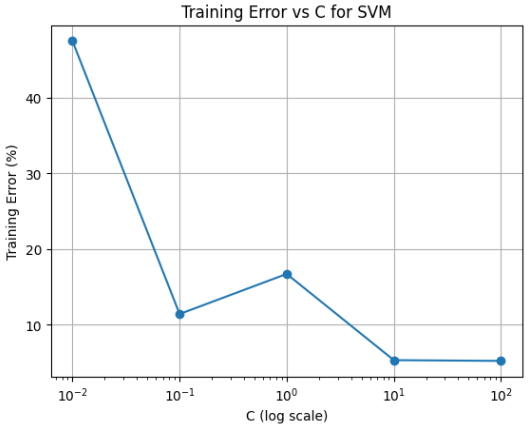
où $\mathbf{1}_{\,y_i(w^\top x_i+b)\le 1}$ est la fonction indicatrice qui vaut 1 lorsque $y_i(w^\top x_i + b) \le 1$ et 0 sinon.

6.2 Expériences numériques

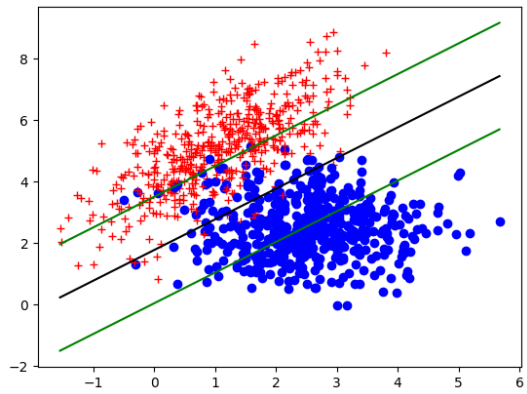
Nous étudions l'impact du paramètre C dans l'objectif SVM en testant plusieurs valeurs $C \in \{0.01,\, 0.1,\, 1,\, 10,\, 100\}$ sur un même jeu de données. Pour chaque valeur de C, nous lançons l'algorithme du sous-gradient stochastique pendant un nombre fixé d'itérations. Nous suivons (i) l'évolution de la valeur de la fonction objectif au fil des itérations, et (ii) l'erreur d'entraînement en fonction de C. Enfin, pour illustrer qualitativement l'effet de C sur la séparation, nous visualisons la frontière de décision obtenue pour une petite valeur de C et pour une grande valeur de C.



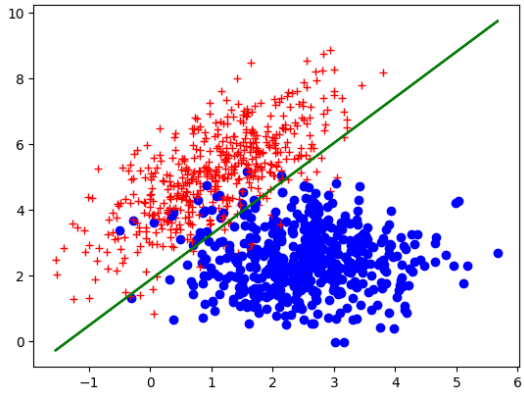
(a) Valeur de l'objectif en fonction des itérations pour différentes valeurs de C.



(b) Erreur d'entraînement en fonction de C (échelle logarithmique).



(c) Frontière de décision pour un C petit (ex : $C = 0.01$).



(d) Frontière de décision pour un C grand (ex : $C = 100$).

Figure 15: Étude expérimentale de l'influence du paramètre C pour le SVM.

On observe que l'erreur d'entraînement diminue lorsque C augmente (Fig. 15b), ce qui est attendu car un C plus grand donne plus de poids à la minimisation des erreurs de classification sur l'ensemble d'apprentissage. En revanche, les courbes de l'objectif (Fig. 15a) montrent que pour des valeurs élevées de C, l'optimisation devient moins régulière, avec des pics et des oscillations plus visibles. À l'inverse, des valeurs plus petites de C conduisent à une convergence plus lisse et plus stable, mais généralement avec une erreur d'entraînement plus élevée. Cela illustre le compromis contrôlé par C : si C est grand, on pondère davantage l'erreur de classification (ce qui peut réduire la marge et mener à un surapprentissage), tandis que si C est petit, on tolère plus d'erreurs et on favorise une marge plus grande, ce qui peut améliorer la généralisation.

7 Conclusion

En conclusion, le choix de la méthode d'optimisation reste étroitement lié à la nature du problème considéré, aux contraintes de calcul ainsi qu'aux propriétés de la fonction objectif. Cette étude souligne le rôle des paramètres d'optimisation et de la structure du problème dans le comportement des algorithmes. Les résultats obtenus peuvent ainsi servir de repère pour sélectionner une méthode d'optimisation adaptée à des contextes d'apprentissage automatique et d'optimisation numérique.

Annexes

Repository github avec les notebooks:Notebooks