Méthodes du premier ordre pour la régression à grande échelle

Dexter METSANOU TSAFACK et Komlan Epe-Nsin Merveilles AGBETI-MESSAN

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

February 9, 2024

Abstract

Cette étude approfondit les algorithmes d'optimisation, en mettant l'accent sur la Descente de Gradient et ses dérivés tels que le Gradient Accéléré, la Descente de Sous-Gradient et le Gradient Proximal. L'analyse explore l'impact de paramètres comme les taux d'apprentissage sur la vitesse de convergence, compare différentes méthodes, et élargit l'examen aux problèmes d'apprentissage machine à grande échelle. Les expériences englobent des considérations sur la taille des ensembles de données et des méthodes de gradient stochastique. Les résultats fournissent des perspectives sur les choix optimaux pour divers scénarios en matière d'optimisation.

1 Introduction

Les algorithmes d'optimisation jouent un rôle clé en apprentissage machine et en modélisation statistique. Cette étude se penche sur les principes de la Descente de Gradient et de ses dérivés, explorant des paramètres clés comme les taux d'apprentissage, les critères de convergence, et les techniques d'accélération. La motivation réside dans l'amélioration de l'efficacité de l'optimisation pour divers problèmes, y compris la régression et la classification. En expérimentant avec différentes méthodes et paramètres, nous visons à découvrir des informations sur leurs caractéristiques de performance et leur applicabilité.

L'étude s'appuie sur des concepts fondamentaux de la littérature sur l'optimisation [1] et utilise des techniques telles que le Momentum [6], le Gradient Proximal [4] et la Descente de Gradient Stochastique [3]. Le choix de la méthode d'optimisation dépend de facteurs tels que la nature du problème, la précision souhaitée et les ressources disponibles [5].

2 Le problème de régression linéaire en grande taille

La Régression Linéaire est une méthode statistique qui se positionne comme un outil permettant de modéliser la relation entre une variable cible et un ensemble étendu de variables explicatives. Son application principale réside dans

la prédiction de la valeur d'une variable cible en se basant sur les valeurs des variables explicatives. Le processus implique la détermination des paramètres d'une fonction visant à modéliser cette relation de manière à minimiser l'erreur par rapport aux données disponibles.

Formellement, pour une matrice $A \in \mathbb{R}^{n*d}$ et un vecteur $b \in \mathbb{R}^n$, l'objectif est de minimiser la fonction suivante :

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

Nous pouvons voir que la fonction f est μ fortement convexe avec $\mu = \lambda_{min}(\nabla^2 f(x)) = \frac{1}{n}\lambda_{min}(A^T A)$ et L lisse avec $L = \lambda_{max}(\nabla^2 f(x)) = \frac{1}{n}\lambda_{max}(A^T A)$, puisqu'ici la matrice Hessienne est constante, indépendante de x.

En grande taille, on dispose d'une très grande base de données en terme de nombre d'observations. L'accroissement de la masse et de la taille des données a en effet nécessité la proposition de nouvelles approches statistiques adaptées aux caractéristiques des données modernes. En particulier, la grande dimension des données (nombre important de variables) pose un ensemble de problèmes à la statistique multivariée classique que l'on résume usuellement par le terme "fléau de la dimension", dû à Richard Bellman [2]. Parmi les problèmes que posent la grande dimension des données ont peut citer les problème numériques, les problèmes d'inférence ou les problèmes de biais des estimateurs. Il a donc été nécessaire de développer ces dernières années des méthodes capables de pallier ces problèmes.

3 Les algorithmes du premier ordre utilisés dans cette étude

Le large spectre des méthodes d'apprentissage est de résoudre les problèmes d'optimisation qui en résulte à l'optimalité. Lorsque $A^T A$ est inversible (cas d'etude), nous avons une solution optimale unique sous forme: $x^* = \underset{x\in\mathbb{R}^d}{argmin} f(x)$. En effet, résoudre $\nabla f(x^*)$ conduit à $x^* = (A^T A)^{-1} A^T b$. Cependant, lorsque n et d sont grands (ce qui est le cas dans les problèmes de "big data" modernes), l'utilisation de cette formule est prohibitive d'un point de vue informatique, d'où la nécessité d'utiliser des algorithmes d'optimisation du premier ordre. Ainsi avec les caractéristiques de la fonction à optimiser, il est très intéressant d'utiliser les algorithmes du gradient ou du sous gradient.

3.1 L'algorithme du gradient

L'algorithme du gradient, aussi appelé algorithme de descente de gradient, désigne un algorithme d'optimisation différentiable. Il est destiné à minimiser une fonction réelle différentiable définie sur un espace euclidien (par exemple $\mathbb{R}^n$, l'espace des n-uplets de nombres réels, muni d'un produit scalaire) ou, plus généralement, sur un espace hilbertien. L'algorithme est itératif et procède donc par améliorations successives. Au point courant, un déplacement est effectué dans la direction opposée au gradient, de manière à faire décroître la fonction. il est souvent utilisé pour entraîner les modèles de Machine Learning, tels que les

réseaux de neurones, les modèles de régression linéaire, etc... Cette description montre que l'algorithme fait partie de la famille des algorithmes à directions de descente.

Principe de l'algorithme du gradient

Algorithm 1: Algorithme de la Descente de Gradient

Input: Point initial $x_0 \in \mathbb{R}^n$, pas initial α_0, critère d'arrêt
Output: Itéré x_{k+1}
Initialisation: Choisir x_0 comme point initial et α_0 comme pas initial;
while *le critère d'arrêt n'est pas satisfait* **do**
  Nouvel itéré $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$;
  Actualiser le pas de descente α_k ;
end

les critères d'arrêt couramment utilisés dans l'algorithme de descente de gradient sont :

1. **Norme du gradient :** Arrêtez l'algorithme lorsque la norme du gradient devient suffisamment petite.

   Exemple : $\|\nabla f(x_k)\| < \epsilon$, où ϵ est une petite valeur seuil.

2. **Nombre maximum d'itérations :** Limitez l'algorithme à un nombre prédéfini d'itérations.

   Exemple : Arrêtez après $k_{\max}$ itérations.

3. **Tolérance sur la variation de la fonction objectif :** Arrêtez lorsque la variation de la fonction objectif devient suffisamment petite.

   Exemple : $|f(x_{k+1}) - f(x_k)| < \epsilon$, où ϵ est un seuil défini.

Le choix du critère d'arrêt dépend souvent de la nature du problème, de la précision souhaitée et des ressources disponibles. Il peut être nécessaire d'expérimenter différents critères pour trouver celui qui convient le mieux à votre cas particulier.

Pour le choix du pas, il est possible de prendre un pas fixe α pour toutes les itérations, dans ce cas on a $\alpha_k = \alpha \quad \forall k$. Il existe des variantes de cet algorithme où à chaque itération on calcule un pas optimal qui permet de descendre beaucoup plus rapidement. Pour la résolution du problème de régression, on utilise l'algorithme de descente de gradient avec un pas de 0.1. Par la ensuite, on utilise un pas de $1/L$ (L : *Constante de gradient Lipschitz*) qui est le pas optimal pour le problème [5].

3.2 Accélération du gradient

La méthode de Momentum (la methode de la boule pesante) est une technique utilisée pour accélérer la convergence de l'algorithme de descente de gradient en ajoutant un terme de momentum à la mise à jour des paramètres[6]. Nous utilisons cette méthode d'accélération pour notre problème de régression. La mise à jour a chaque itération est donnée par la formule suivante :

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

où :

- α_k est le pas d'apprentissage, défini comme $\frac{4}{(\sqrt{L}+\sqrt{\mu})^2}$,
- β_k est le coefficient de momentum, défini comme $\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$,
- μ est la plus petite valeur propre de la matrice hessienne $\nabla^2 f(x)$, spécifiquement $\mu = \lambda_{\min}(A^T A)/n$,
- L est la plus grande valeur propre de la matrice hessienne $\nabla^2 f(x)$, spécifiquement $L = \lambda_{\max}(A^T A)/n$.

Le pas d'apprentissage α_k est adaptatif en fonction des valeurs propres de la matrice hessienne, favorisant une convergence plus rapide. Le coefficient de momentum β_k ajoute une "mémoire" à l'algorithme en tenant compte de la différence entre les itérations actuelle et précédente.

En exploitant les itérés précédents, le gradient accéléré améliore l'anticipation de la direction minimale, optimisant ainsi la convergence, notamment pour des fonctions peu fortement convexes.

3.3 L'agorithme du sous-gradient

L'algorithme du sous-gradient [7] est une variante de l'algorithme du gradient utilisé pour résoudre les problèmes de minimisation dont la fonction objectif est convexe mais non différentiable. L'algorithme génère les itérés $\{x_k\}$ à travers la formule de mise à jour suivante :

$$x_{k+1} = x_k - \alpha_k * \partial g_k$$

où ∂g_k est l'un des sous-gradients de la fonction f. Cet algorithme n'est pas toujours un algorithme de descente, il est donc plus judicieux de s'intéresser à mettre à jour une liste contenant la meilleure valeur de l'objectif à chaque itération.

Dans ce travail, nous utilisons l'algorithme du sous-gradient pour résoudre le problème de Régression LASSO où un terme de régularisation en norme L_1 est ajouté à notre problème initial :

$$f(x) = \frac{1}{2n}\|Ax - b\|_2^2 + \lambda\|x\|_1$$

À chaque itération k, on calcule le sous-gradient ∂g^k de la fonction f avec la formule :

$$\partial g^k = \frac{1}{n}\left((A^T A)x_k - A^T b\right) + \lambda * sg^k$$

où sg^k est le sous-gradient de $\|x\|_1$ défini par :

$$sg^k = \begin{pmatrix} sg_1^k(x) \\ \vdots \\ sg_n^k(x) \end{pmatrix}$$

tel que

$$sg_i^k(x) = \begin{cases} -1 & \text{si } x_i < 0 \\ 0 & \text{si } x_i = 0 \\ 1 & \text{si } x_i > 0 \end{cases}$$

.

$\lambda > 0$ est un hyper-paramètre.

Principe de l'algorithme du sous-gradient

Algorithm 2: Algorithme de la Descente de Sous-Gradient

Input: Point initial $x_0 \in \mathbb{R}^n$, pas initial α_0, critère d'arrêt
Output: Itéré x_{k+1}
Initialisation: Choisir x_0 comme point initial et α_0 comme pas initial;
while *le critère d'arrêt n'est pas satisfait* **do**
  Nouvel itéré $x_{k+1} = x_k - \alpha_k \partial f(x_k)$;
  Actualiser le pas de descente α_k ;
end

Dans nos expériences numériques pour cet algorithme, on choisira un pas α_k constant à valeur variable afin d'observer son influence sur la vitesse de convergence. On comparera aussi la performance de cet algorithme à celle de l'algorithme du gradient proximal.

3.4 L'algorithme du gradient proximal

L'algorithme du gradient proximal [4] représente une autre variante de l'algorithme du gradient, il est spécifiquement conçu pour résoudre des problèmes de minimisation où la fonction objectif s'exprime de la manière suivante :

$$f(x) = g(x) + h(x)$$

où g est une fonction convexe différentiable et h est convexe non différentiable. Les étapes de cet algorithme sont presques similaires à celles de l'algorithme du gradient, avec une différence lors du calcul de (x_{k+1}), où l'opérateur proximal $\text{prox}_{\alpha_k} h$ est appliqué à la formule de mise à jour.

Dans le cadre de notre travail, l'algorithme du gradient proximal est appliqué au problème de Régression LASSO, défini par :

$$g(x) = \|Ax - b\|_2^2$$

et

$$h(x) = \lambda \|x\|_1$$

La formule de mise à jour s'exprime de la manière suivante :

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

L'idée générale de cette algorithme est d'approcher la fonction g par une linéarisation, tout en conservant la fonction h inchangée, mettant en évidence l'expression du proximal $\text{prox}_{\alpha_k h}$, qui se définit pour notre problème de la manière suivante:

$$\text{prox}_{\alpha\lambda\|.\|}(x_i) = \begin{cases} x_i - \alpha & \text{si } x_i > \alpha \\ 0 & \text{si } x_i \in [-\alpha, \alpha] \\ x_i + \alpha & \text{si } x_i < -\alpha \end{cases}$$

Dans nos expériences numériques pour cet algorithme, nous utiliserons un pas fixe que nous ferrons varier pour observer son influence sur la vitesse de convergence et nous comparons la performance de cet algorithme à celle de l'algorithme du sous-gradient.

4 Variantes stochastiques des algorithmes

Les algorithmes stochastiques en apprentissage sont utilisés pour résoudre des problèmes d'optimisation lorsque la fonction objectif dépend de la somme ou de la moyenne de nombreuses fonctions individuelles, généralement liées aux données d'entraînement[1].

Le principe fondamental de ces algorithmes repose sur l'utilisation d'une estimation stochastique du gradient de la fonction objectif pour mettre à jour les paramètres du modèle. A chaque itération, le gradient n'est pas calculé sur l'ensemble complet des données mais plutôt sur un petit groupe aléatoire de données, souvent appelé lot (batch). Ces algorithmes offrent plusieurs avantages parmi lesquels:

- **Efficacité de calcul** : Le calcul du gradient sur l'ensemble complet des données peut être coûteux en termes de ressources. En utilisant les bacth, les calculs sont plus rapides, ce qui est crucial pour de grandes quantités de données.
- **Réduction de la variance**
- **Évitement de la stagnation** : La variabilité introduite par l'estimation stochastique peut empêcher l'algorithme de rester coincé dans des minima locaux, facilitant une exploration plus dynamique de l'espace des paramètres.
- **Prévention du surajustement** : L'utilisation d'un petit lot à chaque itération introduit du bruit dans le processus d'optimisation, aidant à prévenir le surajustement en fournissant une régularisation stochastique.

Pour notre travail, nous avons étudié la gradient stochastique dont le problème de minimisation de façon général s'exprime de la manière suivante:

$$\min_{x} F(x) = \mathbb{E}_{\epsilon}[f(x, \epsilon)]$$

où ϵ est une variable aléatoire.

L'algorithme du gradient stochastique se présente comme suit :

Algorithm 3: Algorithme du Gradient Stochastique

Initialisation: Choisir $x_0 \in \mathbb{R}^d$;
Répéter pour $k = 1, 2, 3, ...$
- choisir ϵ_k : Réalisation de ϵ ;
- $x_k = x_{k-1} - \alpha_k \nabla f(x_k, \epsilon_k)$ avec α_k bien choisi.;

Il existe plusieurs variantes de la méthode du gradient stochastique :

- **Momentum Stochastique :** Introduit un terme de momentum pour accélérer la convergence.
- **Adagrad (Adaptive Gradient Algorithm) :** Adapte le taux d'apprentissage en fonction de l'historique des gradients.
- **RMSprop (Root Mean Square Propagation) :** Adapte le taux d'apprentissage en normalisant les gradients.

- **Adam (Adaptive Moment Estimation) :** Combine les idées d'Adagrad et de RMSprop[3].
- **Mini-batch SGD :** Utilise des mini-lots de données pour estimer le gradient.

Dans nos expériences numériques pour cet algorithme, on choisit de comparer les comportements de la méthode du gradient et du gradient stochastique avec mini-batching sur des instances du problème de régression.

5 Expériences numériques

Dans cette section nous allons procéder à une description détaillée de chaque expérience réalisée pour chaque algorithme présenté ci-dessus. Ensuite nous présenterons les résultats obtenus avec des commentaires à l'appui.

5.1 Algorithme du gradient

Expérience 1: Influence du pas sur la vitesse de convergence

Dans cette section, nous explorons l'influence du taux d'apprentissage sur la rapidité de convergence. En ajustant le taux d'apprentissage, nous évaluons son impact sur la vitesse de convergence de l'algorithme. Plus précisément, en variant le choix parmi plusieurs taux, nous cherchons à identifier celui qui offre la meilleure performance. Nous effectuons la descente de gradient pour chaque taux sélectionné et comparons les résultats à travers des graphiques.

Résultat

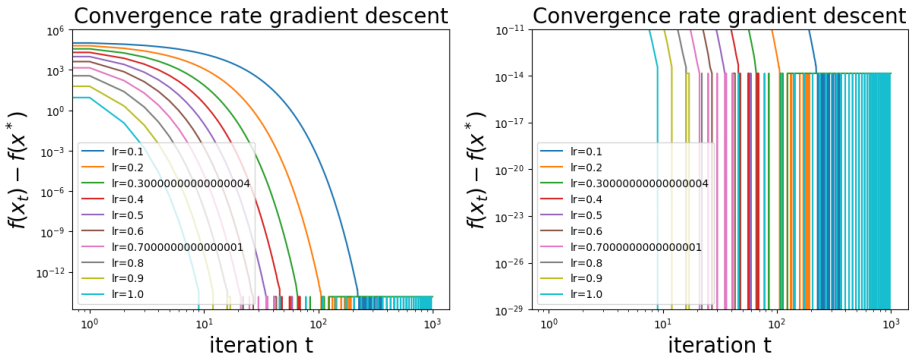


Figure 1: Convergence par rapport au learning rate (à gauche) et Oscillation après convergence (à droite)

Il apparaît que la convergence est plus rapide avec un taux d'apprentissage croissant situé entre 0.4 et 0.9. Le taux optimal dans cette expérimentation est de 0.9. Bien que le taux de 1.0 conduise à une convergence plus rapide, nous notons plusieurs oscillations. Ces oscillations sont limitées pour le taux de 0.9.

Expérience 2: Comparaison avec le learning rate optimal 1/L

Dans cette phase de l'étude, nous examinons l'influence de la taille du dataset sur le choix du taux d'apprentissage optimal, calculé avec la méthode 1/L. Nous optons pour un dataset de taille 1000, calculons le taux d'apprentissage optimal, puis choisissons deux autres taux pour les comparer en termes de vitesse de convergence. Nous reproduisons cette analyse avec un dataset de taille 10000.

Résultat

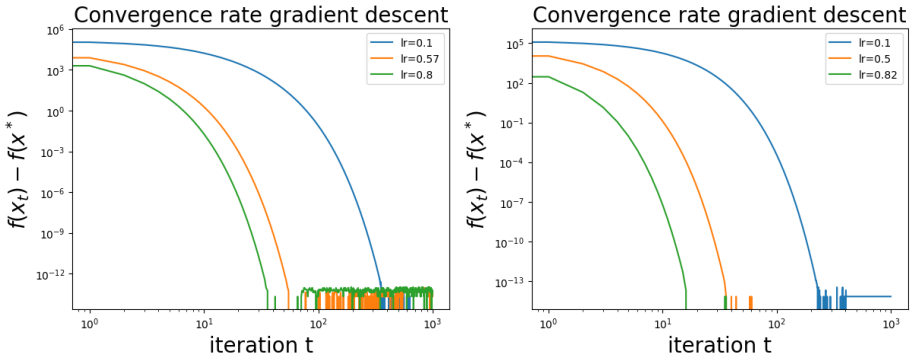


Figure 2: Comparaison avec le learning rate optimale pour un dataset de taille 1000 (à gauche) et taille 10000 (à droite)

Pour le premier graphique, relatif à un dataset de taille 1000 avec un taux optimal calculé de 0.57, nous constatons que le taux de 0.8 conduit à une meilleure convergence. Quant au deuxième graphique, pour un dataset de taille 10000, le taux optimal calculé de 0.87 se révèle être celui assurant la meilleure convergence. En conclusion, une taille de dataset plus importante semble favoriser une meilleure vitesse de convergence pour le taux optimal calculé.

5.2 Algorithme de l'accélération du gradient

Expérience: Comparaison du gradient accéléré par rapport à la descente de gradient

Dans cette expérience, nous collectons les valeurs objectives générées par l'algorithme du gradient accéléré pour les comparer avec celles obtenues lors de la descente de gradient classique. Cette analyse comparative vise à évaluer la performance relative des deux méthodes en termes de convergence et d'objectifs atteints.

Résultat

Nos observations indiquent que l'accélération du gradient conduit à une convergence plus rapide par rapport à la descente de gradient classique. Cette accélération peut être expliquée par le poids du momentum, qui favorise une descente plus rapide en pondérant le gradient. Cette constatation met en lumière l'effet positif de l'introduction du momentum dans le processus d'optimisation.

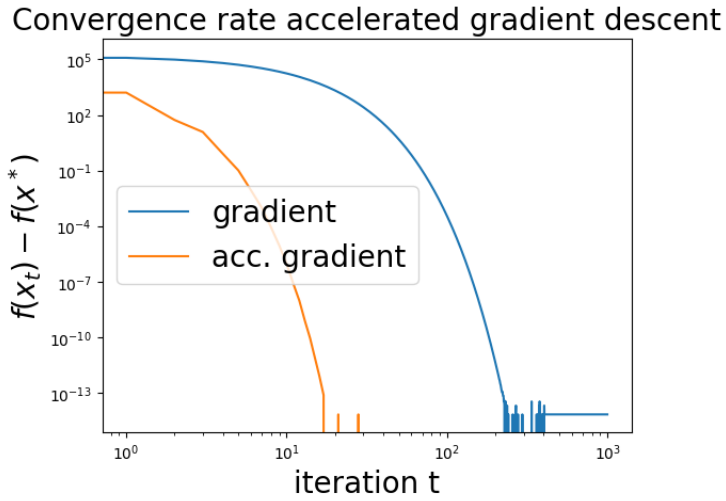


Figure 3: Comparaison de la vitesse de convergence du gradient accéléré par rapport à la descente de gradient

5.3 Algorithme de la descente du sous gradient

Expérience: Influence du pas sur la vitesse de convergence

Dans cette étape de l'expérimentation, nous ajustons le pas de convergence pour évaluer son influence sur la vitesse de convergence de la descente du sous-gradient. En variant le choix parmi plusieurs pas, notre objectif est de déterminer celui qui offre la meilleure performance. Pour ce faire, nous effectuons la descente du sous-gradient pour chaque pas sélectionné, puis comparons les résultats à travers des graphiques.

Résultat

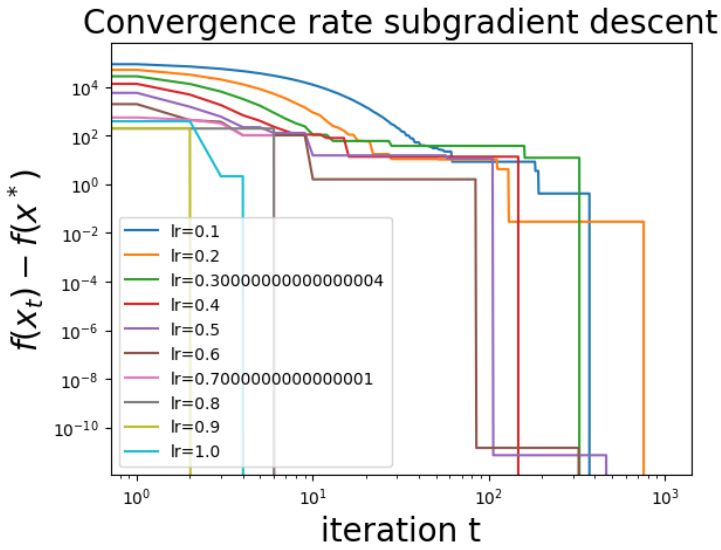


Figure 4: Influence du pas sur la vitesse de convergence

Nos observations mettent en évidence une meilleure convergence pour un taux d'apprentissage de 0.9.

5.4 Algorithme du gradient proximal

Expérience 1: Influence du pas sur la vitesse de convergence

Au cours de cette expérience, nous ajustons le taux de convergence pour évaluer son impact sur la rapidité de convergence de l'algorithme du gradient proximal. En variant le choix parmi plusieurs taux, notre objectif est de déterminer celui qui offre la meilleure performance. Nous effectuons la descente du gradient proximal pour chaque taux sélectionné et comparons les résultats à travers des graphiques.

Résultat

Dans le cadre de l'algorithme du gradient proximal, nos observations révèlent qu'un taux d'apprentissage de 0.775 offre une convergence optimale, cependant, les oscillations conséquentes nous incitent à ne pas le choisir. En revanche, un taux de 0.325 présente une convergence satisfaisante avec des oscillations réduites, ce qui en fait un choix plus judicieux malgré une convergence légèrement moins rapide.

Expérience 2: Comparaison de l'algorithme du sous-gradient et du gradient proximal

Dans cette phase de l'expérimentation, nous procédons à une comparaison entre l'algorithme du sous-gradient et l'algorithme du gradient proximal. Pour ce faire, nous ajustons le taux de convergence dans chaque algorithme et observons la vitesse de convergence ainsi que les résultats obtenus. Cette comparaison

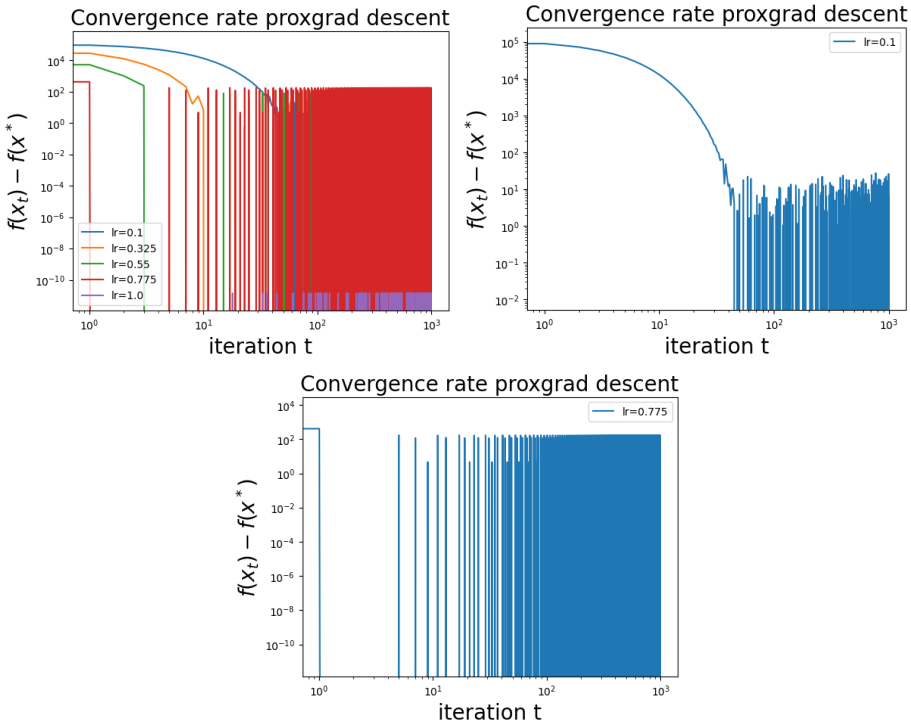


Figure 5: Influence du pas sur la vitesse de convergence

vise à évaluer la performance relative de ces deux approches dans le contexte spécifique de notre étude.

Résultat

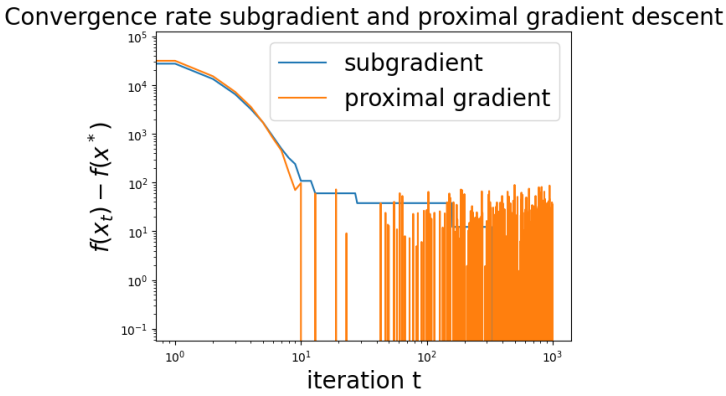


Figure 6: Comparaison de l'algorithme du sous-gradient et du gradient proximal

La comparaison entre l'algorithme du gradient proximal et celui du sous-gradient, avec un pas égal à 0.1 dans les deux cas, démontre que le gradient

proximal présente une convergence légèrement plus rapide. Ces observations confirment la tendance générale selon laquelle le sous-gradient converge moins rapidement par rapport au proximal. Toutefois, il est important de noter que dans certains cas particuliers, illustrés par des figures distinctes pour le sous-gradient et le proximal, le sous-gradient peut se révéler plus efficace en termes de performances. Ces nuances soulignent l'importance de considérer la nature spécifique du problème et les caractéristiques des données lors du choix de l'algorithme d'optimisation.

5.5 Algorithme du Gradient Stochastique

Expérience 1: Influence du pas sur la vitesse de convergence

Au cours de cette expérience, nous ajustons le taux de convergence afin d'évaluer son impact sur la rapidité de convergence de l'algorithme du gradient stochastique. En explorant divers choix de taux, notre objectif est de déterminer celui offrant la meilleure performance. La descente du gradient stochastique est réalisée pour chaque taux sélectionné, et les résultats sont comparés à travers des graphiques.

Résultats

Résultat

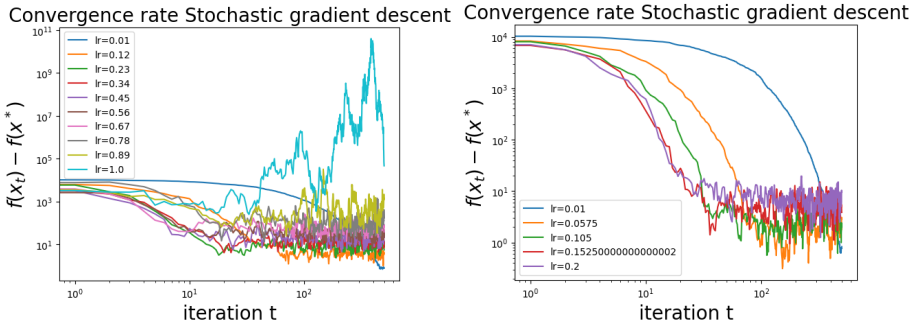


Figure 7: Influence du pas sur la vitesse de convergence

Les résultats obtenus mettent en évidence l'influence significative du taux sur l'algorithme. Un pas dépassant un seuil critique peut conduire à une divergence, tandis qu'en dessous d'une certaine valeur, l'influence sur la convergence est limitée. Une convergence similaire est observée dans la plage appropriée de choix de pas.

Expérience 2: Comparaison de l'algorithme du gradient stochastique et du gradient normal (full batch)

Dans cette phase de l'expérimentation, nous procédons à une comparaison entre l'algorithme du gradient stochastique et celui du gradient normal (full batch). Pour ce faire, nous ajustons le taux de convergence dans chaque algorithme et observons la vitesse de convergence ainsi que les résultats obtenus. Cette

comparaison vise à évaluer la performance relative de ces deux approches dans le contexte spécifique de notre étude.

Résultats

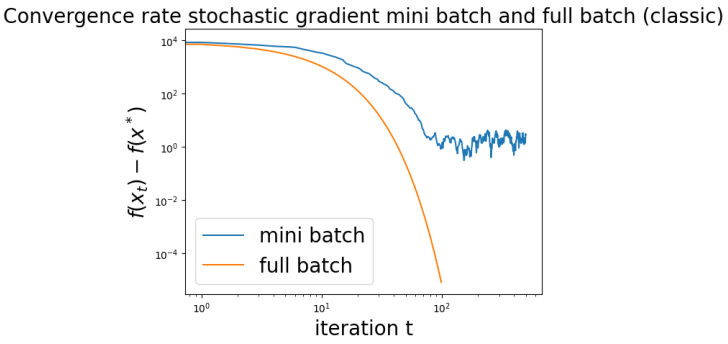


Figure 8: Comparaison de la convergence entre le gradient stochastique et le gradient normal (full batch)

Les résultats mettent en évidence une convergence beaucoup plus rapide du full batch par rapport à un mini-batch de taille 10 ou 100. Cette observation souligne l'influence significative de la taille du lot sur la vitesse de convergence de l'algorithme du gradient stochastique.

6 Extension au cas de la classification par support vector machines

Dans cette section, nous allons mettre en œuvre un algorithme de gradient stochastique pour résoudre le problème de classification par support vector machines(SVM).

Description du problème

Le problème classification par SVM consiste à minimiser la fonction objective suivante :

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

où C est un paramètre de régularisation, (x_i, y_i) sont les données d'entraînement, et w et b sont les paramètres du modèle SVM.

Algorithme

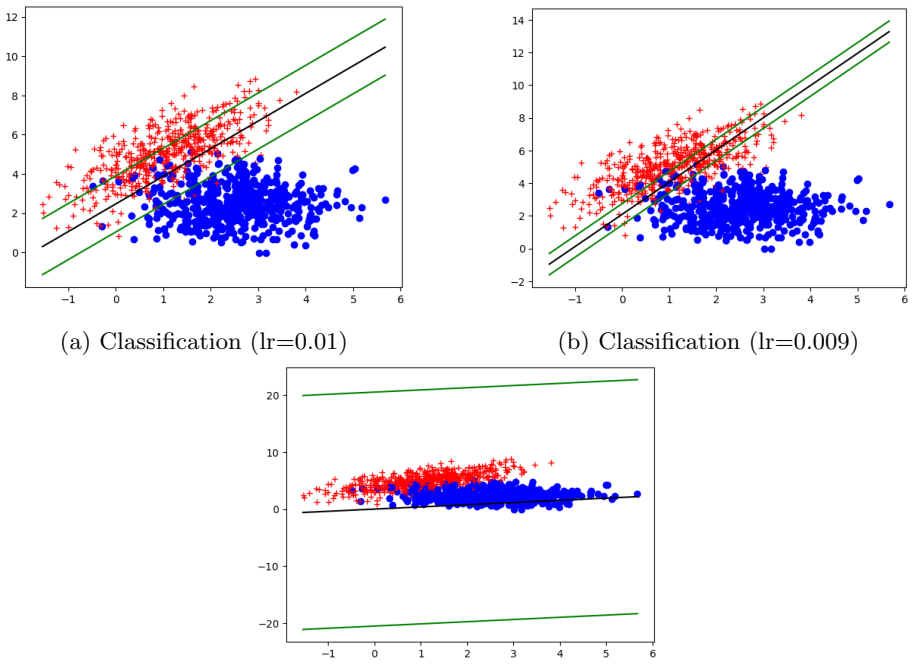
Algorithm 4: Algorithme de gradient stochastique pour le problème de classification par SVM

Input: Données (x_i, y_i), paramètre de régularisation C, nombre d'epocs $nmax$, La taille du lot(b_{size}), le pas lr
Output: Paramètres optimaux $\theta = (w, b)$
Initialiser les paramètres $\theta = (w, b)$;
for $t = 1$ **to** $nmax$ **do**
  for $i = 1$ **to** n/b_{size} **do**
    Sélectionner aléatoirement b_{size} indices.;
    Calculer le sous-gradient sur les indices sélectionnés;
    Mettre à jour θ;
  end
end
return θ;

Expérience numérique

Pour implémenter cet algorithme, nous avons généré nos données de taille 1000. Comme valeur pour nos paramètres nous avons $C = 10$, $nmax = 100$, $b_{size} = 100$, $lr = 0.01$ *et* 0.009.

Résultats



(a) Classification (lr=0.01)

(b) Classification (lr=0.009)

(c) Classification (lr=0.005)

Figure 9: Influence du pas sur le problème de classification par SVM

Ces observations soulignent l'importance critique du choix du pas dans le

contexte de la SVM. Un pas de 0.01 parvient à bien séparer les données en deux clusters distincts, à l'exception de quelques points. De même, un pas de 0.009 offre une séparation satisfaisante en resserrant un peu plus la classification. Cependant, une diminution du pas à 0.005 conduit à un élargissement soudain de la séparation, regroupant pratiquement tous les points dans un seul cluster.

Ainsi, la visualisation confirme l'influence significative du paramètre de pas sur la convergence du sous-gradient stochastique et la qualité de la séparation des données dans le contexte de la SVM.

7 Conclusion

En résumé, cette recherche offre une exploration approfondie des algorithmes d'optimisation, mettant l'accent sur leurs applications en apprentissage machine. À travers des expériences approfondies et des comparaisons, nous soulignons l'importance des choix de paramètres, tels que les taux d'apprentissage, pour atteindre une convergence optimale. Les résultats apportent des informations précieuses aux praticiens et chercheurs pour choisir des approches d'optimisation adaptées à divers scénarios.

Annexes

Vous trouverez les notebooks sur le site de l'ENAC accompagné de ce rapport ou encore via ce lien GitHub.
https://github.com/MarcoPerson/OMA.

References

[1] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[2] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Parikh Neal and Boyd Stephen. *Proximal Algorithms*. SIAM, 2014.

[5] COUELLAN Nicolas. Cours Optimastion Mathematiques pour l'Apprentissage. 2024.

[6] Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[7] N. Shor. *Minimization Methods for Non-differentiable Functions*. Springer Series in Computational Mathematics. Springer, 1985.