

TP 1 - Descente de gradient

L'objectif de ce TP est d'expérimenter la descente de gradient dans le cadre de la régression linéaire et du coût de classification hinge loss, vus en TD. Dans toute la suite, l'espace d'entrée est de d dimensions, les labels sont des réels (dans le cas de la régression linéaire) ou dans $\{-1, 1\}$ (dans le cas de la classification), l'espace de recherche fonctionnel est considéré paramétré par un vecteur de poids $\mathbf{w} \in \mathbb{R}^d$ et n dénotera le nombre d'exemples. Nous ne considérerons pas de biais dans ce TP (pas de poids \mathbf{w}_0).

Attention !

Toutes vos fonctions dans la suite doivent pouvoir prendre en entrée des matrices d'exemples et des vecteurs de labels - et non pas uniquement un exemple et un label. Nous suivrons les conventions suivantes : $X \in \mathbb{R}^{n,d}$, $\mathbf{w} \in \mathbb{R}^{d,1}$, $Y \in \mathbb{R}^{n,1}$.

Il y a quelques pièges pour la manipulation des matrices avec `numpy` :

- l'opérateur `*` permet de multiplier termes à termes deux matrices de même dimension, mais parfois il fait le produit matriciel lorsque les matrices n'ont pas des tailles compatibles (par exemple $(1, d)$ et $(d, 1)$) !
- l'opérateur `@` permet de faire la multiplication matricielle (attention aux dimensions!).
- faites le moins possible de boucles (aucune boucle n'est requise dans l'implémentation des fonctions de coût et des gradients!). Python est très lent dans ce cas ...
- parfois vous passerez une matrice en entrée, parfois un vecteur (lorsque vous ne sélectionnez qu'une ligne des exemples), or les opérateurs n'auront pas le même comportement selon les cas ... Pensez à transformer vos entrées au tout début de toutes vos fonctions afin d'éviter les bugs (par exemple : `y = y.reshape(-1,1)`; `w = w.reshape(-1,1)`; `x = x.reshape(y.shape[0],w.shape[0])`)
- Penser à utiliser `np.where`, `np.sign` et `np.maximum`

Implémentation des fonctions de coût

Implémentez (sans utiliser aucune boucle! chaque fonction fait 1 à 2 lignes) :

- une fonction `mse(w,x,y)` qui renvoie le coût aux moindres carrés pour une fonction linéaire de paramètre \mathbf{w} sur les données \mathbf{x} (de taille n, d) et les labels \mathbf{y} . Votre fonction devra sortir le coût sous la forme d'une matrice de taille $n, 1$ (le coût pour chaque exemple).
- une fonction `hinge(w,x,y)` qui renvoie le coût de classification de type hinge-loss (le coût du perceptron) : $\Delta(\hat{y}, y) = \max(0, 1 - y \times \hat{y})$.
- les fonctions `mse_grad(w,x,y)` et `hinge_grad(w,x,y)` qui renvoient le gradient des moindres carrés et du coût hinge sous la forme d'une matrice n, d .

Vous pouvez tester vos fonctions avec la fonction `check_fonctions()`.

Descente de gradient

Implémentez une fonction `descente_gradient(datax, datay, f_loss, f_grad, eps, iter)` qui réalise une descente de gradient pour optimiser le coût `f_loss` (dont le gradient est donné par `f_grad`) sur les données `datax` et les labels `datay`, avec un pas de descente de `eps` et `iter` itérations. Votre fonction devra renvoyer le paramètre optimal \mathbf{w} trouvé, la liste des \mathbf{w} et les valeurs de la fonction de coût au fur et à mesure des itérations.

Expérimentations

Vous trouverez dans le fichier `mltools.py` une fonction `gen_arti(nbex=1000,data_type=0,epsilon=0.02)` qui permet de générer des jeux de données de `nbex` points selon : 2 gaussiennes (`data_type=0`), 4 gaussiennes (`data_type=1`) et un échiquier (`data_type=2`) avec un bruit `epsilon`. Vous avez également les fonctions :

- `plot_data(data,labels)` qui permet de visualiser les données ;
- `plot_frontiere(data,f,step)` qui permet de tracer les frontières de décision de la fonction `f` pour un problème 2D en discrétisant l'espace en `step` intervalles ;
- `make_grid(data,xmin,xmax,ymin,ymax,step)` qui permet de construire une grille de discrétisation d'un espace 2D (en considérant comme bornes de l'espace soit les minimums et maximums de `data`, soit les valeurs passées en paramètre).

Des exemples d'utilisation sont données dans le squelette du code du TP.

Testez votre implémentation de la descente du gradient sur le problème à deux gaussiennes. Comparez les résultats obtenus entre la régression linéaire et le coût de hinge. Visualisez les frontières de décision et tracez l'évolution du coût en fonction des itérations. Que se passe-t-il lorsque le pas de gradient est augmenté ou diminué fortement ? Dans le cas d'un problème séparable et dans le cas d'un problème non séparable (par exemple en augmentant le bruit fortement).

Visualisez la fonction de coût dans l'espace des poids selon les deux dimensions du problème (cf squelette du code du TP). Tracez sur le même graphique la trajectoire suivie par l'algorithme d'optimisation (les `w` au fur et à mesure des itérations).

Expérimentez avec et sans l'ajout d'un biais.

Expérimentez sur les autres types de données artificielles.

Bonus : Expérimenter l'effet d'une régularisation L2 (ajout de la norme des paramètres dans le coût à minimiser), en observant l'impact du poids λ associé sur les frontières obtenues.