TP 4 - Perceptron, SVMs

Consignes

- Pour ce TP, vous travaillerez par équipe de deux, et rendrez un seul rapport pour l'équipe.
- Vous pouvez rendre au choix [code + pdf] ou notebook ipython.

Intro Le perceptron est un modèle connexionniste inspiré des neurosciences : il modélise de manière très simpliste le fonctionnement d'un neurone. Le neurone dispose de d signaux d'entrées $x_i \in \mathbb{R}$ - le nombre de dimension de l'espace de description des exemples - et dans le cas simple de la classification binaire, une sortie y. L'interaction entre les entrées et la sortie est supposée linéaire : $y = \sum_{i=0}^{d} w_i x_i$ (la dimension 0 permet l'ajout d'un signal constant - le biais). Nous noterons par la suite $f_{\mathbf{w}} : \mathbb{R}^{d+1} \to \mathbb{R}$ la fonction linéaire paramétrée par le vecteur $\mathbf{w}$. Dans le cas de la classification binaire dans un ensemble $\{-1, 1\}$, la décision se fait selon le signe de y ; $+1$ si $f_{\mathbf{w}}(\mathbf{x}) > 0$, -1 sinon.
L'algorithme d'apprentissage du perceptron consiste à corriger itérativement les poids afin de réduire l'erreur sur la base d'exemples :

1. Initialiser le vecteur de poids $\mathbf{w}$ aléatoirement
2. Répéter jusqu'à convergence : pour chaque couple $(\mathbf{x}^n, y^n)$ de la base d'apprentissage :
   - calculer $f_{\mathbf{w}}(\mathbf{x}^n)$
   - Si $f_{\mathbf{w}}(\mathbf{x}^n)y^n > 0$, alors ne rien faire (pas d'erreur sur l'exemple)
   - Sinon, mettre à jour les poids : $\mathbf{w}^{t+1} \leftarrow \mathbf{w} + \epsilon y^n \mathbf{x}^n$, ϵ un pas d'apprentissage.

Ce modèle est équivalent dans sa formulation à l'optimisation d'un classifieur linéaire avec une fonction de coût *hinge loss* : $\ell^{\alpha}(y, f_{\mathbf{w}}(x)) = max(0, \alpha - y f_{\mathbf{w}}(x))$ avec $\alpha = 0$. L'algorithme de résolution correspond à une variante d'une descente de gradient stochastique sur ce problème d'optimisation car les instances sont utilisées de façon cyclique, au lieu d'être tirées au hazard parmis la base d'entraînement.

1 Implémentation du perceptron

Le fichier source du TP contient un squelette de code pour le perceptron. Il est très proche du TP précédent sur la descente de gradient.

Q 1.1 Implémenter les fonctions `hinge(w,data,y)` et `grad_hinge(w,data,y)` qui calculent le coût hinge et le gradient du coût pour le vecteur de paramètres `w`, la matrice d'exemples `data` et le vecteur de labels `y`. Votre code doit fonctionner dans le cas où les différents paramètres sont des vecteurs et des matrices (utiliser `np.reshape` pour vous assurez de la nature des paramètres). Transformer de préférence vos entrées de la manière suivante :
`data,y,w=data.reshape(len(y),-1),y.reshape(-1,1),w.reshape(1,-1)`. Penser à utiliser `np.sign` et `np.maximum`.

Q 1.2 Tester sur les données à deux gaussiennes (utiliser la fonction `gen_arti()`). On prendra $\epsilon = 0.001$ et on considerera l'apprentissage terminé après 1000 itérations.

Q 1.3 Tracer la trajectoire de l'apprentissage dans l'espace des poids et les frontières obtenues dans l'espace de représentation des exemples. La solution trouvée est-elle unique ? La frontière de décision trouvée est-elle pertinente ? Comparer dans le cas où les données sont bruitées (en particulier dans le cas non linéairement séparable).

Q 1.4 (bonus) Trois variantes de la descente de gradient sont très utilisées :

- descente batch : le gradient est calculé sur tous les exemples (comme moyenne des gradients associés à chaque instance), puis la mise à jour est effectuée sur les poids. Une prise en compte de tous les exemples une fois s'appelle *une passe*, *'epoch'*.

- descente stochastique : à chaque époque, l'ordre des exemples est tiré aléatoirement. La mise à jour des poids est faîte après traitement de chaque exemple dans cet ordre aléatoire.
- mini-batch : l'ensemble d'apprentissage est partitionné aléatoirement en k sous-ensembles appelés bloc ; la mise à jour est faîte après traitement de chaque bloc d'exemples. Si $k = 1$, c'est équivalent à une descente stochastique ; si $k = N$, c'est équivalent à une descente batch.

Implémenter les trois variantes. Tester en fonction du bruit (paramètre sigma dans la fonction `gen_arti()`) sur les données les performances en terme de pourcentage d'instances bien classifiées et la vitesse de convergence (par rapport au nombre de passes).

Q 1.5 (bonus) Pas de gradient variable : plutôt que de garder un pas constant tout au long des itérations, des variantes le font décroître en fonction du nombre d'itérations. Implémenter une décroissance linéaire et une décroissance quadratique (en $\frac{1}{t^2}$). Comparer les variantes en particulier dans le cas non séparable.

2 Données USPS

Se reporter au TP précédent pour charger les données USPS et les fonctions utiles.

Q 2.1 Sur quelques exemples de problèmes 2 classes (6 vs 9, [1,2] vs [6,8] par exemple), entraîner votre perceptron et visualiser la matrice de poids obtenue.

Q 2.2 En utilisant les données de test, tracer les courbes d'erreurs en apprentissage et en test en fonction du nombre d'itérations. Observez-vous du sur-apprentissage ? Expérimenter dans différents settings : 1 chiffre contre 1, 1 chiffre contre tous, un sous-ensemble de chiffres contre un autre.

3 Expressivité et feature map

Q 3.1 Tester votre perceptron sur les autres données artificielles fournies (toujours avec la fonction `gen_arti`). Que remarquez vous ? Est-ce normal ?

Afin d'augmenter l'expressivité des classifieurs linéaires, une opération très courante est de plonger les données dans un espace de dimension supérieure. Soit $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ une fonction de plongement (feature map), l'apprentissage est fait alors sur l'ensemble $\{\phi(\mathbf{x}^n), y^n\}$ avec un vecteur de poids $\mathbf{w}$ de dimension d'.

La variable `projection` de la classe `Perceptron` permet de pré-traiter les données en appliquant une fonction contenue dans la variable avant tout traitement.

Q 3.2 Plongement polynomiale dans le cas 2D

Soit le plongement (en 2D) polynomiale suivante : $poly(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. A quoi devraient correspondre les frontières de décision dans l'espace projeté ? Implémenter et tester.

Q 3.3 Plongement gaussien

Soit $B = \{o^i \in \mathbb{R}^d\}_{i=1}^{N_b}$ un ensemble de points dans l'espace de description des exemples. Soit $gauss_B(\mathbf{x}) = (k(\mathbf{x}, o^1), k(\mathbf{x}, o^2), \ldots, k(\mathbf{x}, o^{N_b}))$ avec $k(x, x') = e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$.

A quoi correspond la i-ème dimension dans l'espace projeté ? Que signifie une valeur faible ou forte ? A quoi correspondent dans ce cas les poids w^i ? Quelle est la signification d'un poids nul ? d'un poids positif élevé ? Coder et expérimenter en tirant 1) aléatoirement des points dans l'espace de description 2) en utilisant un sous-échantillon de l'ensemble d'apprentissage comme base de projection.

Q 3.4 Quelle est l'expressivité de ce modèle par rapport au modèle linéaire ? Comment évolue-t-elle en fonction du nombre de dimensions nulles de $\mathbf{w}$? Afin de *régulariser* le modèle, on ajoute au problème d'optimisation un terme en $\|\mathbf{w}\|^2$, la nouvelle fonction de coût est : $L(f_{\mathbf{w}}(x), y) = hinge(\mathbf{w}, y) + \lambda\|\mathbf{w}\|^2$. Que change cette formulation ? Proposer une méthode d'optimisation et tester.