Gradient Stochastique et Perceptron

Les fichiers Python `tp_perceptron_source.py` et `tp_perceptron_script.py` sont disponibles sur le site du cours. Ils contiennent le code et les fonctions utiles pour ce TP.

- Découverte de Python -

Consulter les pages suivantes pour démarrer ou bien trouver quelques rappels de Python :

⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/liesse_python/1-Intro-Python.html

⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/liesse_python/2-Numpy.html

⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/liesse_python/3-Scipy.html

⋆⋆⋆ http://scikit-learn.org/stable/index.html

⋆⋆ http://www.loria.fr/~rougier/teaching/matplotlib/matplotlib.html

⋆⋆ http://jrjohansson.github.io/

- Introduction -

Définitions et notations

On rappelle ici le cadre de la classification binaire supervisée, et l'on présente les notations que l'on utilisera :

- $\mathcal{Y}$ l'ensemble des étiquettes des données (*labels* en anglais). On traite le cas binaire : il y a donc deux classes. Il est confortable de raisonner avec $\mathcal{Y} = \{-1, 1\}$ pour représenter les étiquettes, car on va considérer des signes dans au cours de ce travail. [1]
- $\mathbf{x} = (x_1, \ldots, x_p)^\top \in \mathcal{X} \subset \mathbb{R}^p$ est une observation, un exemple, un point (ou un *sample* en anglais). La $j$ème coordonnée de $\mathbf{x}$ est la valeur prise par la $j$ème variable (*feature* en anglais).
- $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots n\}$ un ensemble d'apprentissage contenant n exemples et leurs étiquettes,
- Il existe un modèle probabiliste qui gouverne la génération des nos observations selon des variables aléatoires X et Y : $\forall i \in \{1, \ldots, n\}, (\mathbf{x}_i, y_i) \overset{i.i.d}{\sim} (X, Y)$.
- On cherche à construire à partir de l'ensemble d'apprentissage $\mathcal{D}_n$ une fonction appelée classifieur, $\hat{f} : \mathcal{X} \mapsto \{-1, 1\}$ qui pour un nouveau point $\mathbf{x}_{\text{new}} \in \mathcal{X}$ fournit une étiquette $\hat{f}(\mathbf{x}_{\text{new}})$.

Génération artificielle de données

Dans un but d'expérimentation, il est plus aisé de travailler sur des données générées artificiellement. On considère afin de faciliter la visualisation que les variables explicatives (*features*, en anglais) sont de dimension deux. Cela consiste à prendre $p = 2$ dans le formalisme ci-dessus.

1. Étudiez la fonction `rand_gauss(n,mu,sigmas)` qui engendre n observations selon la loi normale multi-dimensionnelle de moyenne le vecteur `mu` et de matrice de covariance la matrice diagonale de diagonale `sigmas` $= [\sigma_1, \sigma_2]$, *i.e.,* la matrice de variance covariance est : $\begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}$. Appliquez ensuite les fonctions `rand_bi_gauss`, `rand_clown` et `rand_checkers` pour différents paramètres d'entrée `n,mu,sigma`. Que renvoient ces fonctions ? À quoi correspond la dernière colonne ?

1. Noter que pour d'autres méthodes, il peut-être plus pratique de prendre $\mathcal{Y} = \{0, 1\}$.

2. Conservez quelques jeux de données afin de les utiliser dans la suite : pour chacun, il faudra sauver sous forme d'un tableau numpy à deux colonnes `dataX` les données, et dans une vecteur `dataY` les labels correspondants à chaque exemple.
3. Utilisez la fonction `plot_2d` disponible dans `tp_perceptron_source.py` et qui permet de visualiser quelques jeux de données en fonction des étiquettes associées. Changer éventuellement la couleur.

- Perceptron -

Les classifieurs linéaires (affines)

Un classifieur linéaire est un classifieur linéaire qui associe à chaque observation $\mathbf{x}$ une étiquette dans $\mathcal{Y}$ (ici $\mathcal{Y} = \{-1, 1\}$) selon sa position par rapport à un hyperplan affine. L'ensemble des classifieurs linéaires est donc lié à l'ensemble des hyperplans (affines) de $\mathbb{R}^p$ que l'on définit pour un certain vecteur directeur (aussi dit vecteur normal) $\mathbf{w} = (w_0, w_1, \cdots, w_p)^\top \in \mathbb{R}^{p+1}$ par

$$H_{\mathbf{w}} = \left\{ \mathbf{x} \in \mathbb{R}^p : \hat{f}_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^{p} w_i x_i = 0 \right\}.$$

Pour classer une observation $\mathbf{x}$ (*i.e.,* affecter une étiquette 1 ou -1) on utilise alors $\text{sign}\left(\hat{f}_{\mathbf{w}}(\mathbf{x})\right)$, où la fonction sign est définie par

$$\text{sign}(x) = \begin{cases} 1 & \text{si } x \geq 0, \\ -1 & \text{si } x < 0 \end{cases}.$$

Ainsi, la fonction $\mathbf{x} \mapsto \text{sign}(\hat{f}_{\mathbf{w}}(\mathbf{x}))$ est un classifieur binaire avec une frontière linéaire (affine) définie par $\mathbf{w}$.

L'objectif du perceptron est de trouver un hyperplan qui sépare le mieux possible les données en deux groupes. On aimerait donc que de chaque côté de l'hyperplan séparateur, les étiquettes soient le plus possible homogènes. Le vecteur $\mathbf{w}$ est appelé vecteur de poids. Le coefficient w_0 est l'ordonnée à l'origine (*intercept* en anglais). Pour plus de détails sur l'intérêt de cette méthode on pourra se référer à [HTF09].

Nous supposerons dans cette partie introductive que nos données sont en deux dimensions ($p = 2$). Utilisez les données artificielles déjà construites pour illustrer les questions suivantes.

1. A quoi correspond la frontière de décision du perceptron ? Trouvez (à la main) une bonne séparatrice. Quand est-ce que $\hat{f}_{\mathbf{w}}(\mathbf{x})$ est grand ? négatif ? positif ? Quelle est la signification géométrique de cette fonction ? À quoi correspond w_0 ?
2. Vérifiez que la fonction `predict(x,w)` prend en entrée un vecteur $\mathbf{x} \in \mathbb{R}^p$ et un vecteur poids $\mathbf{w} \in \mathbb{R}^{p+1}$ renvoie le vecteur de prédiction $\hat{f}_{\mathbf{w}}(\mathbf{x})$. Verifiez ensuite que `predict_class(x,w)` renvoie bien l'étiquette prédite $\text{sign}\left(\hat{f}_{\mathbf{w}}(\mathbf{x})\right)$.

Fonction de coût

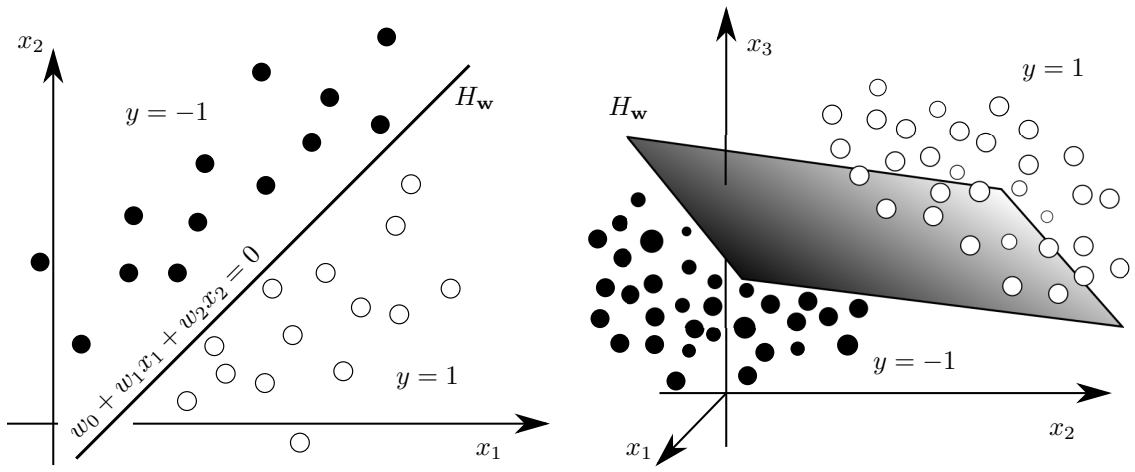
Afin de mesurer l'erreur commise sur l'ensemble d'un jeu de données $\mathcal{D}_n$ il est nécessaire de se fixer une fonction de perte $\ell : \mathbb{R} \times \mathcal{Y} \mapsto \mathbb{R}^+$ qui mesure le coût d'une erreur lors de la prédiction d'un exemple. Le coût que l'on veut minimiser est $C_{\mathbf{w}} = \mathbb{E}\left[\ell(\hat{f}_{\mathbf{w}}(\mathbf{x}), y)\right]$, l'espérance de la fonction de perte sur l'ensemble des données. Trois fonctions de perte sont utilisées habituellement et définies ci-dessous :

— le pourcentage d'erreur : $\text{ZeroOneLoss}(\hat{f}_{\mathbf{w}}(\mathbf{x}), y) = |y - \text{sign}(\hat{f}_{\mathbf{w}}(\mathbf{x}))|/2$

— l'erreur quadratique : $\text{MSELoss}(\hat{f}_{\mathbf{w}}(\mathbf{x}), y) = (y - \hat{f}_{\mathbf{w}}(\mathbf{x}))^2$

— l'erreur *hinge* (*i.e.,* charnière en français) : $\text{HingeLoss}(\hat{f}_{\mathbf{w}}(\mathbf{x}), y) = \max(0, 1 - y \cdot \hat{f}_{\mathbf{w}}(\mathbf{x}))$

Cette partie à pour but d'étudier ces différentes fonction de pertes. Ces trois fonctions sont codées dans le fichier source (ainsi que les gradients associés)



(a) $p = 2$ (b) $p = 3$

Figure 1 – Exemple de séparation de classe (1 pour les points blanc, -1 pour les noirs) par des hyperplans affines, en dimension $p = 2$ et $p = 3$

3. Fixer momentanément un choix de $\mathbf{x}$ et $\mathbf{w}$ et afficher le graphe des fonctions réelles :

$$\begin{aligned} \mathbb{R} &\to \mathbb{R} \\ y &\mapsto \ell(\hat{f}_{\mathbf{w}}(\mathbf{x}), y) \end{aligned}$$

pour les trois choix de fonctions de pertes décrites ci-dessus.

4. Maintenant si $\mathbf{x}$ et y sont supposés fixe. Quelle est la nature des fonctions

$$\mathbb{R}^{p+1} \to \mathbb{R}^+ \tag{1}$$

$$\mathbf{w} \mapsto \hat{f}_{\mathbf{w}}(\mathbf{x}) \quad \texttt{l(y, fw(x))} \tag{2}$$

pour les trois pertes étudiées : constante, linéaire, quadratique, constante par morceaux, linéaire par morceaux, quadratique par morceaux ?

- Algorithme de descente de gradient stochastique -

Dans le cas général, il est bien sûr impossible de faire une recherche exhaustive de l'espace $\mathbb{R}^{p+1}$ où évolue $\mathbf{w}$ afin de trouver le minimum. La méthode du perceptron propose pour cela d'utiliser une variante de l'algorithme de descente du gradient, une méthode usuelle et générale d'optimisation de fonction différentiable. La méthode est itérative : à chaque pas, le point courant est corrigé dans la direction du gradient, mais en sens opposé. L'algorithme converge dans le cas général vers un minimum local pour peu que le pas soit bien choisi. Le minimum atteint est global en particulier pour les fonctions convexes. La méthode de gradient stochastique propose de ne pas pas utiliser le gradient complet, qui requiert de calculer une somme sur les n observations, mais plutôt de tirer (aléatoirement ou non) une valeur sur laquelle on calcul un gradient

L'algorithme du perceptron est décrit de la manière suivante :

5. Proposer une variante "Perceptron version aléatoire" qui visite les observations en effectuant un tirage aléatoire uniforme (faire le cas avec remise ou sans remise au choix). C'est cette version qui est généralement appelée méthode de descente de gradient stochastique. Remarquer que c'est celle qui est proposée par défaut dans la fonction `gradient`.
6. On va observer graphiquement avec `plot_cout3d` l'évolution de $\frac{1}{n}\sum_{i=1}^{n} \ell(\hat{f}_{\mathbf{w}}(\mathbf{x}_i), y_i)$ selon $\mathbf{w}$. En réalité on n'affiche que la dépendance en w_1 et w_2 dans la fonction `plot_cout3d`. Pourquoi ne peut-on pas afficher la dépendance en w_0 ?

Algorithme 1 : Perceptron (Version cyclique)

Data : les observations et leurs étiquettes $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq n\}$; le pas de gradient : ϵ ; le nombre maximal d'itérations : n_{iter} ;
Result : $\mathbf{w}$
initialiser (aléatoirement) $\mathbf{w}$; initialiser $j = 0$
while $j \leq n_{\text{iter}}$ **do**
  $\mathbf{w} \leftarrow \mathbf{w}$
  for $i = 1$ *to* n **do**
    $\mathbf{w} \leftarrow \mathbf{w} - \epsilon \nabla_{\mathbf{w}} \ell(\hat{f}_{\mathbf{w}}(\mathbf{x}_i), y_i)$
  $j \leftarrow j + 1$

7. La fonction `frontiere` permet d'afficher des couleurs différentes selon la classe choisie par un classifieur. Un exemple est donné pour faire afficher la prédiction proposée par la méthode `SGDClassifier` de `sklearn` (dont une description est donnée sur la page : http://scikit-learn.org/stable/modules/sgd.html)

```
from sklearn.linear_model import SGDClassifier
```

   Note : SGD est abréviation *Stochastic Gradient Descent*)
   Adapter cet exemple pour visualiser la frontière de votre implémentation.
8. Expérimentez sur différents jeux de données : utiliser soit les fonctions fournies dans le fichier source, soit la fonction de `sklearn`. Étudiez les performances selon les points suivants : le nombre d'itérations, la fonction de coût, la difficulté du problème (si les classes sont facilement séparables ou non par un hyperplan). Observez-vous des comportements étranges, si oui quelle en est la raison ?
9. La descente de gradient simple est aussi appelée *batch* (*i.e.,* lot ou stock en français), et consiste à calculer le vrai gradient $\frac{1}{n}\sum_{i=1}^{n} \nabla_{\mathbf{w}} \ell(\hat{f}_{\mathbf{w}}(\mathbf{x}_i), y_i)$. Étudiez comme précédemment le comportement de l'algorithme, en remarquant que l'option `stoch=True` peut-être désactivé.
10. Question optionnelle : Étudiez numériquement la vitesse de convergence dans le cas suivant : les $\mathbf{x}_i$ sont des points uniformément repartis sur les segments $\{0\} \times [0, M]$ (alors les y_i valent -1) ou bien sur le segment $\{\delta\} \times [0, M]$ (alors les y_i valent -1). De plus la proportion de 1 est égal à $1/2$. On regardera l'impact de δ, M et n sur le temps de convergence du perceptron.
11. Proposez des variantes sur les conditions d'arrêt de l'algorithme.
12. Quel est le principal problème du perceptron ?
13. Question optionnelle : Trouver un fonction de perte telle que l'algorithme du perceptron soit équivalent à la version donnée par l'Algorithme 2 (qui est la version initiale de l'algorithme). Interprétez la condition suivante : $\hat{f}_{\mathbf{w}}(\mathbf{x}_i) \cdot y_i \leq 0$.

Algorithme 2 : Perceptron "classique" (Version cyclique)

Data : les observations et leurs étiquettes $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq n\}$; le pas de gradient : ϵ ; le nombre maximal d'itérations : n_{iter} ;
Result : $\mathbf{w}$
initialiser (aléatoirement) $\mathbf{w}$; initialiser $j = 0$
while $j \leq n_{\text{iter}}$ **do**
  $\mathbf{w} \leftarrow \mathbf{w}$
  for $i = 1, \cdots, n$ **do**
    if $\hat{f}_{\mathbf{w}}(\mathbf{x}_i) \cdot y_i \leq 0$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \epsilon(1, \mathbf{x}_i) \cdot y_i$
  $j \leftarrow j + 1$

Pour aller plus loin sur le point de vue classique on peut consulter hagan.okstate.edu/4_Perceptron.pdf. Un point de vue plus moderne sur la technique du gradient stochastique est proposé par [Sha11].

Perceptron : linéaire... vraiment ?

14. Quelle est la formule analytique d'une ellipse, d'une hyperbole et d'une parabole en 2D ?
15. Proposez une méthode pour réussir à classifier le jeu de données `clown`. Peut-on généraliser au delà ? On pourra utiliser la fonction `poly2` du fichier source associé au TP, qui plonge les données bi-dimensionnel dans l'espace des fonction polynomiale de degré 2 en les données. Comment l'utiliser pour apprendre un perceptron ?
16. Sur le jeu de données `clown`, faites quelques expériences en transformant vos données et tracez les frontières de décision.

Références

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. http://www-stat.stanford.edu/~tibs/ElemStatLearn/. 2

[Sha11] S. Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2) :107–194, 2011. http://www.cs.huji.ac.il/~shais/papers/OLsurvey.pdf. 4