A2DI - TP n°3

Dans ce troisième TP, nous étudierons l'évolution de l'erreur de train par rapport à celle de l'erreur de test. Nous pourrons vérifier que pour l'algorithme du perceptron, ces deux erreurs ont un comportement similaire nous permettant de conclure que cet algorithme est suffisamment simple pour éviter de tomber dans les dangers du sur-apprentissage.

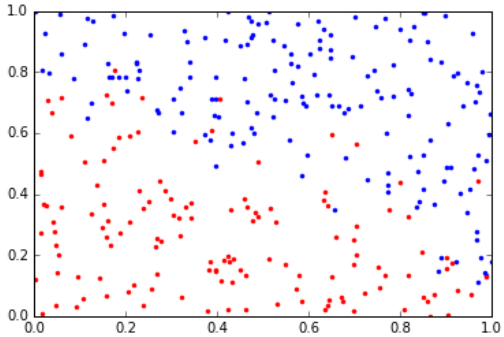
Nous verrons également également l'algorithme de régression linéaire et sa généralisation à la régression polynomiale. En faisant varier le degré du polynôme, nous pourrons jouer sur la capacité de l'algorithme et nous verrons alors un décrochage entre erreur de train et de test.

Exercice n°1 : Algorithme "*pocket*" pour le perceptron

Dans cet exercice, nous poursuivons notre étude du perceptron en rendant sa tâche toujours plus difficile. Nous allons maintenant violer l'hypothèse principale pré-requise à l'utilise du perceptron : la séparabilité linéaire des données. Sans cette hypothèse, le perceptron ne converge pas. L'algorithme "*pocket*" consiste juste alors à définir un nombre maximum d'itérations et à garder trace de l'erreur de train Err_{train}. Après l'itération finale, on choisit les paramètres qui, à une itération donnée, ont produit l'erreur la plus faible.

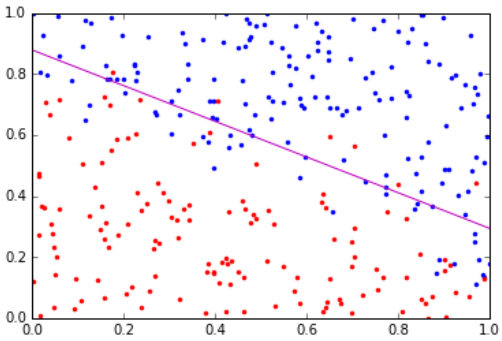
Questions :

1. Nous allons commencer par générer un jeu de données qui ne soit pas linéairement séparable. Nous allons modifier la fonction `datagen` écrite au TP précédent. Au moment de déterminer la classe d'un exemple, vous devrez :
   — déterminer la distance d entre ce point généré et la droite d'équation $x_2 = -0.5x_1 + 0.75$,
   — calculer $\theta = e^{-\frac{d^2}{2\sigma^2}}$ avec $\sigma = 0.05$,
   — échantillonner une perturbation $Z \sim \text{Ber}\left(\frac{\theta}{2}\right)$,
   — Si l'échantillon $z = 1$ alors, permuter la valeur de la classe.
   Vous afficherez ce dataset pour vérification pour $n = 300$ dont 20% pour le train et 80% pour le test. Voici un exemple de visualisation souhaitée :



2. Créez une fonction `ptrain_v2` à partir de `ptrain` et suivant le principe de l'algorithme "*pocket*".
3. Affichez l'évolution de l'erreur de train et de l'erreur de test au fil des itérations de `ptrain_v2`. Vous observez un comportement assez chaotique dû à l'incapacité du perceptron à traiter de telles données. Néanmoins, les deux erreurs suivent-elles une même dynamique ?

4. Superposez l'hyperplan séparateur correspondant à la valeur de θ minimisant l'erreur de train (c.f. question 5 du TP n°1). Voici un exemple de visualisation souhaitée :

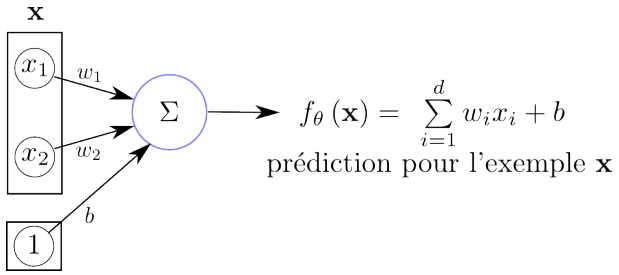


Etant donné, le comportement chaotique du perceptron, la solution obtenue peut être un peu moins bonne que celle ci-dessus.

Exercice n°2 : Régression linéaire et polynomiale

La régression linéaire est une des plus vieilles méthodes statistiques connues. Encore aujourd'hui, elle reste extrêmement populaire du fait d'un bon compromis entre efficacité et facilité de mise en œuvre.

Le point de départ de cette méthode est un problème de régression pour lequel on dispose d'un jeu de données $\mathcal{D} = \left\{\left(\mathbf{x}^{(i)}, y_i\right)\right\}_{i=1}^{n}$. Les exemples $\mathbf{x}^{(i)}$ appartiennent à espace $\mathbb{X} = \mathbb{R}^d$. Les réponses y_i appartiennent à un espace $\mathbb{Y} = \mathbb{R}$. Notre but est de trouver une fonction linéaire $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}$:



On note la similitude avec la fonction apprise par le perceptron. Le seule différence est l'absence d'une étape de binarisation (fonction sign). Comme pour le perceptron, nous noterons

$$\boldsymbol{\theta} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{pmatrix}. \tag{1}$$

Maintenant que nous avons un modèle, nous allons voir comment obtenir une solution satisfaisante, c'est à dire trouver une bonne valeur de $\boldsymbol{\theta}$. C'est là qu'interviennent nos données $\mathcal{D}$:

— Intuitivement, on veut $f_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) \approx y_i$ pour tout i.

— Une façon mathématique de dire ça est de vouloir que $\sum_{i=1}^{n}\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y_i\right)^2$ soit petit.

— Une façon mathématique d'obtenir ça est de poser $J(\boldsymbol{\theta}) = \sum_{i=1}^{n}\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) - y_i\right)^2$ et de minimiser la fonction J.

La fonction J est appelée **fonction de coût** (concept ultra-important en ML). Le succès de la régression linéaire vient du fait qu'il est très facile de trouver le minimum de J. On montre en effet sans difficulté que le

minimiseur $\boldsymbol{\theta}^*$ de J est l'unique point tel que :

$$\frac{dJ}{d\boldsymbol{\theta}}(\boldsymbol{\theta}^*) = 0. \tag{2}$$

Grâce à des résultats d'algèbre linéaire, résoudre cette équation est tout aussi facile. Pour commencer, observez que

$$f_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) = \boldsymbol{\theta}^T.\mathbf{x}^{(i)} \text{ (produit scalaire).} \tag{3}$$

Adoptons également, une notation similaire à celle vue en cours :

matrice des exemples $\mathbf{X}=$ $\begin{array}{cccc} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & & \mathbf{x}^{(n)} \\ \left[\begin{array}{c} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ \cdots \\ x_d^{(1)} \end{array}\right. & \left[\begin{array}{c} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ \cdots \\ x_d^{(2)} \end{array}\right. & \cdots & \left[\begin{array}{c} x_1^{(n)} \\ x_2^{(n)} \\ x_3^{(n)} \\ \cdots \\ x_d^{(n)} \end{array}\right. \\ 1 & 1 & \cdots & 1 \end{array}$ dimension d $+1$

nbr d'exemple n

De la même manière, notons

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}. \tag{4}$$

On peut alors écrire

$$J(\boldsymbol{\theta}) = \left(\mathbf{X}^T\boldsymbol{\theta} - \mathbf{y}\right)^T \left(\mathbf{X}^T\boldsymbol{\theta} - \mathbf{y}\right) \tag{5}$$

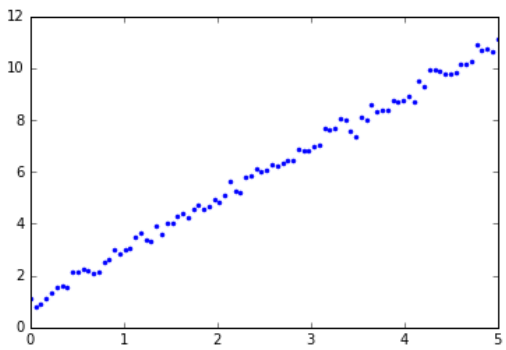
1. A l'aide du formulaire de dérivation matricielle, montrez que :

$$\frac{dJ}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) = 2\mathbf{X}\left(\mathbf{X}^T\boldsymbol{\theta} - \mathbf{y}\right). \tag{6}$$

   En déduire que

$$\boldsymbol{\theta}^* = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\mathbf{y}. \tag{7}$$

2. Nous allons à présent tester l'efficacité de cette méthode sur un jeu de donnée synthétique. Répartissez n =90 points sur l'intervalle $[0;5]$ (fonction `numpy.linspace`). Ces points constituent nos exemples et pour bien visualiser le comportement cet algorithme, nous travaillerons avec $d = 1$
3. Générez les réponses y_i selon une distribution Gaussienne : $Y|X = x_i \sim \mathcal{N}(2x_i + 1, \sigma)$ avec $\sigma = 0.2$. Affichez ces données. Voici un exemple de visualisation souhaitée :

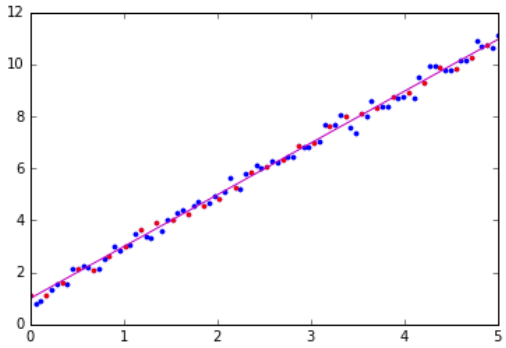


4. Découpez à présent ce dataset en train et en test avec un ratio $\frac{1}{3}$ / $\frac{2}{3}$. On conservera les noms de variable habituels :

— `X_train` de type `numpy array` 1D contenant les exemples d'apprentissage,
— `X_test` de type `numpy array` 1D contenant les exemples de test,
— `y_train` de type `numpy array` 1D contenant les réponses des exemples d'apprentissage,
— `y_test` de type `numpy array` 1D contenant les réponses des exemples de test.
Le découpage se fera de manière déterministe (boucle `for`, tous les 3 tours x_i est attribué à `X_train` et à `X_test` le reste du temps).

5. Calculez θ^* selon la formule de la question 1. Attention, veillez à augmenter la variable `X_train` avec une ligne de 1 (fonctions `numpy.ones` et `numpy.vstack`). Les opérations d'algèbre linéraire se feront aussi via le module `numpy` : `numpy.dot` et `numpy.linalg.inv`

6. Affichez la droite estimée d'équation $\theta_1^* x + \theta_2^*$. Voici un exemple de visualisation souhaitée :



Nous allons à présent généraliser l'algorithme de la régression linéaire au cas **polynomial**, c'est à dire qu'on suppose que la relation liant les $\mathbf{x}^{(i)}$ aux y_i est polynôme de degré p supérieur à 1. Pour simplifier la présentation supposons que $p = 2$ et surtout que $d = 1$, c'est à dire que

$$f_{\boldsymbol{\theta}}(\mathbf{x}) \quad = \quad w_{1,2}x^2 + w_{1,1}x + b \tag{8}$$

L'indexation du paramètre $w_{1,k}$ fait référence au terme du polynôme de degré k pour l'unique dimension de $\mathbf{x}$. Dès qu'on choisit $d > 1$, le nombre de termes du polynôme croît très rapidement. Par exemple, pour $d = 2$, on aurait :

$$f_{\boldsymbol{\theta}}(\mathbf{x}) \quad = \quad w_{2,2}x_1^2x_2^2 + w_{2,1}x_1^2x_2 + w_{1,2}x_1x_2^2 + w_{2,0}x_1^2 + w_{0,2}x_2^2 + w_{1,1}x_1x_2 + w_{1,0}x_1 + w_{0,1}x_2 + b. \tag{9}$$

Dans tous les cas, on remarque qu'on peut ré-écrire l'expression de la fonction $f_{\boldsymbol{\theta}}(\mathbf{x})$ dans une forme proche de celle de la régression linéaire. Pour le cas de l'équation (8), posons

$$\boldsymbol{\theta} = \begin{pmatrix} w_{1,2} \\ w_{1,1} \\ b \end{pmatrix} \text{ et } \mathbf{z} = \begin{pmatrix} x^2 \\ x \\ 1 \end{pmatrix}. \tag{10}$$

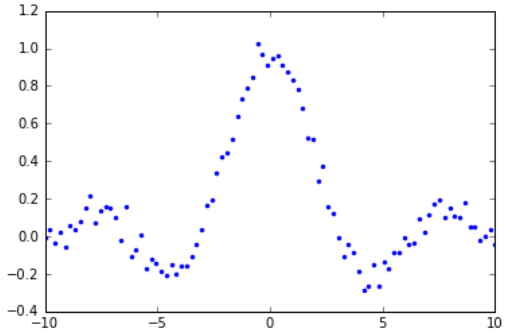
On a donc

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{z}. \tag{11}$$

En voyant $f_{\boldsymbol{\theta}}$ comme une fonction de $\mathbf{z}$, on se ramène bien au cas linéaire. Les vecteurs $\mathbf{z} \in \mathcal{Z}$ forment une représentation intermédiaire et $\mathcal{Z}$ est appelé **espace d'attributs**.

7. Nous allons essayer d'appliquer une régression polynomiale à la fonction $\frac{\sin(x)}{x}$. Comme précédemment, nous allons commencer par générer des données. Répartissez n =90 points sur l'intervalle $[-10; 10]$.

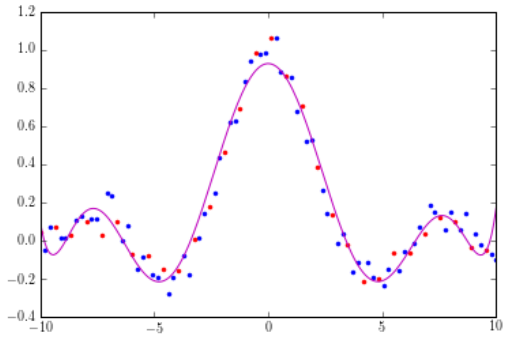
8. Générez les réponses y_i selon une distribution Gaussienne : $Y|X = x_i \sim \mathcal{N}\left(\frac{\sin(x_i)}{x_i}, \sigma\right)$ avec $\sigma = 0.05$. Affichez ces données. Voici un exemple de visualisation souhaitée :



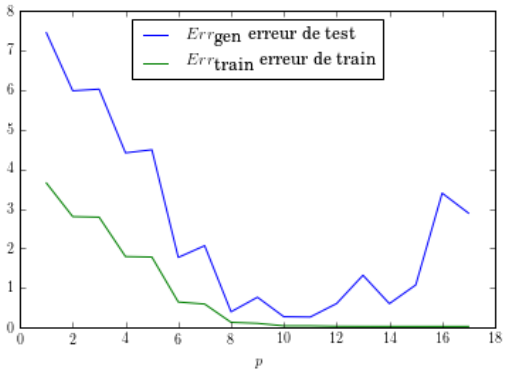
9. Créez des ensembles de train et de test de la même manière que pour le cas linéaire.

10. Créez une fonction `polyreg` qui prend en entrée `X_train`, `y_train` et le degré du polynôme p. Cette fonction retourne le vecteur $\boldsymbol{\theta}^*$. Pour le calculer, il suffit d'appliquer l'équation (7) en remplaçant les $\mathbf{x}_i$ par les $\mathbf{z}_i$.

11. Utilisez la fonction `polyreg` sur vos données avec $p = 8$. Tracez le polynôme correspondant au vecteur $\boldsymbol{\theta}$ retourné par la fonction. Voici un exemple de visualisation souhaitée :

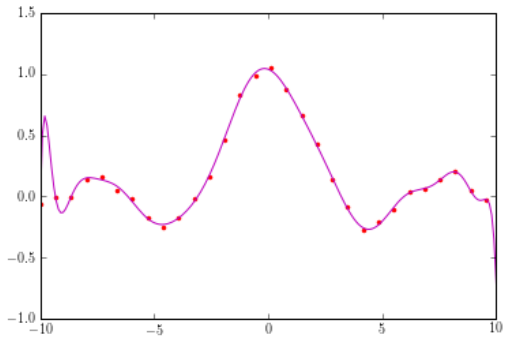


12. Calculez et affichez les erreurs de train et de test en faisant varier p de 1 à 17. Vous devez obtenir des courbes semblables aux suivantes :



D'après ce graphique, quel est le degré optimal pour cette regression polynomiale ?

13. Tracez le polynôme correspondant au degré 19 en le superposant aux données d'apprentissage seulement. Voici un exemple de visualisation souhaitée :



En quoi ce résultat illustre le phénomène de sur-apprentissage (overfitting) ?