MASTER DE CHIMIE · MODÉLISATION MOLÉCULAIRE

TP de Chimiométrie / Machine Learning

Classification de molécules actives par le perceptron de Rosenblatt

3 novembre 2026

1 Introduction et contexte chimique

En chimiométrie et en conception de médicaments (*drug design*), le but est de prédire l'activité biologique d'une molécule directement à partir de sa structure chimique.

Le **perceptron de Rosenblatt** est l'algorithme de classification linéaire qui permet de séparer un espace de descripteurs moléculaires en deux classes :

- **Classe $+1$:** molécules actives (ex. : inhibiteur efficace d'une enzyme).
- **Classe -1 :** molécules inactives ou toxiques.

1.1 Signification des descripteurs moléculaires

Pour caractériser nos molécules, nous utilisons 3 descripteurs physico-chimiques normalisés :

1. x_1 **: le coefficient de partage lipophile** ($\log P$) $\rightarrow$ mesure l'hydrophobie de la molécule (capacité à traverser les membranes biologiques).
2. x_2 **: la surface polaire de Van der Waals (PSA)** $\rightarrow$ exprime l'accessibilité aux liaisons hydrogène.
3. x_3 **: le volume molaire normalisé** $\rightarrow$ traduit l'encombrement stérique de la molécule.

1.2 Fondements mathématiques

Pour chaque molécule k décrite par le vecteur $\mathbf{x}^{(k)} = [x_1, x_2, x_3]^\top$, le perceptron calcule une combinaison linéaire pondérée à laquelle on ajoute un biais b :

$$z = \sum_{i=1}^{3} w_i x_i + b = \mathbf{w}^\top \mathbf{x}^{(k)} + b$$

La **fonction d'activation** évalue le résultat z pour assigner une étiquette binaire de prédiction y_{pred} :

Fonction d'activation

$$f(z) = \begin{cases} +1 & \text{si } z \geq 0 \\ -1 & \text{si } z < 0 \end{cases}$$

Si la prédiction y_{pred} est différente de la classe réelle $y^{(k)}$, les poids synaptiques $\mathbf{w}$ et le biais b sont mis à jour (avec un taux d'apprentissage $\eta = 1$) selon la règle d'apprentissage du perceptron :

Règle d'apprentissage

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \left(y^{(k)} - y_{\text{pred}}^{(k)}\right) \mathbf{x}^{(k)}$$
$$b^{(t+1)} = b^{(t)} + \left(y^{(k)} - y_{\text{pred}}^{(k)}\right)$$

2 Jeu de données expérimental

Le tableau ci-dessous regroupe 6 molécules de test avec leurs 3 descripteurs et leur classe biologique réelle :

Molécule	x_1 ($\log P$)	x_2 (PSA)	x_3 (Vol. molaire)	Classe réelle (y)
M1	1.0	2.0	0.5	-1
M2	2.0	1.0	1.5	$+1$
M3	0.5	2.5	0.2	-1
M4	2.5	2.0	1.0	$+1$
M5	1.5	1.2	0.8	-1
M6	3.0	2.5	2.0	$+1$

3 Résolution numérique détaillée (première itération)

Note méthodologique importante

L'algorithme n'isole pas une molécule pour boucler dessus jusqu'à ce qu'elle soit correcte. Il réalise un parcours séquentiel unique de M1 à M6 (ce qu'on appelle **une époque**). Chaque molécule est testée **une seule fois** par époque. Si une erreur est commise, les poids sont mis à jour immédiatement, et l'algorithme passe à la molécule suivante. C'est en répétant plusieurs époques complètes sur l'ensemble du jeu de données que le modèle converge.

Initialisons les poids et le biais à zéro :

$$\mathbf{w}^{(0)} = [0, 0, 0]^\top \quad \text{et} \quad b^{(0)} = 0$$

Étape 1 · Examen unique de la molécule M1

$\mathbf{x}_1 = [1.0, 2.0, 0.5]^\top$ et $y_1 = -1$:

$$\begin{aligned} z_1 &= (0 \times 1.0) + (0 \times 2.0) + (0 \times 0.5) + 0 = 0 \\ y_{\text{pred}} &= f(0) = +1 \quad (\text{car } z \geq 0) \end{aligned}$$

Analyse : la classe prédite est $+1$ alors que la réalité est $y_1 = -1$. Il y a **erreur de classification**. On met à jour les poids une seule fois, puis on passe directement à M2 :

$$\begin{aligned} \mathbf{w}^{(1)} &= \mathbf{w}^{(0)} + (y_1 - y_{\text{pred}})\,\mathbf{x}_1 = [0,0,0] + (-1-(+1)) \times [1.0, 2.0, 0.5] \\ &= [0,0,0] + (-2) \times [1.0, 2.0, 0.5] = [\mathbf{-2.0},\ \mathbf{-4.0},\ \mathbf{-1.0}]^\top \\ b^{(1)} &= b^{(0)} + (y_1 - y_{\text{pred}}) = 0 + (-2) = \mathbf{-2} \end{aligned}$$

Étape 2 · Examen unique de la molécule M2

$\mathbf{x}_2 = [2.0, 1.0, 1.5]^\top$ et $y_2 = +1$, avec les nouveaux paramètres $\mathbf{w}^{(1)}$ et $b^{(1)}$:

$$\begin{aligned} z_2 &= (-2.0 \times 2.0) + (-4.0 \times 1.0) + (-1.0 \times 1.5) - 2 \\ &= -4.0 - 4.0 - 1.5 - 2 = -11.5 \\ y_{\text{pred}} &= f(-11.5) = -1 \quad (\text{car } z < 0) \end{aligned}$$

Analyse : la classe prédite est -1 alors que $y_2 = +1$. Il y a **erreur**. On met à jour :

$$\begin{aligned} \mathbf{w}^{(2)} &= \mathbf{w}^{(1)} + (1-(-1))\,\mathbf{x}_2 = [-2.0, -4.0, -1.0] + 2 \times [2.0, 1.0, 1.5] \\ &= [-2.0+4.0,\ -4.0+2.0,\ -1.0+3.0] = [\mathbf{2.0},\ \mathbf{-2.0},\ \mathbf{2.0}]^\top \\ b^{(2)} &= b^{(1)} + (1-(-1)) = -2 + 2 = \mathbf{0} \end{aligned}$$

L'algorithme continue ainsi séquentiellement pour M3, M4, M5 et M6. Si des erreurs subsistent à la fin de M6, une **deuxième époque** recommence en repartant de M1 avec les poids $\mathbf{w}^{(2)}$.

4 Code MATLAB d'implémentation

perceptron_rosenblatt.m

```
% ========================================================================
% TP Chimiometrie : Perceptron simple pour la classification de molecules
% ========================================================================
clear; clc;

% 1. Definition du jeu de donnees (6 molecules, 3 descripteurs)
% Descripteurs : [logP, PSA, Volume Molaire]
X = [
```

```
    1.0, 2.0, 0.5;   % M1
    2.0, 1.0, 1.5;   % M2
    0.5, 2.5, 0.2;   % M3
    2.5, 2.0, 1.0;   % M4
    1.5, 1.2, 0.8;   % M5
    3.0, 2.5, 2.0    % M6
];

% Classes biologiques reelles (+1 ou -1)
Y = [-1; 1; -1; 1; -1; 1];

% 2. Initialisation des poids et du biais
w = [0; 0; 0];
b = 0;
max_epochs = 100;
learning_rate = 1.0;

% 3. Boucle d'apprentissage (Epochs)
for epoch = 1:max_epochs
    errors = 0;

    % Parcours sequentiel de chaque molecule (une seule fois par epoque)
    for i = 1:size(X, 1)
        xi = X(i, :)';
        yi = Y(i);

        % Calcul de la sommation lineaire et de la fonction d'activation
        z = dot(w, xi) + b;
        if z >= 0
            y_pred = 1;
        else
            y_pred = -1;
        end

        % Mise a jour immediate si la prediction est incorrecte
        if y_pred ~= yi
            w = w + learning_rate * (yi - y_pred) * xi;
            b = b + learning_rate * (yi - y_pred);
            errors = errors + 1;
        end
    end

    % Condition d'arret : si aucune erreur sur l'epoque complete
    if errors == 0
        fprintf('Convergence parfaite atteinte a l''epoque %d !\n', epoch);
        break;
    end
end

% 4. Affichage des resultats finaux
disp('----------------------------------------');
disp('Resultats de l''apprentissage :');
disp('Vecteur de poids optimises w :');
disp(w');
fprintf('Biais optimise b : %.2f\n', b);
disp('----------------------------------------');
```