Objectifs du TP :
- étudier et comprendre le fonctionnement de l'algorithme du perceptron,
- analyser et implémenter plusieurs variantes de l'algorithme,
- tester les algorithmes sur des données simulées.

1 Algorithme du perceptron pour la classification binaire

L'algorithme de perceptron, dans sa version standard, est dédié à la classification binaire. Les données d'apprentissage utilisées dans cette section s'écrivent donc sous la forme $S = \{(x_i, y_i)\}_{i=1}^n$ avec $x_i \in \mathbb{R}^d$ (d est le nombre d'attributs des données d'entrée) et $y_i \in \{-1, 1\}$. Les étiquettes y_i ne peuvent prendre que deux valeurs (1 ou -1), d'où l'appellation classification binaire.

Un perceptron binaire est un classifieur linéaire. À partir des données $\{(x_i, y_i)\}_{i=1}^n$, l'objectif est d'apprendre un vecteur $w \in \mathbb{R}^d$ tel que $\langle w, x_+ \rangle > 0$ pour tout x_+ appartenant à la classe 1 et $\langle w, x_- \rangle < 0$ pour tout x_- appartenant à la classe -1, où $\langle w, x \rangle := \sum_{j=1}^d w^{(j)} x^{(j)}$ est le produit scalaire entre les deux vecteurs w et x, et $w^{(j)}$ et $x^{(j)}$ sont les j-èmes composantes des vecteurs w et x.

L'idée de l'algorithme du perceptron est d'initialiser w au vecteur nul, itérer un nombre de fois (fixé a priori ou jusqu'à convergence) sur les données d'apprentissage, et ajuster le vecteur de pondération w à chaque fois qu'une donnée est mal classée.

Algorithme Perceptron binaire (sans biais)

Entrée : une liste S de données d'apprentissage, $(x_1, y_1), \ldots, (x_n, y_n)$ où $x_i \in \mathbb{R}^d$ et $y_i \in \{-1, 1\}$.
Sortie : le vecteur de pondération w.

---

1. Initialiser le vecteur $w \leftarrow 0$
2. **Répéter**
3. **Pour** chaque exemple $(x_i, y_i) \in S$ **faire**
4. **Si** $y_i \langle w, x_i \rangle \leq 0$ **alors** # exemple mal classé
5. Ajuster w : $w \leftarrow w + y_i x_i$
6. **Fin si**
7. **Fin pour**
8. **Jusqu'à** ce que tous les exemples soient bien classés

---

1) Soit la fonction `genererDonnees` ci-dessous qui permet de générer aléatoirement un jeu de données en 2 dimensions. Utilisez cette fonction pour générer un jeu de données d'apprentissage. Représentez sur un graphique les données d'apprentissage générées.

2) Implémentez l'algorithme du perceptron décrit ci-dessus. Dans sa version *sans biais*, il suppose que les données sont séparables par un hyperplan passant par l'origine. Appliquer l'algorithme du perceptron sur le jeu de données d'apprentissage. Représentez sur le graphique le classifieur (l'hyperplan) appris par le perceptron.
Indications : en utilisant la librairie `pylab`, pour représenter un point de coordonnées `(x,y)` par un 'o' bleu : `plot(x,y,'ob')` ; pour représenter une droite passant par les points `(x1, y1)` et `(x2, y2)` : `plot([x1, x2],[y1, y2])`.

3) Utilisez la fonction `genererDonnees` pour générer un jeu de données test. Calculez l'erreur de prédiction du perceptron appris à la question précédente sur le jeu de données de test.

```
from pylab import rand

def generateData(n):
    """
    generates a 2D linearly separable dataset with 2n samples.
    returns X an array of 2D samples, and Y the samples label
    """
    xb = (rand(n) * 2 - 1) / 2 - 0.5
    yb = (rand(n) * 2 - 1) / 2 + 0.5
    xr = (rand(n) * 2 - 1) / 2 + 0.5
    yr = (rand(n) * 2 - 1) / 2 - 0.5
    inputs = []
    for i in range(n):
        inputs.append([xb[i], yb[i], -1])
        inputs.append([xr[i], yr[i], 1])

    data = np.array(inputs)
    X = data[:, 0:2]
    Y = data[:, -1]
    return X, Y
```

4) Écrivez une fonction `generateData2(n)` en remplaçant les 4 premières lignes par

```
xb = (rand(n) * 2 - 1) / 2 + 0.5
yb = (rand(n) * 2 - 1) / 2
xr = (rand(n) * 2 - 1) / 2 + 1.5
yr = (rand(n) * 2 - 1) / 2 - 0.5
```

Visualisez les données générées. Elles sont séparables, mais par un hyperplan qui ne passe pas par l'origine. Écrivez une fonction `def complete(sample):` qui complète un échantillon en rajoutant à chaque exemple une coordonnée égale à 1. Appliquez l'algorithme du perceptron sur les nouvelles données. Déduisez-en l'équation d'une droite séparant les données et tracez-la.

2 Perceptron à noyau

1) La fonction `genererDonnees3` ci-dessous permet de générer aléatoirement un jeu de données en 2 dimensions. Utilisez cette fonction pour générer un jeu de données d'apprentissage. Représentez sur un graphique les données d'apprentissage générées.

```
def generateData3(n):
    """
    generates a non linearly separable dataset with about 2n samples.
    The third element of the sample is the label
    """
    xb = (rand(n) * 2 - 1) / 2
    yb = (rand(n) * 2 - 1) / 2 # (xb, yb) est dans le carré centré à l'origine de côté 1
    xr = 3 * (rand(4 * n) * 2 - 1) / 2
    yr = 3 * (rand(4 * n) * 2 - 1) / 2 # (xb, yb) est dans le carré centré à l'origine de côté 3
    inputs = []
    for i in range(n):
        inputs.append(((xb[i],yb[i]),-1))
```

```
    for i in range(4 * n):
        # on ne conserve que les points extérieurs au carré centré à l'origine de côté 2
        if abs(xr[i]) >= 1 or abs(yr[i]) >= 1:
            inputs.append(((xr[i],yr[i]),1))
    return inputs
```

(2) On considère le plongement $\phi : \mathbb{R}^2 \to \mathbb{R}^6$ défini par $\phi(x, y) = (1, x, y, x^2, x * y, y^2)$. Transformez l'échantillon d'apprentissage en utilisant ce plongement. Apprenez le en utilisant le perceptron (pas besoin de compléter l'échantillon : pourquoi ?). Soit w le vecteur calculé. Déduisez-en la fonction f donnant l'équation de la courbe séparatrice. Pour la tracer de manière rudimentaire, vous pourrez utiliser le code suivant (on sait que les données sont contenues dans le carré centré à l'origine et de côté 3) :

```
res = 100
for x in range(res):
    for y in range(res):
        if abs(f(w,-3/2+3*x/res,-3/2+3*y/res))<0.01:
            plot(-3/2+3*x/res,-3/2+3*y/res,'xb')
```

Ce plongement correspond à la fonction noyau k_1 définie par

$$k_1((x_1, y_1), (x_2, y_2)) = 1 + x_1x_2 + y_1y_2 + x_1^2x_2^2 + x_1y_1x_2y_2 + y_1^2y_2^2.$$

(3) Étant donné un échantillon $S = ((x_1, y_1), \ldots, (x_n, y_n))$ où chaque $x_i \in \mathbb{R}^2$, une fonction noyau $k : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$, un ensemble de coefficients $c_1, \ldots, c_n \in \mathbb{N}$, l'ensemble VS des exemples de coefficients non nuls ($VS = \{(x_i, y_i) | c_i \neq 0\}$), on en déduit la fonction $f : \mathbb{R}^2 \to \mathbb{R}$ définie par

$$f(x) = \sum_{(x_i,y_i)\in VS} c_i y_i k(x_i, x).$$

On appelle VS l'ensemble de support. Écrivez la fonction `def f_from_k(coeffs, support_set, k, x)` qui calcule la fonction f.

(4) Écrivez la fonction `perceptron_k` du perceptron à noyau, selon l'algorithme ci-dessous :

Algorithme Perceptron à noyau

Entrée : une liste S de données d'apprentissage, $(x_1, y_1), \ldots, (x_n, y_n)$ où $x_i \in \mathbb{R}^d$ et $y_i \in \{-1, 1\}$, une fonction noyau k.
Sortie : le vecteur des coefficients *coef* et l'ensemble support *support_set*.

---

1. Initialiser $coef \leftarrow []$ et $support\_set \leftarrow []$
2. **Répéter**
3. **Pour** chaque exemple $(x_i, y_i) \in S$ **faire**
4. **Si** $y_i f\_from\_k(coef, support\_set, k, x_i) \leq 0$ **alors** # exemple mal classé
5. Ajuster *coef* et *support_set* : si x_i n'est pas dans *support_set*, l'ajouter à *support_set* et ajouter 1 à à la liste $coeff$; sinon, ajouter 1 au coefficient correspondant
6. **Fin si**
7. **Fin pour**
8. **Jusqu'à** ce que tous les exemples soient bien classés

Appliquez cet algorithme aux données précédentes, pour la fonction noyau k_1 : vous devez trouver exactement la même fonction séparatrice.

(5) Appliquez un noyau Gaussien aux données précédentes.

```
def kg(x,y):
    sigma=10
    return exp(-((x[0]-y[0])**2+(x[1]-y[1])**2)/sigma**2)
```

Refaites l'expérience en prenant $\sigma = 20, \sigma = 1, \sigma = 0.5, \sigma = 0.2$. Interprétez-ce que vous observez.

3 Exercice

Chargez les données `learn.data` à l'adresse `http://pageperso.lif.univ-mrs.fr/~francois.denis/IAAM1/learn.data`. Visualisez-les et programmez un perceptron capable de les apprendre au mieux. Pour lire un jeu de données :

```
f = open("./learn.data","r")
training_set=[]
x=f.readline()
while x:
    training_set.append(eval(x))
    x=f.readline()
f.close()
```

Votre perceptron sera testé sur un jeu de données test généré dans les mêmes conditions.
Écrivez la fonction `def score(S, coeffs, support_set, k)` qui calcule le score de la fonction définie par `coeffs, support_set,k` sur `S`.