TIPE: Le diagnostic de tumeurs mammaires par SVM

Simon Legris

Épreuve de TIPE

Session 2022

Plan de l'exposé

Introduction

Problème

- On cherche à aider au diagnostic de tumeurs du sein, c'est à dire déterminer numériquement la présence d'une tumeur ou non et sa dangerosité.
- Pour cela, il faut un système capable de classifier les mammographies.
- Il existe une méthode particulièrement adaptée : celle des SVM.

Introduction

Problème

- On cherche à aider au diagnostic de tumeurs du sein, c'est à dire déterminer numériquement la présence d'une tumeur ou non et sa dangerosité.
- Pour cela, il faut un système capable de classifier les mammographies.
- Il existe une méthode particulièrement adaptée : celle des SVM.

Introduction

Problème

- On cherche à aider au diagnostic de tumeurs du sein, c'est à dire déterminer numériquement la présence d'une tumeur ou non et sa dangerosité.
- Pour cela, il faut un système capable de classifier les mammographies.
- Il existe une méthode particulièrement adaptée : celle des SVM.

Introduction

Présentation des SVM

- SVM : Support Vectors Machine ou Séparateur à Vaste Marge.
- Ensemble de techniques d'apprentissage supervisé pour résoudre des problèmes de discrimination.Les SVM sont très utilisés dans la classification d'images.
- Développés à partir des années 1990.
- Utilisés dans le diagnostic médical à partir de la toute fin du vingtième siècle.

Introduction

Principe de fonctionnement

- On se donne un ensemble composé de 2 types de points, des croix et des ronds par exemple. On dit que ces points sont séparables linéairement s'il existe une droite telle que les ronds et les croix soient de part et d'autre de la droite.

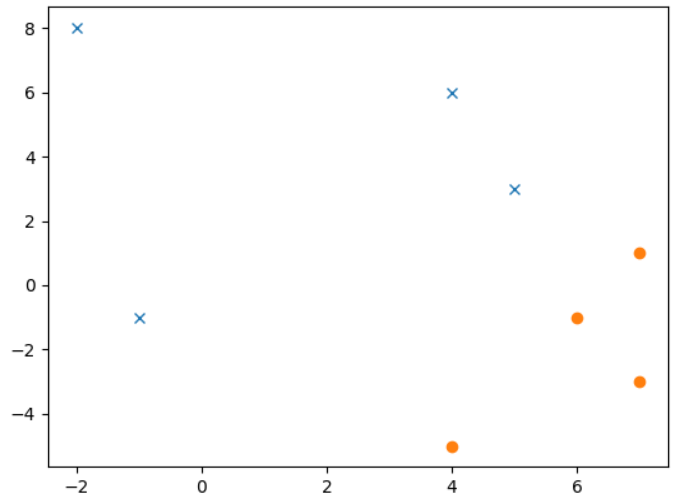

Figure – exemple d'ensemble de points séparables linéairement

Introduction

Principe de fonctionnement

- Il existe cependant une infinité de droites pouvant séparer les données. Laquelle choisir pour classifier au mieux les futures données ?

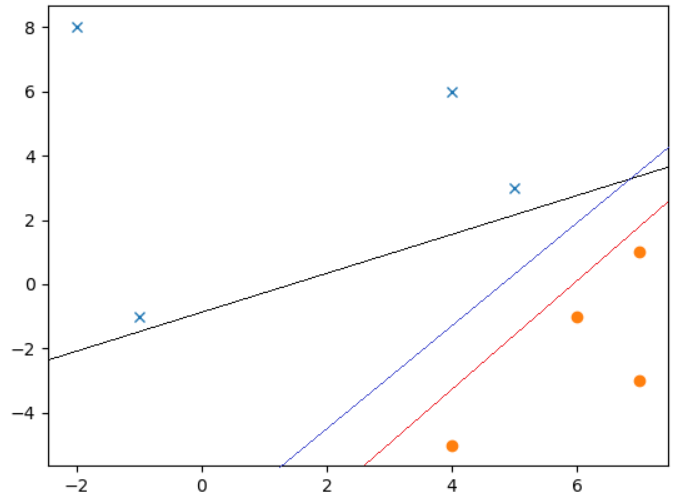

Figure – Exemples de droites séparant linéairement les données

Introduction

Principe de fonctionnement

- Le rôle du SVM est de trouver la "meilleure" de ces droites, c'est à dire celle qui maximise la distance des vecteurs les plus proches de la droite à celle-ci.
- On appelle cette distance la **marge**.
- On appelle ces vecteurs les **vecteurs supports**.

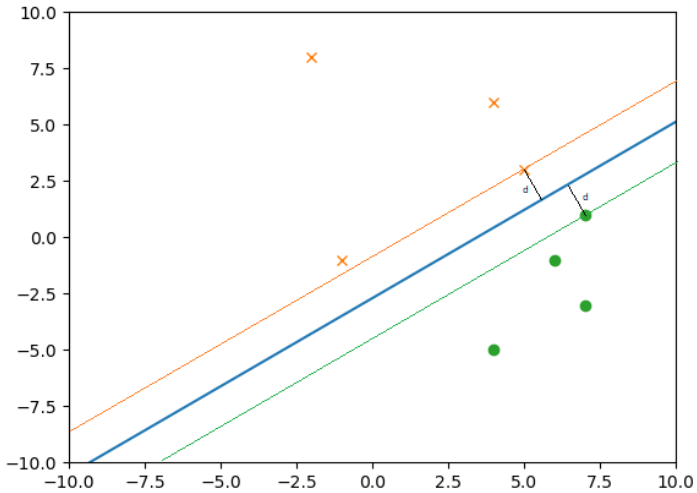

Figure – droite séparatrice maximisant la marge d

Implantation

Trouver les vecteurs supports

- On cherche x_1, x_2 tq $\|x_1x_2\| = min\{\|cr\|, (c,r) \in Croix \times Rond\}$
- Il suffit de calculer toutes les longueurs et de trouver le minimum. Algorithmiquement cette opération a un coût en $O(n^2)$ qui peut être optimisé.

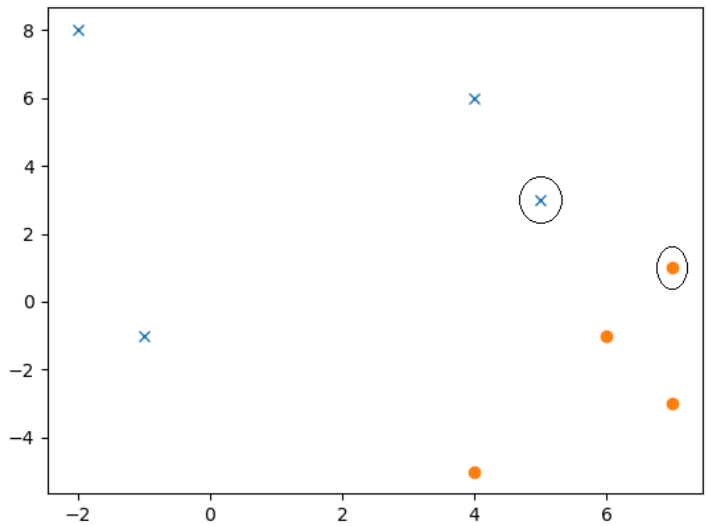

Figure – Vecteurs supports

Implantation

Trouver la droite, formalisme mathématique

- Soit $\vec{w}$ un vecteur orthogonal à la droite séparatrice.
- L'équation de la droite est de la forme :

$$\vec{X} \cdot \vec{w} + b = 0$$

- On appelle b **le biais**.

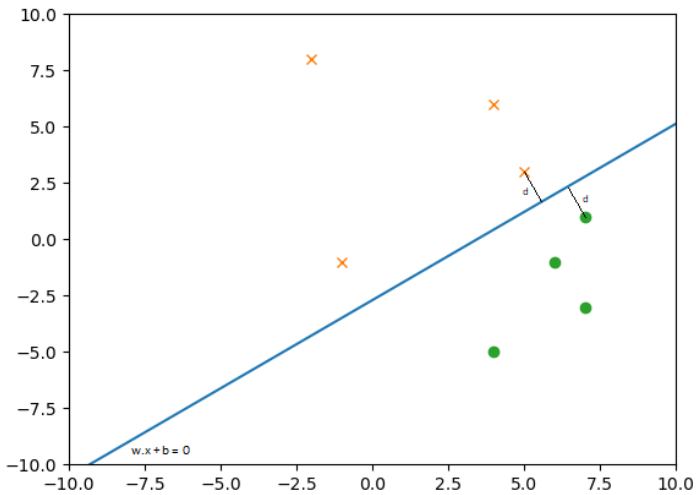

Figure – La droite d'équation $\vec{X} \cdot \vec{w} + b = 0$

Implantation

Trouver la droite, formalisme mathématique

- Soit $\vec{X}$ un vecteur du plan. Si X est du côté gauche de la droite, $\vec{X} \cdot \vec{w} \leq -b$. Si X est du côté droit, $\vec{X} \cdot \vec{w} \geq -b$
- On pose :

$$y = \begin{cases} +1 \ si \ \vec{X} \cdot \vec{w} + b \geq 0 \\ -1 \ si \ \vec{X} \cdot \vec{w} + b < 0 \end{cases}$$

Implantation

Formalisme mathématique

- On pose arbitrairement $\vec{X} \cdot \vec{w} + b = -1$ et $\vec{X} \cdot \vec{w} + b = 1$ les équations respectives des droites D1 et D2.
- On a donc $\vec{X} \cdot \vec{w} + b \geq 1$ et $\vec{X} \cdot \vec{w} + b \leq -1$ respectivement pour les ronds et les croix.

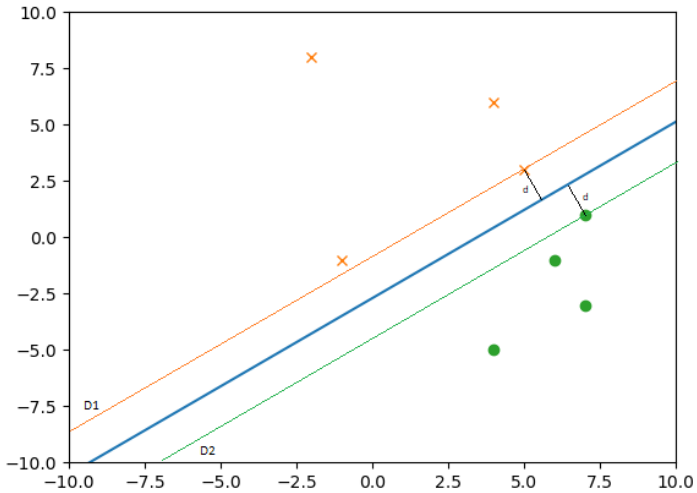

Figure – Les droites passant par les vecteurs supports

Implantation

Formalisme mathématique

- On a donc pour n points sur le plan :

$$\forall i \in [\![1, n]\!], y_i(\vec{X_i} \cdot \vec{w} + b) \geq 1$$

- Comme $d = \frac{1}{\|\vec{w}\|}$, on doit résoudre le problème d'optimisation suivant :

$$\max_{(w,b)} \frac{1}{\|\vec{w}\|} \ tq \ \forall i \in [\![1, n]\!], \ y_i(\vec{X_i} \cdot \vec{w} + b) \geq 1$$

- Ce qui revient à résoudre :

$$\min_{(w,b)} \frac{1}{2}\|\vec{w}\|^2 \ tq \ \forall i \in [\![1, n]\!], \ y_i(\vec{X_i} \cdot \vec{w} + b) \geq 1$$

Implantation

Résolution du problème d'optimisation

- On appelle problème primal un problème de la forme :

$$\min_{(w,b)} \max_{\alpha} L(w, b, \alpha)$$

- On appelle problème dual un problème de la forme :

$$\max_{\alpha} \min_{(w,b)} L(w, b, \alpha)$$

- Dans ce paragraphe, on suppose admis les résultats concernant les multiplicateurs de Lagrange, les conditions de Karush-Kuhn-Tucker (KKT) et les conditions d'égalité pour les problèmes primaux et duaux.

Implantation

Résolution du problème d'optimisation

- On pose :

$$\forall i \in [\![1, n]\!],\ g_i(w) = -y_i(\vec{X_i} \cdot \vec{w} + b)\ +\ 1$$

- On note :

$$\alpha = (\alpha_1, ..., \alpha_n)$$

- On a donc l'expression du lagrangien :

$$L(w, b, \alpha)\ =\ \frac{1}{2}\|\vec{w}\|^2 + \sum_{i=1}^{n} \alpha_i g(w)_i$$

Implantation

Résolution du problème d'optimisation

- On cherche à déterminer le minimum de cette fonction suivant w et b. On dérive donc par rapport à ces deux variables. D'où :

$$\frac{\partial L(w,b,\alpha)}{\partial w} = \vec{w} - \sum_{i=1}^{n} \alpha_i y_i \vec{X_i} = 0$$

$$\frac{\partial L(w,b,\alpha)}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0$$

- en remplaçant w dans $L(w,b,\alpha)$, on obtient

$$L(w,b,\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{X_1} \cdot \vec{X_j}$$

Implantation

Résolution du problème d'optimisation

- Par égalité des solutions du problème primal et dual, résoudre notre problème revient à résoudre :

$$\max_{\alpha} L(w, b, \alpha) \ tq \ \forall i \in [\![1, n]\!], \alpha_i \geq 0 \ et \ \sum_{i=1}^{n} \alpha_i y_i = 0$$

- C'est ce problème que l'on va résoudre algorithmiquement.

Implantation

Résolution du problème d'optimisation

- Pour résoudre ce problème, on peut utiliser la méthode de la descente du gradient afin de déterminer les valeurs des α_i. On pourra ensuite déterminer l'expression de $\vec{w}$ puis en prenant l'un des vecteurs supports, remonter à l'expression de b. On a donc réussi a déterminer l'équation de la meilleure droite séparatrice..

Implantation

Résultats

Pour l'ensemble de données
$E = \{(5, 3, -1), (-2, 8, -1), (4, 6, -1), (-1, -1, -1), (7, 1, 1), (4, -5, 1), (6, -1, 1), (7, -3, 1)\}$, on obtient :

- **Vecteurs supports :** $(5, 3, -1), (7, 1, 1)$
- **Biais :** $b = -2.13$
- **Droite séparatrice :** $y = 0.783x + 2.70$

Expériences

Explications générales

- Pour chaque expérience, les données d'entrée sont des images.
- Afin d'exploiter ces données, il faut associer un point à chaque image c'est à dire deux nombres correspondant à des critères que l'on jugera pertinents.

Première expérience

Objectifs

- Construire un SVM capable de séparer linéairement les mammographies suivant la présence ou non de tumeurs.
- Détecter la présence de tumeurs avec une plus grande efficacité qu'avec une analyse des mammographies à l'oeil nu.
- Évaluer l'efficacité de la prise en compte de l'incertitude dans le diagnostic par rapport à un jugement binaire.

Première expérience

Protocole

- Données d'entrainement composées de 30 mammographies, 15 saines et 15 avec la présence d'une tumeur.
- Étiquettage suivant la présence potentielle d'une tumeur et la taille de cette tumeur.
- 2 groupes de données à étudier, un qui passera par le SVM, l'autre à l'oeil nu.

Première expérience

Mise en œuvre

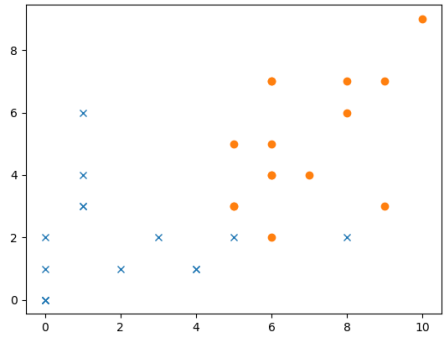

Figure – ensemble des données correspondant aux mammographies

- On remarque que les données ne sont pas linéairement séparables à cause d'un unique point.

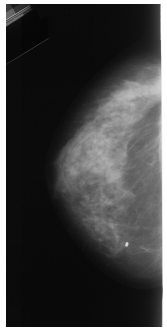

Figure – La mammographie correspondant au point "parasite"

Première expérience

Mise en œuvre

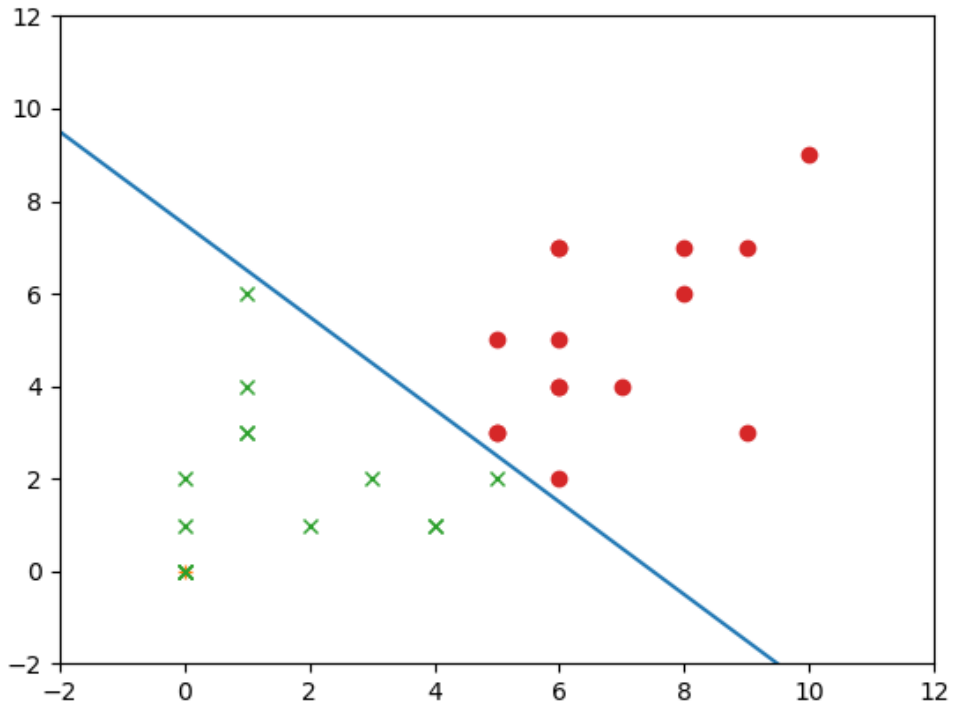

Figure – Données linéairement séparées

Première expérience

Résultats numériques.

	Résultats	Corrects	Faux positifs	Faux négatifs	Total
Méthode	N/A	N/A	N/A	N/A	N/A
SVM	N/A	64	8	8	80
Œil nu	N/A	52	12	16	80

Figure – Résultats statistiques de la première expérience

Deuxième expérience

Objectifs

- Construire un SVM capable de séparer linéairement les mammographies suivant le caractère malin ou non des tumeurs.

Deuxième expérience

Protocole

- Données d'entrainement composées de 30 mammographies contenant des tumeurs, 15 bénignes et 15 malignes.
- Étiquettage suivant l'aire d'une tumeur sur une mammographie et sa régularité.
- La régularité d'une tumeur est obtenue en comparant son aire à celle de son cercle circonscrit.
- Les données sont obtenues à l'aide des masques binaires des tumeurs et d'un algorithme qui calcule en pixel-carré les aires exploitées.
- Étude avec un groupe de données inconnues.

Deuxième expérience

Protocole

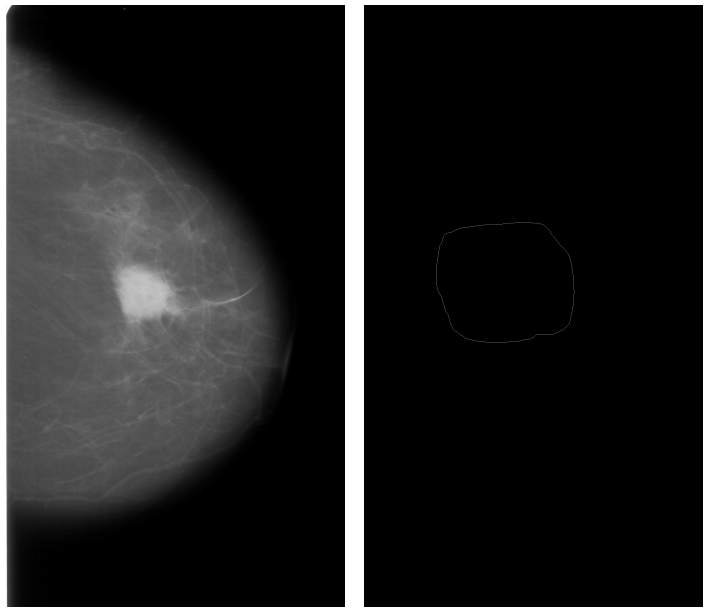

Figure – Exemple de masque binaire correspondant à une mammographie

Deuxième expérience

Mise en œuvre

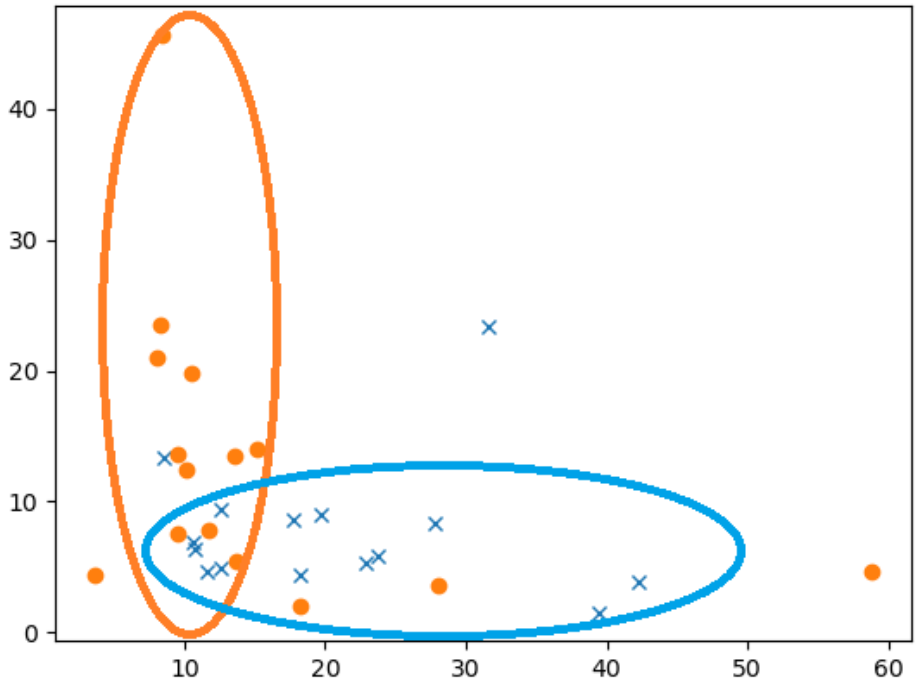

Figure – Répartition des points correspondants aux masques analysés

Deuxième expérience

Résultats

- Bien qu'une tendance soit observée dans la répartition des données, celles-ci ne sont pas linéairement séparables et donc non exploitables par le SVM
- On pourrait intégrer une marge souple pour classifier les cas en acceptant des points mal-classés mais cela serait risqué dans le cadre du diagnostic.

Exploitation des résultats

Première expérience

Le SVM a pu être crée avec succès. Les résultats sont meilleurs que ceux obtenus à l'œil nu mais insuffisants par rapport aux méthodes de diagnostic réelles. La prise en compte de l'incertitude permet néanmoins une amélioration des résultats.

Exploitation des résultats

Deuxième expérience

- La modélisation n'est pas complètement inadaptée mais sans doute trop simpliste pour être exploitable avec les outils développés dans ce cadre. Il faut revoir soit la méthode dans son ensemble soit prendre en compte d'autres facteurs.

Pistes d'amélioration

Prise en compte d'autres facteurs

- On pourrait comparer l'intensité en nuances de gris de zones particulièrement claires par rapport au milieu environnant. En effet, les tumeurs ont tendance à apparaître comme plus blanches que le reste du sein sur les mammographies.

Pistes d'amélioration

Revoir la méthode de classification

- Dans les cas où les SVM sont réellement utilisés dans le diagnostic, on travaille dans un espace de dimension le nombre de pixel de l'image afin de différencier des différences de texture et d'organisation entre tumeurs bénignes et malignes plus précisément.

Codes

Introduction

- Dans cette partie sont présents tous les codes qui ont servi à un moment ou à un autre dans ce travail. Certains sont obsolètes dans la version finale du problème et sont indiqués comme tels.

Codes

Codes utilitaires

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import math

def appartient(truc, liste):
    for i in range (len(liste)):
        if liste[i] == truc :
            return True
    return False

def enleve_doublon(l):
    n = len(l)
    rep = []
    for i in range (0, n) :
        if not(appartient(l[i], l[(i+1)::])) :
            rep.append(l[i])
    return rep

def segment(m1,m2):
    return [(m1[0],m1[1]),(m2[0],m2[1])]
```

Codes

Codes utilitaires

```
def intersection(s1,s2):
    X1=s1[0][0]
    X2=s1[1][0]
    Y1=s1[0][1]
    Y2=s1[1][1]
    X3=s2[0][0]
    X4=s2[1][0]
    Y3=s2[0][1]
    Y4=s2[1][1]
    Ia=[max(min(X1,X2),min(X3,X4)),min(max(X1,X2),max(X3,X4))]
    if  (max(X1,X2) < min(X3,X4)):
        return False
    A1= (Y1-Y2)/(X1-X2)
    A2= (Y3-Y4)/(X3-X4)
    b1=Y1-A1*X1
    b2=Y3-A2*X3
    if (A1==A2):
        return False
    Xa=(b2-b1)/(A1-A2)
    return not(Xa<Ia[0] or Xa>Ia[1])
```

Codes

Codes utilitaires

```
def resolution_systeme(a,b):
    return np.linalg.solve(a,b)

def distance(m1,m2):
    return ((m1[0]-m2[0])**2+(m1[1]-m2[1])**2)**0.5

def minimum(l):
    min=l[0]
    for x in l:
        if x <= min:
            min=x
    return min

def ante_minimum(l): #Fonction obsolete
    mini=minimum(l)
    k=0
    while l[k] != mini:
        k+=1
    l.pop(k)
    return minimum(l)
```

Codes

Codes utilitaires

```
def produit_scalaire(x,y):
    n=len(x)
    return sum(x[k]*y[k] for k in range(n))

def vecteur_augmente(x): #Fonction obsolete
    n=len(x)
    l=[x[k] for k in range(n-1)]+[1]+[x[n-1]]
    return tuple(l)

def multiplication_vecteur(a,x):
    l=[a*u for u in x]
    return tuple(l)

def somme_vecteurs (l):
    n=len(l[0])
    x=[0 for i in range(n)]
    for u in l:
        for k in range(n):
            x[k] += u[k]
    return tuple(x)
```

Codes

Codes pour l'implantation des SVM

```
def separation_donnees(l):
    n=len(l[0])
    l1=[]
    l2=[]
    for x in l:

        if x[2]==-1:
            l1.append(x)
        else:
            l2.append(x)
    return l1,l2

def trace_de_point_initial(l):
    l1,l2= separation_donnees(l)
    X1=[x[0] for x in l1]
    Y1=[x[1] for x in l1]
    X2=[x[0] for x in l2]
    Y2=[x[1] for x in l2]
    plt.plot(X1,Y1, "x")
    plt.plot(X2,Y2,"o")
    plt.show()
```

Codes

Codes pour l'implantation des SVM

```
def est_separable_2D(l):
    l1,l2=separation_donnees(l)
    n=len(l1)
    S1=[segment(l1[i],l1[j]) for i in range(n) for j in range(i)]
    S2=[segment(l2[i],l2[j]) for i in range(n) for j in range(i)]
    for i in range(n):
        for j in range(n):
            if intersection(S1[i],S2[j]):
                return False
    return True
```

Codes

Codes pour l'implantation des SVM

```
def vecteurs_supports(l): #Fonction obsolete
    l1,l2=separation_donnees(l)
    vectsup=[[],[]]
    vect=[]
    for m1 in l1:
        for m2 in l2:
            vect.append((distance(m1,m2),m1,m2))
    l=[vect[i][0] for i in range(len(vect))]
    mini = minimum(l)
    ante_mini=ante_minimum(l)
    for x in vect:
        if x[0]==mini :
            vectsup[0].append(x[1])
            vectsup[0].append(x[2])
        elif x[0]==ante_mini:
            vectsup[1].append(x[1])
            vectsup[1].append(x[2])
    return vectsup

def bias_imput(l): #Fonction obsolete
    n=len(l[0])
    a=[[x[k]for k in range(n-1)]+[1]for x in l]
    c=[x[n-1] for x in l]
    s=resolution_systeme(a,c)
    return s[n-1]
```

Codes

Codes pour l'implantation des SVM

```
def droite_s paratrice(l): #Fonction obsol te
    vectsup=vecteurs_supports(l)
    n=len(l1[0])
    if len(vectsup[0])<=2:
        vectsup1=enleve_doublon(vectsup[0]+[vectsup[1][0]])
        vectsup2=enleve_doublon(vectsup[0]+[vectsup[1][1]])
        n=len(vectsup1)
        m=len(vectsup2)
        if n<=2 or m<=2:
            if n>m:
                #b1=bias_imput(enleve_doublon(vectsup1))
                a1=[[produit_scalaire(vecteur_augmente(vectsup1[i]),vecteur_a
                c1=[x[n-1] for x in vectsup1]
                s1=resolution_systeme(a1,c1)
                w1=somme_vecteurs([multiplication_vecteur(s1[k],vecteur_augm
                W1=produit_scalaire(w1[:3],w1[:3])
                return w1[:3]
            else:
                b2=bias_imput(enleve_doublon(vectsup2))
                a2=[[produit_scalaire(vecteur_augmente(vectsup2[i]),vecteur_a
                c2=[x[m-1] for x in vectsup2]
                s2=resolution_systeme(a2,c2)
                w2=somme_vecteurs([multiplication_vecteur(s2[k],vecteur_augm
                W2=produit_scalaire(w2[:3],w2[:3])
```

Codes

Codes pour l'implantation des SVM

```
                return w2[:3]
        #b1=bias_imput(enleve_doublon(vectsup1))
        #b2=bias_imput(enleve_doublon(vectsup2))
        a1=[[produit_scalaire(vecteur_augmente(vectsup1[i]),vecteur_augmente
        a2=[[produit_scalaire(vecteur_augmente(vectsup2[i]),vecteur_augmente
        c1=[x[n-1] for x in vectsup1]
        c2=[x[n-1] for x in vectsup2]
        s1=resolution_systeme(a1,c1)
        s2=resolution_systeme(a2,c2)
        w1=somme_vecteurs([multiplication_vecteur(s1[k],vecteur_augmente(vect
        w2=somme_vecteurs([multiplication_vecteur(s1[k],vecteur_augmente(vect
        W1=produit_scalaire(w1[:3],w1[:3])
        W2=produit_scalaire(w1[:3],w1[:3])

        if max(1/W1,1/W2)==1/W1:
            return w1[:3]
        else:
            return w2[:3]
    else:
        vectsup0=enleve_doublon(vectsup[0])
        #b=bias_imput(vectsup0)
        a=[[produit_scalaire(vecteur_augmente(vectsup0[i]),vecteur_augmente(
        c=[x[n-1] for x in vectsup0]
        s=resolution_systeme(a,c)
        n=len(s)
```

Codes

Codes pour l'implantation des SVM

```
        w=somme_vecteurs([multiplication_vecteur(s[k],vecteur_augmente(vectsu
        return w[:3]

def trac_droite(w,b):
    X=np.linspace(-40,40, 1000)
    if abs(w[1])<=10**-10:
        plt.plot([-b]*1000,X)
    else:
        Y=[-(w[0]/w[1])*x - b/w[1] for x in X]
        plt.plot(X,Y)

def svm_2d(l): #Fonction obsolete
    l1,l2 = separation_donnees(l)
    if not est_separable_2D:
        return "les donnees ne sont pas separables lin airement"
    w_tilde = droite_separatrice(l)
    n=len(w_tilde)
    b=w_tilde[n-1]
    w=w_tilde[:n-1]
    return w,b
    trac_droite(w,b)
    trace_de_point_initial(l)
```

Codes

Codes pour l'implantation des SVM

```
def probleme_optimisation(l): #Fonction obsolete
    n=len(l)
    a=[]
    vectsup=vecteurs_supports(l)
    for j in range(n):
        L=[l[i][2]*l[j][2]*produit_scalaire(l[i][:2],l[j][:2]) for i in range
        a.append(L)
    a.append([l[i][2] for i in range(n)]+[0])
    p=[1]*(n+1)
    alpha=resolution_systeme(a,p)
    w=somme_vecteurs([multiplication_vecteur(l[k][2],multiplication_vecteur(
    b=vectsup[0][0][2]-produit_scalaire(w,vectsup[0][0][:2])
    return w,b

def SVM_v2(l):  #Fonction obsolete
    w,b=probleme_optimisation(l)
    trac_droite(w,b)
    trace_de_point_initial(l)
```

Codes

Codes pour l'implantation des SVM

```
def SVM_vf(l):
    w,b =descente_de_gradient(l, max_iter = 100000, lr = 0.00001,  c1=100,
epsilon= 1E-7)
    trac _droite(w,b)
    trace_de_point_initial(l)
```

Codes

Codes pour l'analyse d'image

```
image = Image.open(r"C:\Users\smnle\Desktop\Mask\Mask15C.jpg")
n, m = image.size
image = image.load()

tab = []
for i in range(n):
    for j in range(m):
        if image[i, j] == 1:
            tab.append([i, j])

max = 0
for i in range(len(tab)):
    for j in range(i):
        d = (tab[i][0] - tab[j][0]) ** 2 + (tab[i][1] - tab[j][1]) ** 2
        if d > max:
            max = d

aire_circonscrit = math.pi * max / 4 # en pi
```

Codes

Codes pour l'analyse d'image

```
image = Image.open(r"C:\Users\smnle\Desktop\Mask\Mask15C.jpg")
n, m = image.size
image = image.load()

# cette partie l   sert      paissir   le trait pour combler les trous
im = []
for i in range(n):
    im.append([0 for j in range(m)])
for i in range(n):
    for j in range(m):
        if image[i, j]:
            for i1 in range(i-2, i+3):
                for j1 in range(j-2, j+3):
                    im[i1][j1] = 1
aire = 0
for i in range(n):
    intersection = 0
    for j in range(1,m):
        intersection += (im[i][j - 1] != im[i][j])
        if image[i, j]:
            aire += 1
        elif (intersection / 2) % 2:
            aire += 1
    aire -= 2 * (intersection // 2)
```

Codes

Descente du gradient

```
def descente_de_gradient(l, max_iter = 100000, lr = 0.00001,  c1=100,
epsilon= 1E-7):
    C=np.Inf
    x=[u[:2] for u in l]
    Matx=np.array(x)
    y=[u[2] for u in l]
    Maty=np.array(y)
    n = len(x[0])
    w = np.random.normal(n)
    b = np.random.normal()
    nb_echantillons = len(x)
    Mat_noy = np.zeros([nb_echantillons, nb_echantillons])
    for i in range(nb_echantillons):
        for j in range(nb_echantillons):
            Mat_noy[i,j] = np.dot( x[i], x[j] )


    Lambda = np.absolute(np.random.normal(len(x)))
    tour_boucle = 0

    while True:
        dLdLambda = -1 + np.matmul (Mat_noy, Lambda*Maty)*Maty + c1*np.dot(La
        Lambda -= lr * dLdLambda
```

Codes

Descente du gradient

```
        Lambda =  np.minimum(np.maximum(Lambda, 0), C )

        tour_boucle+=1
        if tour_boucle % 20000 ==0:

            p1 = -1*np.sum(Lambda) + 0.5*np.linalg.norm(np.matmul(Matx.T, La
            p2 = c1 * 0.5 * np.dot(Lambda, y) **2
            p = p1 + p2
            print("p1: %f, p2: %f, perte: %f, Lambda max: %f"%( p1, p2, p, np
            w = np.matmul( Matx.T, Lambda*Maty )

            W_support = np.argwhere( Lambda >epsilon )[:,0]
            b_support = np.argwhere( np.logical_and( Lambda > epsilon, Lambda
            w = np.matmul( Matx[ W_support].T, Lambda[ W_support ] * Maty[W
            b = np.mean( Maty[b_support] - np.matmul( Matx[b_support] ,w ))


        if tour_boucle >= max_iter:
            break
    W_support = np.argwhere( Lambda >epsilon )[:,0]
    b_support = np.argwhere( np.logical_and( Lambda > epsilon, Lambda< C - e
    w = np.matmul( Matx[W_support].T, Lambda[W_support]*Maty[W_support])
    b = np.mean( Maty[b_support] - np.matmul( Matx[b_support] ,w ))
    return w,b
```