DUCHANOY Colin
LANCERY Hugo
QUIN Hugo
SOUCOURRE Vincent

Bureau d'étude Ma324 : Optimisation Différentiable 2

AERO 3

M. COUFFIGNAL – M. EL MAHBOUBY

Avril 2022

Table des matières

.....

I. Les moindres carrés multi-classes avec régularisation

1. Version linéaire

On cherche à apprendre une fonction linéaire :

$$f: \mathbb{R}^{d+1} \to \mathbb{R}^{c}$$

De la forme :

$$f(x) \coloneqq x^T W$$

Où x : = $(Data_{ligne,i}, 1)^T$ avec $Data_{ligne,i}$ le i -ème vecteur de Data et qui renvoie un vecteur ligne de taille C de la forme :

$$b = \begin{cases} b_j = 1 \ \ si\ le\ label\ de\ x\ est\ j \in [1, C] \\ 0\ sinon \end{cases}$$

1) Taille de la matrice W

La matrice des paramètres W est de taille d+1,C. Avec C le nombre de catégories :

2) Justifions que le problème d'apprentissage de la fonction f s'écrit sous la forme matricielle :

On cherche à minimiser l'écart entre la fonction et la valeur retourné.
On a donc :

$$\left\| f\left(\left(Data_{ligne,i}, 1\right)^T\right) - b_i \right\|_F^2$$
$$\Leftrightarrow \left\| \left(Data_{ligne,i}, 1\right)^T W - b_i \right\|_F^2$$

Ce qui nous donne sous forme matricielle :

$$\left\| \begin{pmatrix} Data_{ligne,1}, 1 \\ Data_{ligne,2}, 1 \\ \vdots \\ Data_{ligne,m}, 1 \end{pmatrix}^T W - \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \right\|_F^2$$

$$\Leftrightarrow \arg\min_W \|DW - B\|_F^2$$

On peut rajouter un coefficient multiplicateur :

$$\arg\min_W \frac{1}{2} \|DW - B\|_F^2$$

Avec D=$\begin{pmatrix} Data_{ligne,1}, 1 \\ Data_{ligne,2}, 1 \\ \vdots \\ Data_{ligne,m}, 1 \end{pmatrix} = (Data\ 1_m)$

Et B la matrice dont les coefficients sont de la forme :

$$b_{ij} = \begin{cases} 1 \ \ si\ le\ label\ de\ Data_{ligne,i}\ est\ j \in [1, C] \\ 0\ sinon \end{cases}$$

3) Soit $\rho \in \mathbb{R}_+^*$ le facteur de régularisation. Nous allons nous intéresser à la forme dîtes régularisée du problème d'apprentissage :

$$(P_k)\ \arg\min_W \frac{1}{2}\|DW - B\|_F^2 + \frac{\rho}{2}\|W\|_F^2$$

a) Expliquons l'intérêt d'ajouter le terme $\frac{\rho}{2}\|W\|_F^2$

L'intérêt du terme $\frac{\rho}{2}\|W\|_F^2$ est d'éviter le surapprentissage

b) Montrons que le problème (P) s'écrit sous la forme d'une forme quadratique matricielle :

$$(P_1)\ \arg\min_W \frac{1}{2}\langle \mathrm{W}, \mathrm{AW}\rangle - \langle \mathrm{C}, \mathrm{W}\rangle$$

$$\frac{1}{2}\|DW - B\|_F^2 + \frac{\rho}{2}\|W\|_F^2$$

$$\Leftrightarrow \frac{1}{2}\,Tr\big((DW - B)^T(DW - B)\big) + \frac{\rho}{2}Tr(W^TW)$$

$$\Leftrightarrow \frac{1}{2}\,Tr(\,W^TD^TDW - W^TD^TB - B^TDW + B^TB + \rho W^TW)$$

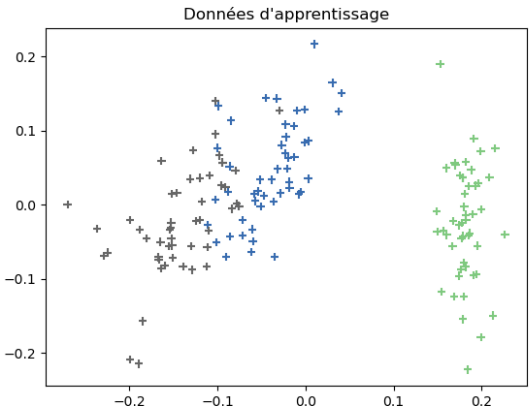
$$\Leftrightarrow \frac{1}{2}Tr\big(W^T(D^TD + \,\rho I)W - W^T(2D^TB)\big)$$

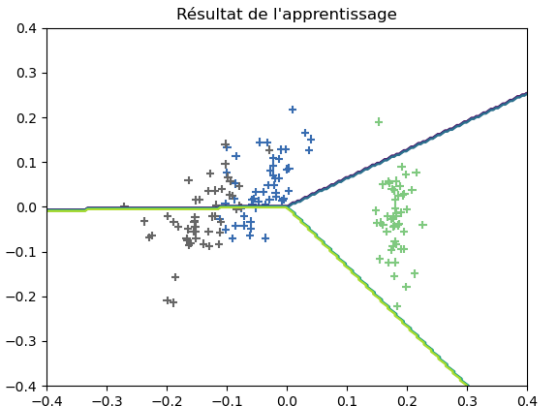
$$\Leftrightarrow \frac{1}{2}Tr\big(W^T(D^TD + \,\rho I)W) - Tr(W^T(D^TB)\big)$$

$$\Leftrightarrow \boldsymbol{\frac{1}{2}\langle W, AW\rangle - \langle C, W\rangle\ avec} \begin{cases} \boldsymbol{A = D^TD + \,\rho I} \\ \boldsymbol{C = D^TB} \end{cases}$$

c) Ecriture de l'algorithme et du programme sous Python

ii. Test sur l'ACP de dimension 2 de Iris.csv

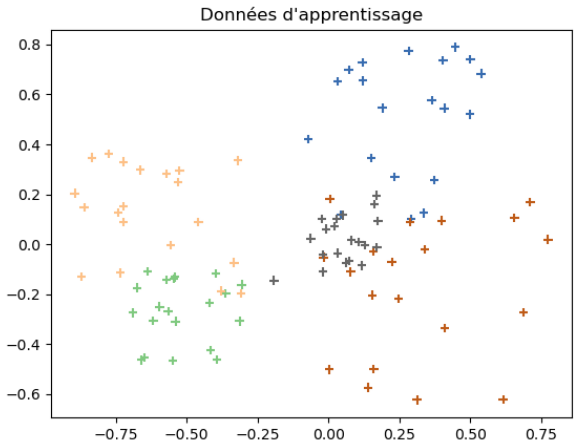


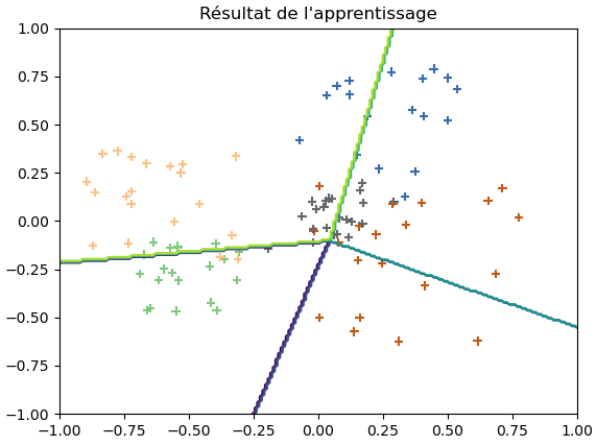


On voit que les catégories éloignées sont bien séparées visuellement, mais lorsqu'elles se rapproche la séparation devient compliquée. Cela illustre bien la limite de la version linéaire.

iii. Test sur les données train_data1.npy et train_data2.npy

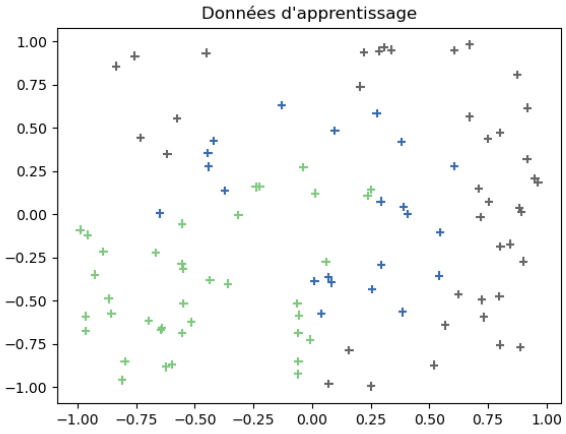
Données train_data1.npy :

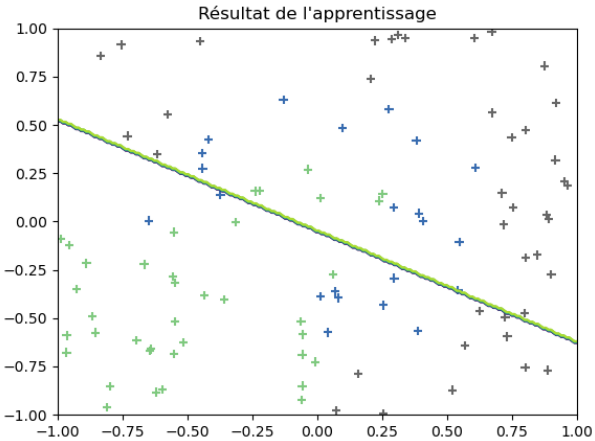




On peut faire la même remarque que précédemment la version linéaire sépare

Données train_data2.npy

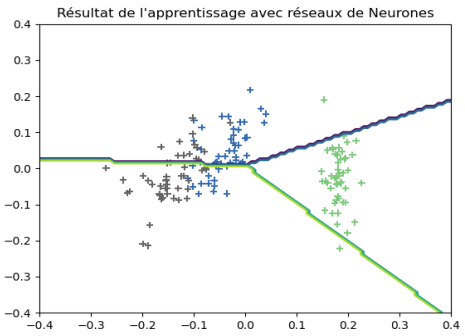




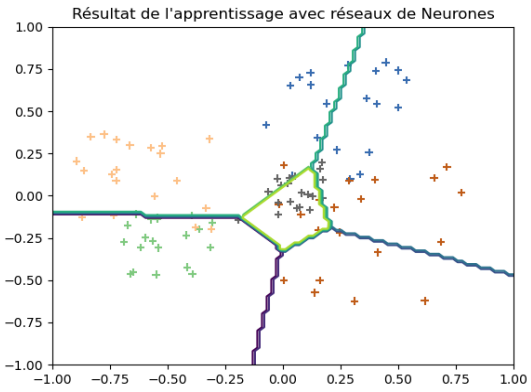
iv. Test sur d'autres données 2D

On remarque pour les données 2D qui ne sont pas séparables linéairement que le programme ne parvient pas à générer une séparation des données selon le bon nombre de catégories existantes

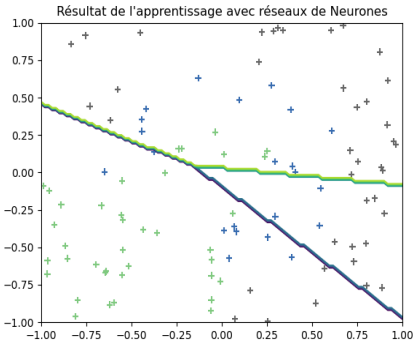
4) Comparaison pour les données 2D avec un apprentissage avec un réseau de neurone de la forme vu en T



Données iris



Données data1



Données data2

5) Matrices de confusions pour les données Mnist et Fashion_Mnist

```
Matrice de confusion :
 [[529.   8.  74.  27.  27. 106.  40.  56.  29.  84.]
 [  0. 450.   4.  42.  61. 179.  38. 147.  37. 177.]
 [ 29.  12. 360.  33.  67.  61. 142. 210.  23.  95.]
 [ 18.   8.  46. 193.  35. 159.  41. 225.  44. 241.]
 [ 32. 179. 105. 117. 117.  55. 113.   5. 139. 120.]
 [ 46. 116.  15. 146.  10.  70.  47.  89.  58. 295.]
 [ 48.  29. 255.  35.  50.  83. 311.  31.  60.  56.]
 [ 92.  82. 208. 296.  14. 104.  26. 119.  48.  39.]
 [ 37.  45.  28. 135.  66. 134.  76.  39. 141. 273.]
 [ 64. 104.  62.  84.  18. 276.  75.   1. 276.  49.]]
Taux de réussite 23.39 %
```

Première matrice de confusion. Le taux de réussite est bien trop faible. Nous avions oublié de « recentrer » les données

On obtient finalement :

-La matrice de confusion suivante pour les données mnist, avec un taux de réussite de 86.47 %

	0	1	2	3	4	5	6	7	8	9
0	936	0	1	2	0	13	19	2	6	1
1	0	1107	3	2	2	1	5	1	14	0
2	16	59	831	23	15	0	28	22	33	5
3	4	17	24	881	5	13	10	24	21	11
4	0	21	4	1	880	2	13	1	9	51
5	21	16	5	76	24	645	28	21	37	19
6	15	10	5	0	16	19	891	0	2	0
7	4	44	16	6	15	1	2	897	0	43
8	14	53	8	33	27	33	19	12	753	22
9	19	11	2	13	65	1	1	64	7	826

-La matrice de confusion suivante pour les données fashion_mnist, un taux de réussite de 81,59%

	0	1	2	3	4	5	6	7	8	9
0	771	1	14	94	1	19	63	1	35	1
1	4	951	14	27	0	2	2	0	0	0
2	25	1	699	8	179	17	54	0	17	0
3	28	12	11	874	29	25	18	0	2	1
4	0	2	78	33	797	6	81	1	2	0
5	2	0	0	0	0	838	0	91	10	59
6	177	5	107	60	90	26	504	0	31	0
7	0	0	0	0	0	46	0	859	0	95
8	0	1	2	12	6	26	14	4	933	2
9	0	0	0	0	0	21	0	46	0	933

Nous avons également essayé d'appliquer cette matrice de confusion pour les données 2D

Voici ce que renvoie le code pour :

```
Matrice de confusion :
 [[50.  0.  0.]
 [ 0. 28. 22.]
 [ 0.  3. 47.]]
Taux de réussite 83.33 %
```

Données iris

```
Matrice de confusion :
 [[19.  1.  0.  0.  0.]
 [ 3. 17.  0.  0.  0.]
 [ 0.  0. 18.  2.  0.]
 [ 0.  0.  1. 19.  0.]
 [ 1.  0.  6. 13.  0.]]
Taux de réussite 73.0 %
```

Données data1

```
Matrice de confusion :
 [[34.  0.  4.]
 [ 9.  0. 13.]
 [ 8.  0. 32.]]
Taux de réussite 66.0 %
```

Données data2

Pas réellement cohérent de calculer cette matrice de confusion dans le sens où nous ne possédons pas de base de données test pour ces données-là.

2. Version non-linéaire avec l'astuce des noyaux

Nous allons utiliser l'astuce des noyaux. On suppose fixée un noyau de type positif *kern* : $\mathbb{R}^{d+1}$ x $\mathbb{R}^{d+1} \rightarrow \mathbb{R}$ et un ensemble de taille $P \in \mathbb{N}^*$:

$$P_k \coloneqq \{x \in Data\}$$

On sait d'après le théorème de Mercer qu'il existe une fonction de redescription $\phi: P_k \rightarrow H$ et un produit scalaire $\langle \cdot, \cdot \rangle$ sur H tel que :

$$kern(a, b) = \langle \phi(a), \phi(b) \rangle$$

Pour tout $(a, b) \in {P_k}^2$

1) Justifions que pour tout élément $w \in H$ et $x := (d, 1)^T$ avec d un vecteur ligne de Data s'écrit sous la forme :

$$\langle \phi(x), w \rangle = (kern(x_1, x) \ \dots \ kern(x_P, x) \begin{pmatrix} a_1 \\ \vdots \\ a_P \end{pmatrix}$$

Nous savons que $w \in H$. Or tout élément de H s'écrit sous la forme :

$$w = \sum_{i=1}^{n} a_i k_{x_i}$$

Ainsi :

$$\langle \phi(x), w \rangle = \left\langle k_{x_j}, \sum_{i=1}^{p} a_i k_{x_i} \right\rangle = \sum_{i=1}^{p} a_i k(x_j, x_i)$$

Ce qui s'écrit matriciellement :

$$\langle \phi(x), w \rangle = (kern(x_1, x) \ \dots \ kern(x_P, x) \begin{pmatrix} a_1 \\ \vdots \\ a_P \end{pmatrix}$$

2) Déduisons-en que l'on peut transformer la version linéaire du problème de l'apprentissage de la partie précédente en un problème de la forme :

$$(P_k)\ \arg\min_W \frac{1}{2}\|D^K W^K - B\|_F^2 + \frac{\rho}{2} trace((W^K)^T K W^K)$$

Où :

- D^K est une matrice de taille m x P ne dépendant que des points de P_k , de $Data$ et de $kern$
- W^K:Une matrice de paramètres de taille P x C
- B: La matrice de la partie précédente
- K: La matrice de Gram associée à P_k et $kern$

On rappelle qu'on cherche à résoudre le problème suivant :

$$W^* = \arg\min_W \sum_{i=0}^{m} (\langle \phi(x), w\rangle - b_i) + \frac{\rho}{2}\|W\|_F^2$$

Grace à la question précédente on sait que :

$$\langle \phi(x), w\rangle = (kern(x_1, x)\ \dots\ kern(x_P, x) \begin{pmatrix} a_1 \\ \vdots \\ \vdots \\ a_P \end{pmatrix}$$

On obtient donc :

$$\sum_{i=0}^{m} (\langle \phi(x), w\rangle - b_i)^2 = \left\| \begin{pmatrix} kern(x_1, x_1) & \cdots & kern(x_1, x_p) \\ \vdots & \ddots & \vdots \\ kern(x_m, x_1) & \cdots & kern(x_m, x_p) \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & a_{1p} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mp} \end{pmatrix} - \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_m \end{pmatrix} \right\|_F^2$$

$$\Leftrightarrow \|D^K W^K - B\|_F^2$$

Pour l'autre terme nous avons :

$$\frac{\rho}{2}\|W\|_F^2$$

$$\Leftrightarrow \frac{\rho}{2} trace(W^T W)$$

$$\Leftrightarrow \frac{\rho}{2} trace\left(\begin{pmatrix} {w_1}^T & \dots & {w_m}^T \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ \vdots \\ w_m \end{pmatrix} \right)$$

$$\Leftrightarrow \frac{\rho}{2} trace((W^K)^T K W^K)$$

En additionnant les deux termes trouvés, on obtient bien le résultat souhaité.
Donc le problème d'apprentissage défini précédemment peut s'écrire sous la forme suivante :

$$(P_k) \ \arg\min_W \frac{1}{2} \|D^K W^K - B\|_F^2 \ + \frac{\rho}{2} trace((W^K)^T K W^K)$$

Remarque pour la suite de l'étude :
On peut se ramener à la même forme quadratique de dans la version linéaire avec des matrices différentes :

$$\frac{1}{2}\langle \boldsymbol{Wk}, \boldsymbol{AWk} \rangle - \langle \boldsymbol{C}, \boldsymbol{Wk} \rangle \ \boldsymbol{avec} \begin{cases} \boldsymbol{A = Dk^T Dk + \ \rho K} \\ \boldsymbol{C = Dk^T B} \end{cases}$$

Nous pouvons utiliser la même fonction du gradient conjugué pour résoudre ce problème.

3) Adaptons à la « version noyau » les questions 3 et 4 de la partie précédente

Nous pouvons maintenant définir la fonction app_global_MCK. Dans un premier temps nous allons définir les différentes matrices définis à la question précédente avec les bonnes dimensions.

```
m, d = np.shape(train_data)
p=len(Pk)
Dk=np.zeros((m,p))
B=np.zeros((m,nbcat))
K = np.zeros((p,p))
Wko=np.zeros((p,nbcat))
```

Tous d'abord nous allons créer la matrice B identique à la partie précédente. Ensuite nous réalisons une boucle qui nous permettra d'obtenir la matrice Dk, pour cela nous appelons un noyau (choisi) entre un vecteur de Pk (qui contient des lignes au hasard de Data) et une ligne de train_data.
On réalise une boucle pour la création de K qui contiendra les noyaux entre les lignes de Pk.
On applique ensuite le gradient conjugué comme dans la fonction linéaire tout en modifiant les valeurs de A et C.
Ainsi on obtient la matrice des paramètres W.

```
def app_global_MCK(train_data,train_labels,nbcat,rho,epsilon,kern, Pk):

for j in range (nbcat):
        for i in range(m):
            if train_labels[i]==j:
                B[i,j]=1

    for i in range(m):
        for j in range(p):
            x1=train_data[i].reshape(-1,1)
            x1=np.vstack((x1,1))
            x2=Pk[j].reshape(-1,1)
            x2=np.vstack((x2,1))
            Dk[i,j]=kern(x1,x2)

    for i in range(p):
        for j in range(p):
            K[i,j]=kern(Pk[i].reshape(-1,1),Pk[j].reshape(-1,1))

    A= Dk.T@Dk+rho*K
    C=Dk.T@B
    W ,cpt = GPCmat(A,C,Wko,epsilon)
    return W
```

Malheureusement, à part avec Iris.csv, notre fonction ne permet pas de retourner une matrice d'apprentissage à cause d'un problème de division par 0 dans le calcul du gradient conjugué

II. Un algorithme « alternatif » pour les SVM

Dans cette partie, nous allons décrire un problème de classification « linéaire ». Supposons données un ensemble fini E partitionne en deux sous-ensembles disjoints de $\mathbb{R}^n$:

$$E_1 = \{u_i \in \mathbb{R}^n | i \in [1,p]\} et\ E_2 = \{v_j \in \mathbb{R}^n | j \in [1,q]\}$$

L'idée est de trouver un hyperplan séparateur. On peut se représenter cette situation de la manière suivante : par exemple les points de E_1 vérifient une propriété P alors que les points de E_1 non. On cherche alors à séparer les deux ensembles de points par un hyperplan H. L'espace $\mathbb{R}^n$ sera alors divisé en deux parties disjointes. L'espace situé « au-dessus » de H et l'espace situé « en-dessous » de H. Supposons alors que l'on se donne un nouveau point x $\in \mathbb{R}^n$. Prédire si x vérifie ou non la propriété P revient à regarder où x est situé par rapport à l'hyperplan H.

Nous avons vu en TD que la formulation des SVM revient à résoudre un problème de la forme suivante :

$$(P_2) = \begin{cases} min \frac{1}{2}\|w\|^2 \\ Sous\ les\ contraintes: \begin{cases} w^T u_i - b \geq 1 \ \ \forall i \in [1,\mathrm{p}] \\ -w^T v_j + b \geq 1 \ \ \forall j \in [1,\mathrm{q}] \end{cases} \end{cases}$$

1) Montrons que $J(w,b) := \frac{1}{2}\|w\|^2$ est 1-elliptique et ∇J est 1-lipschitzienne

Soit $(w_1, w_2) \in \mathbb{R}^n$ x $\mathbb{R}^n$

J est 1 elliptique, Démonstration :

$$\langle \nabla J(w_1) - \nabla J(w_2), w_1 - w_2 \rangle$$

$$\Leftrightarrow \langle w_1 - w_2, w_1 - w_2 \rangle$$

$$\Leftrightarrow (w_1 - w_2)^T (w_1 - w_2)$$

$$\Leftrightarrow \alpha \|w_1 - w_2\|_2^2$$

Donc J est bien elliptique avec $\alpha = 1$

J est 1lipschitzienne, Démonstration :

$$\|\nabla J(w_1) - \nabla J(w_2)\|_2$$

$$\Leftrightarrow \|w_1 - w_2\|_2$$

$$\Leftrightarrow M\|w_1 - w_2\|_2$$

Donc J est bien lipschitzienne avec $M = 1$

a) Convexité

Démontrons que l'ensemble $C = \{(w, b) \in \mathbb{R}^n \mathbb{R} \ | w^T u_i - b \geq 1, -w^T v_j + b \geq 1, i \in [1, p], \forall j \in [1, q]\}$ est convexe :

L'espace des contraintes est ferme et borné (compact) donc l'ensemble C est convexe.

b) Solution du problème

Nous avons vu que l'espace des contraintes était convexe, de plus J est linéaire.
Nous pouvons donc en déduire que le problème P_2 admet au minimum une solution.

c) Expression des contraintes

Nous avons l'espace des contraintes suivants :

$$C = \{(w, b) \in \mathbb{R}^n \mathbb{R} \ | w^T u_i - b \geq 1, -w^T v_j + b \geq 1, i \in [1, p], \forall j \in [1, q]\}$$

Nous pouvons ainsi le réécrire sous la forme matricielle suivante :

$$\begin{pmatrix} -{u_1}^T & 1 \\ \vdots & \vdots \\ -{u_p}^T & 1 \\ {v_1}^T & -1 \\ \vdots & \vdots \\ {v_q}^T & -1 \end{pmatrix} \begin{pmatrix} w \\ b \end{pmatrix} \geq 1_{p+q}$$

$$\Leftrightarrow \begin{pmatrix} -{u_1}^T & 1 \\ \vdots & \vdots \\ -{u_p}^T & 1 \\ {v_1}^T & -1 \\ \vdots & \vdots \\ {v_q}^T & -1 \end{pmatrix} \begin{pmatrix} w \\ b \end{pmatrix} + 1_{p+q} \leq O_{\mathbb{R}^{p+q}}$$

$$\Leftrightarrow C\begin{pmatrix} w \\ b \end{pmatrix} + 1_{p+q} \leq O_{\mathbb{R}^{p+q}}$$

Avec C=$\begin{pmatrix} -u_1{}^T & 1 \\ \vdots & \vdots \\ -u_p{}^T & 1 \\ v_1{}^T & -1 \\ \vdots & \vdots \\ v_q{}^T & -1 \end{pmatrix}$

2) Soit $z \coloneqq (z_1, z_2, \dots, z_{p+q})^T$ tel que $C\begin{pmatrix} w \\ b \end{pmatrix} + 1_{p+q} \leq O_{\mathbb{R}^{p+q}}$

On considère le lagrangien augmente suivant :

$$\mathcal{L}_\rho(x, z, \lambda) = \frac{1}{2}\|w\|^2 + \lambda^T\left(C\begin{pmatrix} w \\ b \end{pmatrix} + z + 1_{p+q}\right) + \frac{\rho}{2}\left\|C\begin{pmatrix} w \\ b \end{pmatrix} + z + 1_{p+q}\right\|_2^2$$

Avec $\lambda \in \mathbb{R}^{p+q}$ est le multiplicateur de Lagrange et ρ>0 est le paramètre dit de pénalité quadratique des contraintes.

Nous allons utiliser pour la résolution numérique du problème précèdent, le schéma itératif d'ADMM défini par :

$$\begin{cases} w^{k+1} = argmin\mathcal{L}_\rho(x, z^k, \lambda^k) \\ z^{k+1} = argmin\mathcal{L}_\rho(w^{k+1}, z, \lambda^k) \\ \lambda^{k+1} = \lambda^k - \rho(C\begin{pmatrix} w \\ b \end{pmatrix}^{k+1} + z^{k+1} + 1_{p+q}) \end{cases}$$

d) En posant $x = (w, b)^T$ montrons que :

$$\mathcal{L}_\rho(x,z,\lambda) = \frac{1}{2}\|Ax\|^2 + \lambda^T\left(Cx + z + 1_{p+q}\right) + \frac{\rho}{2}\left\|Cx + z + 1_{p+q}\right\|_2^2$$

On cherche A tel que :

$$Ax = A\begin{pmatrix} w \\ b \end{pmatrix} = w$$

$$\Leftrightarrow A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Donc :

$$\mathcal{L}_\rho(x,z,\lambda) = \frac{1}{2}\left\|\begin{pmatrix} 1 \\ 0 \end{pmatrix}^T \begin{pmatrix} w \\ b \end{pmatrix}\right\|^2 + \lambda^T\left(C\begin{pmatrix} w \\ b \end{pmatrix} + z + 1_{p+q}\right) + \frac{\rho}{2}\left\|C\begin{pmatrix} w \\ b \end{pmatrix} + z + 1_{p+q}\right\|_2^2$$

$$\Leftrightarrow \mathcal{L}_\rho(x,z,\lambda) = \frac{1}{2}\|Ax\|^2 + \lambda^T\left(Cx + z + 1_{p+q}\right) + \frac{\rho}{2}\left\|Cx + z + 1_{p+q}\right\|_2^2$$

e) Montrons que :

$$\begin{cases} \dfrac{\partial}{\partial z}\mathcal{L}_\rho(x,z,\lambda) = (A + \rho C^T C)x + C^T(\rho(z+d) + \lambda) \\ \dfrac{\partial}{\partial x}\mathcal{L}_\rho(x,z,\lambda) = \lambda + \rho(z + Cx + d) \end{cases}$$

On développe l'expression du lagrangien pour se simplifier les calculs :

$$\mathcal{L}_\rho(x,z,\lambda) = \frac{1}{2}\|Ax\|^2 + \lambda^T\left(Cx + z + 1_{p+q}\right) + \frac{\rho}{2}(C^T C\, x^2 + 2C^T xz + 2C^T x 1_{p+q} + z^T z + 2z^T 1_{p+q} + 1_{p+q}{}^2)$$

On calcule pour chaque dérivée :

$$\frac{\partial}{\partial x}\mathcal{L}_\rho(x,z,\lambda) = 2 * \frac{1}{2}Ax + \lambda^T C + \frac{\rho}{2}(2C^T C\, x + 2C^T z + 2C^T 1_{p+q})$$

$$\Leftrightarrow Ax + \rho C^T C\, x + \lambda^T C + \rho(C^T z + C^T 1_{p+q})$$

$$\Leftrightarrow (A + \rho C^T C)x + C^T(\rho(z+d) + \lambda)$$

On fait de même pour la seconde dérivée :

$$\frac{\partial}{\partial z}\mathcal{L}_\rho(x,z,\lambda) = \lambda + \frac{\rho}{2}(2C\,x + 2z + 2 * 1_{p+q})$$

$$\Leftrightarrow \lambda + \rho\left(Cx + z + 1_{p+q}\right)$$

$$\Leftrightarrow \lambda + \rho(z + Cx + d)$$

Avec d=1_{p+q}

f) Ecriture de la fonction python Admm() permettant d'implémenter l'algorithme (P)

Nous allons maintentant ecrire la fonction python Admm. Dans un premier temps nous initialisons λ, x et z en posant $\frac{\partial}{\partial x}\mathcal{L}_\rho(x,z,\lambda) = 0$ et $\frac{\partial}{\partial z}\mathcal{L}_\rho(x,z,\lambda) = 0$

```
def x0_update(A,rho,C,d,z0,lambd0):
    return -np.linalg.inv(A+rho*C.T@C)@(C.T@(rho*(z0+d)+lambd0))


def z0_update(C,rho,d,lambd0,x1):
    return -(lambd0/rho)-C@x1-d

def lambd0_update(A,rho,C,d,lambd0,x1,z1):
    return lambd0-rho*(C@x1+z1+d)
```

Nous pouvons définir les différents paramètres ainsi que les matrices nécessaires à l'implémentation de notre algorithme.

```
rho=10**(-3)
A=np.hstack([np.eye(n+1,n),np.zeros((n+1,1))])
d=np.ones((p+q,1))
E=np.block([[np.ones((p,1))],[-np.ones((q,1))]])
C=np.block([X,E])
x0=np.zeros((n+1,1))
z0=np.zeros((p+q,1))
lambd0=np.zeros((p+q,1))
#Boucle d'itération de l'ADMM
compteur=0
Norm=1
```

Nous pouvons donc commencer à écrire notre algorithme. Nous réalisons une boucle permettant de s'assurer que quand la méthode converge, l'algorithme s'arrête.
A chaque itération, nous réinitialisons x, z, λ en faisant appels aux différentes fonctions définis (`x0_update, z0_update, lambd0_update`)

```
def Admm(X,p,q,u,v,rho,epsi,n=2,vue=0):
    #Initialisation rho<epsi
    rho=10**(-3)
    A=np.hstack([np.eye(n+1,n),np.zeros((n+1,1))])
    d=np.ones((p+q,1))
    E=np.block([[np.ones((p,1))],[-np.ones((q,1))]])
    C=np.block([X,E])
    x0=np.zeros((n+1,1))
    z0=np.zeros((p+q,1))
    lambd0=np.zeros((p+q,1))
    #Boucle d'itération de l'ADMM
    compteur=0
    Norm=1
    while Norm>epsi and compteur<100:
        x1=x0_update(A,rho,C,d,z0,lambd0)
        z1=z0_update(C,rho,d,lambd0,x1)
        lambd1=lambd0_update(A,rho,C,d,lambd0,x1,z1)
        Norm=np.linalg.norm(x1-x0,2)
        compteur+=1
        x0=copy.deepcopy(x1)
        z0=copy.deepcopy(z1)

        #Le tracé interviendra ici

    print("nombre d'itérations à e(-2): ",compteur)
    return x1,lambd1,compteur
```

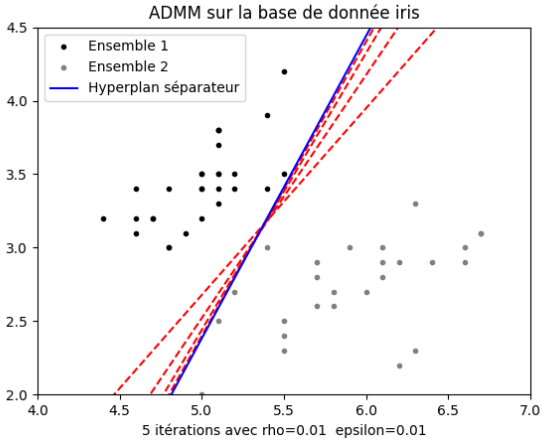
Nous incluons aussi une fonction permettant de tracer les graphiques dans notre fonction ADMM Faisant appel à une fonction Affichage défini précédemment :

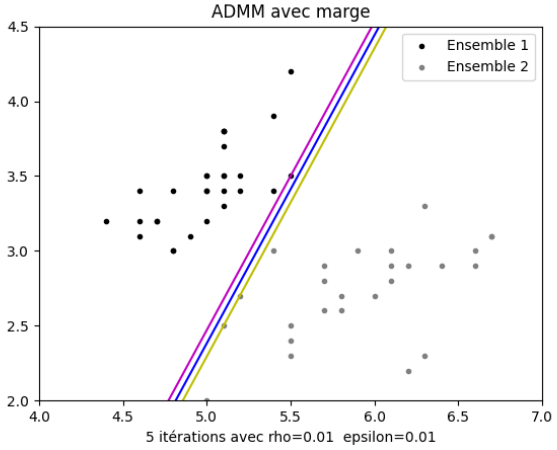
```
#Tracé
if vue==1:
      nbt=100
      t=np.linspace(3,8,nbt)
      w=x1[:2].T[0]
      b=(np.min(w@u.T)+np.max(w@v.T))/2
      delta=(np.min(w@ u.T)-np.max(w@v.T))/2
      affichage(Norm,epsi,rho,u,v,delta,b,w,t,nbt)
```

Ainsi en appelant les différents paramètres de notre fonction ADMM nous pouvons réaliser la fonction :

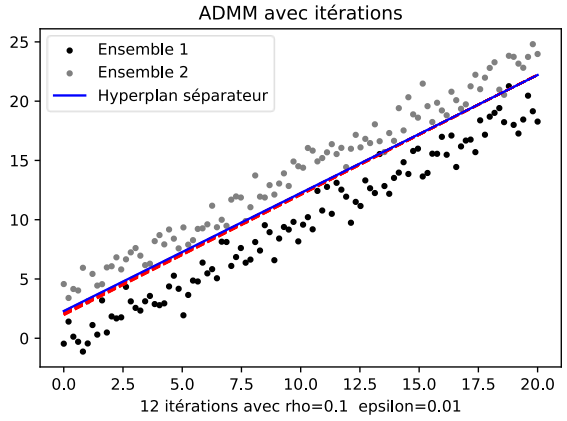
```
epsi=10**(-2)
rho=10**(-4)
u=u_train
v=v_train
X=np.concatenate((-u,v),axis=0)
p,q=np.shape(u)[0],np.shape(v)[0]
xsol,lambdsol,compteur=Admm(X,p,q,u,v,rho,epsi,n=2,vue=1)
```

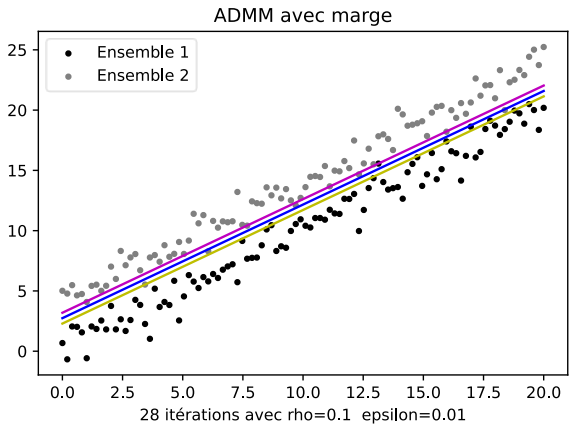
Nous obtenons les traces suivants avec la base de données Iris :





On observe ici que notre algorithme réalise 5 itérations avant le trouver l'hyperplan séparant les 2 ensembles. Il converge vers la solution rapidement avec le pas et la précision choisi ici.
Nous voyons sur la figure de droite que l'on obtient un hyperplan séparateur avec deux marges +/-.





3) Programme Python de l'algorithme ADMM

Utilisons l'algorithme ADMM pour réaliser un programme Pyhton reconnaissant des chiffres manuscrits à partir de la base de données MNIST, et puis des habits avec Fashion MNIST. On comparera les résultats avec ceux de la première partie.

En utilisant l'ADMM pour la base de données MNIST, nous obtenons les solutions suivantes :

Nous observons que pour des bases de données plus conséquentes, notre algorithme converge quand même assez rapidement pour une précision de 0,01.

4) Donnons pour chacun des cas la matrice de confusion sur les données tests de chacune des deux bases de données MNIST

Nous réalisons maintenant la matrice de confusion pour la base de données MNIST :

$$M_{conf} := \begin{pmatrix} N_{vp} & N_{fn} \\ N_{fp} & N_{vn} \end{pmatrix}$$

Où N_{vp} le nombre de vrais positifs, N_{fn} le nombre de faux négatifs, N_{fp} le nombre de faux positifs et N_{vn} le nombre de vrais négatifs. Les nombres de vrais positifs et négatifs correspondent au nombre de fois où le programme a eu juste sur la détection du chiffre et les nombres faux, le nombre de fois où il s'est trompé sur la prédiction.

Si on prend l'exemple avec le chiffre 0 et un inconnu x à détecter :

- N_{vp} : notre programme pense qu'il s'agit d'un 0 et x est bien un 0.
- N_{fp} : notre programme pense qu'il s'agit d'un 0 et x n'est pas un 0.
- N_{fn} : notre programme pense qu'il ne s'agit pas d'un 0 et x est bien un 0.
- N_{vn} : notre programme pense qu'il ne s'agit pas d'un 0 et x n'est pas un 0.

Nous obtenons donc la matrice de confusion suivante :

$$M_{conf} := \begin{pmatrix} 8168 & 226 \\ 1319 & 287 \end{pmatrix}$$

Taux de réussite : 84.55%
Sensibilité : 0.973076006671432

De plus si l'on compare avec la version non linéaire on observe que notre algorithme admet un taux de réussite plus faible que pour la version non linéaire qui est de 86.3 %

5) (Bonus) : Appliquons l'algorithme ADMM à la formulation avec noyau de la SVM