Data Mining
TP8 - Optimisation sans contraintes

Rémi MUSSARD - Thomas ROBERT

Le but du TP est de faire de la régression logistique afin de faire de classification par apprentissage supervisé.

1 Régression logistique binomiale

On commence par une régression logistique binomiale, c'est à dire avec deux classes.

1.1 Fonctions codées

On code pour cela plusieurs fonctions :

1.1.1 EncoderLabel01

Cette fonction permet de passer d'un label -1 / 1 en un label 0 / 1.

```
function z = EncoderLabel01(y)
    z = (y+1)/2;
end
```

1.1.2 probaAPosteriori

Cette fonction permet de calculer les probabilités à posteriori. Elle fonctionne pour 2 classes ou plus.

```
function p = probaAPosteriori(theta, phi)

    p = exp(phi*theta);
    p = p./repmat((1+sum(p,2)), 1, size(theta,2));

end
```

1.1.3 ma_reg_log

Cette fonction calcule la régression logistique.

```
function [theta,L,p] = ma_reg_log(phi, z)

    theta = zeros(size(phi,2), 1);
    theta_old = theta + 5;
    i=1;

    L = [];

    while norm(theta_old - theta) > 1e-1
        theta_old = theta;
```

```
        p = probaAPosteriori(theta, phi);
        l = -sum(z.*log(p) + (1 - z).*log(1-p));
        L = [L l];
        W = diag(p.*(1-p));
        r =  phi*theta + W\(z - p);
        theta = (phi'*W*phi)\(phi'*W*r);
        i = i + 1;
    end
end
```

1.2 Application aux données clowns

Après avoir séparé les données en un ensemble de test et un ensemble d'apprentissage (80% en test / 20% en apprentissage), on appliquera les fonctions vues ci-dessus.

```
load clownsv7.mat

% Labels réencodés de -1 / 1 à 0 / 1
z = EncoderLabel01(y);

% Partage des données en deux : Apprentissage et Test
[xapp, zapp, xtest, ztest] = splitdata(X, z, 0.2);
n = size(X, 1);
napp = size(xapp, 1);
ntest = size(xtest, 1);
```

1.3 Frontière de décision linéaire

On calcul d'abord la régression sur une matrice $\phi = [1\ x_1\ x_2]$ afin d'obtenir une frontière de décision linéaire.

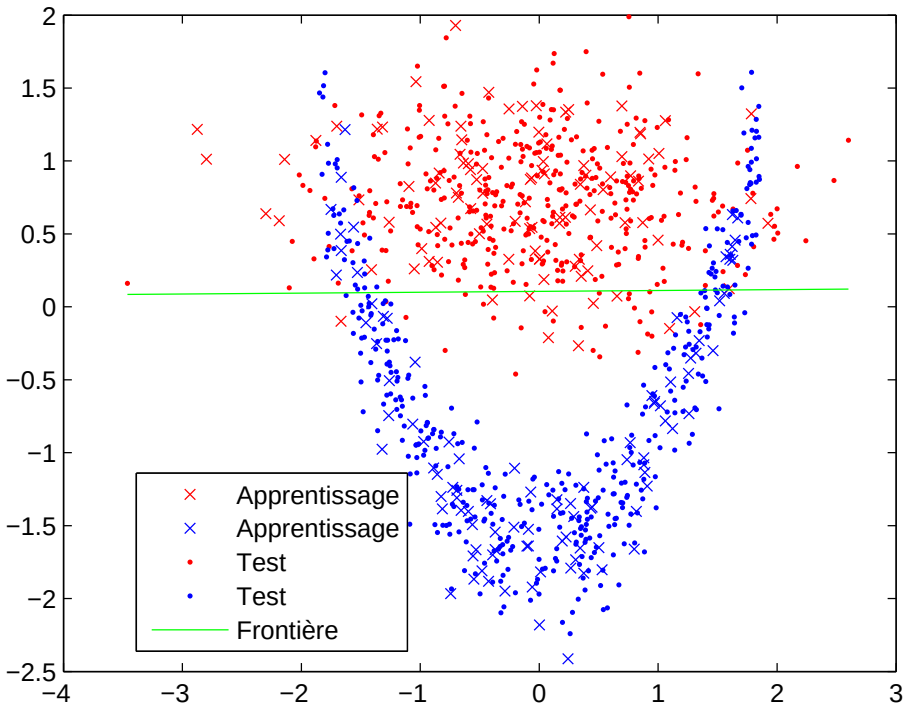
On obtient alors une erreur d'environ 18% avec cette frontière linéaire. Cette erreur est raisonnable mais reste assez importante, d'autant qu'on pourrait sans doute beaucoup l'améliorer en utilisant une frontière de décision quadratique plus complexe et plus adaptée aux données.

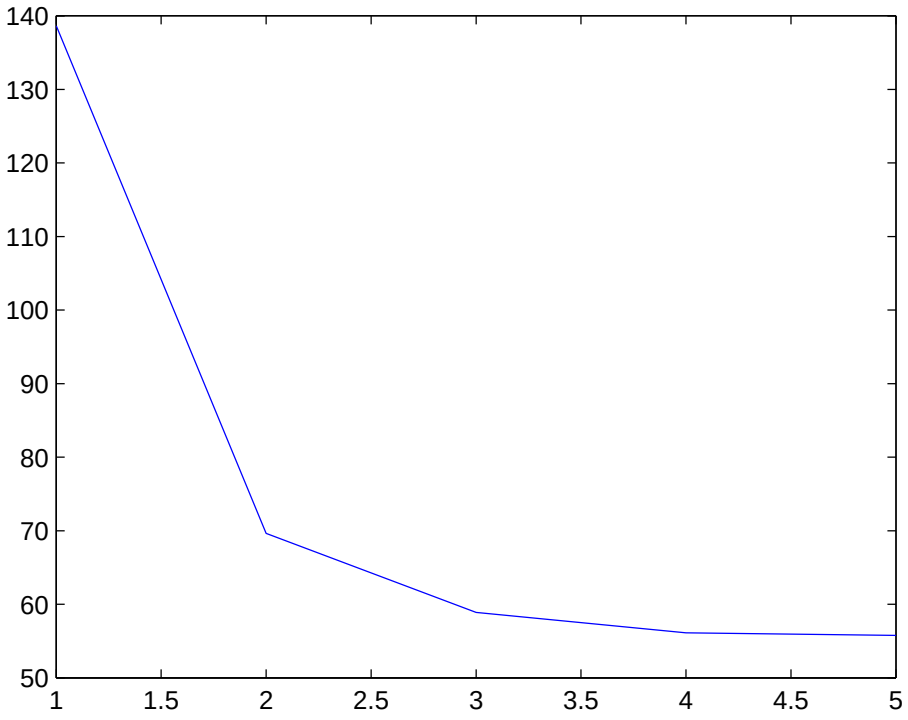
Lorsque l'on cherche à classifier les données de test à partir des paramètres de décisions obtenus avec les données d'apprentissage, on se rend compte que la différence d'erreurs est assez faible entre les données d'apprentissage et de tests.

```
% matrices de données
phiApp = [ones(napp,1) xapp];
phiTest = [ones(ntest,1) xtest];

% application de la regression logistique
[theta,L] = ma_reg_log(phiApp, zapp);

% calcul des classes
zappExp = round(probaAPosteriori(theta, phiApp));
ztestExp = round(probaAPosteriori(theta, phiTest));

% erreur
errApp = sum(zappExp ~= zapp)/napp;
errTest = sum(ztestExp ~= ztest)/ntest;

fprintf('Erreur en apprentissage : %i %%\n', round(errApp*100));
fprintf('Erreur en test : %i %%\n', round(errTest*100));

% calcul de la frontière de décision
xFront = [min(X(:,1)) max(X(:,1))]';
yFront = -(theta(1) + theta(2)*xFront)/theta(3);

% affichage
figure;
plot(xapp(zapp == 1,1),xapp(zapp == 1,2),'.r'); hold on;
plot(xapp(zapp == 0,1),xapp(zapp == 0,2),'.b');
```

```
27 plot(xtest(ztest == 1,1),xtest(ztest == 1,2),'xr');
28 plot(xtest(ztest == 0,1),xtest(ztest == 0,2),'xb');
29 plot(xFront, yFront, '-g');
30
31 legend('Apprentissage', 'Apprentissage', 'Test', 'Test', 'Frontière', 'Location', 'Best');
32
33 % evolution critere de convergence
34 figure;
35 plot(L);
```

```
Erreur en apprentissage : 11 %
Erreur en test : 17 %
```





Evolution du maximum de vraisemblance

1.4 Frontière de décision quadratique

On réalise donc une frontière de décision quadratique afin d'améliorer la performance de l'algorithme.

On calcule pour cela la régression logistique avec la matrice $\phi = [1 \;\; x_1 \;\; x_2 \;\; x_1x_2 \;\; x_1^2 \;\; x_2^2]$.

Le reste fonctionne de la même façon que pour une frontière linéaire, on a simplement changé d'espace. La seule différence est que la frontière est quadratique, donc à tracer avec la fonction `contour` de Matlab.

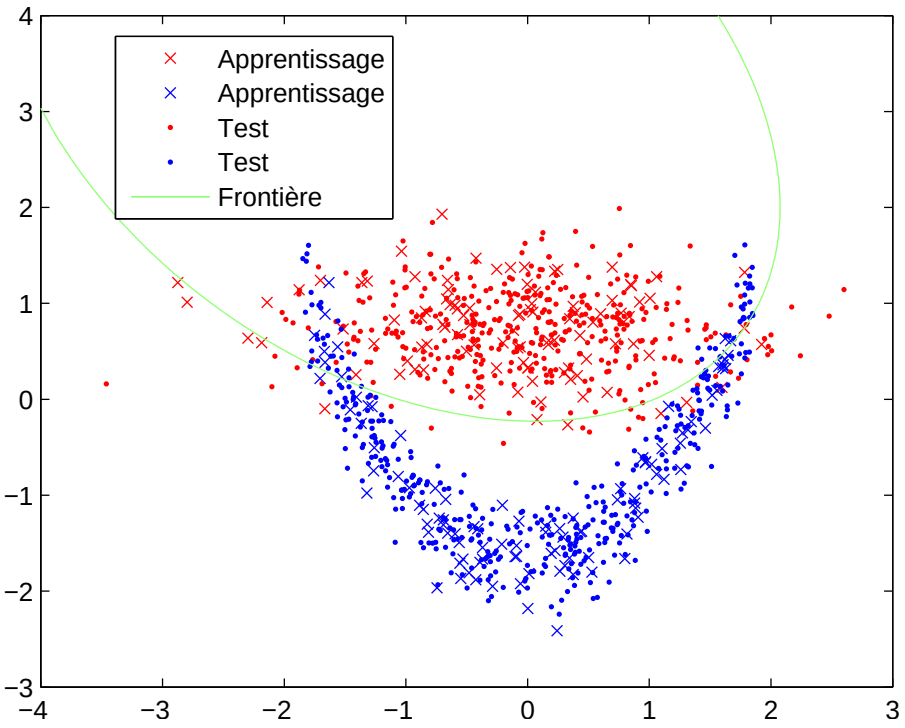
L'erreur tombe à 10%, soit presque la moitié de l'erreur avec une frontière linéaire. Cette erreur reste assez important car les données sont très mélangées autour de la frontière, il semble donc difficile de mieux séparer les données le plus excentrées des milieux de classes.

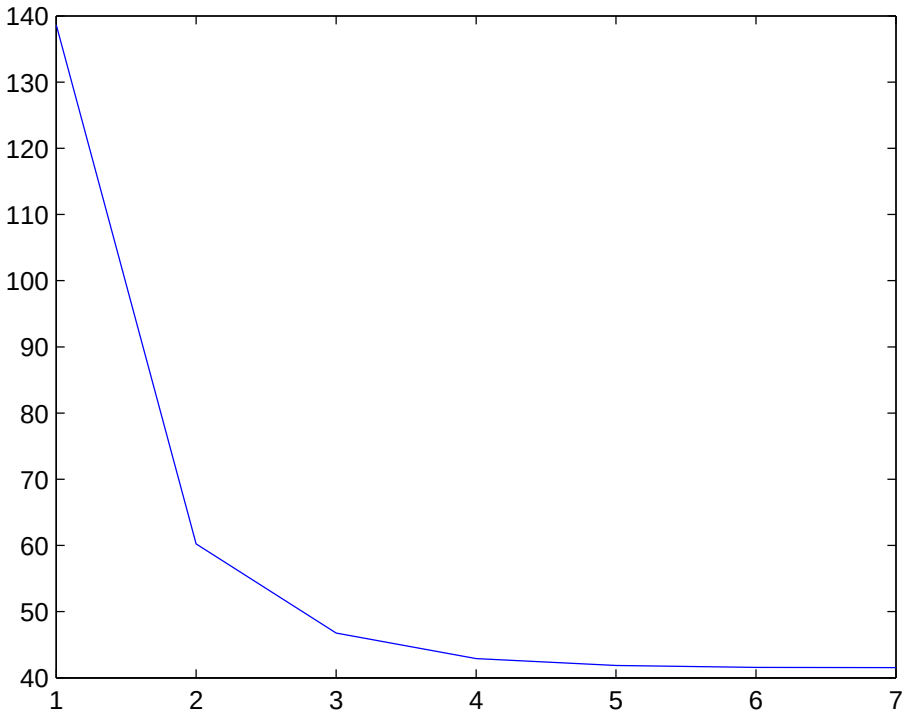
La meilleure solution pour diminuer le taux d'erreur serait de rejeter les points incertains trop proche de la frontière de décision. Cependant, l'inconvénient de cette méthode est qu'on classifie moins de points.

```
% matrices de données
phiApp = [ones(napp,1) xapp xapp(:,1).*xapp(:,2) xapp.^2];
phiTest = [ones(ntest,1) xtest xtest(:,1).*xtest(:,2) xtest.^2];

% application de la regression logistique
[theta,L] = ma_reg_log(phiApp, zapp);

% calcul des classes
zappExp = round(probaAPosteriori(theta, phiApp));
ztestExp = round(probaAPosteriori(theta, phiTest));

% erreur
errApp = sum(zappExp ~= zapp)/napp;
errTest = sum(ztestExp ~= ztest)/ntest;

fprintf('Erreur en apprentissage : %i %%\n', round(errApp*100));
fprintf('Erreur en test : %i %%\n', round(errTest*100));

% calcul de la frontière de décision
[xx yy] = meshgrid(-4:0.1:4,-4:0.1:4);
```

```
zz = theta(1) + theta(2)*xx + theta(3)*yy + theta(4)*(xx.*yy) + ...
    theta(5) * (xx.^2) + theta(6) * (yy.^2);

% affichage
figure;
plot(xapp(zapp == 1,1),xapp(zapp == 1,2),'.r'); hold on;
plot(xapp(zapp == 0,1),xapp(zapp == 0,2),'.b');
plot(xtest(ztest == 1,1),xtest(ztest == 1,2),'xr');
plot(xtest(ztest == 0,1),xtest(ztest == 0,2),'xb');
contour(xx, yy, zz, [0 0]);

legend('Apprentissage', 'Apprentissage', 'Test', 'Test', 'Frontière', 'Location', 'Best');

% evolution critere de convergence
figure;
plot(L);
```

```
Erreur en apprentissage : 10 %
Erreur en test : 12 %
```





Variation du maximum de vraisemblance

2 Régression logistique multimodale

Pour adapter le programme à 3 classes, il faudrait répéter l'opération de manière à séparer chaque groupe les uns des autres. Donc, pour l'ajout d'un (k+1)ième groupe, on aura k frontières de plus à celles déjà existantes.

Il faut donc une matrice d'appartenance aux clusters $Z \in \{0,1\}^{n \times k}$ qui se calcule grâce à la fonctions :

```
function [z, nbClasse] = EncoderLabel01_Multi(y)

    vals = unique(y);
    nbClasse = length(vals);

    z = zeros(size(y, 1), nbClasse - 1);

    for i = 1:nbClasse - 1
        z(y == vals(i),i) = 1;
    end

end
```

En notant $\Theta = [\theta_1 \ \theta_2 \ ... \ \theta_k]$ la matrice avec chaque paramètre de frontière en colonne, on peut simplement calculer les probabilités à posteriori. On aura dans la matrice de probabilités $P \in \mathbb{R}^{n \times k}$ contenant la probabilité pour chaque frontière de décision P_i dans chaque colonne.

$$P_i = \frac{e^{\theta_i \phi}}{1 + \sum_{j=1}^{k} e^{\theta_j \phi}}$$

La fonction de calcul de probabilité a été généralisée et est la même que celle disponible dans la partie 1.

L'algorithme itératif de calcul de la régression logistique multimodale est plus compliquée. Nous n'avons pas réussi à faire fonctionner cet algorithme correctement.

La méthode que nous avons appliqué était de calculer W, r et θ_i pour chaque colonne de la matrice Θ. Malheureusement, cette méthode ne semble pas fonctionner correctement.

Nous avons finalement le code suivant :

```
function [theta,L,p] = ma_reg_log_multi(phi, z)

    nbFront = size(z, 2);

    theta = zeros(size(phi,2), nbFront)+1;
    theta_old = theta + 5;
    i=1;

    L = [];

    %while norm(theta_old - theta) > 1e-1
    for i = 1:100
        theta_old = theta;
        p = probaAPosteriori(theta, phi);
        %l = -sum(z.*log(p) + (1 - z).*log(1-p));
        %L = [L l];
        for j = 1:nbFront
            W = diag(p(:,j) .* (1 - p(:,j)));
            r =  phi*theta(:,j) + W\(z(:,j) - p(:,j));
            theta(:,j) = (phi'*W*phi)\(phi'*W*r);
        end
        %i = i + 1;
    end
end
```