Machine Learning

Part-B

Members :

AOUCHICHE Kamel
FERHOUM Ryad
HABBICHE Lotfi
MEDJBER Hamza
ZOBIRI Okba

Chapitre1 : Régression

Introduction :

Dans ce premier chapitre nous allons documenter les différentes méthodes de régression utilisées sur notre jeu de données de convertisseurs d'énergie houlomotrice .

Pour rappel, nous possédons 4 bases de données représentants la simulation des positions des convertisseurs dans 4 parties de la côte sud de l'Australie (Sydney, Adelaïde, Tasmanie, Perth), chacune possède 49 attributs, 32 pour les positions de chaque convertisseur 16 pour l'énergie générée par chacun des convertisseurs et l'énergie totale générée par la ferme.

Nous possédons pour chacune de ces bases plus 72.000 instances, mais pour nous tests nous n'utiliserons que la base de Sydney car elle contient assez d'instance pour tester nous méthodes.

Objectifs :

Dans le cas générale de l'apprentissage automatique, la régression sert à estimer une valeur de sortie à partir d'un groupe de données donnés en entrée, dans notre cas cela reviens à estimer la puissance totale générée par la ferme d'énergie en se basant sur la position des convertisseurs et l'énergie générée par chacun d'eux.

Prétraitements :

Notre jeu de données comporte un grand nombre d'attributs et d'instances, il est donc nécessaire de s'assurer que les données d'entrainement soient de qualité afin que nous obtenions des résultats du même type.

- Transformation des données : il existe un énorme écart entre les attributs de positions qui représentent des valeurs de centaines de mètres et ceux de la puissance générée qui eux sont autours des dizaines de millions. Il sera donc necessaire de reduire cet ecat en divisant ces valeurs par un facteur de 10.
- Réduction des données : la fouille de données sera extrêmement longue sur les données complètes de notre base, il faudrait donc concevoir une représentation réduite qui pourrait reproduire les mêmes résultats que la version complète.

[x1….x16]
[y1….y16]
[p1….p16]
Y

[x1,x2 ,x3]
[y1,y2 ,y3]
[p1,p2 ,p3]
Y

1.Régression linéaire :

La régression linéaire a comme but de décrire la variation d'une variable dépendante (y) associée aux variations de plusieurs variables indépendantes, Donc l'objectif est de trouver une fonction d'estimation en se basant sur les données dites d'entrainement :

$$y(x) = f(x1\ldots, y1\ldots, z1\ldots)$$

- *Pré-lancement : - mettre en place nos données d'entrainement dans une matrice X, et les données de sortie dans un vecteur Y.*
- *Fonction de calcul du cout : une fonction qui permet de calculer la différence entre le résultat obtenue et le résultat estimé, il sera donc nécessaire de minimiser cette fonction.*

$$J = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}(x^{(i)}) - y^{(i)})^2$$

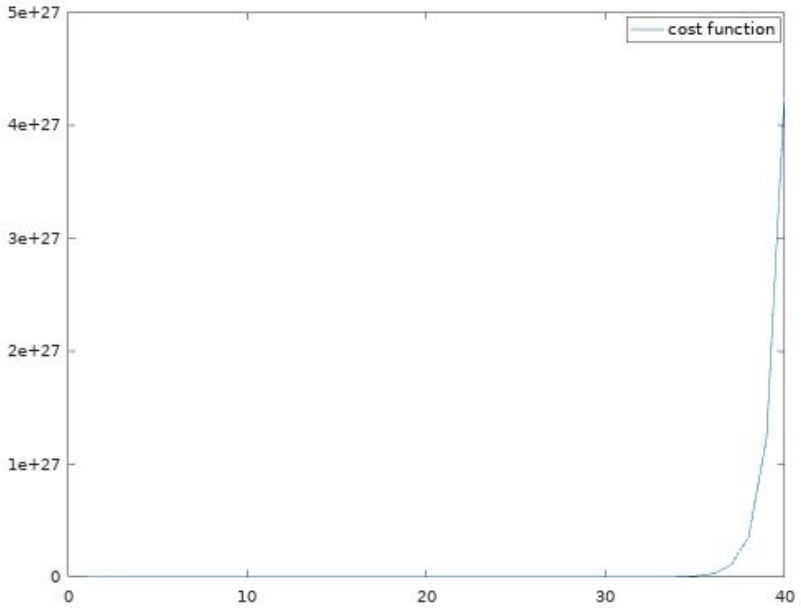
1.1.Gradient descent :

un algorithme itératif qui permet de mettre à jour les poids de la fonction linéaire à chaque itération en se basant sur la fonction du cout, il sera nécessaire de fixer le taux d'apprentissage alpha a une valeur adéquate afin de réussir a converger vers le minimum globale.

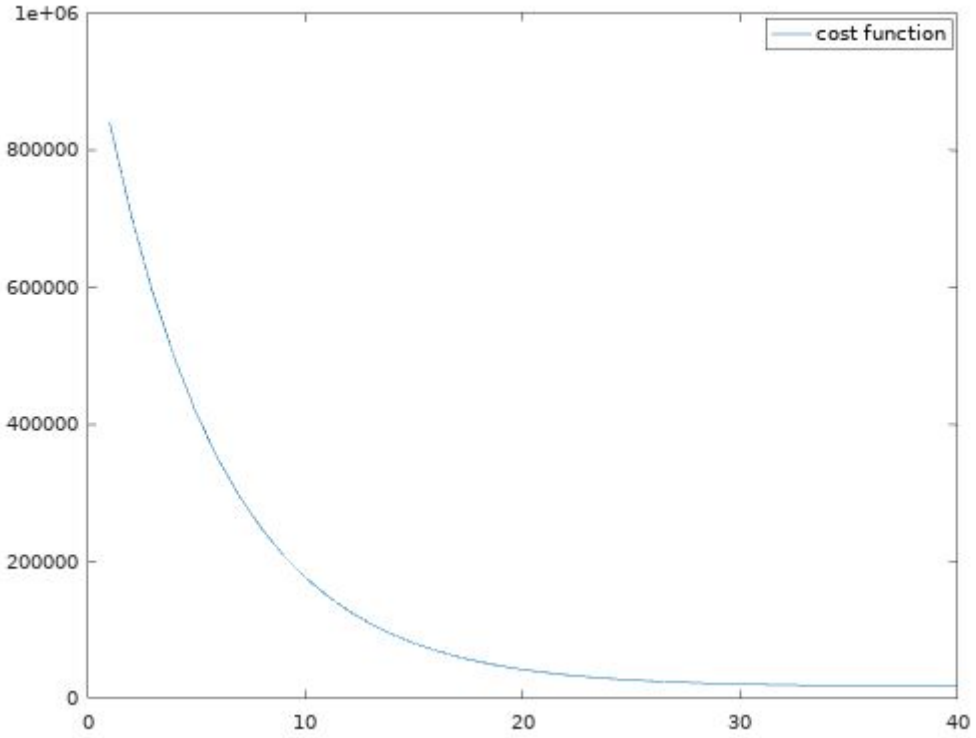
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \text{(for all } j\text{)}$$

Implémentation et tests :

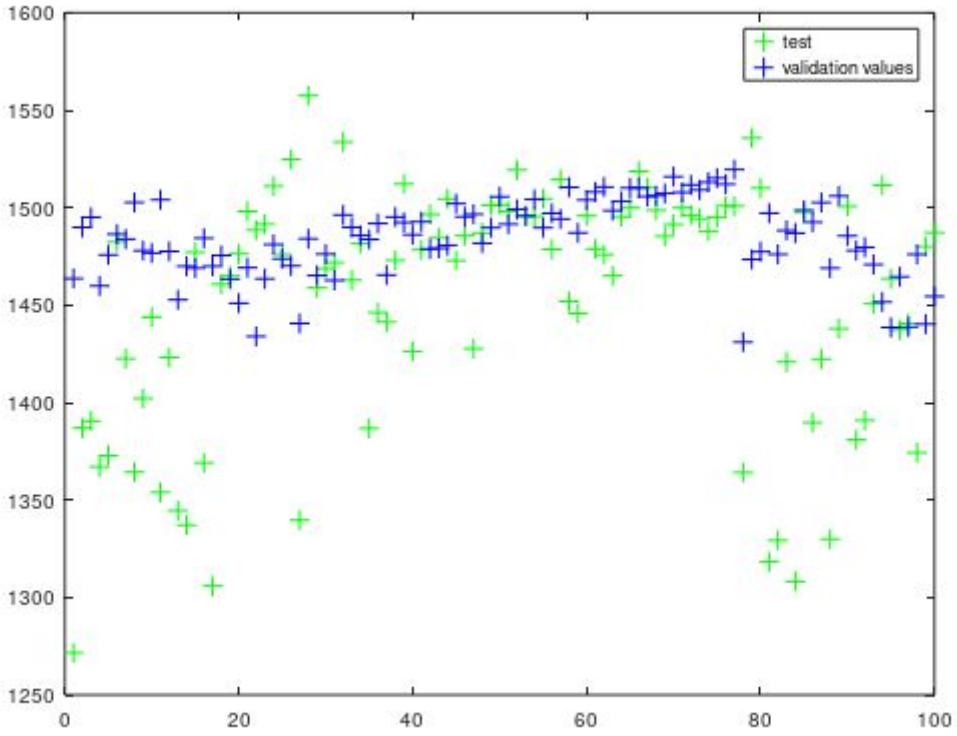
Apres avoir fixé le nombre d'itérations et le taux d'apprentissage a 1, nous ne sommes pas arrivé a un point de convergence dans la fonction du cout.



Après avoir tester différents taux d'apprentissage nous somme arrivés a converger en initialisant alpha a 0.03.



Les résultats de ce test seront représenté dans ce graphe ou le nuage de points en vert représente les résultats du test et ceux en bleu les données visés.



Nous remarquerons une légère marge d'erreur sur les resultats obtenus, Marge qui sera a minimiser en modifiant le taux d'apprentissage et les nombre d'iterations.

1.2.Normal Equation:

La deuxième méthode pour calculer le vecteur theta c'est la «Normal Equation » qui est une approche analytique de la régression linéaire avec la fonction du moindre coût carré. Nous pouvons directement trouver la valeur de θ sans utiliser Gradient Descent. Suivre cette approche est une option efficace et plus rapide[1],

le principe est le suivant :

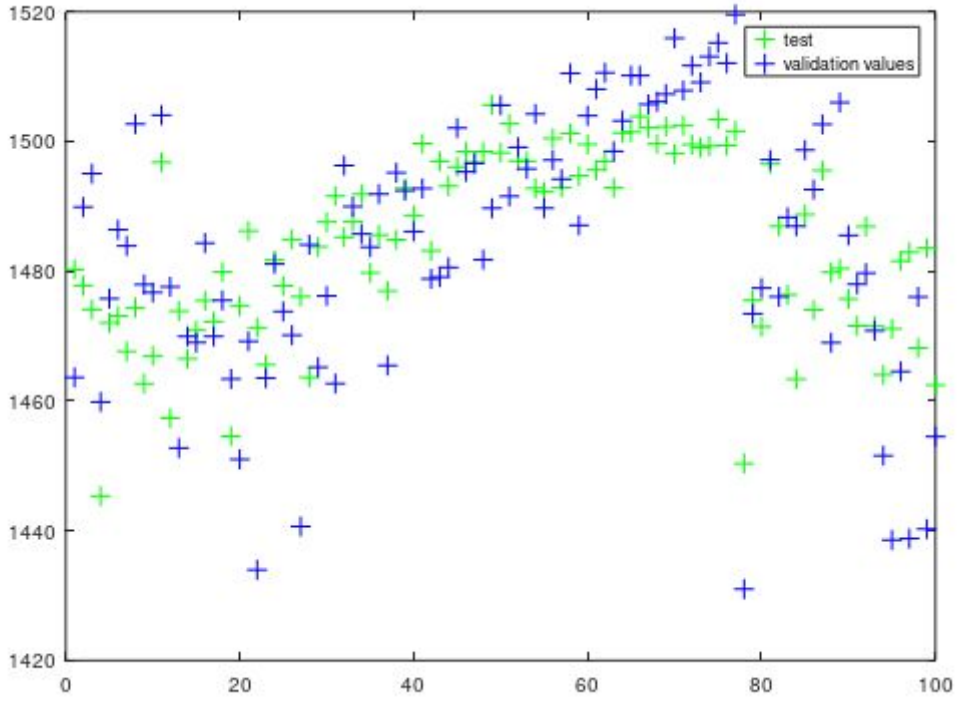
- Afin d'avoir le minimum local de chaque fonction mathématique, on calcul la fontion de dérivé et on calcule l'équation f'(x) = 0.
- Dans notre cas, notre fonction est J(theta) présentée sous la forme :

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$Cost'(\theta) = 0$ est équivalent à écrire : $\theta = \left(X^T X\right)^{-1} . \left(X^T y\right)$

Note : Dans le cas du Normal Equation et contrairement à la méthode de Gradient Decent, on ne peut pas suivre le déroulement de la fonction du cout car la Normal Equation n'est pas une fonction itérative.

Pour la phase de test et validation nous avons utilisé nos données de validation qui représente 20% des données et on a réussis à réaliser le graphe suivant : en vert la vrai valeur de Y et en bleu la valeur obtenue par le modèle :



Conclusion

Il n'existe pas une méthode meilleure qu'une autre, le choix de celle-ci dépond des caractéristiques des données et du type du problème.

Chapitre2 : Classification

Introduction :

Ce chapitre va contenir les différentes méthodes de classifications appliquées sur le jeu de donnée wifi_localization choisis dans la partie A.

Mais d'abord un petit rappelle sur la base de donnée:

La base de données contient 8 attributs (7 routeurs,1 entier) 2000 instances regroupée en 4 classes qui sont :

Nom de la classe	Numero de la classe	Nombre d'espèces appartenant à la classe
Chambre	1	500
Salle de conférence	2	500
Cuisine	3	500
Salle de bain	4	500

La base de données étant assez cohérente aucune normalisation n'a été appliquée.

1.Régression logistique :

La régression logistique est une méthode statistique largement utilisée en apprentissage automatique. Elle constitue un cas particulier de modele lineaire.

Son travaille est d'analyse si une ou plusieurs variables indépendantes déterminent ou non un résultat (pour lequel il n'y a que deux résultats possibles).

Or son but finale est de prédire si une donnée test fais partie d'une classe ou d'une autre.

Conception de la solution:

pour la régression logistique nous avons implementer plusieurs fonction:

Le gradient descent :Cette fonction consiste a calculer les meilleures valeurs des paramètres thêta et aussi les valeurs de cost que on va detailler plus tard. Elle prend comme parametre :

theta: Vecteur des paramètres initialisé a 0 de taille m+1, m étant le nombre d'instance dans notre training data.
X :les inputs sous forme de matrice.
Y : un vecteur de taille M qui prend la valeur 1 ou 0.
Alpha : un paramètre qui permettra à cost de converger.
Lambda : paramètre pour la régularisation.

La fonction Hypothèse: retourne la valeur de l'hypothese vu en cours

Elle prend en entrée:

x : vecteur contenant thêta*X.

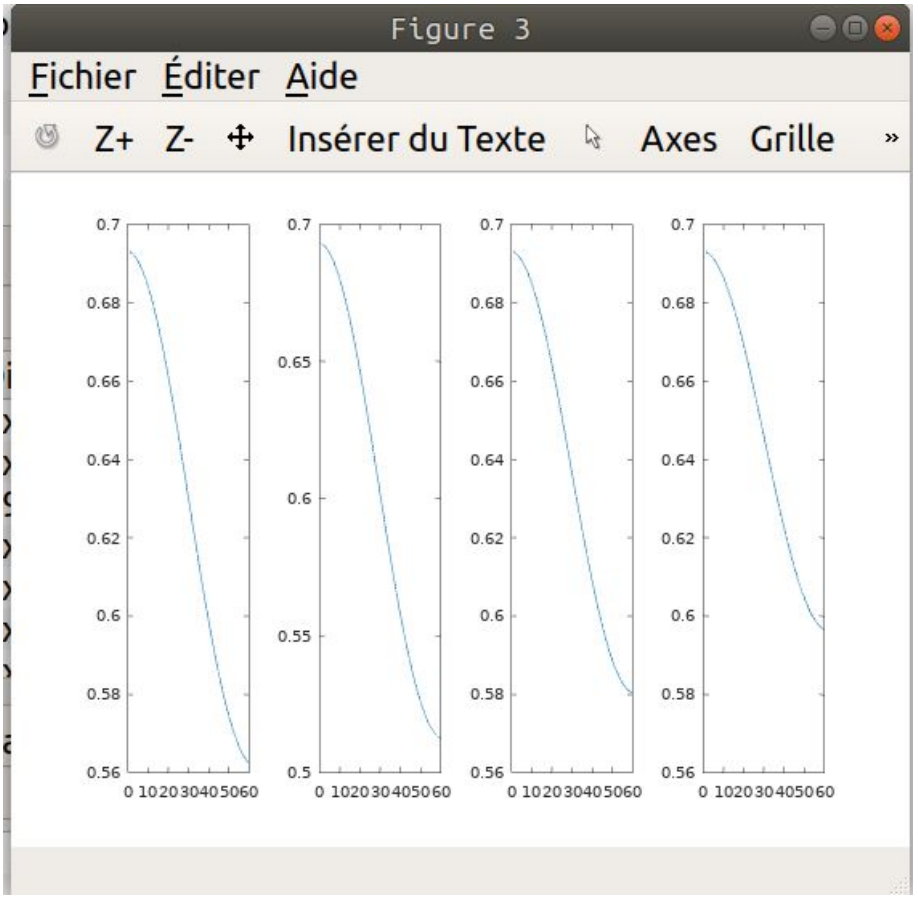
La procédure suivit :

D'abord on s'est restri a 60 % des données pour le training data et les 40 % restante seront dédiée aux test et pour la validation.

Ensuite on a initialiser les paramètres d'entrée : thêta, Nombre d'Itération, Alpha et Lambda.

On arrive ensuite au gradient descent qui va nous permettre de calculer les meilleures valeurs de Thêta, on appelle donc 4 fois cette fonction en suivant la méthode one vs all .

La figure ci-dessous nous montre les cost des quatre classe.



Pour obtenir de meilleurs résultats nous avons testé plusieurs valeurs de Lambda et Alpha, le tableau suivant résume cette opération :

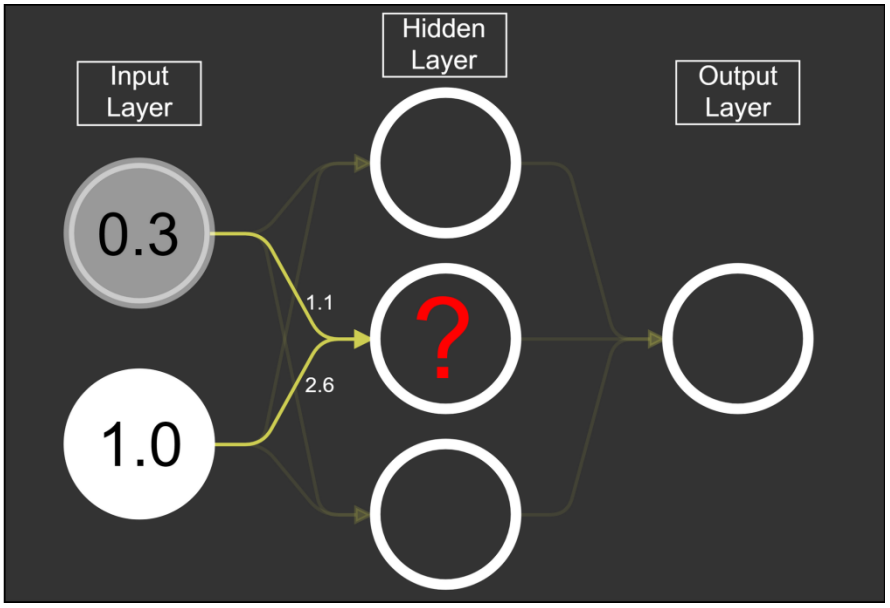
itération	Alpha	Lambda	COST
120	0.00004	0.5	70%
200	0.001	1	30%
300	0.01	0.05	50%

2.Les réseaux de neurones :

Les réseaux de neurones sont des algorithmes inspirés du cerveau humain, ils consistent de neurones aussi appelés nœuds.

Les nœuds sont connecté entre eux, chacun nœud possède un identifiant et chaque liaison un poids.

Ils sont répartis en couche d'entrée, multiple couches cachés et couche de sorties.



Back Propagation :

C'est le cœur de chaque réseau de neurones, on l'utilise pour calculer les gradients d'une manière efficace, afin d'ajuster les poids des relations.

$$Loss(y, \hat{y}) = \sum_{i=1}^{n} (y - \hat{y})^2$$

$$\frac{\partial\, Loss(y,\hat{y})}{\partial W} = \frac{\partial Loss(y,\hat{y})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial W} \quad \text{where } z = Wx + b$$

$$= 2(y - \hat{y}) * \text{derivative of sigmoid function} * x$$

Feed Forward :

La fonction qui nous permettra de calculer le poids.

Conception de la solution :

Nous avons 4 classe pour l'emplacement des utilisateur ce qui nous amène a transformé la sortie «classes» en 4 output, notre but et de prédire à quelle classe appartient nos données de test.

On crée les matrice poid1 et poid2 aléatoirement avec les taille des inputs la couche caché et le output. On retire les y des inputs puis on fait l'entraînement sur 43données/58, pour chaque itération

on fait un feedforward ou un back propagation en même temps et tout cela MaxItteration jus-qua avoir des résultats correcte.

La figure suivante représente le cost et les résultats des test pour y nous avons pris 20 échantillons et pour le deuxième test 25 échantillons:

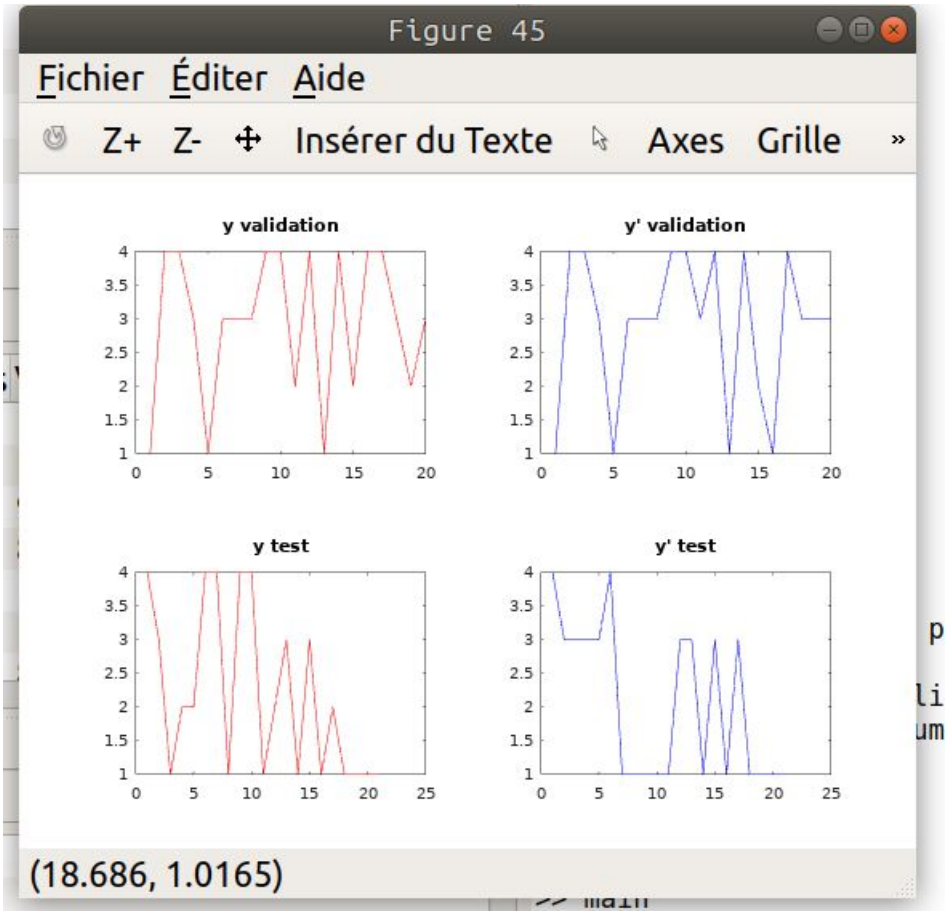


Figure 45
Fichier Éditer Aide
Z+ Z- Insérer du Texte Axes Grille »
y validation
y' validation
y test
y' test
(18.686, 1.0165)

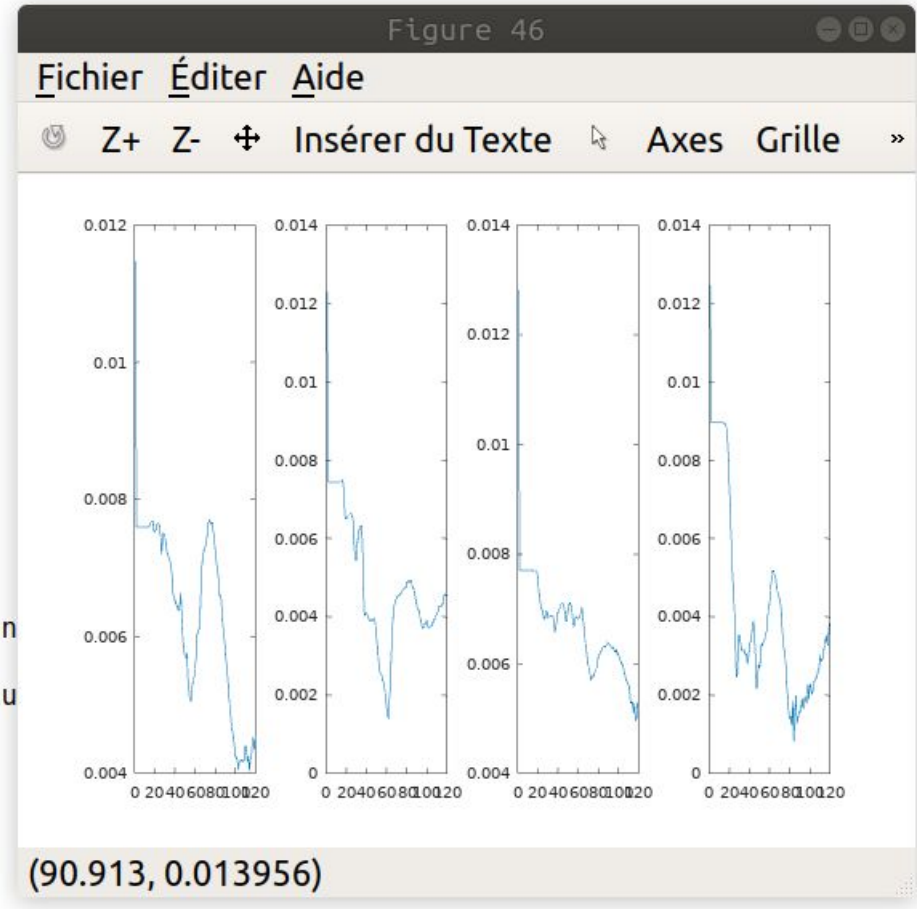


Figure 46
Fichier Éditer Aide
Z+ Z- Insérer du Texte Axes Grille »
(90.913, 0.013956)

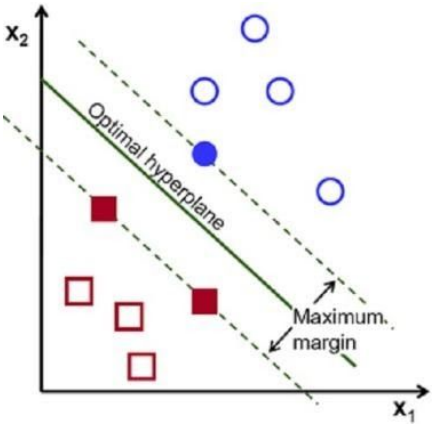
Pour de meilleurs résultats on a effectué plusieurs essais en utilisant différente valeur d'alpha :

ALPHA	RESULTAT
0,001	Converge rapidement
0,01	Ne Converge pas
1	Ne converge pas

RESULTAT : Validation 15/20 juste, Test 15/25 juste

3.Support Vector Machine(SVM) :

Les SVM ont été développés dans les années 1990 à partir des considérations théoriques de « Vladimir Vapnik » sur le développement d'une théorie statistique de l'apprentissage : la théorie de « Vapnik-Chervonenkis ». Ils ont rapidement été adoptés pour leur capacité à travailler avec des données de grandes dimensions, le faible nombre d'hyper paramètres, leurs garanties théoriques, et leurs bons résultats en pratique. (2)



Outils utilisés :

Pour cette partie on a utilisé la bibliothèque ***LIBSVM***.

LIBSVM est une bibliothèque qui regroupe un ensemble d'algorithmes d'apprentissage automatique (classification, régression...), toutes issues de l'approche par les machines à support de vecteurs (Support Vector Machine)

Afin d'utiliser cette dernière nous avons téléchargé la bibliothèque sur le site officiel : https://www.csie.ntu.edu.tw/~cjlin/libsvm/ et nous avons choisis la dernière version : Libsvm-3.24

Cette bibliothèque contient des fichiers écrits en C++ et en C qui permettent de faire l'apprentissage, les fichiers peuvent être compilés avec un compilateur C++, elle contient aussi un fichier nommée make qui va lancer la compilation de ces fichiers et charger les fichiers : 'libsvmread.mex', 'libsvmwrite.mex', 'svmtrain.mex', et 'svmpredict.mex'.

Démarche :

Afin de procéder à notre apprentissage nous avons suivi les étapes suivantes :

- Préparation du Data set : en le divisant en 30% Test et 70% Train.
  Où nous avons utilisé la méthode des 60% 20% 20% pour diviser la base de donnée respectivement en donnée d'apprentissage, donnée de test et données de validation et cela de manière aléatoire.

- Phase d'apprentissage : Trouver les meilleurs paramètres et générer le modèle.
  Nous avons utilisé la méthode des 60% 20% 20% pour diviser la base de donnée respectivement en donnée d'apprentissage, donnée de test et données de validation et cela de manière aléatoire.

- Modifier les paramètres : Dans le but de trouver les meilleure paramètres , nous avons travaillé avec la fonction « svmtrain » qui prend comme paramètres d'entrée « Training_data » et « classes » et l'ensemble d'options qui sont :
  - -s ---SVC : pour dire qu'il s'agit de multi-classes SVM
  - -t –radial : pour la fonction du kernel « radial basis funtion »
  - -c : pour désigner le paramètre C
  - -g : pour donner le paramètre Gamma

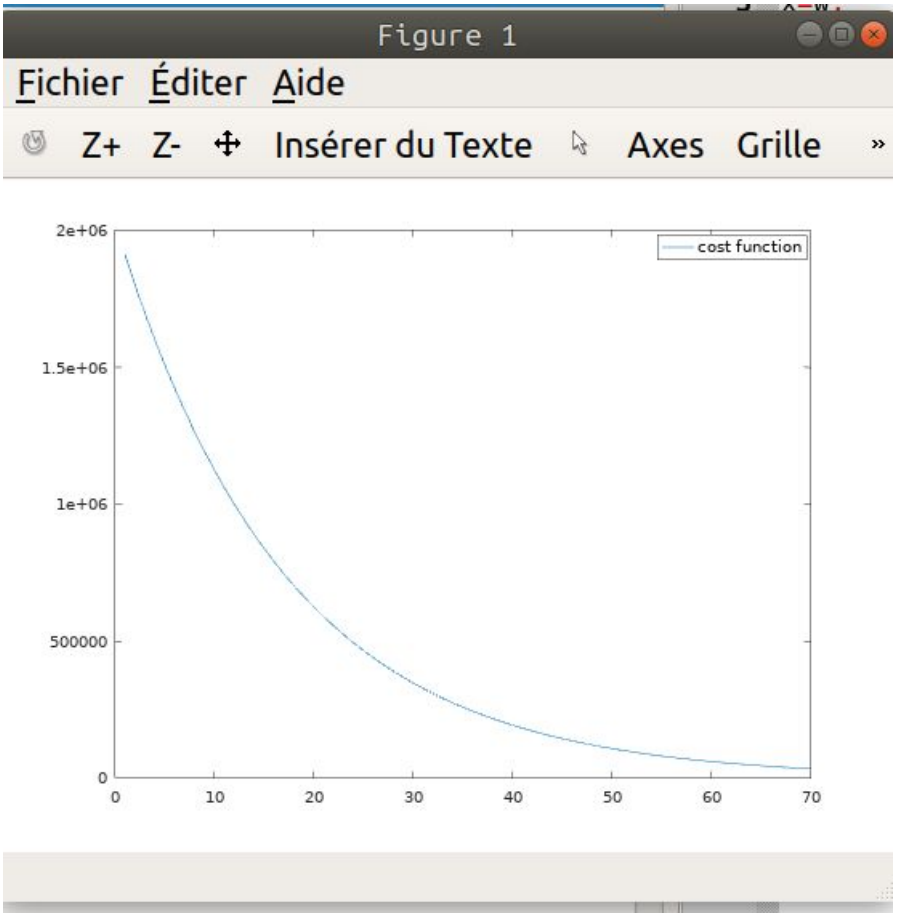
- Tester le modèle résultat : Afin de tester le modèle obtenu, nous avons utilisé la fonction « svmpredict » qui exige le paramètres d'entrées « Test_data » et « Test_classes ».
  En appliquant cette fonction qui retourne le pourcentage de précision (Accuracy), on a obtenu le résultat suivant :

Accuracy = 95.2381% (20/21) (classification)

4.Regression linéaire :

Nous avons 4 classe pour l'emplacement des utilisateur ce qui nous amène a transformé la sortie «classes» en 4 output, notre but et de prédire à quelle classe appartient nos données de test on appliquant l'algorithme de régression.

La figure suivante représente le cost :



Conclusion

Il n'existe pas une méthode meilleure qu'une autre, le choix de celle-ci dépond des caractéristiques des données et du type du problème.

Annexe :

Logistic regression :

cost.m

```
function resu=cost(x,y,theta,lambda)
 s=0;
 m=size(x,1);
 for i=1:m
  if y(i)==0,
       s=s-log(1-hypothese(x,theta,i)); #pour y=0 la classe negative

  endif
  if y(i)==1,
   s=s-log(hypothese(x,theta,i));#pour y=1 la classe positive
  endif
 endfor

resu =
-(1/m)*sum(y.*log(sigmoid(x(:,2:8)*theta))+(1-y).*log(1-sigmoid(x(:,2:8)*theta)))+(lambda/(2*m))*sum(theta.^2);

endfunction
```

gradient.m

```
function [vecteur_cost,theta]=gradient(x,y,theta,alpha,iter,lambda)
 t1=0;
```

```
  t0=0;
  t2=0;
  t3=0;
  t4=0;
t5=0;
t6=0;
  m=size(x,1);
  vecteur_cost=[];
  for i=1:iter
    for j=1:m
      t0=t0+(hypothese(x,theta,j)-y(j));
    endfor
      for j=1:m
      t1=t1+(hypothese(x,theta,j)-y(j))*x(j,2);
    endfor
    for j=1:m
      t2=t2+(hypothese(x,theta,j)-y(j))*x(j,3);
    endfor
      for j=1:m
      t3=t3+(hypothese(x,theta,j)-y(j))*x(j,4);
      endfor

      for j=1:m
      t4=t4+(hypothese(x,theta,j)-y(j))*x(j,5);
    endfor
       for j=1:m
      t5=t5+(hypothese(x,theta,j)-y(j))*x(j,6);
    endfor
       for j=1:m
      t6=t6+(hypothese(x,theta,j)-y(j))*x(j,7);
      endfor
```

```
    theta(1)=theta(1)-(alpha*t0)/m;
    theta(2)=theta(2)-(alpha*t1)/m;
      theta(3)=theta(3)-(alpha*t2)/m;
           theta(4)=theta(4)-(alpha*t3)/m;
                    theta(5)=theta(5)-(alpha*t4)/m;
                    theta(5)=theta(5)-(alpha*t4)/m;
      theta(6)=theta(6)-(alpha*t5)/m;
                        theta(7)=theta(7)-(alpha*t6)/m;


  vecteur_cost=[vecteur_cost;cost(x,y,theta,lambda)];

endfor



endfunction
```

hypothese.m

```
function g = hypothese(t)
g = 1.0 ./ (1.0 + exp(-t));
end
```

hypo.m

```
function h=hypo(x,theta,i)
 h=theta(1)+theta(2)*x(i,2)+theta(3)*x(i,3)+theta(4)*x(i,4);
 endfunction
```

script.m

```
clear ; close all; clc

load w.txt;
data=w;
```

```
x=data(:,1:7);


x=x*-1;
x=[ones(size(x,1),1),x];
x=x(1:160,:);


y=data(:,8);
y=y(1:160,:);


theta=zeros(5);


alpha=0.0000001;
iter=60;
lambda=0.9;
z=y;
v=x;
vecteur_cost=[];
index_theta = zeros(size(x, 2)-1, 1);
coste=zeros(4,iter);


 for i=1:4,
  pos = find(y == i);
       neg = find(y ~= i);
         z(pos ,1) = 1;
         z(neg ,1) = 0;


         [vecteur_cost,theta] = gradient(x,z,index_theta,alpha,iter,lambda);
         theta_result(i,:)=theta';
         coste(i,:)=vecteur_cost;
endfor
```

```
plotCost(coste);
```

plotCost.m

```
function []=plotCost(vecteur)
 figure(3);
 subplot(1,4,1);
 plot(vecteur(1,:));
  subplot(1,4,2);
 plot(vecteur(2,:));

    subplot(1,4,3);
    plot(vecteur(3,:));
      subplot(1,4,4);
       plot(vecteur(4,:)) ;
 endfunction
```

Regression linaire :

classification

main.m

```
clear ;close all;
load w.txt;
x=w;
y=x(1:150,8);

x=x(1:150,1:7);
x=[ones(size(x,1),1) x];

d=size(x,2);
x=x*-1;
```

```
theta = randperm(8);
theta=theta';
alpha=0.000001;
nbrIter=70;

theta=gradient(x,y,alpha,theta,nbrIter);
x=x(1:100,:);
y=y(1:100,:);
figure(2);
plot(x*theta,'+','MarkerSize',10,'Color','g');
hold on;
plot(y,'+','MarkerSize',10,'Color','b');
hold on;
legend('test','validation values');
```

Hypothèse.m

```
function hyp=Hypothese(x,theta)
  hyp=x*theta;
endfunction
```

gradient.m

```
function theta=gradient(x,y,alpha,theta,iter)
  m=size(x,1);
  vecteur=[];
  t=zeros(8,1);
 for i=1:iter
     hypot = Hypothese(x,theta);
   for k = 1:8,
       t(k) = theta(k)-(alpha/m)*sum((hypot-y).*x(:, k));
```

```
      theta(k) = t(k);
   endfor

 vecteur=[vecteur;cost(x,y,theta)];

endfor
figure(1);
plot(vecteur(:,1),'-');
 legend('cost function');

endfunction
```

cost.m

```
function y=cost(x,y,theta)
     m = length(y);
   hypothese = Hypothese(x,theta);
   y= (sum((hypothese-y).^2)+sum(theta).^2)/(2*m);
endfunction
```

SVM

main.m

```
clc;
clear;
load w.txt;
data=w;
x=data(1:100,1:7);
y=data(1:100,8);
```

```
rand_nums=randperm(101);
%60% pour le training

donnee_apprentissage=x(rand_nums(1:60),:);
classe_label=y(rand_nums(1:60),:);
%20% pour le test
donnee_test=x(rand_nums(61:80),:);
classe_test=y(rand_nums(61:80),:);
%et enfin 20% pour la validation
donnee_de_valid=x(rand_nums(81:101),:);
classe_valid=y(rand_nums(81:101),:);

% J'ai choisis la finction rbf et un C=1 et gamma=1 cross validation de 5
% folds et h pour une heuristique pour allez plus rapide
training = svmtrain(classe_label, donnee_apprentissage, '-s ---SVC -t --radial basis function -c 1 -g 1  -h -v 5 ');
[test_predi] = svmpredict(classe_test, donnee_test, training);
[class_predi] = svmpredict(classe_valid, donnee_de_valid, training);
```

Neural Network

Classification

deriv_hypothse.m

```
function [result] = deriv_hypothse(x)
  for i=1:size(x,2)
   result(1,i)=exp(-x(1,i))/((1+exp(-x(1,i)))^2);
   end
endfunction
```

hypothese.m

```
function s = hypothese(x)


 for i=1:size(x,2)
   s(1,i)=1/(1+exp(-x(1,i)));
   end
endfunction
```

feedforward.m

```
function [layer,output] = feedforward (input,poid1,poid2)
   layer1 = hypothese(input*weight1);


   output= hypothese(layer1*weight2);


endfunction
```

backprobagation.m

```
function [poid_result1,poid_result2] = backprobagation(input,layer,poid1,poid2,y,output)
   densite_poid2=layer'*(2*(y(1,:)-output).*deriv_hypothse(output))  ;#derivé d1

densite_poid1=input'*(((2*(y(1,:)-output).*deriv_hypothse(output))*poid2').*deriv_hypothse(layer));
#dérivé d2



   poid_result2=poid2+densite_poid2*0.1;#gradient theta2
   poid_result1=poid1+densite_poid1*0.001;#gradient theta1
endfunction
```

main.m

```
 load w.txt;
 x=w;
nbr_output=4;
```

```
y=zeros(size(x,1),nbr_output);#x=x(1:500,1:19);ableau pour le nombre de output

nb_couche_cache=8;

layer1 = zeros(1,nb_couche_cache); #martrice 1x8 de la couche caché



 for i=1:size(y,1) #modifier le features en une valeur entiere en tableau de 1 et 0

   y(i,x(i,8))=1;

  end

x(:,8) = [];#enlever la colonne de y de la matrice des input

x=x*-1;

poid1= rand(size(x,2),nb_couche_cache); #generer aleatoirement le tableau des poids de la premiere couche

poid2 = rand(nb_couche_cache,nbr_output); #generer aleatoirement le tableau des poids de la 2eme couche

coste=[];

nbr_iteration=120;


for k=1:nbr_iteration

  c_intermediaire =[];

for j=1:100

  [layer1,output]= feedforward(x(j,:),poid1,poid2); #caclule du output 'hypothese' avec les poid en entrer

  [poid1,poid2]=backprobagation(x(j,:), layer1,poid1,poid2,y(j,:),output); #modifier les poids en fonctions du output trouvé

  c_intermediaire= [c_intermediaire; (y(j,:)-output).^2];#calcul du coste


end

coste=[coste;sum(c_intermediaire)];

end

m=size(x,1);


coste=coste/(m*2);#division sur 2*m
```

```
plot_result(x,poid1,poid2,output);
plot_cost(coste);
```

plot_cost.m

```
function [] = plot_cost(cost)
 figure
  subplot(1,4,1);
  plot(cost(:,1))
  subplot(1,4,2);
   plot(cost(:,2))
  subplot(1,4,3);
  plot(cost(:,3))
  subplot(1,4,4);
  plot(cost(:,4))

endfunction
```

plot_result.m

```
function  plot_result (input,w1,w2,y)
 load w.txt;
 in=w;

  resultat_validation=[];
  resultat_test=[];
  for i=1:20
    [xx,output]= feedforward(input(i,:),w1,w2);
    [Max1,result1] = max(output(1,:));
    resultat_validation=[resultat_validation;in(i,8),result1 ];
   end
   for i=21:41
```

```
 [xx,output]= feedforward(input(i,:),w1,w2);
 [Max2,result2] = max(output(1,:));
 resultat_test=[resultat_test;in(i,8),result2];
end
figure
subplot(2,2,1);
plot(resultat_validation(:,1),'r');
title("y validation");
subplot(2,2,2);
plot(resultat_validation(:,2),'b');
title("y1validation");
subplot(2,2,3);
plot(resultat_test(:,1),'r');
title("y test");
subplot(2,2,4);
plot(resultat_test(:,2),'b');
title("y1 test");
```

endfunction

NormalEquation :

regression

normalequ.m

```
function theta=normalequ(x,y,theta)

 theta= zeros(size(x,2),1);

 theta=pinv(x'*x)*x'*y;

endfunction
```

```
clear;close all;

alpha=0.0000000000000008;

nbrIter=200;

x=importdata('Sydney_Data.csv');

x=x/1000;

y=x(1:200,49);

z=[];

x=x(1:200,1:48);

z=[x(:,1:3),x(:,16:18),x(:,32:34)];

size(z)


d=size(x,2);

theta = randperm(10);

theta=theta';

z=[ones(size(z,1),1),z];

z=z(1:100,:);

y=y(1:100,:);

theta1=[]

theta1=normalequ(z,y,theta);

 figure(4);

plot(z*theta1,'+','MarkerSize',10,'Color','g');

hold on;

plot(y,'+','MarkerSize',10,'Color','b');
```

```
hold on;
legend('test','validation values');
```

Regression linaire

regression

main.m

```
clear;close all;
alpha=0.0000000000000008;
nbrIter=200;
x=importdata('Sydney_Data.csv');
x=x/1000;
y=x(1:200,49);
z=[];
x=x(1:200,1:48);
z=[x(:,1:3),x(:,16:18),x(:,32:34)];
size(z)

d=size(x,2);
theta = randperm(10);
theta=theta';
z=[ones(size(z,1),1),z];

theta=Gradient_decent(z,y,alpha,theta,nbrIter);
z=z(1:100,:);
```

```
y=y(1:100,:);
figure(2);
plot(z*theta,'+','MarkerSize',10,'Color','g');
hold on;
plot(y,'+','MarkerSize',10,'Color','b');
hold on;
legend('test','validation values');
```

Hypothese.m

```
function hyp=Hypothese(x,theta)
  hyp=x*theta;
endfunction
```

Gradient_decent.m

```
function theta=Gradient_decent(x,y,alpha,theta,iter)
  m=size(x,1);
  vecteur=[];
  t=zeros(10,1);
 for i=1:iter
      hypot = Hypothese(x,theta);
    for k = 1:10,
        t(k) = theta(k)-(alpha/m)*sum((hypot-y).*x(:, k));
        theta(k) = t(k);
    endfor

  vecteur=[vecteur;cost(x,y,theta)];
vecteur
```

```
endfor
figure(1);
plot(vecteur(:,1),'-');
 legend('cost function');


endfunction
```

cost.m

```
function y=cost(x,y,theta)
     m = length(y);
   hypothese = Hypothese(x,theta);
   y= (sum((hypothese-y).^2)+sum(theta).^2)/(2*m);
endfunction
```

plot_cost.m

```
function plot_cost(inter,cost)
  plot(inter,cost);
endfunction
```

Références

[1] *https://www.geeksforgeeks.org/ml-normal-equation-in-linear-regression/*

[2] *https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support*

Who Did What

AOUCHICHE Kamel & FERHOUM Ryad *Régression(Code + Report)*

HABBICHE Lotfi & MEDJBER Hamza & ZOBIRI Okba *Classification(Code + Report)*