République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Ben Youcef Benkhedda Alger 1

Faculté des sciences

Département mathématiques et informatique

Mini projet parité B

Apprentissage automatique

Machine Learning

Réalisé par :

- BALA Abdelkarim.
- ADDAD Malek
- AITALLALA Yazid.
- CHETOUAH Hadjer.
- MEHADJBIA Abdelhak.
- ALILICHE Fares.

- Class : M1
- Team : 09
- 2020/2019

introduction

Machine Learning (apprentissage automatique) est une nouvelle technologie informatique, elle a été apparue la seconde moitié de 20 éme siècle, né à partir du domaine de l'intelligence artificielle permet aux ordinateurs de prédire des nouveaux résultats en analysant des données.

Il existe deux types d'apprentissage :

1. **Apprentissage supervisé :** apprendre à l'ordinateur comment faire quelque chose au début il faut décrire à la machine les informations nécessaires et les critères à analyser donc la machine apprend de ces entrées et prédit les sorties, il utilise soit la méthode de classification ou régression.

a) Classification : consiste à regrouper l'ensemble de donnée en classes de tel sorte que les nouvelles données sont prédites en fonction des données historiques.

b) Régression : consiste à trouver une fonction mathématique qui permettra de donner une estimation des résultats de nouvelles données

Problème classification

I) introduction

Dans cette partie on veut voir les différentes méthodes de classification (régression logistique, gradient descent, SVM, arbre de décision, équation normale) qui nous avons appliquées sur la data set de BUPA.

- ✓ Rappelle sur la base de données « BUPA »

Nombre d'instants	345
Nombre d'attribues	7
Nombre des class	2
Nom des classes	0(absence de trouble)/1(présence de trouble)
Nombre d'espèces appartenant à la classe	200/145

Remarque :

- ✓ Nous avons choisir la 7eme attribue qui est le sélecteur comme un out put.
- ✓ Les 5 premiers attribues sont transformée en nous divisons sur 100 pour réduire l'intervalle d'étude.

II) régression logistique

La régression logistique est un technique statistique de prédiction, il est définie sur l'intervalle [-∞ ∞+] par f(Z) = 1/1+exp (-Z) tel que **0 < f(Z) < 1**

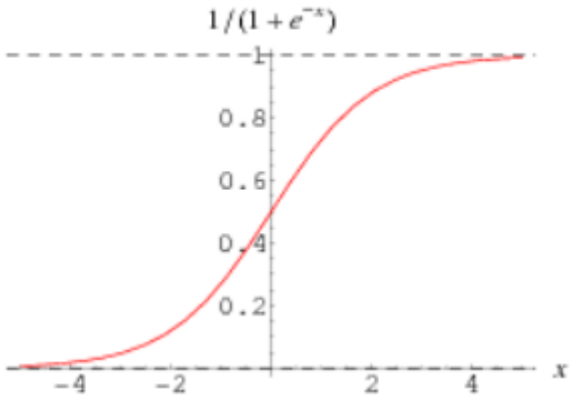


Figure 1 fonction logistique

1) conception de solution

On utilise cette fonction pour calculée la prédiction de chaque individu pour les quatre modèle choisi, si la valeur obtient pour un individu est supérieur à 0.5 donc l'individu appartient à la classe 1 (présence de trouble hépatique) si inférieur à 0.5 c à d appartient à la classe 0 (absence de trouble hépatique).

Donc pour calculer tous les prédictions des individus, on utilise la fonction « **hypotses** » qu'il fait appelle au model et la fonction logistique.

Pour calculer le cout de prédiction, on utilise la fonction « **cost_function** » qu'il fait la somme du vecteur d'hypothèse.

L'objectif de notre étude est de déterminé les meilleures valeurs des thêtas relié à un modèle parmi les modèles choisi tel que la valeur de coût est la valeur la plus minimale.

Pour calculé les thêtas on utilise les fonctions « **gradian_discent** », « **gradian** » et « **normal équation** ».

Fonctions utilises :

- ✓ model_x (input data, thêta) : float

Calculer l'output pour chaque individu

- ✓ sigmoid (input tableau) : float

Calculer la prédiction des valeurs qu'ils sont calculer par le modèle « x »

- ✓ hypotses (input data, thêta, output data) : tableau float

Un tableau de démentions (M*1) contient les valeurs de prédiction calculer par la fonction sigmoid, le « M » est le nombre des individus

- ✓ cost function (input tableau) : float

Sommer la prédiction et diviser sur le nombre des individus (une moyenne)

- ✓ gradian (input data, thêta, output data) tableau float

Calculer la dérivée de la fonction de coût.

- ✓ gradian discent (input data, thêta, output data, alfa, nombre itération) : tableau float

Calculer les thêtas.

- ✓ normal equation (input data, output data) : tableau float

Calculer les thêtas.

- ✓ J erreur (prédiction, output) : float

Calculer le pourcentage d'erreurs de classification.

2) La procédure suivit

Tout d'abord nous avons découpé notre base de données en 3 parties.

- ✓ 60% pour l'apprentissage.
- ✓ 20% pour la validation.
- ✓ 20% pour le test.

Après on initialise les valeurs de thêta, alfa, nombre d'itération et lambda.

(i) Gradient descent

On peut donc faire un appel à la fonction de « gradian_discent » pour calculer les meilleurs paramètres (thêta) pour chaque modèle.

On peut voir le changement de valeur de coût par rapport au nombre des itérations sur les figures suivantes.

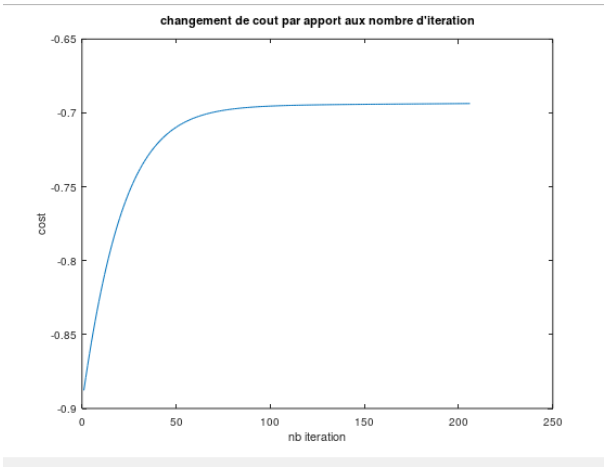


Figure 2 changement de coût de modèle 1

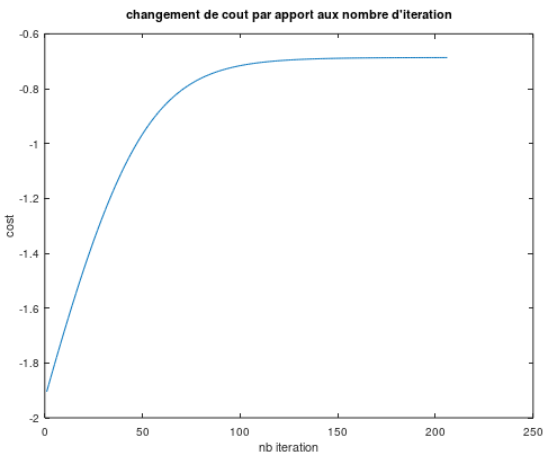


Figure 3 changement de coût de modèle 2

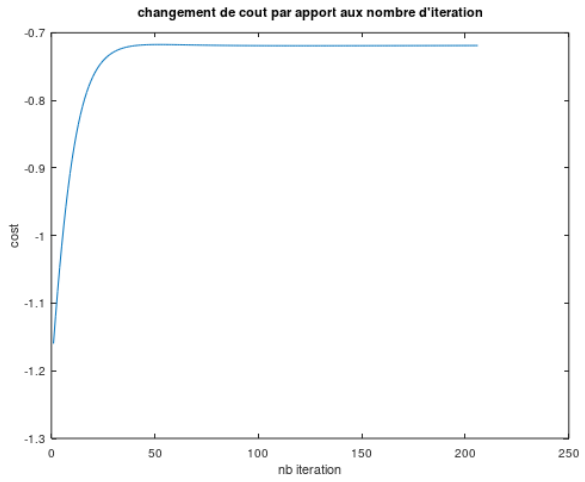


Figure 4 changement de coût de modèle 3

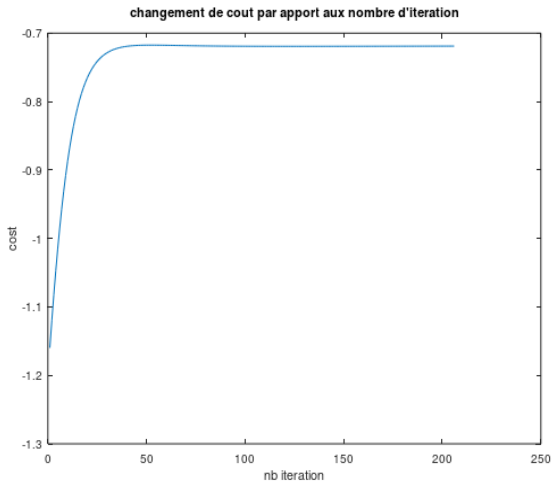


Figure 5 changement de coût de modèle 4

On peut obtenir des valeurs différents de coût pour chaque modèle en changèrent la valeur de lambda et alfa, on résume ça dans les tableaux suivant.

Modèle	Nombre d'itération	lambda	alfa	Le coût
1	100	0.02	0.03	-0.68
1	100	0.1	0.1	-0.9
1	100	1	0.3	-1.5

Figure 6 changements de coût par apport au lambda et alfa

La question qui se pose, quelle est le meilleur modèle qui représente ce problème ?
La réponse est de calculer le taux d'erreur pour le cross validation data set et training data set de chaque modèle on utilise la fonction « **J_erreur** », après on obtient le modèle qui a le taux d'erreur le plus petite de cross validation data set, comme cet exemple.

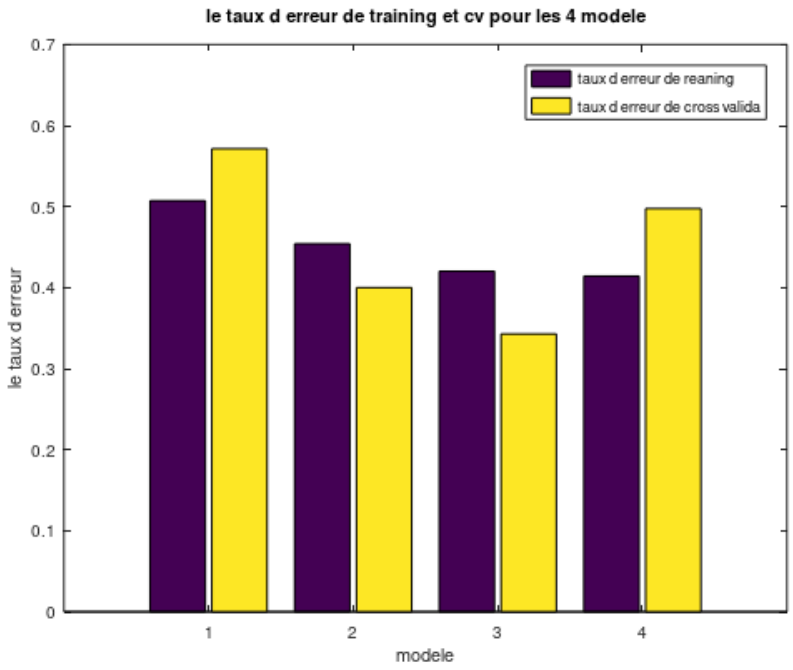


Figure 7 le taux d'erreur de training et cross validation pour 4 modèles

Après la détermination de modèle on peut voir le changement de taux d'erreur pour plusieurs itérations telles que les thêtas sont initialisée pour chaque itération à différent valeurs (utilisation de fonction « randn »). Le résultat est affiché dans la figure ci-dessus.

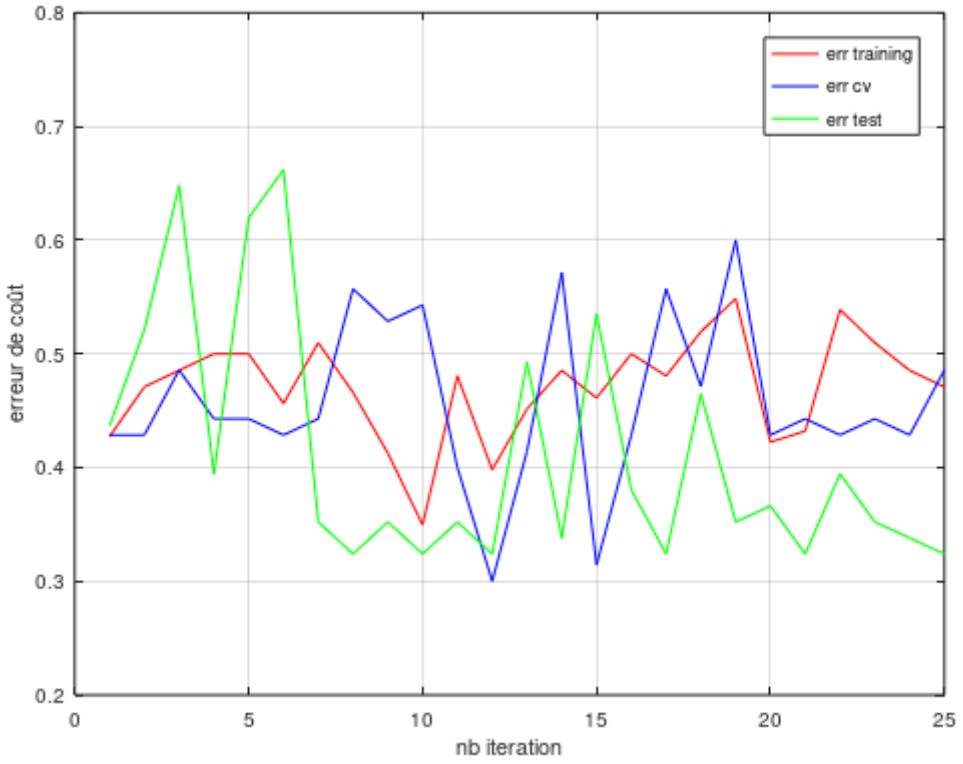


Figure 8 taux d'erreur par apport nombre d'itération

(ii) Equation normale

Une application mathématique qui calcule les valeurs des thêtas directement, on utilise cette équation :

$$\theta = \left(X^T X\right)^{-1} . \left(X^T y\right)$$

Figure 9: équation normal

```
>> main_normal_equation
 ------------------------------------------------------
 ------------------------------------------------------
algoritheme d'aprentissage Equation Noramle
 ------------------------------------------------------
 ------------------------------------------------------
taux d erreur de training data  est
 0.52913
taux d erreur de cross validation data  est
 0.57143
taux d erreur de cross test data  est
 0.57143
 ------------------------------------------------------
>>
```

Figure 10: résultat de l'application de l'équation normal

III) Decision Tree :

1) Définition

L'arbre de décision ou connu sous le nom de decision tree est l'un des algorithmes de l'apprentissage automatique supervise les plus utilisées car il est facile à mettre en place et de plus il est facilement interprétable.

I. Domaine d'application : Il permet de prédire des résultats dans des problèmes de classification et de régression.

II. Principe : Il se base sur un arbre comme modèle prédictif appelé arbre de décision dans lequel les nœuds représentent caractéristiques, les branches représentent les règles et les feuilles représentent les différentes classes ou les résultats qui sont de nature catégorique ou continue.

2) Présentation Problème :

Energy efficiency est un problème qui représente la simulation de plusieurs caractéristiques de bâtiments virtuel afin d'estimer la charge nécessaire du réchauffement et de la climatisation.
En utilisant 8 attributs (caractéristique) et deux sortie Y1 (cout réchauffement) Y2(cout climatisation)

3) Application

Apres vérification de notre dataset on a déterminé que l'application de prétraitement n'est pas nécessaire (absence de valeur null, ni de outliers).
Chargement de données

```
%% loading and dividing the dataset
data = xlsread('myData');
X=data(:,1:8);
y1=data(:,9);
y2=data(:,10);
```

Figure 11: chargement de données

- Transformation au moment de l'exécuter afin de catégoriser nos Y (output) par l'utilisation de la fonction prédéfinie discretize().

```
Y1_categoriezed = discretize(y1, 'categorical', ...
    {'low', 'medium','high'});
Y2_categoriezed = discretize(y2, 'categorical', ...
    {'low', 'medium','high'});
```

Figure 12: application de la fonction discretize()

- Diviser notre dataset en deux ensembles train et test en utilisant la fonction randperm pour avoir des ensembles choisie aléatoirement

```
indx=randperm(768);

X_train=X(indx(1:538),:);
Y1_trainLabels=Y1_categoriezed(indx(1:538),:);
Y2_trainLabels=Y2_categoriezed(indx(1:538),:);
X_test=X(indices(539:end),:);
Y1_testLabels=Y1_categoriezed(indices(539:end),:);
Y2_testLabels=Y2_categoriezed(indices(539:end),:);
```

Figure 13: division de Data Set

- Entrainement et creation de l'arbre en utilisant fitctree

```
Arbre1= fitctree(X_train,Y1_trainLabels);

Arbre2= fitctree(X_train,Y2_trainLabels);
```

Figure 14: entrainement et création de l'arbre

- View nous permet de visualiser notre arbre et predict prend en argument le model et retourne un tableau avec les predictions

```
view(Arbre1,'mode','graph');
view(Arbre2,'mode','graph');
labels_y1=predict(arbre1,X_test);

labels_y2=predict(arbre2,X_test)

accuracy1=sum(labels_y1 == Y1_testLabels )/length(Y1_testLabels)
accuracy2=sum(labels_y2 == Y2_testLabels )/length(Y2_testLabels)
```

Figure 15: visualisation de l'arbre

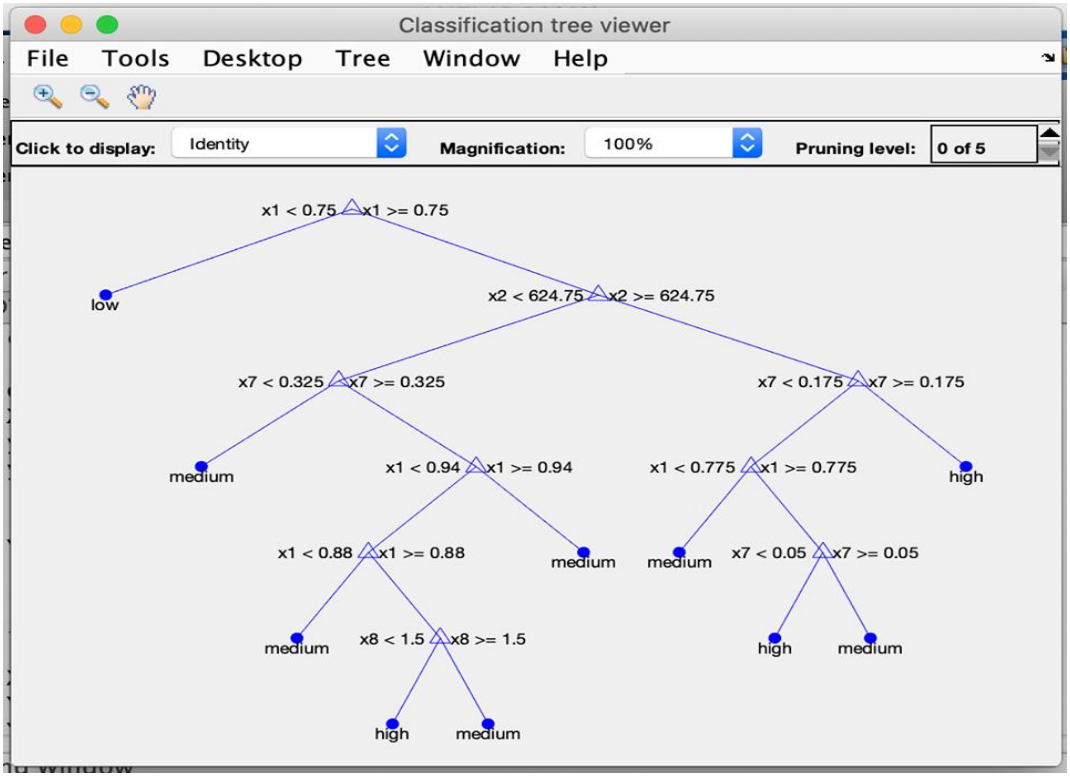


Figure 16: affichage de l'arbre

IV) Support Vector Machine :

1) Définition :

Une machine à vecteur de support (SVM) est un algorithme d'apprentissage automatique qui analyse les données pour la classification et l'analyse de régression. SVM est une méthode d'apprentissage supervisé qui examine les données et les trie dans l'une des deux catégories. Un SVM génère une carte des données triées avec les marges entre les deux aussi éloignées que possible. Les SVM sont utilisés dans la catégorisation des textes, la classification des images, la reconnaissance de l'écriture manuscrite et dans les sciences.

2) Application

- Apres vérification de notre dataset on a déterminé que l'application de prétraitement n'est pas nécessaire (absence de valeur null, ni de outliers).
- Chargement de données

```
%% feature selection

opts = statset('display','iter');
regF1 = @(X_train, Y1_trainLabels, X_test, Y1_testLabels)...
    sum(predict(fitrsvm(X_train, Y1_trainLabels,'KernelFunction','gaussian'), X_test) ~= Y1_testLabels);

[fs, history] = sequentialfs(regF1, X_train, Y1_trainLabels, 'cv', c1, 'options', opts,'nfeatures',5);
```

Figure 17:chargement des donnees

- Sélection des paramètres les plus significatifs

```
%% loading and dividing the dataset
data = xlsread('myData');
X=data(:,1:8);
y1=data(:,9);
y2=data(:,10);
```

Figure 18:découpage de data set

- L'entrainement du model en utilisant fitrsvm

```
X_train_w_best_feature = X_train(:,fs);

Md1 = fitrsvm(X_train_w_best_feature,Y1_trainLabels,'KernelFunction','gaussian','OptimizeHyperparameters','auto',...
      'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName',...
      'expected-improvement-plus','ShowPlots',true));

Md2 = fitrsvm(X_train_w_best_feature,Y2_trainLabels,'KernelFunction','gaussian','OptimizeHyperparameters','auto',...
      'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName',...
      'expected-improvement-plus','ShowPlots',true));
```

Figure 19: utilisation de fitrsvm

- Utilisation de la fonction predict afin de prédire un résultat apartir de l'ensemble test

```
X_test_w_best_feature = X_test(:,fs);
test_accuracy_1 = sum((predict(Md1,X_train_w_best_feature) == Y1_trainLabels))/length(Y1_trainLabels)*100

X_test1_w_best_feature = X_test(:,fs1);
test_accuracy_2 = sum((predict(Md2,X_test1_w_best_feature) == Y2_testLabels))/length(Y2_testLabels)*100
```

Figure 20: utilisation de fonction predict

V) LES RESEAUX DE NEURONES

1) DEFINITION

Les réseaux de neurones sont un domaine très important au sein de l'intelligence artificielle. Inspiré par le comportement du cerveau humain (principalement référé aux neurones et à leurs connexions), son principe est de créer des modèles artificiels pour résoudre des problèmes difficiles.
Chaque Réseau de neurones à une couche d'entrée et une couche de sortie, les couches entre ces deux sont appelées « couches cachées », leur nombre varie par rapport au besoin.

Cette implémentation nous permettra d'avoir un réseau de neurone qui apprend de ses erreurs et qui prédira les valeurs de sorties d'inputs nouvellement entrés.

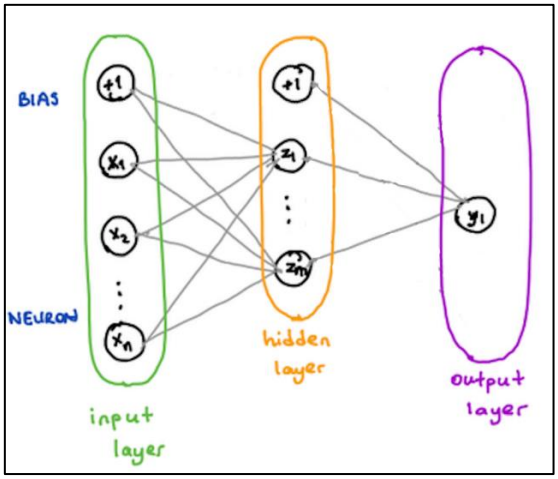


Figure 21 Architecture réseaux de neurones

2) Problème de Classification (Liver Disorders) :

Ce problème de classification est constitué d'une data-set composée de 7 features + le biais et une valeur de sortie.

Afin d'implémenter nos algorithmes de réseaux de neurones nous avons procédé comme suit :

(i) Initialisation des thêtas :

On initialise les thêtas hasardement pour des valeurs comprises dans l'intervalle [-epsilon , epsilon] et on ajoute le biais qui est égal à 1 à l'input comme le montre la figure précédente.

```
#initialiser les theta
theta_one=rand(4,size(input, 2))* 2 * epsilon - epsilon;
theta_two=rand(1,4)* 2 * epsilon - epsilon;
```

Figure 22: Initialisation des thêtas

(ii) Sigmoid :

La fonction Sigmoid est utilisée comme fonction d'activation.

```
%%% Fonction sigmoid

function g=sigmoid(theta,x)
  g=1 ./(1+ e.^-(theta*x));
endfunction
```

Figure 23: La fonction Sigmoid

(iii)Forward Propagation :

Forward Propagation sert à calculer les fonctions d'activation de chaque couche grâce la fonction Sigmoid appliquée à chaque sortie de valeur (n'est pas appliquée dans la dernière sortie dans l'algorithme de régression).

```
%forward propagation

#fonction d'activation a2= theta * instence J (1*4)

a2=sigmoid(theta_one,base(:,j));

#fonction d'activation a3 (1*obeservation)
a3=sigmoid(theta_two,a2);
```

Figure 24 :Forward Propagation (classification)

(iv)Backward Propagation :

Back Propagation sert à calculer le gradient (calcul de dérivée) pour modifier les poids ; c'est à dire calculer le cout d'erreur afin que notre réseau apprenne tout seul.

```
%backpropagation implementation

% erreur de troisieme coche
error_three=a3-output(j);
e=output(j)-a3;


error_two=(theta_two' * error_three) .* a2 .*(1-a2);
theta_grad1= theta_one - (error_two * base(:,j)');
theta_grad2= theta_two - (error_three * a2');

theta_2_reg=theta_two; theta_2_reg(:,1)=0;
theta_1_reg=theta_one; theta_1_reg(:,1)=0;


J += -output(j,:)*log(a3)-(1-output(j,:))*log(1-a3);
 s=J/size(input,2);
```

Figure 25: Backward propagation

(v) Affectation des nouvelles valeurs des thêtas après gradientDescent

Génération de nouveaux thêtas en dehors de la boucle qui génère les observations.

```
% generer un nouveau theta_one
%et theta_two apres calcule du gradient
theta_one=theta_grad1;
theta_two=theta_grad2;
```

Figure 26: Thêtas du gradient

3) RESULTATS :

(i) Modèle de classification :

Résultats de l'exécution du modèle:

On voit dans le graphe que le cout diminue à chaque itération tendant vers 0 ce qui veut dire que c'est un bon apprentissage.

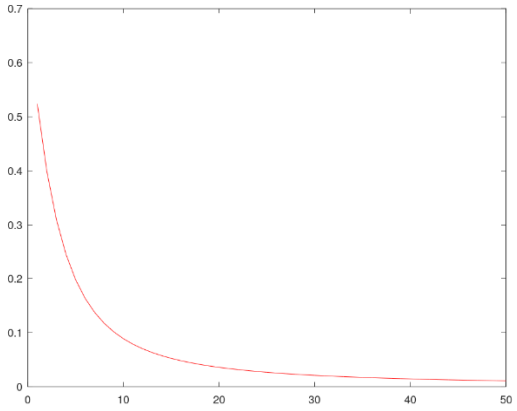


Figure 27: Cost plot (classification)

La meilleure itération avec les meilleurs thêtas :

```
>> nn_classification(cv_in,cv_out,50,0.12)

O1 =
   0.619293   0.010369
   0.619106   0.077048
   0.570533   0.113143
   0.547627   0.078661

O2 =
 -1.8487  -1.7867  -1.7051  -1.6952
```

Figure 28: Valeurs de thêtas optimales

Problème de Régression

ENERGY EFFICIENCY

I) Introduction

Aujourd'hui, l'efficacité énergétique est un terme très utilisé dans le monde. Le domaine du bâtiment est l'un des principaux domaines où l'efficacité énergétique peut être utilisée car au cours de la phase de conception d'un bâtiment, de nombreuses simulations sont effectuées pour estimer l'énergie requise afin de maintenir une température idéale, ces dernières peuvent prendre beaucoup de temps, donc appliquer l'apprentissage automatique est utile pour éliminer le besoin d'effecteur ces simulations avec un gain de temps et d'argent considérable pour les constructeurs et les occupants de l'édifice.

II) Présentation du problème:

Le but de ce problème de régression est de prédire la charge de réchauffement et de refroidissement d'une future construction en utilisant l'ensemble de données de 768 instances comportant 8 attributs

III) Présentation de l'ensemble de données sélectionné

L'ensemble de données étudié est fourni par l'ingénieur en génie-civil Angeliki Xifara et traité par Athanasios Tsanas (centre d'Oxford pour les mathématiques industrielles et appliquées) et a été présenté à UCI machine learning le 30/11/2012.

L'ensemble de données contient 8 attributs avec des valeurs numériques idéales pour un analyse en régression.

1) Information sur l'ensemble de données :

Caractéristiques de l'ensemble de données:	Multivariée	**Nombre d'instances:**	768	**Zone:**	**Ordinateur**
Caractéristiques des attributs:	Entier, réel	**Nombre d'attributs:**	8	**Date de don**	2012-11-30
Tâches associées:	Classification, régression	**Des valeurs manquantes?**	N / A		

2) Les entrées:

- X1 : Compacité relative Réel avec un intervalle fixe de valeurs
  [0.62,0.64,0.66,0.69,0.714,0.74, 0.76,0.79,0.82,0.86,0.90,0.98)]
- X2 : Superficie (Réel)
- X3 : Superficie du mur (Réel)
- X4 : Superficie du toit (Réel)
- X5 : Hauteur totale (Réel)
- X6 : Orientation Codification :
  North=2 ; East=3 ;South=4 ;West=5
- X7 : Superficie du vitrage (Réel)
- X8 : Superficie du vitrage Répartition (Réel)

3) Les sorites:

- Y1 : Charge de chauffage (Réel)
- Y2 : Charge de refroidissement (Réel)

I) La technique Régression linéaire (LR)

1) Définition :

La régression linéaire multivariable est utilisée lorsque nous souhaitons prédire une seule valeur de sortie y à partir de plusieurs valeurs d'entrée x , nous faisons un apprentissage supervisé à l'aide de ces fonctions :

- ✓ Hypothèse
- ✓ Cost function : fonction pour minimiser les erreurs
- ✓ Gradient descent

II) Réalisation :

1) La fonction Hypothèse

La fonction hypothèse tente de nous fournir la meilleure ligne droite qui passe par le maximum de nombre en affectant à chaque paramètre d'entrée un poids appelée thêta, dans notre cas il s'agit d'une régression linaire multiple ou i > 1

$$\sum_{i=1}^{m} \theta_i * X_i$$

Hypothèse de la régression linéaire

Ou X_i sont les attributs de Dataset

➢ Dans notre étude on essaye de choisir thêta à partir des algorithmes Gradient descente ou équations normal pour que h(x) (résultat de notre hypothèse) soit proche de y

2) La fonction de coût

La fonction de cout est la somme des carres des différences(la différence entre les valeurs donnée par le modèle .

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

➢ Nous essayons de minimiser le coût afin de :

• Obtenir la meilleure ligne possible.

• La ligne doit passer par le maximum des points de notre ensemble de données d'entraînement.

3) Calcule de thêta

On peut utiliser deux méthodes :

(i) La fonction Gradient Descent

Cette fonction est une implémentation de l'algorithme Gradient Descent qui permet de calculer les thêta θ_i et de minimiser notre fonction de coût, et en appliquant de manière répétée ces équations de gradient descent notre hypothèse deviendra de plus en plus précise.

La formule pour calculer les θ_i est :

$$\text{repeat until convergence: } \{$$
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0..n}$$
$$\}$$

Figure 29: lq fonction grqdient descent

Où α est un paramètre à choisir appelé le learning rate

(ii) Equation normal :

Cette une fonction qui prend en paramètre les input X et output Y,elle permet de trouver les meilleurs valeurs de thêta
Nous devons ajouter la colonne de 1 à X.

$$\theta = (X^T X)^{-1} X^T Y$$

Figure 30:equation normal

La transposé de la matrice des inputs

(iii) La différence entre Gradient Descent et Équation normal

Gradient Descent	Equation normal
Nécessite de choisir alpha	Ne nécessite pas de choisir alpha
Nécessite beaucoup d'itérations	Il n'est pas itératif
Fonctionne rapidement si N est large	Nécessite de calculer $(X^T X)^{-1}$
	Il va être long si le N est trop large

Vu le nombre d'entités qui est inférieur à 1000, la méthode d'équation normale est meilleure que le gradient discent

4) Étape d'implémentation :

Afin de réaliser l'implémentation nous avons suivi les étapes suivantes :

1. Chargement des données

```
%importer la data set
data= load ('energy1.txt');
```

Figure 31:partie de code qui représente le chargement de données

2. Normalisation et prétraitement des données

Afin d'éviter les anomalies et réduire la complexité dans notre modèle on utilise la technique de normalisation ; cela va servir à simplifier le problème d'apprentissage. Pratiquement on utilise le logiciel EXCEL pour effectuer ces traitements.

3. Diviser notre ensemble de données en 3 parties :

- 70% de la base est consacré pour le training, c'est à dire pour permettre à notre modèle d'apprendre à partir de cet échantillon.

- 15% pour la partie de validation où nous avons comparé le résultat obtenu avec celui obtenu de la partie training et décider si on valide le modèle ou bien on doit changer les paramètres fixés.

- 15% restante pour la partie test où nous allons tester notre modèle avec les nouvelles valeurs restantes de la base.

4. Initialiser les variables nécessaires pour l'exécution de gradient descent :

```
%Enregister la taille de output dans une varriable m
m=length(y_train);

% initilaliser alpha et nombre d'iterations
alpha = 0.5;
iteration = 100000;

% Initialiser le vecteur de theta
theta = rand(9,1);
```

Figure 32: code grqdient descent

5. Les parties du code qui représentent les plots :

- **Le plot de l'hypothèse :**

```
%Ploter l'hypothèse H avec les données
figure('name','hypothese avec gradientDescent');
plot(x_train(:,2)+x_train(:,3)+x_train(:,4)+x_train(:,5)+x_train(:,6)+x_train(:,7)+x_train(:,8), y_train, 'r+', 'MarkerSize', 10);
hold on;
H = hypothese_team9(thetaeq,x_train);
plot (x_train(:,2)+x_train(:,3)+x_train(:,4)+x_train(:,5)+x_train(:,6)+x_train(:,7)+x_train(:,8), H, '-');
legend('data','Hypothèse');
% L'axe des Y
ylabel('OUTPUT');
% L'axe des X
xlabel('INPUT');
hold off;
```

Figure 33: code plot de l'hypothèse

- **Le plot de cout :**

```
function [theta,v]=gradientDescent_team9(X,y,theta,alfa,nb_iteration)

 v=[];
  for i=1:nb_iteration
    g= gradian(theta,X,y);
    theta= theta-((alfa/length(y))*g);
    v=[v;costfunction_team9(theta,X,y)];
  end

  figure(1);
  plot (v, '-b', 'LineWidth', 2);
  % L'axe des X
  xlabel('It?rations');
  % L'axe des Y
  ylabel('Cost J');
  end
```

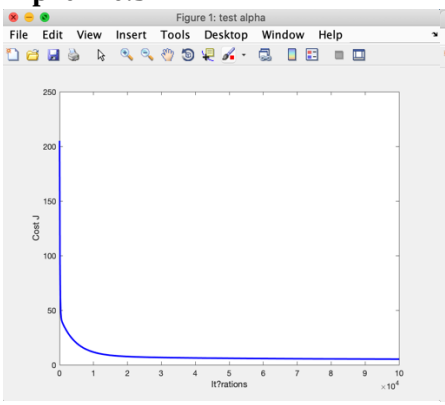
Figure 34:code plot du cout

On a ploter les valeurs de coût à partir de la fonction gradient descent :

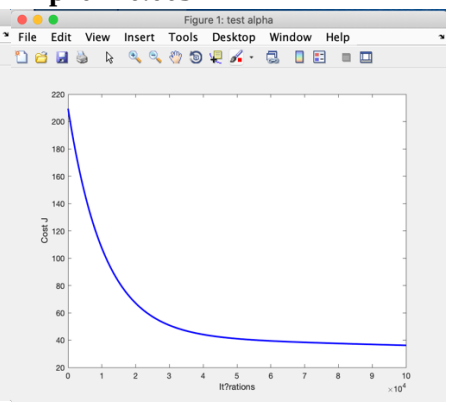
✓ **Le plot de cost avec différentes valeurs d'alpha :**

On a essayé de lancer une descente de gradient avec différentes valeurs de alpha et de voir son influence sur les valeurs de couts.

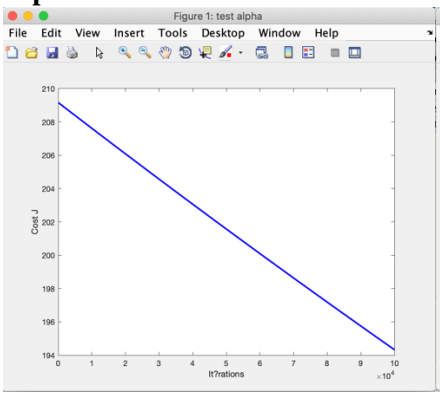
Alpha = 0.5



Alpha = 0.005



Alpha = 0.00005



Alpha= 500

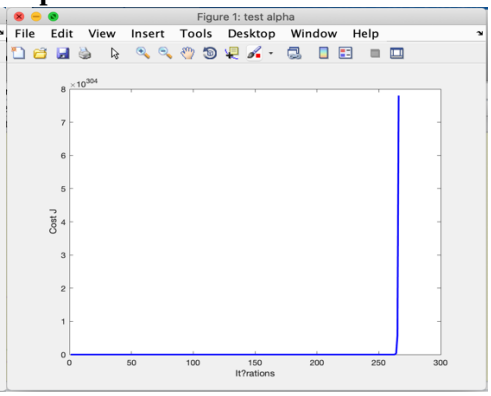


Figure 35:plot de cout avec différentes valeurs d'alpha

6. Prédiction de valeurs restantes (test) de notre ensemble de données :

```
%% partie 2 predition des valeurs restantes
x_rest = data (653:768,1:8);
y_rest = data (653:768,9);
x_rest = [ones(size(x_rest,1), 1), x_rest(:,:)];

%predicter les y
y_predict=x_rest * thetaeq;

%plot predict et reel dans le mm graphe
plotprediction(y_rest,y_predict);
diff = abs(y_rest-y_predict);
precision=mean(diff);
fprintf('la precision using normal equation  est  %f \n',precision);

y_predictgd=x_rest * theta;
diff = abs(y_rest-y_predictgd);
precisiongd=mean(diff);
fprintf('la precision using gradient descent  est %f \n',precisiongd);
```

Figure 36 Code de prédiction valeurs

7. Afficher le résultat prédit et valeur réelle sur le même graphe à l'aide de plot

```
function plotprediction(y_rest_reel,y_rest_predict)
  figure ('name','Valeurs output');
  plot(y_rest_reel,'Displayname','Y_predict'); % plot les y reel par des ligne rouge
  title('prediction des valeurs');
  hold on;
  plot(y_rest_predict,'Displayname','Y_reel'); % plot les y predit par des ligne bleu
  legend('Y reel','Y predict');
  hold off;
  end
```

Figure 37 code de plot de prédiction des valeurs

- **Le plot de prédiction des valeurs :**

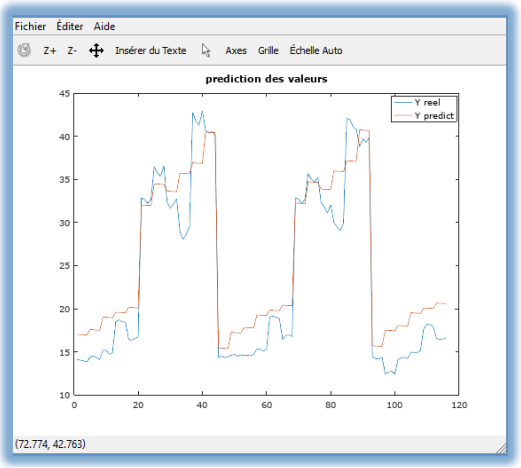


Figure 38 La propagation des données. prédits et reel

8. Tester un echantillon :

Afin de comparer entre les deux algorithme Gradient descent est normal eqation nous avons effectuer un test sur un seul echantillon

```
y =

   1.00000   1.00000   0.64000   0.71000   0.50000   1.00000   1.00000   0.00000   0.00000

y_predicteq =  21.518
y_predictgd =  24.859
```

Figure 39 Teste d'un echantillon

-le tableau ci dessous reprensente les valeurs predits et reel de Y

Y réel 21.55	
Y prédit avec gradient descent est 21.518	Y prédit avec équation normal est 24.859

➢ D'après le résultat obtenu de la presicion on constate que eqationnormal est plus optimal et meilleure que Gradient descenet

9. **Predir la partie train et test**

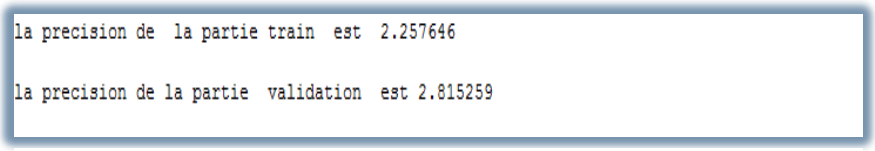


Figure 40 Precision des deux partie (train et validation)

L'interprétation de ces valeurs : veut dire que par exemple si la charge de : Charge de refroidissement (Réel) est a 21 en réalité donc notre modèle va indiquer qu'il va avoir 24.

10. **Au final** comme nous avons dans notre problème deux valeur de Y a prédire Y1 et Y2 nous avons donc du créé deux fonction main que nous avons appelé main_team9 pour prédire le Y1 Charge de chauffage(Réel) et main_Y2_team5 pour prédire le Y2 Charge de refroidissement(Réel)

III) LES RESEAUX DE NEURONES

1) Problème de Régression (Energy Efficieny) :

Ce problème de régression est constitué d'une data-set composée de 8 features + le biais et 2 valeurs de sortie qu'on a décidé de limiter à une valeur Y1 c'est-à-dire « la charge de chaleur ».

(i) Initialisation des thêtas :

On initialise les thêtas hasardement pour des valeurs comprises dans l'intervalle [-epsilon , epsilon] et on ajoute le biais qui est égal à 1 à l'input comme le montre la figure précédente.

```
#initialiser les theta
theta_one=rand(4,size(input, 2))* 2 * epsilon - epsilon;
theta_two=rand(1,4)* 2 * epsilon - epsilon;
```

Figure 41: Initialisation des thêtas

(ii) Sigmoid :

La fonction Sigmoid est utilisée comme fonction d'activation.

```
%%% Fonction sigmoid

function g=sigmoid(theta,x)
  g=1 ./(1+ e.^-(theta*x));
endfunction
```

Figure 42: La fonction Sigmoid

(iii)Forward Propagation :

Forward Propagation sert à calculer les fonctions d'activation de chaque couche grâce la fonction Sigmoid appliquée à chaque sortie de valeur (n'est pas appliquée dans la dernière sortie dans l'algorithme de régression).

```
%forward propagation

 #fonction d'activation a2= theta * instence J (1*4)
 a2=sigmoid(theta_one,base(:,j));

 #fonction d'activation a3 (1*1)
 a3=theta_two*a2;
```

Figure 43:forwor probagation

(iv)Backward Propagation :

Back Propagation sert à calculer le gradient (calcul de dérivée) pour modifier les poids ; c'est à dire calculer le cout d'erreur afin que notre réseau apprenne tout seul.

```
%backpropagation implementation

% erreur de troisieme coche
error_three=a3-output(j);
e=output(j)-a3;


error_two=(theta_two' * error_three) .* a2 .*(1-a2);
theta_grad1= theta_one - (error_two * base(:,j)');
theta_grad2= theta_two - (error_three * a2');

theta_2_reg=theta_two; theta_2_reg(:,1)=0;
theta_1_reg=theta_one; theta_1_reg(:,1)=0;


J += -output(j,:)*log(a3)-(1-output(j,:))*log(1-a3);
 s=J/size(input,2);
```

Figure 44: Backward propagation

(v) Affectation des nouvelles valeurs des thêtas après gradientDescent

Génération de nouveaux thêtas en dehors de la boucle qui génère les observations.

```
% generer un nouveau theta_one
%et theta_two apres calcule du gradient
theta_one=theta_grad1;
theta_two=theta_grad2;
```

Figure 45: Thêtas du gradient

2) RESULTATS :

(i) Modèle de régression :

Résultats de l'exécution du modèle:

On remarque dans le graphe ci-dessous que le coût diminue à chaque itération (5 itérations) tendant vers 0 qui signifie que c'est un bon apprentissage.

Dans certains cas (variation de nombre d'itérations), le graphe atteint 0 car nous n'avons par régulariser la base de données.

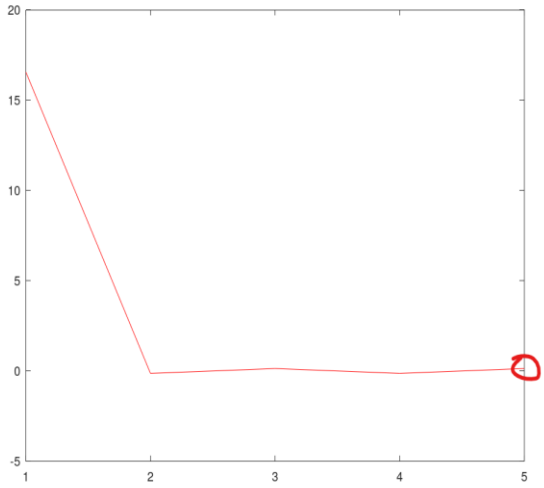


Figure 46: Cost plot (régression)

La meilleure itération avec les meilleurs thêtas :

On remarque que y_train est proche de la valeur réelle y, ce qui veut dire que le réseau a appris et à prédire de vraies valeurs de y.

```
O1 =
Columns 1 through 9:

 -0.09288283   0.01577604  -0.00018868   1.43308460   0.66666893   0.32042088  -0.04648038  -0.10822493   0.10209516
  0.06988026  -0.03957087  -0.02772235   0.11261513   0.10686522   0.02861135  -0.01860022  -0.05206311  -0.06857161
 -0.08290470   0.08405899   0.02172021  -0.10033849  -0.00614561   0.03310468  -0.06023771  -0.04569389  -0.04546824
  0.09309550  -0.05923414  -0.04526116   0.02547933  -0.07201980  -0.04220198   0.04060913  -0.03319684   0.00428233

Column 10:

 -0.05712099
  0.07879842
 -0.05753200
  0.05865397

O2 =
    0.133433   16.639893   -0.022187    0.112849

y_train =
16.507
```

y=16.648 ≈ y_train=16.507

Figure 47: Valeurs de thêtas optimales

Régression logistique

```
function model= model_a(input,theta)
  model=[];
  for i=1 : length(input),
    # m= t1 + x1*t2 + x2*t3 + x3*t4 + x4*t5 + x5*t6 + x6*t7
    model=[model;theta(1)+ theta(2)*input(i,1) + theta(3)*input(i,2) + theta(4)*input(i,3) +
    theta(5)*input(i,4) + theta(6)*input(i,5) + theta(7)*input(i,6)];
  endfor
endfunction
```

```
function y=sigmoid(x)
  y=[];
  for i=1 : length(x),
    y= [y; 1/(1 + exp(x(i)*(-1)))];
  endfor
endfunction
```

```
function h=hypotses(input,theta)
  h=[];
  #  m ---> theta'*input
  m=model_a(input,theta);
  h=sigmoid(m);

endfunction
```

```
function gradian=gradian(input,theta,output)
  h=hypotses(input,theta);
  #ajouter le bais = 1
  input= [ones(length(input),1) input];
  # h-output (m*1)   input'   (7*m)    j (7,1)
  gradian=1/length(output)*(input'*(h-output));
endfunction
```

```
function  j= cost_function(input ,theta,output,lambda)
  cost=0;
  h=hypotses(input,theta);
  for i=1 :length(output),
    # la somme de cout pour chaque linge
     cost=cost+((-output(i)*log(h(i)))-((1-output(i))*log(1-h(i))));
  endfor
  # j(theta)= 1/M * somme (cout);
  j= ((-1/length(output))*cost)+ (lambda/(2*length(output))*sum(theta(2:end).^2));
endfunction
```

```
function theta=gradian_discent(input,theta,output,alfa,nb_it,nb_f,lambda)
  j=[];
  for i=1:length(output),
    # theta (7,1)   gradian   (7,1)
    theta = theta - alfa*gradian(input,theta,output);

    j= [j;cost_function(input,theta,output,lambda)];
  endfor

  figure(nb_f);
  plot(j);
  title("changement de cout de modele 3 par apport aux nombre d'iteration");
  xlabel('nb iteration');
  ylabel('cost');
endfunction
```

```
function sum_err=J_err(input,theta,output)
  h=hypotses(input,theta);
  err=[];

  for i=1 : length(h)
    if((h(i) >= 0.5)&&(output(i)==0))||((h(i) < 0.5)&&(output(i)==1)),
      err=[err,1];
    else
      err=[err,0];
    end
  endfor
  sum_err=(1/length(output))*sum(err);
endfunction
```

```
function data=normalisation(data)

  for i=1 :size(data,2),
    maxl=max(data(:,i));
    minl=min(data(:,i));
    average=(maxl-minl)/2;
    for j=1 :size(data,1),
      data(j,i)=(data(j,i)-average)/(maxl-minl);
    endfor

  endfor
```

```
function theta= normal_equation(input,output,lambda)
  l=size(input,2)+1;
  X=input;
  id = eye(l);
  id(1,1)=0;
  f=lambda*id;
  X=[ones(length(input),1) input];
  Y=output;
  #pinv(X) ----> inv(X'*X)*X
  theta= pinv(X'*X+f)*(X'*output);
endfunction
```

LES RESEAUX DE NEURONE

```
  % charger la base de donnee
load inputdata.txt;
load outdata.txt;
%epsilon pour gerer thetas au debut

epsilon=0.12
%load 'besttheta1.txt'
%load 'besttheta2.txt'
% input data

input=input;
%output data
output=out;
% nombre d'iteration
iter=6;
% vecteur pour stocker les le cout
train=[];
pred=[];
# ajouter la biaiss
input=[ones(rows(input),1),input];
#pred_in=[ones(rows(pred_in),1),pred_out];
J=0;
s=0;
s2=0;
e=0;
g=e;
theta_2_reg=[];
theta_1_reg=[];
theta_grad1=[];
theta_grad2=[];



#initialisation des theta
theta_one=rand(4,size(input, 2))* 2 * epsilon - epsilon;
theta_two=rand(1,4)* 2 * epsilon - epsilon;

#inverser la base
base=input';

#
for i= 1:iter
   for j=1:size(base,2)


 %forward propagation

   #fonction d'activation a2= theta * instence J (1*4)
   a2=sigmoid(theta_one,base(:,j));

   #fonction d'activation a3 (1*1)
   a3=theta_two*a2;
   #â—‹ a3=sigmoid(theta_two,a2);


     %backpropagation implementation

   % erreur de troisieme coche
   error_three=a3-output(j);
   e=output(j)-a3;

   % erreur de deuxieme coche

   error_two=(theta_two' * error_three) .* a2 .*(1-a2);
   theta_grad1= theta_one - (error_two * base(:,j)');
   theta_grad2= theta_two - (error_three * a2');

   %substract gradient for calcule the derivate
   theta_2_reg=theta_two; theta_2_reg(:,1)=0;
   theta_1_reg=theta_one; theta_1_reg(:,1)=0;


 J += -output(j,:)*log(a3)-(1-output(j,:))*log(1-a3);
   s=J/size(input,2);


     #vecteur pour stocker les valeur de (y- hypothese) pour voir le taux d'ereur


  endfor
  # regularisation avec le cout et gradient descent
    # first column is empty
    %theta_2_reg=theta_two; theta_2_reg(:,1)=0;
    #theta_1_reg=theta_one; theta_1_reg(:,1)=0;
   # J=J/m+lambda/(2*m) * (sum(theta_1_reg.^2)(:))+sum(theta_2_reg.^2)(:)));

  %substract  %gradient
    %theta_grad1= theta_one - (error_two * base(:,j)');
    %theta_grad2= theta_two - (error_three * a2');

  theta_one=theta_grad1;
  theta_two=theta_grad2;

  #pred=[pred,nn_predict(input,output,theta_one,theta_two)];
  train=[train,e];


endfor
%pred=nn_predict(input,output,besttheta1,besttheta1);
plot(train,"r-");
%hold on;
%plot(pred,"b-");
%hold off;
printf('\n O1 =  \n');
disp(theta_one)
printf('\n O2 =  \n');
disp(theta_two)
printf('\n y_train =  \n');
% afficher la derniere valeur pour verifier
  disp(a3);
```

```
% la fonction ci dessous est pour tester notre modele avec les meilleures thetas (n'est pas appelée)
function pred= nn_predict(input,output,theta_g1,theta_g2)

    base=input';
    pred=[];
  for j=1:size(base,2)
  a2=sigmoid(theta_g1,base(:,j));
    a3=sigmoid(theta_g2,a2);

  endfor
  pred=[pred,a3];
endfunction
```

```
%%% Fonction sigmoid

function g=sigmoid(theta,x)
  g=1 ./(1+ e.^-(theta*x));
endfunction
```

Regression lineaire :

- Main

```
%importer la data set
data= load ('energy1.txt');
%division des instances
x_train = data(1:537,1:8);
y_train = data (1:537,9);

x_test_1 =data(538:653,1:8);
y_test_1 = data (538:653,9);

x_validation = data(653:768,1:8);
y_validation = data (653:768,9);
%Enregister la taille de output dans une varriable m
m=length(y_train);

%Ajouter une colonne X0 itialise a 1 dans le vecteur X(l'input)
x_train = [ones(m, 1) x_train];

% initilaliser alpha et nombre d'iterations
alpha = 0.5;
iteration = 100000;

% Initialiser le vecteur de theta
theta = rand(9,1);

% Appel de la fonction gradientDescent
v =[];
[theta,v]= gradientDescent_team9(x_train, y_train, theta, alpha, iteration);
fprintf('les theta de gradient descent sont \n');
disp(theta);


%afficher la valeur la valeur de cout
fprintf ('la valeur de cout  %f \n',v(99990) ) ;

%afficher les theta de lequation normal
lambda = 0 ;

thetaeq=equation_normal_team9(x_train ,y_train ,lambda);
fprintf('les theta de lequation normal sont \n');
```

```
%Ploter l'hypothese H avec les donn?es
figure('name','hypothese avec gradientDescent');
plot(x_train(:,2)+x_train(:,3)+x_train(:,4)+x_train(:,5)+x_train(:,6)+x_train(:,7)+x_train(:,8), y_train, 'r+', 'MarkerSize', 10);
hold on;
H = hypothese_team9(thetaeq,x_train);
plot (x_train(:,2)+x_train(:,3)+x_train(:,4)+x_train(:,5)+x_train(:,6)+x_train(:,7)+x_train(:,8), H, '-');
legend('data','Hypoth?se');
% L'axe des Y
ylabel('OUTPUT');
% L'axe des X
xlabel('INPUT');
hold off;

%% partie 2 predition des valeurs restantes
x_rest = data (653:768,1:8);
y_rest = data (653:768,9);
x_rest = [ones(size(x_rest,1), 1), x_rest(:,:)];
%predicter les y
y_predict=x_rest * thetaeq;
%plot predict et reel dans le mm graphe
plotprediction(y_rest,y_predict);
diff = abs(y_rest-y_predict);
precision=mean(diff);
fprintf('la precision using normal equation  est  %f \n',precision);

y_predictgd=x_rest * theta;
diff = abs(y_rest-y_predictgd);
precisiongd=mean(diff);
fprintf('la precision using gradient descent  est %f \n',precisiongd);
%prediction
x_t = [ones(size(x_train,1), 1), x_train(:,:)];
x_v = [ones(size(x_validation,1), 1), x_validation(:,:)];
predictionTrain=hypothese_team9(theta,x_train);
predictionvld=hypothese_team9(theta,x_v);
%calcule precision
precision=mean(abs(predictionTrain-y_train));
precisionvld=mean(abs(predictionvld-y_validation));
fprintf('la precision de  la partie train  est  %f \n\n',precision);
fprintf('la precision de la partie  validation  est %f \n\n',precisionvld);
```

- Hypothese

```
function h=hypothese_team9(theta,X)
  h=X*theta;
end
```

- Gradien descent

```
function [theta,v]=gradientDescent_team9(X,y,theta,alfa,nb_iteration)

v=[];
 for i=1:nb_iteration
   g= gradian(theta,X,y);
   theta= theta-((alfa/length(y))*g);
   v=[v;costfunction_team9(theta,X,y)];
 end
%plot cost function
 figure('name','test alpha');
 plot (v, '-b', 'LineWidth', 2);
 % L'axe des X
 xlabel('It?rations');
 % L'axe des Y
 ylabel('Cost J');
 end
```

```
function g=gradian(theta,X,y)
  m=size(y,1);
  h= hypothese_team9(theta,X);
  dj=h-y ;   % pour matirce dj(m*1)
  dj=X'*dj ;          %pour matirce X'  (2*m)
  g=(1/m)*dj ;   %donc g =  (2*1)
end
```

- Cost function

```
function j=costfunction_team9(theta,X,y)
 %m=length(y);

sub=(hypothese_team9(theta,X)-y).^2;
 sub=sum(sub);
 const=1/(size(y,1)*2);
 j=const*sub;

end
```

- Normal equation

```
function theta= equation_normal_team9(input,output,lambda)
  l=size(input,2);
  X=input;
  id = eye(l);
  id(1,1)=0;
  f=lambda*id;
  %X=[ones(length(input),1) input];
  Y=output;
  %pinv(X) ----> inv(X'*X)*X
  theta= pinv(X'*X+f)*(X'*output);
end
```

SVM

```
clear; close all; clc;

%% loading and dividing the dataset
data = xlsread('ENB2012_data');
X=data(:,1:8);
y1=data(:,9);
y2=data(:,10);
indx=randperm(768);
X_train=X(indx(1:538),:);
Y1_trainLabels=y1(indx(1:538),:);
Y2_trainLabels=y2(indx(1:538),:);
X_test=X(indx(539:end),:);
Y1_testLabels=y1(indx(539:end),:);
Y2_testLabels=y2(indx(539:end),:);

%% CV partition
c1 = cvpartition(Y1_trainLabels,'KFold','Stratify',true);
c2 = cvpartition(Y2_trainLabels,'KFold','Stratify',true);

%% feature selection
opts = statset('display','iter');
regF1 = @(X_train, Y1_trainLabels, X_test, Y1_testLabels)...
    sum(predict(fitrsvm(X_train, Y1_trainLabels,'KernelFunction','gaussian'), X_test) ~= Y1_testLabels);

[fs, history] = sequentialfs(regF1, X_train, Y1_trainLabels, 'cv', c1, 'options', opts,'nfeatures',4);
```

```
%% Best hyperparameter
X_train_w_best_feature = X_train(:,fs);
Md1 = fitrsvm(X_train_w_best_feature,Y1_trainLabels,'KernelFunction','gaussian','OptimizeHyperparameters','auto',...
      'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName',...
      'expected-improvement-plus','ShowPlots',true));

regF2 = @(X_train, Y2_trainLabels, X_test, Y2_testLabels)...
    sum(predict(fitrsvm(X_train, Y2_trainLabels,'KernelFunction','gaussian'), X_test) ~= Y2_testLabels);

[fs1, history] = sequentialfs(regF2, X_train, Y2_trainLabels, 'cv', c2, 'options', opts,'nfeatures',4);

X_train_w_best_feature = X_train(:,fs1);

Md2 = fitrsvm(X_train_w_best_feature,Y2_trainLabels,'KernelFunction','gaussian','OptimizeHyperparameters','auto',...
      'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName',...
      'expected-improvement-plus','ShowPlots',true));
```

```
%% Final test with test set
X_test_w_best_feature = X_test(:,fs);
test_accuracy_1 = sum((predict(Md1,X_test_w_best_feature) == Y1_testLabels))/length(Y1_testLabels)*100

X_test1_w_best_feature = X_test(:,fs1);
test_accuracy_2 = sum((predict(Md2,X_test1_w_best_feature) == Y2_testLabels))/length(Y2_testLabels)*100
```

Decision Tree

```
clear ; close all; clc
data = xlsread('ENB2012_data.xlsx');
X=data(:,1:8);
y1=data(:,9);
y2=data(:,10);
 Y1_categoriezed = discretize(y1, 'categorical', ...
    {'low', 'medium','high'});
Y2_categoriezed = discretize(y2, 'categorical', ...
    {'low', 'medium','high'});
 %%sequentialfs(function,X_train,Y_train,'cv',cvObject)
indx=randperm(768);

X_train=X(indx(1:538),:);
Y1_trainLabels=Y1_categoriezed(indx(1:538),:);
Y2_trainLabels=Y2_categoriezed(indx(1:538),:);
X_test=X(indices(539:end),:);
Y1_testLabels=Y1_categoriezed(indices(539:end),:);
Y2_testLabels=Y2_categoriezed(indices(539:end),:);
Arbre1= fitctree(X_train,Y1_trainLabels);

Arbre2= fitctree(X_train,Y2_trainLabels);
view(Arbre1,'mode','graph');
view(Arbre2,'mode','graph');

   c1 = cvpartition(Y1_testLabels,'k',5);
   c2 = cvpartition(Y2_testLabels,'k',5);
labels_y1=predict(arbre1,X_test);

labels_y2=predict(arbre2,X_test)

accuracy1=sum(labels_y1 == Y1_testLabels )/length(Y1_testLabels)
accuracy2=sum(labels_y2 == Y2_testLabels )/length(Y2_testLabels)
```

Références

https://quora.com/what-is-cost-funciton-in-linera-regression

https://dataanalyticspost.com/Lexique/normalisation/

https://avellano.fis.usal.es/lalonso/RNA/index.htm

http://georges.gardarin.free.fr/Surveys_DM/Survey_SVM.pdf