Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIV ALGER 1

RAPPORT

PROJET

Apprentissage Automatique

Partie 2

Objectif : Application des techniques de régression linéaire, de classification et de prétraitement des données

Elaboré par Team 15 : Akkouche Abderrahmane

Krim Islem

Masdoua Manil

Sassi Kahina

1ère Année Master – ISII

SOMMAIRE

Régression : Online Video Characteristics and Transcoding Time Dataset Data Set

Introduction:

Après avoir présenté en détail notre sujet (Vidéo Transcoding Time) ainsi que sa base de données et réglé les anomalies qui y était, nous allons dans cette partie s'attaquer à l'implémentation de l'apprentissage automatique qui va être appliqué sur notre sujet.
Bien que le type de notre problème soit un problème de *regression*, nous allons implémenter dans cette partie le « linear regression ».

Prétraitement :

Le prétraitement réalisé dans la partie A de notre projet nous a servi à éliminer et traiter plusieurs anomalies et données qui étaient manquantes, incohérentes, excessives (noisy data).

Dans cette partie nous allons traiter les attributs de notre base de données de type chaine de caractère ou bien String. Les attributs qui sont concernés sont : **codec** et **o_codec**.
Pour bien comprendre notre méthode utilisée, une clarification sur le container vidéo ou bien le codec est quasi nécessaire.
« Le codec est un logiciel qui compresse une vidéo pour la stocker et la lire. **»** -Wikipédia
Ce codec sert à compresser et décompresser une vidéo sans toucher à sa qualité, des fois des petites différences inaperçues pour l'œil humain apparaissent lors de la compression ou décompression.

Quant à notre base de données, nous distinguons 4 types de codec qui sont :

Remarque : chaque type de codec diffère de l'un à l'autre, ici nous parlons du temps nécessaire pour la compression d'une vidéo qui affectera notre temps de transcodage.

La méthode la plus précise pour trouver un encodage qui a un sens pour ces chaines de caractères est de remplacer le codec qui prend un temps de compression élevé par le poids le plus fort, ce processus va être appliqué aux autres codecs jusqu'à ce que nous arrivons à celui qui prendra le moins de temps et qui aura le poids le plus faible.

Voici un ordre décroissant des codecs par rapport au temps nécessaire de compression (du codec le plus lourd au plus fin) :

1-mpeg4.
2-flv.
3-vp8.
4-h264.

Nous avons donc attribué pour chaque codec respectant l'ordre un poids significatif :
mpeg4 = > poids = '3'.
flv = > poids = '2'.
vp8 = > poids = '1.5'.
h264 = > poids = '1'.
Avec une fonction simple sur Excel nous avons pu remplacer chaque codec avec son poids ou valeur significative

Regression Linear

Nous allons utiliser linear régression (regression linéaire) pour entrainer l'ensemble de notre dataset, afin d'avoir une fonction linéaire qui pourra être utilisée par la suite pour prévoir le temps de transcodage facilement et précisément dans le futur.
Mathématiquement, notre but est de déterminer une fonction ***f*** qui étant donné un nouveau **X**, elle prédise parfaitement **Y.** La fonction $f(x)$ doit être donc polynomiale de degré 1 (puisqu'elle est linéaire) avec un nombre de variables qui sera dépendant de combien d'entrées nous disposons dans notre base de données.
Nous disposons dans notre dataset un ensemble de 19 variables en entré et une seule en sortie ainsi que plus de 65000 instances, les attributs étaient bien présentés dans la partie précédente

mais une autre visualisation de ces dernières est obligatoire pour justifier nos choix ultérieurement.

40000x20 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	12	320	240
2	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	12	480	360
3	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	12	640	480
4	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	12	1280	720
5	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	12	1920	1080
6	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	15	176	144
7	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	15	320	240
8	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	15	480	360
9	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	15	640	480
10	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	15	1280	720
11	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	15	1920	1080
12	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	24	176	144
13	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	24	320	240
14	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	24	480	360
15	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	24	640	480
16	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	24	1280	720
17	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	24	1920	1080
18	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	25	176	144
19	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	25	320	240
20	1	130.3567	3	176	144	54590	12	27	1537	0	1564	64483	825054	0	889537	3	56000	25	480	360

Un échantillon du dataset

II.1 Feature scalling :

Le feature scalling consiste à réduire le champ des valeurs de chaque colonne (feature) vers [-1,1], pour établir cette normalisation et pour chaque colonne (feature) on doit d'abord calculer la somme de toute la colonne et puis pour chaque valeur d'unecolonne.
Nous prenons exemple ici la colonne numéro 1 et nous appliquons l'algorithme suivant :

Début

somme1 = **mean** (colonne 1)
MaxVal = **Max** (colonne 1)
MinVal = **Min** (colonne 1)
pour chaque valeur de la colonne faire :
valeur = valeur – somme /MaxVal-MinVal
Fin.

Fin.

L'implémentation exacte de cette normalisation avec le langage Matlab se trouve dans l'annexe. Voici deux figures comparatives qui présenteront l'état d'une colonne avant et après la normalisation :

Après		Avant
640	➔	1.7087
1280		1.9710
1920		2.2333
176		3.2826
320		4.3318
480		1.4727
640		1.7087
1280		1.9710
1920		2.2333
176		3.2826
320		4.3318
480		1.4727
640		1.7087
1280		1.9710
1920		2.2333

Cet algorithme est tiré du **site :** Coursera
formation : machine Learning
Auteur: Andrew Ng

Remarque : Notre fonction de normalisation est personnalisée c'est dire elle contient des changements par rapport à la fonction originale

---

I.2 Présentation de l'hypothèse :
Après avoir normalisé toute notre base de données, nous allons maintenant définir une hypothèse afin de pouvoir obtenir un polynôme de degré 1.
Hypothèse :

$$h(x) = \theta 0 \times X0 + \theta 1 \times X1 + \theta 2 \times X2 + \theta 3 \times X3 + \cdots +$$

Cette formule est tirée du cours de Machine Learning de Mme Boutorh 2018/2019

tel que :
X1, X2, X3 … Xn : représentent nos feature d'entrées.
$\theta 0, \theta 1, \theta 2, \theta 3 \ldots \theta n$: représentent les paramètres de notre hypothèse
n : représente le nombre de feature dans notre base de données.
Notre but principal est de trouver ces paramètres $\theta 0, \theta 1, \theta 2, \theta 3 \ldots \theta n$.
Il existe plusieurs méthodes pour le réaliser, parmi ces méthodes nous allons utiliser le **Gradient Decent.**
Nous pouvons représenter l'hypothèse h également par une forme algébrique c'est-à-dire une vectorisation, tels que nos paramètres « $\theta 0, \theta 1, \theta 2, \theta 3 \ldots \theta n$.» vont devenir un vecteur de taille n * 1 et les feature « X1, X2, X3 … Xn» vont devenir une matrices de taille nbr d'instances * n et notre résultat c'est-à-dire le output sera la multiplication des deux matrices
L'implémentation exacte et détaillé avec le langage Matlab sera présentée dans l'annexe.

I.3 Cost Function :
Cost Function nous permet de déterminer le taux d'erreur dans notre hypothèse, elle est calculée par la formule suivante :

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Cette formule est tirée du cours de Machine Learning de Mme Boutorh 2018/2019

tels que :
m : le nombre d'instances ou features.
y : les sorties de notre dataset.
Pour notre cas il faut calculer J (θ0, θ1, θ2, θ3… θn.).
Notre but s'inscrit dans la minimisation du taux d'erreur, donc les paramètres θ0, θ1, θ2, θ3… θn. qui satisferont J (θ0, θ1, θ2, θ3… θn.) = 0 sont les meilleures paramètres et par la suite ce seront ceux qui seront utilisée dans la partie test.

I.4 Gradient Decent :

Le gradient Decent et étant donné l'une des solutions qui va nous permettre de déterminer nos paramètres θ0, θ1, θ2, θ3… θn. consiste à calculer simultanément ce qui suit :
Cette formule est tirée du cours de Machine Learning de Mme Boutorh 2018/2019
Il faut répéter le processus simultanément jusqu'à ce que la convergence de J (θ0, θ1, θ2, θ3… θn.).

-La répétition du processus est représentée par un nombre d'itération.
-La dérivée de θj est calculée par rapport à notre hypothèse *h*
-Le α est déterminé dans la partie de test (il faut le rechanger jusqu'à trouver la bonne valeur)

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

I.5 Equation normale :

Nous avons ajouté une fonction de l'équation normale qui nous permet de récupérer les thêtas sans avoir à utiliser un algorithme de descente du gradient, il est très puissant lorsqu'il y'a assez peut de features.
Son équation :

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

I.6 Test & Validation :

Convergence de la fonction de cout :
Nous avons avec suffisamment d'itérations et avec un pas d'apprentissage suffisamment petit put avoir un cout de la fonction de cout intéressant, ainsi elle converge aux environs de 70 comme le montre la figure suivante, qui est la descente de la fonction cost J à travers les itérations.

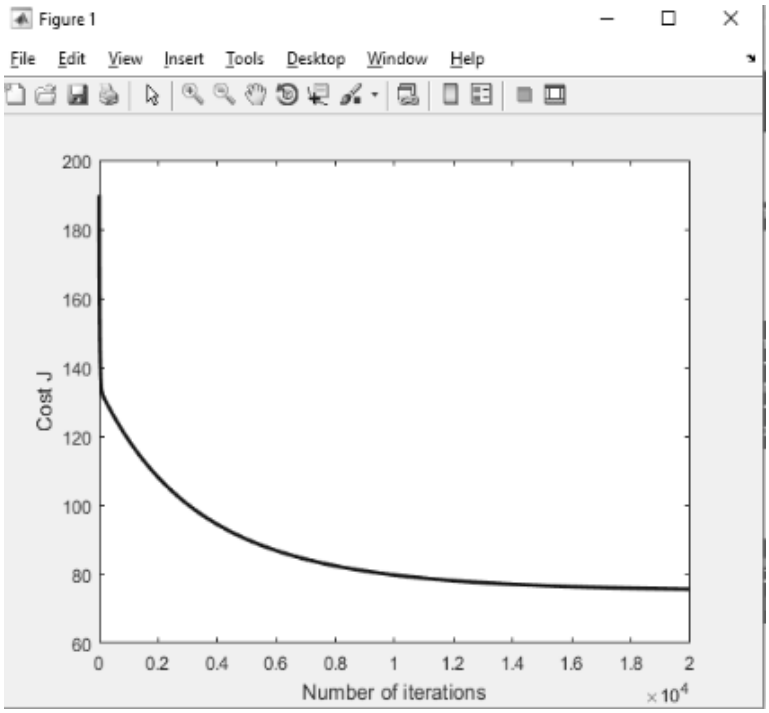


Figure : qui démontre que la fonction de cout J converge

Nous avons aussi un Taux de réussite acceptable qui est de : 76,74%, comme le montre la figure suivante après avoir effectué un test (présent dans le code main tout en bas).

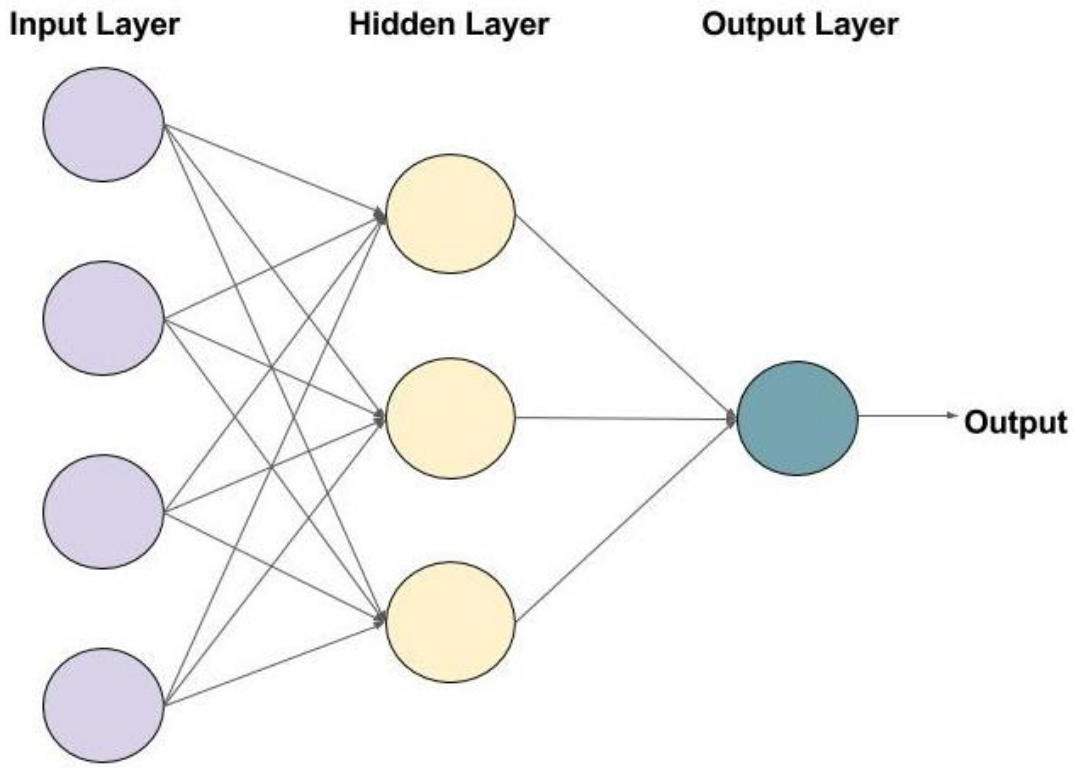
```
ans =

   76.7408
```

Figure qui démontre le taux de succès

NEURAL NETWORK

C'est une méthode inspirée de la neurobiologie, et plus précisément de la manière dont communiquent les neurones dans le cerveau, d'où le nom réseau de neurones.

Dans les systèmes informatiques, c'est une couche d'entrée appelée "Input layer" qui va contenir nos données d'entrées, une ou plusieurs couches cachées que l'on va nommer "Hidden layers", et notre couche de sortie aussi appelée "Output layer".Nous avons conçus et développé une architecture de réseau de neurones comme suit pour le problème de regression.



(Figure tirée du site https://www.learnopencv.com/understanding-feedforward-neural-networks/)

Input layer : Une couche constituée de 20 nœud qui sont aussi appelées les inputs, et qui sont les features de notre DataSet + notre Bias Unit que l'on initialise à 1.

Hidden layer : Il s'agit d'une couche qui comporte 15 nœuds, soit 75% des nœuds de notre "input layer".

Output layer : Il s'agit d'un seul noud qui est la sortie de notre réseau de neurones. Nous avons modifié notre architecture ainsi que la fonction que va utiliser notre Output, notre erreur de back propagation suivant la dérivé.Nous allons détailler cela au cours de la section suivante.

Implémentation:

Main : c'est la fonction principale qui va faire appel à toutes les autres fonctions, c'est aussi la fonction où l'on va spécifier le nombre de couches, où l'on fait l'import des dataset...etc.

ForwardPropagation: C'est la fonction qui fait le forward propagation, ici on ne va pas utiliser la même fonction que dans le cas de la classification (i.e.: qui est la Sigmoïde), par ce que l'on ne cherche pas à avoir des valeurs entre 0 et 1, mais plutôt des valeurs réelles, on va appeler Wight nos poids des layers, nous avons aussi utiliser une fonction polynôme que l'on a appelé «g».

On va donc chercher à utiliser cette fonction :

$$G\left(A^{N+1}\right) = \sum_{i=1}^{n} A^{N}_{i} * weight^{N}_{i}$$

i: nombre de noud.

A: vecteur des nœuds.

n: nombre de nœuds.

N: le numéro de la couche (layer). $N < L$ (L : nombre de couches).

Deriv: C'est la fonction dérivée que l'on va utiliser lors de la back propagation, en fait on va dériver la fonction g(z) comme présenter un peu plus tôt lors de la forward propagation, ce qui va donner:

$$g'(X^{N}) = weight^{N}_{i}$$

Backpropagation: C'est la fonction qui fait la back propagation, nous avons utilisé la nouvelle dérivé que nous venons de montrer, et on finit par faire les calculs des Deltas que nous allons utiliser pour mettre à jour les poids thêtas.

CalculCost : Fonction qui sert à calculer la cost function J, comme représentée ci-dessous

$$J = 1/2m \sum_{i=1}^{m} (g(x_i) - y_i)^2$$

N: numéro du layer (la couche).

m: nombre d'instances.

NeuralNetwork: C'est une fonction très importante qui fait appel au fonctions citées, pour faire les itérations sur toute notre dataset.

Test & Validation : Nous n'avons pas eu de résultats concluants lorsque l'on a implémenté la solution, mais nous avons comme perspectives de refaire l'implémentation de manière plus rigoureuse afin de pallier aux problèmes rencontrés.

Conclusion:

Nous avons conclu que les problèmes linéaires sont encore loin d'être faciles à prédire de manière exacte ou parfaite, mais que l'on pouvait s'approcher d'un bon résultat, en utilisant les bons algorithmes ainsi que les bons outils et avec des données correctes et pré traitées.

Classification: Activity Recognition system based on Multisensor data fusion (AReM) Data Set

Introduction

Un système de reconnaissance d'activité est un système de détection utilisé afin de fournir une assistance aux activités et de soins aux utilisateurs des maisons intelligentes, et qui a comme objectif de classer l'activité exercée par une personne en se basant sur un ensemble de données récupérées depuis un ensemble de dispositifs de capteur WSN placées sur la poitrine et les chevilles.
Ce système doit reconnaitre l'activité courante en utilisant l'ensemble d'informations provenant du changement ou modification implicite du canal sans fils due aux mouvements de l'utilisateur, et appliquer par la suite le processus de reconnaissance en se basant sur la classification qui est un axe d'apprentissage automatique qui sert à faire une classification d'un ensemble de données en se basant sur l'apprentissage initial, et pour cela notre système va considérer activités différentes qui sont :

1. Debout
2. Assis
3. Couché
4. En marchant
5. Faire du vélo
6. Penché (se pencher) en pliant les jambes
7. Penché (se pencher) sans pliant les jambes

Et pour cela nous illustrons par la suite les différentes étapes suivies et les différentes techniques utilisées afin de permettre à notre système d'avoir les meilleurs résultats.

Prétraitement :

Le prétraitement des données est une étape cruciale dans le processus de découverte de connaissances à partir de grandes bases de données. En effet, il permet d'améliorer la qualité des données soumises par la suite aux algorithmes d'apprentissage, Notre base de données qui contient 42240 instances doit être passé par un ensemble d'étapes qui consiste à nettoyer les données, structurer les données …, afin d'améliorer la qualité de base de données, et de clarifier sa structure par la suite aux algorithmes de data mining.
Une étude approfondie sur notre data set : importance et utilité des features ainsi leurs caractéristiques, nous as permis d'extraire ses grands problèmes. Ces problèmes ont était parfaitement soulevé en appliquant les technique de prétraitement :

Problèmes	Traitement établie
-La base de connaissance composée de plusieurs fichiers CSV répartie dans 7 dossiers selon l'activité signifié	-rassemblement de l'ensemble des fichiers dans un seul - transformation du fichier résultant en format matriciel.
Les classes (targets) que model trait sont présenté dans le nom de dossier pas parmi les attribues de data set	-ajouté dans notre data set une colonne **label** qui contient les classes représentant les valeurs des activités. -Ainsi ces classes doivent être représenté d'une manière à rendre l'apprentissage plus facile a effectué, pour mener à ce but les valeurs des classes seront transformé en données numérique ou la valeur de chaque classe (activité) serra prédéfinie par nous-mêmes. Et appliqué pour toute l'instance de mêmes valeurs. -Le codage choisis et le suivant : Penché (se pencher) sans pliant les jambes → 1 Penché (se pencher) en pliant les jambes → 2

	Faire du vélo → 3 Couché → 4 Assis → 5 Debout → 6 En marchant → 7
Duplication des tuples et redondance de données dans l'instance relative à une activité	-suppression des tuples doublant.
L'attribue temps qui es un attribue non significatif et non exploité par l'apprentissage.	-éliminé l'attribue et suppression de la colonne temps correspondante

Le prétraitement nous as permis d'améliorer la qualité de notre base de connaissance avoir une base de connaissance

-Le nombre total d'instance : 37072

-Le nombre totale d'attribue : 8 (**features** : 7, un attribue **target** : représentant les classes)

-absence de redondance.

→Ceci nous a permis de décomposer le data set en 3 parties (**60% trainning, 20% testing, 20% validation**) dans le but d'améliorer la qualité de l'apprentissage (éviter le over et le inder fitting) or les instances de chaque partie sont sélectionnées de la base de données d'une manière aléatoire en utilisant une fonction random.

→Chaque partie est décomposé en partie input représentant les features et partie output représentants les classes à prédire.

Technique d'apprentissage utilisée :

Dans cette partie nous illustrons les différentes techniques utilisées et implémentées et qui sont :

Logistic Regression :

Le modèle de régression logistique permet d'exprimer sous format de probabilité la relation entre une variable de classes, dite variable dépendante, et des variables explicatives.

Dans notre cas, on cherchera à exprimer la probabilité de l'activité exercé par l'individu en connaissant les caractéristiques de cet individu.

Nous avons opté a utilisé ce modèle car il permet d'analyser plus précisément l'impact de chaque variable retenue et de pouvoir quantifier cette relation, ainsi notre base de connaissance s'adapte parfaitement au model du faite que tous nos données sont quantitatif et de type réel ce qui permet de bien faire l'apprentissage.

Comme notre base de connaissance permet de prédire plusieurs classes (7 classes) , il n'est pas possible d'appliquer un modèle de régression logistique binaire pour effectuer les prédiction, il devient donc impératif d'utilisé une technique de Généralisation de la Régression Logistique à la classification multi classe , en utilisant une méthode appelé **ONE VS ALL** :Le principe de cette démarche consiste a associé pour chaque classe à prédire (type de label), une fonction de prédiction **segmoide** cette dernière retourne la probabilité d'appartenir à la classe, or chaque classe doit avoir ses propres valeurs de theta, la prédiction se fera sur l'ensemble des fonctions conçus en sélectionnant la class relatif à la fonction qui correspond à la valeur maximal lors de la prédiction (**max(h(x)**).

Nous avons procédé par trois méthodes différentes pour résoudre les valeurs de theta. Dans le but de mesuré la performance de chaque model et établir une étude comparative sur l'ensemble des résultats abouti.Ceci a données lieux 2 algorithmes :

1- régression logistique optimisé et régularisé avec fmincg.

3- régression logistique régularisé et non optimisé (gradient descent).

1- régression logistique optimisé et régularisé avec fmincg.

La méthode « régularisation et optimisation » appliqué au model permet d'obtenir des valeurs de theta optimaux offrant une meilleure pénalisation aux attribues afin d'éviter le over et l'inder fitting. Dans ce model nous avons opté a utilisé une fonction prédéfinie dans Matlab qui est **fmincg** pour appliquer une optimisation aux différentes valeurs de theta régularisé, cette fonction s'adapte parfaitement à notre model du faite que le nombre de features et petit (6 features) ayant un type réels.

Le schéma ci-dessus illustre la conception globale de l'application de l'algorithme et la prédiction.

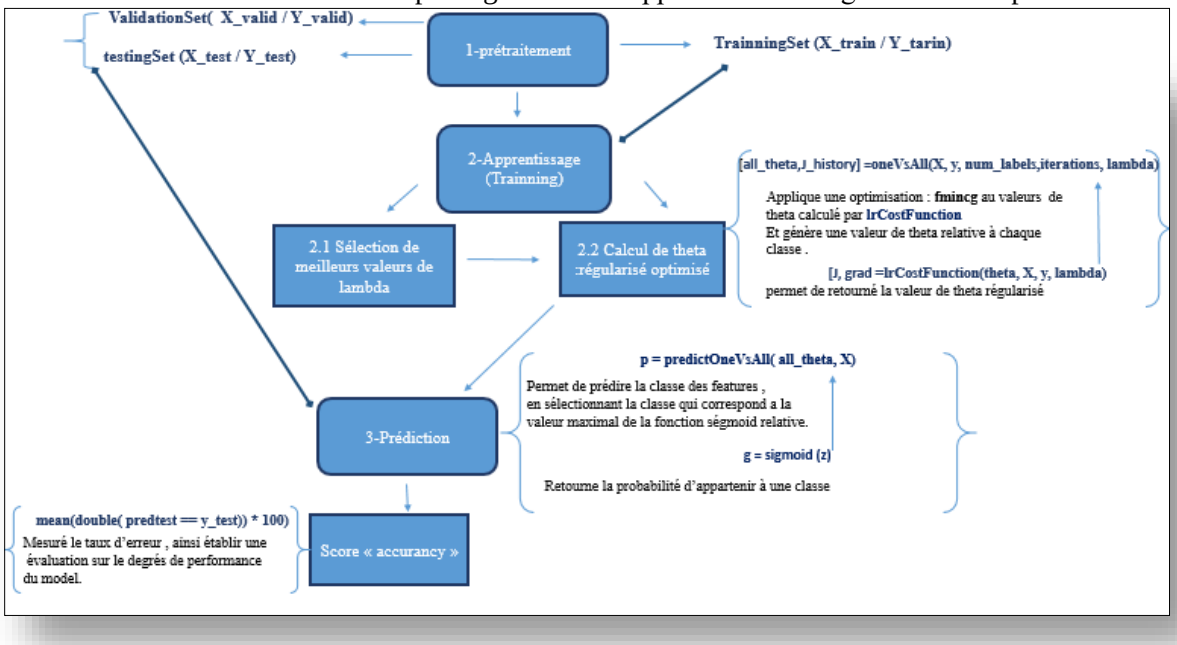


Figure : démarche de conception

En somme, tester les différentes valeurs de lambda sur le model nous as permis de sélectionner la meilleure valeur qui permet d'avoir une erreur minimal correspondante a la valeur de cost.

La performance du model conçus est mesuré en calculant la fonction du cout, cette dernière doit converger vers une valeur minimale très réduite. En fixant le nombre d'itération maximal à 200 et la valeur de lambda à 20 nous avons mesuré la performance de notre model en établissant une visualisation graphique des valeurs de la fonction d'erreur global du model ainsi que la fonction d'erreur relatif à chacune des classes dans les deux figures ci-dessous

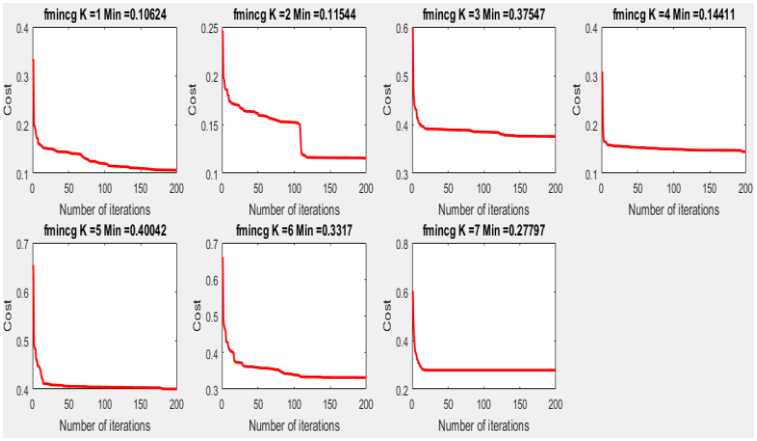


Figure : Cost relatif à chaque classe regression logistique Optimisé et régularisé

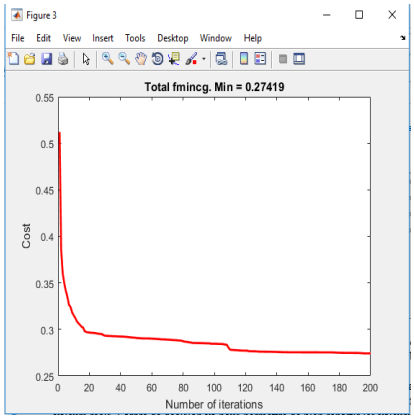


figure : Cost du model optimisé et régularisé

Il est difficile de représenter les lignes de décision (décision boundry) en prenant en compte 2 variables seulement (2D) ; malgré sa nous avons proposé une représentation qui permet de simuler ces lignes de décision.

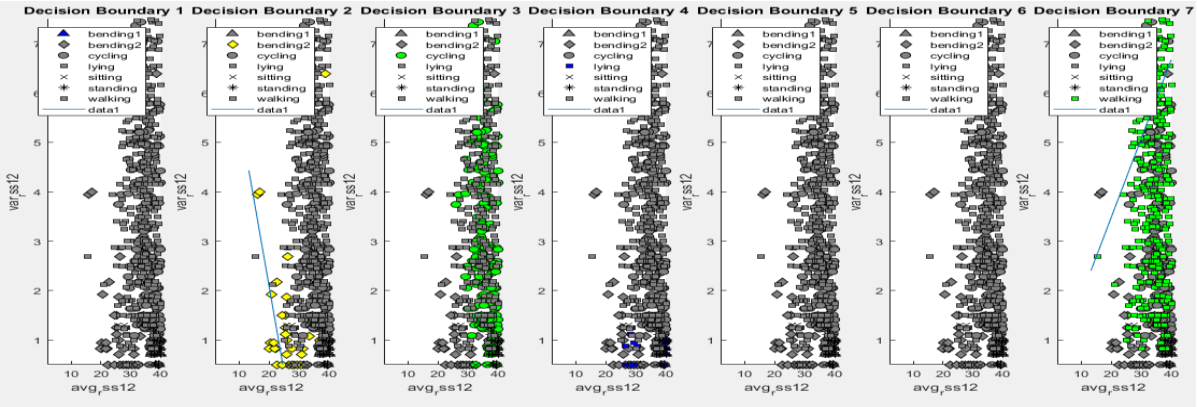


Figure : plot decision boundary model optimisé et régularisé

3. régression logistique régularisé et non optimisé (gradient descent).

En bref cet techenique permet d'obtenir des valeurs de theta régularisé afin de pénalisé les attribue , la régularisation en ajoutant un parametre de régularisation or le calcule du gradien decent et du cost se fera comme suite :

Les résultats abouti de set partie est présent dans la partie annex

L'application de la prédiction dans chaque model sur l'ensemble des données test et validation nous as permis d'evaluer la qualité ainsi que le dégrés de performance de chaque , afin détablir une étude comparatif des résultats différentes méthodes.

	régression logistique optimisé et régularisé avec fmincg.	régression logistique optimisé et régularisé avec fmunc. (QuasiNewton)
Validation accurancy	**69.747134**	**63.413642**
Test accurancy	**69.216347**	**64.546898**

Suite à nos résultats de comparaison entre les meilleures valeurs de score de chaque model, il est clair que le model de **régression logistique régularisé et optimisé fmincg** avec lambda = 20 et un nombre d'itération=200 est le plus précis et efficace globalement.

La méthode de régression logistique avec ces différentes approches permet d'établir une estimation efficace, l'efficacité des résultats abouti est lié en premier lieu au bon choix des paramètres d'apprentissage (lambda et alpha) ainsi qu'aux coefficients (theta) ces paramètres condition la qualité des résultats abouti .le choix des techniques de résolution de la logistique est lié aux caractéristiques de la base de connaissance(corrélation des variable, nombres de variable nombre d'instances…)

Il est donc impératif d'établir une étude approfondie sur le jeu d'entrainement pour arriver à sélectionner le model plus adapté.

Decision Trees :

Arbres de décision[i] est une catégorie utilisée dans l'exploitation des données, qui est basée sur une représentation hiérarchique de la structure de données sous forme d'un ensemble de décision et des règles vus comme une ensemble de nœuds et des arêtes.

Nous avons utilisé cette technique dans notre projet à cause de ses nombreux avantages et utilités, et comme notre jeu de données est composé d'un ensemble de valeurs quantitatifs plus précisément des valeurs réels, l'arbre de décision va nous permettre de bien répartir les valeurs et les conditions sur l'ensemble des nœuds et arêtes de notre arbre, et en quelque sort de construire un bon filtre/processus de filtre qui sera utilisé pour la reconnaissance.

La base de ce travail est de mettre et créer des intervalles comme notre projet et base sur un jeu de donnée quantitatif et donc la division des valeurs qui peuvent être prise par l'ensemble des instances pour chaque attribut / feature , ces valeurs vont générer 6 intervalles, dont chaque intervalle correspond a un attribut / feature, et par la suite le travail est basé sur la division de ces intervalles et de trouver la valeur pertinente qui peut être la médian, la moyenne des valeurs, …etc, ensuite un nœud sera créé qui va contenir une condition qui est la valeur pertinente choisi et par la suite des arêtes doivent être créées qui vont contenir les conditions supérieur ou égale valeur pertinente et inférieur valeur pertinente (la plus des cas le fils droit va contenir les valeurs >= val pertinente et le fils gauche < val pertinente)

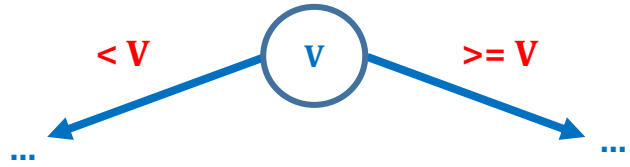


Figure : Structure d'un arbre de decision

Avant de créer les nœuds nous devons choisir quel attribut / feature sera traiter en premier et pour cela une méthode est utilisée qui mesure la pertinence d'un attribut et le calcul d'entropie et gain d'information[ii]

Notre travail est basé sur l'utilisation de la fonction « **Fitctree**[iii] » qui est une fonction prédéfinie qui existe dans Matlab, utilisée pour les problèmes de classification multi-classe/binaire qui nécessite en entrée deux matrices, une matrice qui contient le jeu de données et qui composé de N lignes qui le nombre d'instances utilisé pour la partie « Training » et M colonnes pour le nombre des attributs utilisés, et une deuxième matrice à N lignes comme la première et une seule colonne qui contient les résultats/classes des instances choisis, ces deux matrices corresponds aux deux matrices X et Y qui sont les classes (résultats) après la suppression de cette colonne de la matrice X, cette fonction nous donne en sortie l'arbre final, et les résultats sont les suivants (l'arbre généré est en annexe B)

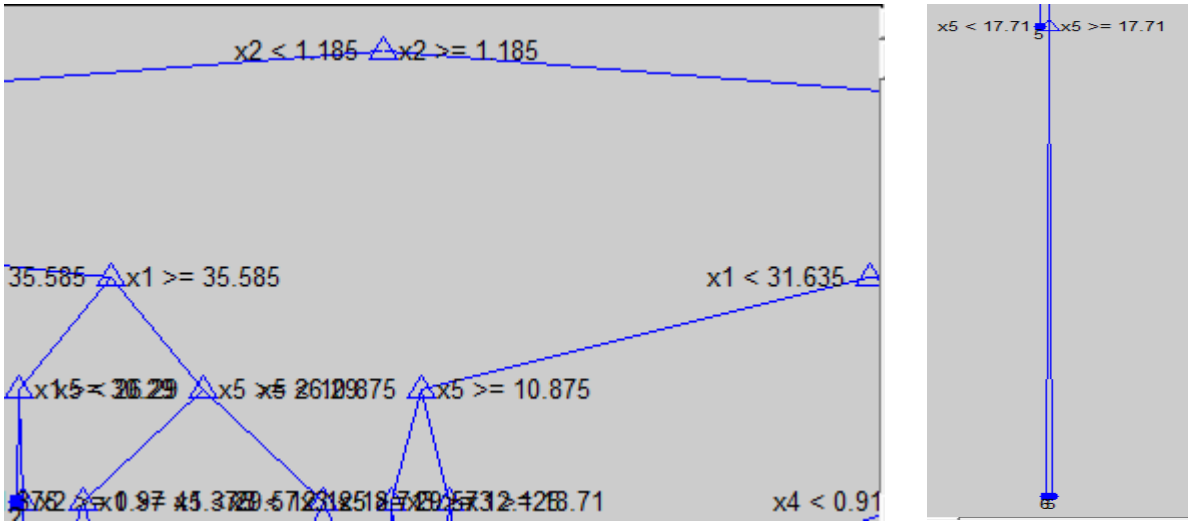


Figure : Résultat l'arbre de décision obtenu (racine, feuille).

Pour la partie apprentissage on a eu les informations suivantes :

PredictorNames: {'x1' 'x2' 'x3' 'x4' 'x5' 'x6'}
ResponseName: 'Y'
ClassNames: [1 2 3 4 5 6 7]
ScoreTransform: 'none'
CategoricalPredictors: []
NumObservations: 22243

Pour la partie Test, nous avons testé toutes les instances, et on vérifie les sorties en augmentant deux compteurs pour le cas valide et invalide, et à la fin on affichera le pourcentage de prédiction valide :

```
Prédiction : %
   71.7292
```

Nous allons utiliser la commande « predict » de Matlab qui est utilisée pour tester et prédire les résultats et qui nécessite en paramètre l'arbre généré et l'instance a testée et elle nous donne en sortie la classe prédite.

L'instance (+ la commande) : `[48,0,1.75,0.43,11,0.71]`

```
predict(M,[48,0,1.75,0.43,11,0.71])
```

Sa prédiction :

```
result =

     4
```

Avec les fonctions prédéfinies de Matlab qui vont nous permettre de manipuler et visualiser les différentes propriétés de notre arbre générée :

Estimation de l'importance de prédiction :

On peut la faire en utilisant quelque commande (écrite et démontré dans le code – Annexe A.c) pour estimer et connaitre le feature le plus important qui nous aide à bien prédire, et pour notre cas le 1ér feature qui est la plus importante

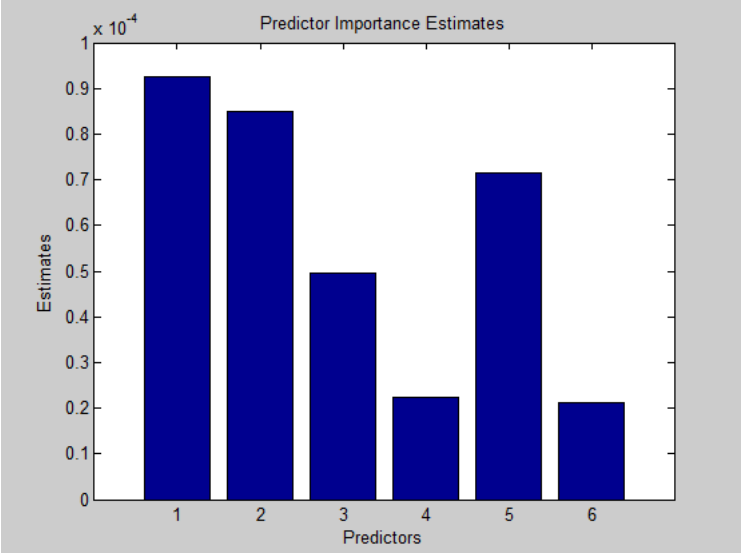


Figure : Histogramme des features les plus importants

Une fonction a été implémenté mais elle ne peut pas être utilisé à cause de Matlab qui ne supporte pas certain condition et traitement, cette fonction applique la méthode expliquée précédemment, et elle repose sur 2 tableaux, un qui contient les nœuds et leurs indices qui ont été construire en faisant une création par longueur préfixe en parcourant l'ensemble des colonnes de notre jeu de donnée pour trouver feature pertinent ensuite on calcul la médiant pour mettre un intervalle et mettre les conditions et le processus se répète récursivement ajoutée en annexe B (partie (a.2))

Les arbres de décisions est une technique performante et efficace, qui est basée sur un ensemble de paramètres qui vont faire la différence et changer la structure de l'arbre en sortie notamment la valeur significative de la condition des nœuds, et elle sans avoir besoin des facteurs supplémentaires comme les autres méthodes, sauf qu'elle nécessite la présence de au moins une instance lors la phase de training sinon cela va engendrer l'ignorance de cette classe dans les résultats.

Neural Network :

C'est une méthode inspirée de la méthode de travail du cerveau humain, en utilisant presque le même principe et qui nécessite les neurones qui sont les cellules vivantes de cette technique, et qui sont répartis sur plusieurs couches (couche d'entrée, couche de sortie et les couches cachées), et à chaque couche - sauf la couche cachée - on associe des poids de sort que chaque couche est multipliée par ces poids (le poids de la couche qui précède cette dernière), nous avons opté à utiliser cette méthode car elle nous permet de bien classier l'ensemble des valeurs en entrée comme ils sont des valeurs distinctes, a travers l'ensemble de neurones crées en pourra bien avoir les bons résultats a travers les différentes traitements.

Et pour cela nous avons utilisé ce système[iv] qui va nous permettre de bien classifier l'activité exercée à un moment donné, et qui va générer un ensemble de poids en se basant sur la partie d'apprentissage et sur les facteurs mise en entrée afin d'avoir les meilleurs résultats par la suite.

Dans ce travail nous avons opté pour utiliser :

- Une couche d'entrée constituée de 6 neurones qui égale au nombre de feature de notre jeu de données.
- Deux couches cachées, dans chacune 75% de nombre de neurones[v] de la couche d'entrée qui égale à 5 neurones.
- Une couche de sortie qui contient 7 neurones qui correspond aux 7 classes de résultats.

L'étape suivante consiste à séparer les entrées et les résultats, donc on doit mettre les entrées dans une matrices de m colonnes = nombre de feature et les résultats dans un vecteur qui va être transformé en une matrice de 0 et 1 ou chaque ligne correspond à une instance et chaque colonne correspond a la classe résultat, si cet instance appartient au début a une classe j elle aura a la colonne j et 1 sinon elle aura un 0.

Par la suite, une génération des poids sera mise en place, et afin de contrôler la fonction aléatoire il est conseillé d'utiliser une méthode[vi] et ne pas avoir des grandes valeurs de poids et donc perturber les résultats par la suite

A la fin on doit préciser le nombre d'itération maximum pour l'exécution de notre fonction pour appliquer le fonctionnement de Gradient Descent qui nécessite ce paramètre sans oublier de préciser Alpha (learning rate) et Lambda pour la régularisation.

Le corps de notre fonction de réseau de neurones et décrite par le pseudocode suivant :

```
Initialisation des Delta
        Répition jussqu'a un nbr d'itération
                Pour chaque instance faire
                        ForwardPropagation() ;
                        BackPropagation() ;
                Fait
                Calculs des D( Delta ) ;
                MiseAJour des poids() ;
                CalculCost() ;
        Fait
```

Et dans notre projet nous avons implémenter trois types de réseau de neurones afin d'avoir les meilleurs résultats avec un meilleur apprentissage, et pour faire une comparaison par la suite, un réseau qui contient une seule couche cachée et le deuxième avec 2 couche cachée et le troisième qui peut contenir un ensemble de couche cachée (n couches) mais on ne l'a pas cité en annexe.

Le model avec 2 couche cachée est vu comme :

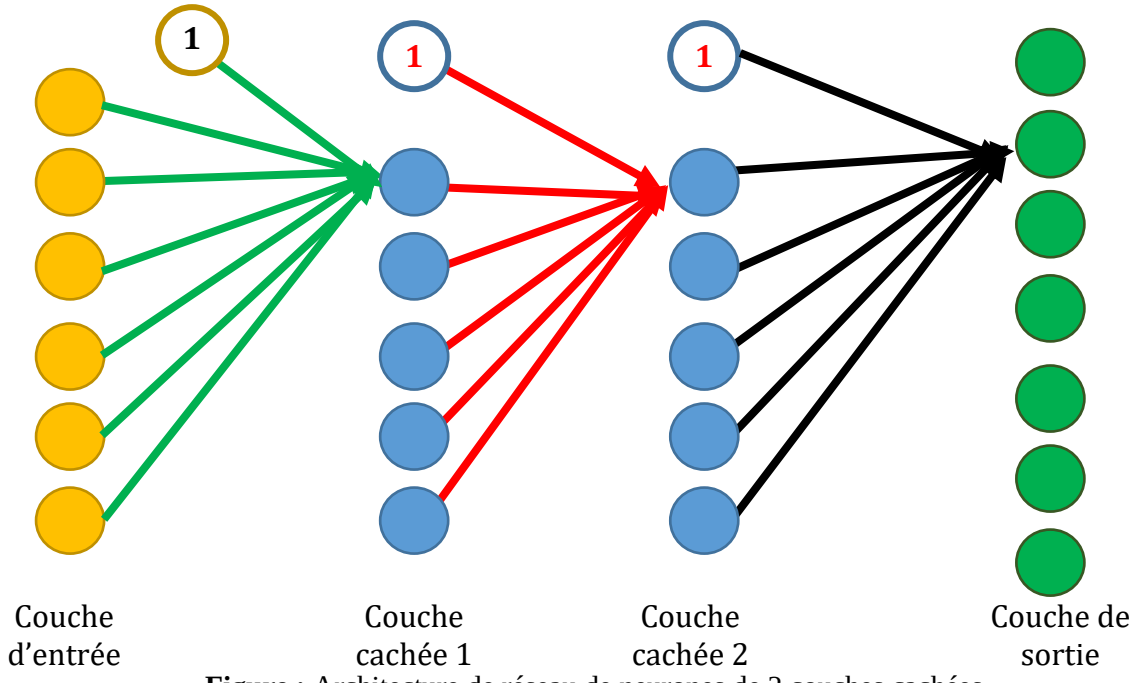
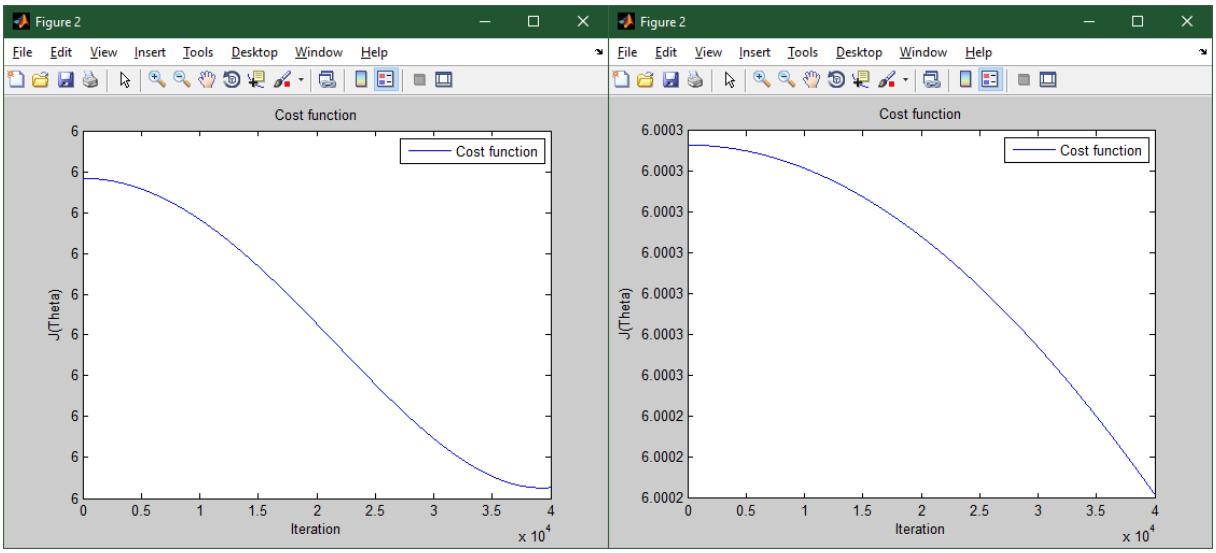


Figure : Architecture de réseau de neurones de 2 couches cachées



RN with 1 hidden layer RN with 2 hidden layers

Figure : Résultat de fonction de coût

La fonction Cost - Coût va nous confirmer que le processus de training est dans la bonne voie, et en utilisant les paramètres suivants :
Nombre d'itération max : 200
Alpha : 0. 00001
Lambda :0.01
Et faisant la partie test on peut confirmer que les bons paramètres ont été trouvée et que la bonne classification sera vu par la suite, et donc nous avons testé toutes les instances, et on vérifie les sorties en augmentant deux compteurs pour le cas valide et invalide, et à la fin on affichera le pourcentage de prédiction valide :

```
Prédiction : %
   18.0604
```

Et pour cela nous allons prendre un exemple :
Instance :

[39,50 0,50 1 0 17,50 0,50]

En sortie :

```
I =

     5
```

Les problèmes rencontrés a la phase d'apprentissage (convergence de la fonction coût) qui est bloqué dans la valeur 6, cela à engendrer de problème de prédiction des valeurs par la suite, e les résultats ne sont pas fiable à 100% (avec une estimation de prédiction correcte 18.06%)
Cette est une méthode efficace et performante mais elle restera toujours bonne en fonction des paramètres lors e la phase de training, et qui sont les facteurs majeurs de la fiabilité et l'efficacité de cette technique, et en fonction de ce choix nous obtiendrons les meilleurs résultats, sauf que elle nécessite un bon moment lors de l'étape de training et validation pour un nombre important d'instance et qui va mettre le processus d'apprentissage dans une grande boucle fermée afin d'avoir les meilleurs paramètres recherchés pour les meilleurs résultats par la suite.

SVM :

Le classfieur SVMs a été conçu pour une séparation de deux ensembles de donnée. Le but de SVM est de trouver un hyperplan qui va séparer et maximiser la marge de séparation entre deux classes. Le problème peut être résolu au moyen de méthode d'optimisation quadratique.
Nous avons choisi ce classifieur car il s'avère particulièrement efficace de par le fait qu'il peut traiter des problèmes mettant en jeu de grands nombres de descripteurs, qu'il assure une solution unique (pas de problèmes de minimum local comme pour les réseaux de neurones) et il a fourni de bons résultats sur des problèmes réels.

Comme notre base de connaissance permet de prédire plusieurs classes (7 classes). Pour chaque classe, on détermine un hyperplan séparant celle-ci de toutes les autres classes. Ainsi, pour 7classes, on doit déterminer 7fonctions de décision. Tous les exemples appartenant à la classe considérée sont étiquetés positivement (+1) et tous les exemples n'appartenant pas à la classe sont étiquetés négativement (-1) une donnée est affectée à la classe qui correspond à la valeur maximale des fonctions de décision.
L'implémentation de la solution est réalisée en utilisant un logiciel matlab « libsvm » qui est un logiciel simple, facile à utiliser et efficace pour SVM.classification et régression. Il résout la classification multi-class : C-SVM, et binaire : nu-SV classification classe unique
, classe-SVM, régression. Il fournit également un outil de sélection automatique de modèles pour Classification C-SVM représentant les différents types de noyaux : linéaire, polynôme, fonction base radia (noyaux gaussien), segmoide, et le noyau recalculé. Deux fonctions sont utilisées pour effectuer l'apprentissage et la prédiction : **svm-train [options] :** permet de faire l'apprentissage sur le jeu d'entrainement (x_train, y_train) en appliquant le noyau choisi cette fonction retourne un model qu'on utilise pour la prédiction ; la prédiction est réalisée en faisant appel à la fonction **svm-predict [options] :** qui utilise le model généré dans l'entrainent et l'applique pour les données de test et validation pour retourner un score de prédiction.
Afin d'évaluer la performance de notre solution nous avons entrainé notre jeu de données sur les différents types de kernel adapté pour notre model et comparé les meilleurs résultats de chaque model les résultats aboutis sont résumés dans le tableau ci-dessus

	RBF : c = 1 g= 8	**Polynôme c =10 g =2**	**Linéaire c= 1 g=0.1**
Validation accurancy	72.7858	66.2725	68.4678
Test accurancy	72.9643	66.1675	68.3754

Suite à nos résultats de comparaison entre les meilleurs valeurs de score de chaque model, il est clair que le model **RBR** avec C = 20 et g=200 est le plus précis et efficace globalement.
Le model SVM est une technique performante ; il permet d'établir un apprentissage efficace et de qualité, le degré de performance des SVM dépend des paramètres d'apprentissage C et Gamma, ainsi que les caractéristiques de la base de connaissance ce qui condition l'utilisation de certain kernel parmi d'autre.

Comparaison & discussion des résultats :

Enfin, selon les données la performance des modèles utilisés pour traiter notre problème de classification est en général de même ordre, malgré cela la performance des SVMs s'avère d'ordre voire supérieure à celle d'un réseau de neurones ou d'un modèle de mélanges gaussiens (logistique) ou arbre de décision ,Il a aussi été montré qu'en utilisant un **noyau RBF,** les SVMs deviennent plus performant ; c'est à dire qu'avec suffisamment de données, l'algorithme peut toujours trouver la meilleure frontière possible pour séparer deux classes .**donc le noyau RBF est le plus adapté**

Répartition des tâches :

Nous avons travaillé en utilisant un système appelé **PEER REVIEW** et **PEER PROGRAMMING**, qui consiste en ce que le travail se code à deux, et aussi à chaque fois qu'un monôme ou binôme effectue sa partie de travail demandé, cette partie-là sera vérifiée et jugé ainsi que rectifié par un autre monôme ou binôme.

KRIM Islam	MASDOUA Manil
-Composition et implémentation du travail de la regression linéaire. -Validation et implémentation de l'architecture du réseau de neurones pour le problème de regression. -Etude et validation de l'implémentation du réseau de neurones pour le problème de classification. -Optimisation et recherche approfondie au sujet du problème de la regression linéaire. -Vérification de l'homogénéité du rapport.	Composition et implémentation de l'architecture du réseau de neurones pour le problème de regression. -Validation et implémentation du travail de la regression linéaire. -Etude et validation de l'implémentation du réseau de neurones pour le problème de classification. -Rédaction et vérification de l'orthographe ainsi que la cohérence des syntaxes utilisées.

AKKOUCHE Abderrahmane	SASSI Kahina
-implémentation de l'algorithme de l'arbre de décision et rédaction de la partie correspondante dans le rapport _implémentation de l'algorithme de réseau de neurone et rédaction de la partie correspondante dans le rapport _le prétraitement des données. -vérification de la cohérence et l'homogénéité du rapport	-implémentation de l'algorithme de regression logistique et rédaction de la partie correspondante dans le rapport _implémentation de l'algorithme de SVM de neurone et rédaction de la partie correspondante dans le rapport _le prétraitement des données. -vérification de la cohérence et l'homogénéité du rapport

ANNEXE A:

1. Online Video Characteristics and Transcoding Time Dataset Data Set:

a.lineare regression

main

```
A = load('dataset.txt');
B = load('datasetY2.txt');
T = load('dataset.txt');
INPUT = A(1:40000,:);
OUTPUT = B(1:40000,:);
A=[ones(size(OUTPUT)),INPUT];
OUTPUTtest = B(40001:68780,:);
lambda = 0.5;
C=projetIAfeatureNormalize_TEAM_15(A);

TEST = T(40001:68780,:);
TESTING = [ones(size(OUTPUTtest)),TEST];
TESTING = projetIAfeatureNormalize_TEAM_15(TESTING);


iterations =20000;
alpha = 0.0001;
theta = zeros(size(A,2), 1);
[theta,Vect] =
projetIAGradientDecent_TEAM_15(C,OUTPUT,theta,alpha,iterations,lambda);
 %Plot the convergence graph

plot(1:numel(Vect), Vect, '-b', 'LineWidth', 2);
xlabel('Number of iterations');
ylabel('Cost J');

% Display gradient descent's result
fprintf('Theta computed from gradient descent: \n');
fprintf(' %f \n', theta);
fprintf('\n');
sum=0;
test = [];
disp(OUTPUT(1));
error = 0;
%test
for i= 1 : 28780
    test= TESTING(i,:)*theta;
    if(sqrt((test - OUTPUTtest(i,:)))^2 > 7)
        error = error +1;
    end
    erreur = sqrt((test - OUTPUTtest(i,:)))^2;
    sum = sum+erreur;
end
disp(sum/ 28780);
taux = (28780 - 6694)*100/28780;
disp(taux); %taux de réussite
```

2 . CostFunctionRegul_TEAM_15

```
function [J] = CostFunctionRegul_TEAM_15(X, y, theta, lambda)

m = length(y); % number of training examples
J = 0;
```

```
h = (X * theta);
cost = (1 / (2 * m)) * sum((h - y) .^ 2);
regularis = (lambda / (2 * m)) * sum(theta(2:20) .^ 2);
% Regularized cost is just summation.
J = cost + regularis;


end
```

3. Normalize_TEAM_15

```
function [XNorm, mu, stddev] = Normalize_TEAM_15(X)
XNorm = X;
mu = zeros(1, size(X, 2));
stddev = zeros(1, size(X, 2));
for i=1:size(mu,2)
    mu(1,i) = mean(X(:,i));
    stddev(1,i) = std(X(:,i));
    XNorm(:,i) = (X(:,i)-mu(1,i))/stddev(1,i);
End
```

4. projetIAfeatureNormalize_TEAM_15

```
function [Xnormalis, moyenne, stddev] = projetIAfeatureNormalize_TEAM_15(X)

Xnormalis = X;
moyenne = zeros(1, size(X, 2));
stddev = zeros(1, size(X, 2));
for i=2:size(moyenne,2)
    if(i ~= 14 && i~= 10  )
        moyenne(1,i) = mean(X(:,i));
        stddev(1,i) = std(X(:,i));
        Xnormalis(:,i) = ((X(:,i)-moyenne(1,i))/stddev(1,i)) + 1;

    end
end

 Xnormalis=Xnormalis + ones(size(Xnormalis))*2.5;
```

projetIAGradientDecent_TEAM_15

```
function [theta, J_history] = projetIAGradientDecent_TEAM_15(X, y, theta,
alpha, num_iters,lambda)

m = length(y);
J_history = zeros(num_iters, 1);

for iter = 1:num_iters
    thetas = zeros(size(X, 2), 1);
    for i = 1:size(X, 2),
        t = theta(i) - alpha * (1 / m) * sum(((X * theta) - y) .* X(:, i));
        thetas(i) = t;
    end
    theta = thetas;
    J_history(iter) = CostFunctionRegul_TEAM_15(X, y, theta, lambda);

end

end
```

neural network

main

```
A = load('dataset.txt');
B = load('datasetY2.txt');
Nbr_result=1;
Nbr_Hidden_Units = 15;

%tableau de la couche caché
Nbr_Hidden_Layer = 1;
Hidden_Layers = zeros(Nbr_Hidden_Units,Nbr_Hidden_Layer);

%Matrices des resultats (Output))
Y = B(4000,:);


X = A(4000,:);



Nbr_Feature = 19;

%Valeur d'epsilon - a ne pas depasser par theta lors d'initialisation
Init_Epsilon =(sqrt(6)/sqrt(7+6));


%les poids relative à la couche input
WeightInput = rand(Nbr_Hidden_Units,Nbr_Feature+1)*Init_Epsilon;

%les poids relative à les couches finale
WeightFinal = rand(Nbr_result,Nbr_Hidden_Units+1)*Init_Epsilon;


%Nombre d'itération max
Nbr_Iteration = 1;
Alpha = 0.0000000001;
Lambda = 0.001;
[Costvect,Result] =
NeuralNetwork_TEAM_15(X,Y,WeightInput,WeightFinal,Hidden_Layers,Nbr_Iterati
on,Lambda,Alpha);

figure;
%Affichage de la fonction J
plot(Costvect)
%Mettre une clé pour le graphe (le nom de la courbe)
legend('Cost function');
%Mettre un nom pour les deux axes
xlabel('Iteration');
ylabel('J(Theta)');
%Mettre un titre pour le graphe
title('Cost function');
```

ForwardPropagation_TEAM_15

```
function [Result] = ForwardPropagation_TEAM_15 (X, WeightInput)
    Result = WeightInput*X;

end
```

Deriv_TEAM_15

```
function S=Deriv_TEAM_15(Weight)
```

```
    S= Weight;
end

function [j]=Cost(X,Y,Output,WeightInput,WeightFinal,Lambda,Position)
    j=0;
    for i=1:size(X,1)
        for k=1:size(Y,2)

            j=j+ WeightInput * X + WeightFinal*(WeightInput * X ));
        end
    end
    j = (-1/size(X,1))*j;
    %Regularisation
    Regul =0;
    for i=1:size(WeightInput,1)
        for j=2:size(WeightInput,2)
            Regul = Regul +(WeightInput(i,j)^2);
        end
    end
    for i=1:size(WeightFinal,1)
        for j=2:size(WeightFinal,2)
            Regul = Regul +(WeightFinal(i,j)^2);
        end
    end
    j=j+(Regul*(Lambda/(2*size(X,1))));
end
```

Cost

```
function [j]=Cost(X,Y,Output,WeightInput,WeightFinal,Lambda,Position)
    j=0;
    for i=1:size(X,1)
        for k=1:size(Y,2)

            j=j+ WeightInput * X + WeightFinal*(WeightInput * X ));
        end
    end
    j = (-1/size(X,1))*j;
    %Regularisation
    Regul =0;
    for i=1:size(WeightInput,1)
        for j=2:size(WeightInput,2)
            Regul = Regul +(WeightInput(i,j)^2);
        end
    end
    for i=1:size(WeightFinal,1)
        for j=2:size(WeightFinal,2)

            Regul = Regul +(WeightFinal(i,j)^2);
        end
    end
    j=j+(Regul*(Lambda/(2*size(X,1))));
end
```

BackPropagation_TEAM_15

```
function[DeltaD,DeltaF]=BackPropagation_TEAM_15(X,Y,Result,WeightFinal,Hidd
en_Layers,DeltaD,DeltaF)
    ErrorF= Result-Y';
    Input = [1;Hidden_Layers(:,1)];
    Error(:,1) = (WeightFinal'*(ErrorF)).*(Input);
    DeltaD=DeltaD+ErrorF*Input';
    Err = Error(2:size(Error,1),1);
```

```
    Input = [1,X];
    DeltaF=DeltaF+Err*Input;
end
```

2. Activity Recognition system based on Multisensor data fusion (AReM) Data Set:

a. Prétraitements :

```
%-------------------------------------------------------------------------
-------------------------------------------------------
                %-----------------------------------------pértaitement
des données-----------------------------------------

 %-----------------------------------------------------bending1-----------
-----------------------------------%
bending11 = readtable('bending1\dataset1.csv', 'HeaderLines',0);
bending12 = readtable('bending1\dataset2.csv', 'HeaderLines',0);
bending13 = readtable('bending1\dataset3.csv', 'HeaderLines',0);
bending14 = readtable('bending1\dataset4.csv', 'HeaderLines',0);
bending15 = readtable('bending1\dataset5.csv', 'HeaderLines',0);
bending16 = readtable('bending1\dataset6.csv', 'HeaderLines',0);
bending17 = readtable('bending1\dataset7.csv', 'HeaderLines',0);
 %-----------------------------------------------------bending2-----------
-----------------------------------%
bending21 = readtable('bending2\dataset1.csv', 'HeaderLines',0);
bending22 = readtable('bending2\dataset2.csv', 'HeaderLines',0);
bending23 = readtable('bending2\dataset3.csv', 'HeaderLines',0);
bending24 = readtable('bending2\dataset4.csv', 'HeaderLines',0);
%bending24 = readtable('bending2\dataset4.csv','delimiter','space',
'HeaderLines',0);
bending25 = readtable('bending2\dataset5.csv', 'HeaderLines',0);
bending26 = readtable('bending2\dataset6.csv', 'HeaderLines',0);
 %-----------------------------------------------------cycling------------
-----------------------------------%
cycling1 = readtable('cycling\dataset1.csv', 'HeaderLines',0);
cycling2 = readtable('cycling\dataset2.csv', 'HeaderLines',0);
cycling3 = readtable('cycling\dataset3.csv', 'HeaderLines',0);
cycling4 = readtable('cycling\dataset4.csv', 'HeaderLines',0);
cycling5 = readtable('cycling\dataset5.csv', 'HeaderLines',0);
cycling6 = readtable('cycling\dataset6.csv', 'HeaderLines',0);
cycling7 = readtable('cycling\dataset7.csv', 'HeaderLines',0);
cycling8 = readtable('cycling\dataset8.csv', 'HeaderLines',0);
cycling9 = readtable('cycling\dataset9.csv', 'HeaderLines',0);
cycling10 = readtable('cycling\dataset10.csv', 'HeaderLines',0);
cycling11 = readtable('cycling\dataset11.csv', 'HeaderLines',0);
cycling12 = readtable('cycling\dataset12.csv', 'HeaderLines',0);
cycling13 = readtable('cycling\dataset13.csv', 'HeaderLines',0);
cycling14 = readtable('cycling\dataset14.csv', 'HeaderLines',0);
cycling15 = readtable('cycling\dataset15.csv', 'HeaderLines',0);
 %-----------------------------------------------------lying--------------
-----------------------------------%
lying1 = readtable('lying\dataset1.csv', 'HeaderLines',0);
lying2 = readtable('lying\dataset2.csv', 'HeaderLines',0);
lying3 = readtable('lying\dataset3.csv', 'HeaderLines',0);
lying4 = readtable('lying\dataset4.csv', 'HeaderLines',0);
lying5 = readtable('lying\dataset5.csv', 'HeaderLines',0);
lying6 = readtable('lying\dataset6.csv', 'HeaderLines',0);
lying7 = readtable('lying\dataset7.csv', 'HeaderLines',0);
lying8 = readtable('lying\dataset8.csv', 'HeaderLines',0);
lying9 = readtable('lying\dataset9.csv', 'HeaderLines',0);
lying10 = readtable('lying\dataset10.csv', 'HeaderLines',0);
lying11 = readtable('lying\dataset11.csv', 'HeaderLines',0);
lying12 = readtable('lying\dataset12.csv', 'HeaderLines',0);
```

```
lying13 = readtable('lying\dataset13.csv', 'HeaderLines',0);
lying14 = readtable('lying\dataset14.csv', 'HeaderLines',0);
lying15 = readtable('lying\dataset15.csv', 'HeaderLines',0);
 %-------------------------------------------------sitting---------------
-----------------------------------%
sitting1 = readtable('sitting\dataset1.csv', 'HeaderLines',0);
sitting2 = readtable('sitting\dataset2.csv', 'HeaderLines',0);
sitting3 = readtable('sitting\dataset3.csv', 'HeaderLines',0);
sitting4 = readtable('sitting\dataset4.csv', 'HeaderLines',0);
sitting5 = readtable('sitting\dataset5.csv', 'HeaderLines',0);
sitting6 = readtable('sitting\dataset6.csv', 'HeaderLines',0);
sitting7 = readtable('sitting\dataset7.csv', 'HeaderLines',0);
sitting8 = readtable('sitting\dataset8.csv', 'HeaderLines',0);
sitting9 = readtable('sitting\dataset9.csv', 'HeaderLines',0);
sitting10 = readtable('sitting\dataset10.csv', 'HeaderLines',0);
sitting11 = readtable('sitting\dataset11.csv', 'HeaderLines',0);
sitting12 = readtable('sitting\dataset12.csv', 'HeaderLines',0);
sitting13 = readtable('sitting\dataset13.csv', 'HeaderLines',0);
sitting14 = readtable('sitting\dataset14.csv', 'HeaderLines',0);
sitting15 = readtable('sitting\dataset15.csv', 'HeaderLines',0);
 %-------------------------------------------------standing--------------
-----------------------------------%
standing1 = readtable('standing\dataset1.csv', 'HeaderLines',0);
standing2 = readtable('standing\dataset2.csv', 'HeaderLines',0);
standing3 = readtable('standing\dataset3.csv', 'HeaderLines',0);
standing4 = readtable('standing\dataset4.csv', 'HeaderLines',0);
standing5 = readtable('standing\dataset5.csv', 'HeaderLines',0);
standing6 = readtable('standing\dataset6.csv', 'HeaderLines',0);
standing7 = readtable('standing\dataset7.csv', 'HeaderLines',0);
standing8 = readtable('standing\dataset8.csv', 'HeaderLines',0);
standing9 = readtable('standing\dataset9.csv', 'HeaderLines',0);
standing10 = readtable('standing\dataset10.csv', 'HeaderLines',0);
standing11 = readtable('standing\dataset11.csv', 'HeaderLines',0);
standing12 = readtable('standing\dataset12.csv', 'HeaderLines',0);
standing13 = readtable('standing\dataset13.csv', 'HeaderLines',0);
standing14 = readtable('standing\dataset14.csv', 'HeaderLines',0);
standing15 = readtable('standing\dataset15.csv', 'HeaderLines',0);
 %-------------------------------------------------walking---------------
-----------------------------------%
walking1 = readtable('walking\dataset1.csv', 'HeaderLines',0);
walking2 = readtable('walking\dataset2.csv', 'HeaderLines',0);
walking3 = readtable('walking\dataset3.csv', 'HeaderLines',0);
walking4 = readtable('walking\dataset4.csv', 'HeaderLines',0);
walking5 = readtable('walking\dataset5.csv', 'HeaderLines',0);
walking6 = readtable('walking\dataset6.csv', 'HeaderLines',0);
walking7 = readtable('walking\dataset7.csv', 'HeaderLines',0);
walking8 = readtable('walking\dataset8.csv', 'HeaderLines',0);
walking9 = readtable('walking\dataset9.csv', 'HeaderLines',0);
walking10 = readtable('walking\dataset10.csv', 'HeaderLines',0);
walking11 = readtable('walking\dataset11.csv', 'HeaderLines',0);
walking12 = readtable('walking\dataset12.csv', 'HeaderLines',0);
walking13 = readtable('walking\dataset13.csv', 'HeaderLines',0);
walking14 = readtable('walking\dataset14.csv', 'HeaderLines',0);
walking15 = readtable('walking\dataset15.csv', 'HeaderLines',0);

%--------------------------------Transformer les tables en matrices--------
-----------------------------------%

%-------------------------------------------------bending1-------------------
-----------------------------------%
bending11 = bending11{:,:};
```

```
bending12 = bending12{:,:};
bending13 = bending13{:,:};
bending14 = bending14{:,:};
bending15 = bending15{:,:};
bending16 = bending16{:,:};
bending17 = bending17{:,:};
 %-----------------------------------------------------bending2--------------
-----------------------------------%
bending21 = bending21{:,:};
bending22 = bending22{:,:};
bending23 = bending23{:,:};
bending24 = bending24{:,:};
bending25 = bending25{:,:};
bending26 = bending26{:,:};
 %-----------------------------------------------------cycling----------------
-----------------------------------%
cycling1 = cycling1{:,:};
cycling2 = cycling2{:,:};
cycling3 = cycling3{:,:};
cycling4 = cycling4{:,:};
cycling5 = cycling5{:,:};
cycling6 = cycling6{:,:};
cycling7 = cycling7{:,:};
cycling8 = cycling8{:,:};
cycling9 = cycling9{:,:};
cycling10 = cycling10{:,:};
cycling11 = cycling11{:,:};
cycling12 = cycling12{:,:};
cycling13 = cycling13{:,:};
cycling14 = cycling14{:,:};
cycling15 = cycling15{:,:};
 %-----------------------------------------------------lying------------------
---------------------------------%
lying1 = lying1{:,:};
lying2 = lying2{:,:};
lying3 = lying3{:,:};
lying4 = lying4{:,:};
lying5 = lying5{:,:};
lying6 = lying6{:,:};
lying7 = lying7{:,:};
lying8 = lying8{:,:};
lying9 = lying9{:,:};
lying10 = lying10{:,:};
lying11 = lying11{:,:};
lying12 = lying12{:,:};
lying13 = lying13{:,:};
lying14 = lying14{:,:};
lying15 = lying15{:,:};
 %-----------------------------------------------------sitting----------------
-----------------------------------%
sitting1 = sitting1{:,:};
sitting2 = sitting2{:,:};
sitting3 = sitting3{:,:};
sitting4 = sitting4{:,:};
sitting5 = sitting5{:,:};
sitting6 = sitting6{:,:};
sitting7 = sitting7{:,:};
sitting8 = sitting8{:,:};
sitting9 = sitting9{:,:};
sitting10 = sitting10{:,:};
sitting11 = sitting11{:,:};
```

```
sitting12 = sitting12{:,:};
sitting13 = sitting13{:,:};
sitting14 = sitting14{:,:};
sitting15 = sitting15{:,:};
%-------------------------------------------------standing-----------------
---------------------------------%
standing1 = standing1{:,:};
standing2 = standing2{:,:};
standing3 = standing3{:,:};
standing4 = standing4{:,:};
standing5 = standing5{:,:};
standing6 = standing6{:,:};
standing7 = standing7{:,:};
standing8 = standing8{:,:};
standing9 = standing9{:,:};
standing10 = standing10{:,:};
standing11 = standing11{:,:};
standing12 = standing12{:,:};
standing13 = standing13{:,:};
standing14 = standing14{:,:};
standing15 = standing15{:,:};
 %-------------------------------------------------walking----------------
---------------------------------%
walking1 = walking1{:,:};
walking2 = walking2{:,:};
walking3 = walking3{:,:};
walking4 = walking4{:,:};
walking5 = walking5{:,:};
walking6 = walking6{:,:};
walking7 = walking7{:,:};
walking8 = walking8{:,:};
walking9 = walking9{:,:};
walking10 = walking10{:,:};
walking11 = walking11{:,:};
walking12 = walking12{:,:};
walking13 = walking13{:,:};
walking14 = walking14{:,:};
walking15 = walking15{:,:};

%---------------------------------Fusionner les matrices par activité----
-------------------------------------%

bending1=[bending11;bending12 ;bending13 ;bending14;bending15
;bending16;bending17];
bending2=[bending21;bending22;bending23;bending24 ;bending25;bending26];
cycling=[cycling1;cycling2;cycling3;cycling4;cycling5
;cycling6;cycling7;cycling8;cycling9;cycling10 ;cycling11
;cycling12;cycling13 ;cycling14 ;cycling15 ;];
lying=[lying1;lying2;lying3;lying4;lying5;lying6 ;lying7 ;lying8;lying9
;lying10;lying11 ;lying12 ;lying13 ;lying14 ;lying15 ];
sitting=[sitting1 ;sitting2 ;sitting3;sitting4 ;sitting5 ;sitting6
;sitting7;sitting8 ;sitting9 ;sitting10 ;sitting11 ;sitting12 ;sitting13
;sitting14 ;sitting15 ];
standing=[standing1;standing2 ;standing3 ;standing4 ;standing5
;standing6;standing7 ;standing8;standing9 ;standing10 ;standing11
;standing12 ;standing13 ;standing14 ;standing15 ];
walking =[walking1;walking2 ;walking3 ;walking4;walking5 ;walking6
;walking7 ;walking8;walking9 ;walking10;walking11 ;walking12
;walking13;walking14 ;walking15 ];
```

```
%----------------------------------Suppression de la colonne de temps------
----------------------------------%

bending1(:,1) = [];
bending2(:,1) = [];
cycling(:,1) = [];
lying(:,1) = [];
sitting(:,1) = [];
standing(:,1) = [];
walking (:,1) = [];

%---------------------Remplacer la colonne des résultats par des numeros
des classes-------------------------%

bending1(:,7) = 6;
bending2(:,7) = 7;
cycling(:,7) = 5;
lying(:,7) = 3;
sitting(:,7) = 2;
standing(:,7) = 1;
walking (:,7) = 4;

%-------------------------------------------Rassemblage-------------------
----------------------------------%
dataa = [bending1;bending2;cycling;lying;sitting;standing;walking];

%-------------------------------Suppression du duplication des instances--
----------------------------------%
dataC=unique(sort(dataa,7), 'rows');

%---------------------------------------Selection des données-------------
----------------------------------%
[m,n] = size(dataC) ;
%60% pour le test
P = 0.60 ;
% 20% 20% Test & Validation
k=0.80;
idx = randperm(m);
Training = dataC(idx(1:round(P*m)),:) ;
Testing = dataC(idx(round(P*m)+1:round(k*m)),:) ;
validation=dataC(idx(round(k*m)+1:end),:) ;
```

b. Logistic Regression

1 .main

```
%-------------------------------------------------------------------------
-------------------------------------------------------------
        %=============================================optimisation des
parameteres avec diférents valeur de
lambda=========================================
fprintf('\n Optimazing parameters, testing differents lambda...')


lambda_test = [ 0.1,0.09,0.08,0.07,0.06,0.05,0.04,0.03,0.02,0.01,0];
plotBiasVsVariance15(x_train, y_train, x_test, y_test, iterations,
num_labels, lambda_test);
%-------------------------------------------------------------------------
-------------------------------------------------------------
        %=============================================1- training
logistique regression fmincg=========================================
```

```
fprintf('\nTraining One-vs-All Logistic Regression...\n')

lambda = 20;
iterations=1000;
[all_theta,j_h] = oneVsAll_team15(x_train,y_train, num_labels,iterations,
lambda);
% Plot the convergence graph
plotCost_team15(j_h,'fmincg');

fprintf('Program paused. Press enter to continue.\n');
pause;

%************************************ predict value one vs
all*************************************************
%1- training
pred = predictOneVsAll_team15(all_theta, x_train);

%2- validation
predvalid = predictOneVsAll_team15(all_theta, x_valid);

%3- testing
predtest = predictOneVsAll_team15(all_theta, x_test);
plotDecisionBoundary_team15(all_theta, x_test, y_test);

%*****************************************Calcule de Score
("Accurancy")*****************************************

fprintf('\nvalidation Set Accuracy: %f\n', mean(double(predvalid ==
y_valid)) * 100);
fprintf('\nTesting Set Accuracy: %f', mean(double(predtest == y_test)) *
100);
```

oneVsAll_team15

```
function [all_theta,J_history] = oneVsAll_team15(X, y,
num_labels,iterations, lambda)
% Some useful variables
m = size(X, 1);
n = size(X, 2);

% You need to return the following variables correctly
all_theta = zeros(num_labels, n + 1);% n+1 le 1 correspond a la collone de
1 ajouté

% Add ones to the X data matrix
X = [ones(m, 1) X];
J_history = zeros(iterations, num_labels);

for c=1:num_labels
initial_theta=zeros(n+1,1);
%     % Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', iterations);
[theta,J]= fmincg (@(t)(lrCostFunction_team15(t, X, (y == c),
lambda)),initial_theta, options);
all_theta(c,:)=theta(:);
```

```
    % history for the iteration.
    if (length(J)<iterations)
        aux = zeros(iterations - length(J),1);
        aux(:) = J(length(J));
        J = [J;aux];
        J_history(:, c) = J;
    else
        J_history(:, c) = J;
    end
end


end
```

3-lrCostFonctionteam15

```
function [J, grad] = lrCostFunction_team15(theta, X, y, lambda)

% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = 0;
grad = zeros(size(theta));


h=sigmoid_team15(X*theta);
reg_term_0=theta.^2;
reg_term_0(1)=0;
J=sum(-y.*log(h)-(1-y).*log(1-h))*(1/m)+sum(reg_term_0)*((lambda)/(2*m));
reg_term=theta*(lambda/m);
reg_term(1)=0;
grad=(X'*(h-y))*(1/m)+reg_term;

end
```

function plotBiasVsVariance_team15

```
function plotBiasVsVariance_team15( X_train, y_train, X_test, y_test,
alpha, iterations, num_labels, lambda, optFunct)

m = size(X_train, 1);
n= size(X_test, 1);
J = zeros(length(lambda),3);
i = 1;

for l = lambda
    if optFunct == 1
        [all_w, j_h] = GradientDescen_team15t(X_train, y_train, alpha,
iterations, num_labels, l);
    elseif optFunct == 2
        [all_w, j_h] = quasiNewton_team15(X_train, y_train, num_labels, l,
iterations);
    else
        [all_w, j_h] = fmincgFunction_team15(X_train, y_train, num_labels,
l, iterations);
    end
```

```
X = [ones(m, 1), X_train];
disp(X)
X_tes = [ones(n, 1), X_test];

    for k = 1:size(all_w, 1)
        [J_train(k), grad1] = lrCostFunction_team15(all_w(k,:)', X,
(y_train == k),0.1);
        [J_test(k), grad2] = lrCostFunction_team15(all_w(k,:)', X_tes,
(y_test == k),0.1);
    end
    J(i,:) = [l, mean(J_train), mean(J_test)];
    i = i+1;
end

figure
plot(J_train(:,1),J_train(:,2),J_train(:,1),J_train(:,3), 'LineWidth', 2)


ylabel('Cost');
xlabel('Lambda');
legend('Train','Test', 'Location','northwest')
title('Bias vs Varance for Lambda Values')

end
```

plotCost_team15

```
 function plotCost_team15(j_h, vtitle)

% Plot Data
num_labels = size(j_h,2);

if num_labels > 3
    figure('position', [100, 100, 1400, 800])
    for j = 1:2
        for k = 1:num_labels
            subplot(3,4,k)
            plot(1:numel(j_h(:,k)), j_h(:,k), '-r', 'LineWidth', 2);
            xlabel('Number of iterations');
            ylabel('Cost');
            title(k);
            title([vtitle, ' K =',num2str(k), ' Min =',
num2str(j_h(end,k))]);
        end
    end
else
    figure('position', [100, 100, 1200, 300])
    for k = 1:num_labels
        subplot(1,num_labels,k)
        plot(1:numel(j_h(:,k)), j_h(:,k), '-r', 'LineWidth', 2);
        xlabel('Number of iterations');
        ylabel('Cost');
        title([vtitle, ' K =',num2str(k), ' Min =', num2str(j_h(end,k))]);
    end
end

figure()
for k = 1:num_labels
    m = mean(j_h(:,2:end),2);
    plot(1:numel(j_h(:,k)), m, '-r', 'LineWidth', 2);
```

```
    xlabel('Number of iterations');
    ylabel('Cost');
end
title(['Total ', vtitle, '. Min = ', num2str(m(end))]);

end
```

plotDecisionBoundary_team15

```
function plotDecisionBoundary_team15(all_theta, x, Y)
%Plot the data with the aditional decision boundary find whit the weight
%vector w.
 %REDUCTION DE NOMBRE D'instance pour le plot
P = 0.30 ;
dataC=[x Y];
[m,n] = size(dataC);
idx = randperm(m);
Train = dataC(idx(1:round(P*m)),:) ;
X=Train(:,1:6);
y=Train(:,7);

m = size(X, 1);
num_labels = size(all_theta,1);
X = [ones(m, 1) X];

figure('position', [100, 100, 1200, 300])
for k = 1:num_labels
    %subplot(2,4,k)
    subplot(1,num_labels,k)
    plotDataA_team15(X(:,2:3), y, k);
    axis([2 40 0.5 7.5])
    hold on

    % Only need 2 points to define a line, so choose two endpoints
    plot_x = [min(X(:,2))-2,  max(X(:,2))+2];

    % Calculate the decision boundary line
    plot_y = (-1./all_theta(k,3)).*(all_theta(k,2).*plot_x +
all_theta(k,1));

    % Plot, and adjust axes for better viewing
    plot(plot_x, plot_y);
    title(sprintf('Decision Boundary %d',k));
    hold off
end

end
```

predictOneVsAll_team15

```
function p = predictOneVsAll_team15(all_theta, X)

m = size(X, 1);
num_labels = size(all_theta, 1);


% Add ones to the X data matrix
X = [ones(m, 1) X];
A=sigmoid_team15(all_theta*X');
```

```
[maxVal maxInd] = max(A);
p=maxInd';


end
```

sigmoid_team15

```
function g = sigmoid_team15(z)
      g = 1 ./ (1 + exp(-z));
end
```

Regression logistique régularisé non optimisé

main

```
%definir les label et les features
input_layer_size  = 6;
num_labels = 7;

%-------------------------------------------------------------------------
-------------------------------------------------------
        %==============================================optimisation des
parameteres avec diférents valeur de
lambda=========================================
fprintf('\n Optimazing parameters, testing differents lambda...')

iterations = 60;           % Number of iterations gradient descent
alpha = 0.005;
lambda_test = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100];
plotBiasVsVariance_team15(x_train, y_train, x_test, y_test, alpha,
iterations, num_labels, lambda_test);


%-------------------------------------------------------------------------
-------------------------------------------------------
        %==============================================1- training
logistique regression gradiant
descent=========================================

fprintf('\nTraining One-vs-All Logistic Regression...\n')
lambda = 0;                % Regularization parameter

[all_theta, j_h] = GradientDescent_team15(x_train, y_train, alpha,
iterations, num_labels, lambda);
% Plot the convergence graph
plotCost_team15(j_h,'gradiant descent');
plotDecisionBoundary_team15(all_theta, x_train, y_train);

fprintf('Program paused. Press enter to continue.\n');
pause;

%************************************ predict value one vs
all*************************************************

%1- testing
predtest = predictOneVsAll_team15(all_theta, x_test);
%2- validation
predvalid = predictOneVsAll_team15(all_theta, x_valid);
```

```
%*****************************************Calcule de Score
("Accurancy")*****************************************
```

GradientDescent team15

```
function [all_w, J_history] = GradientDescent_team15(X, y, alpha,
num_iters, num_labels, lambda)

% History of the cost function in each iteration
J_history = zeros(num_iters, num_labels);

% Some useful variables
n = size(X, 2);

% Variable of all optimal weight found for each class or label
all_w = zeros(num_labels, n + 1);

for c = 1:num_labels
    fprintf('\nTraining k: %f', c);
    w = all_w(c, :)';
    for iter = 1:num_iters
        [J, grad] = lrCostFunction_team15(w, X, (y == c), lambda);
        w = (w - (alpha*grad));
        J_history(iter, c) = J;
    end
    all_w(c, :) = w';
end
end
```

plotBiasVsVariance team15

```
function plotBiasVsVariance_team15( X_train, y_train, X_test, y_test,
alpha, iterations, num_labels, lambda)
% Funtio to test differents values of lambda (regulariazation) and mesure
% the cost.

m = size(X_train, 1);
l = size(X_test, 1);
J = zeros(length(lambda),3);
i = 1;

for l = lambda
        [all_w, j_h] = GradientDescent_team15(X_train, y_train, alpha,
iterations, num_labels, l);
    for k = 1:size(all_w, 1)
        [J_train(k), grad1] = lrCostFunction_team15(all_w(k,:)', X_train,
(y_train == k), 0);
        [J_test(k), grad2] = lrCostFunction_team15(all_w(k,:)', X_test,
(y_test == k), 0);
    end
    J(i,:) = [l, mean(J_train), mean(J_test)];
    i = i+1;
end

figure
plot(J(:,1),J(:,2),J(:,1),J(:,3), 'LineWidth', 2)
ylabel('Cost');
xlabel('Lambda');
legend('Train','Test', 'Location','northwest')
```

```
title('Bias vs Varance for Lambda Values')
end
```

plotCost_team15

```
function plotCost_team15(j_h, vtitle)

% Plot Data
num_labels = size(j_h,2);

if num_labels > 3
    figure('position', [100, 100, 1400, 800])
    for j = 1:2
        for k = 1:num_labels
            subplot(3,4,k)
            plot(1:numel(j_h(:,k)), j_h(:,k), '-r', 'LineWidth', 2);
            xlabel('Number of iterations');
            ylabel('Cost');
            title(k);
            title([vtitle, ' K =',num2str(k), ' Min =',
num2str(j_h(end,k))]);
        end
    end
else
    figure('position', [100, 100, 1200, 300])
    for k = 1:num_labels
        subplot(1,num_labels,k)
        plot(1:numel(j_h(:,k)), j_h(:,k), '-r', 'LineWidth', 2);
        xlabel('Number of iterations');
        ylabel('Cost');
        title([vtitle, ' K =',num2str(k), ' Min =', num2str(j_h(end,k))]);
    end
end

figure()
for k = 1:num_labels
    m = mean(j_h(:,2:end),2);
    plot(1:numel(j_h(:,k)), m, '-r', 'LineWidth', 2);
    xlabel('Number of iterations');
    ylabel('Cost');
end
title(['Total ', vtitle, '. Min = ', num2str(m(end))]);

end
```

plotData_team15

```
function plotData_team15(data)

%   and o for the negative examples. X is assumed to be a Mx2 matrix.

m = size(data, 1);
y=data(:,7);

% Create New Figure
figure; hold on;

% Find Indices of Positive and Negative Examples
a = find(y==1);
b = find(y == 2);
c = find(y == 3);
d = find(y == 4);
e = find(y == 5);
f = find(y == 6);
```

```
g = find(y == 7);


% Plot Examples
plot(data(a,1 ), data(a, 4), 'k+', 'LineWidth', 2, 'MarkerSize', 7);
plot(data(b, 1), data(b, 4), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize',
7);
plot(data(c, 1), data(c, 4), 'kx', 'MarkerFaceColor', 'b', 'MarkerSize',
7);
plot(data(d, 1), data(d, 4), 'ks', 'MarkerFaceColor', 'g', 'MarkerSize',
7);
plot(data(e, 1), data(e, 4), 'kd', 'MarkerFaceColor', 'r', 'MarkerSize',
7);
plot(data(f, 1), data(f, 4), 'k-', 'MarkerFaceColor', 'y', 'MarkerSize',
7);
plot(data(g, 1), data(g, 4), 'k*', 'MarkerFaceColor', 'r', 'MarkerSize',
7);
legend('bending 1', 'bending
2','cycling','lying','sitting','standing','walking');
title('ARMe');
xlabel('features');
ylabel('Activity');

hold off;

end
```

plotDataA_team15

```
function plotDataA_team15(X, y, a)
%PLOTDATA Plots the data points X and y into a new figure
%   PLOTDATA(x,y) plots the data points with + for the positive examples
%   and o for the negative examples. X is assumed to be a Mx2 matrix.

% Create New Figure
hold on;

% Find Indices of Positive and Negative Examples
bending1 = find(y==1); bending2 = find(y == 2); cycling = find(y == 3);
lying = find(y==4); sitting = find(y == 5); standing = find(y == 6);
walking = find(y == 7);

% Plot Examples
if (a==1)||(a==0)
    plot(X(bending1, 1), X(bending1, 2),'k^', 'MarkerFaceColor', 'b', ...
    'MarkerSize', 7);
else
    plot(X(bending1, 1), X(bending1, 2),'k^', 'MarkerFaceColor', [0.5 0.5
0.5], ...
    'MarkerSize', 7);
end
if (a==2)||(a==0)
    plot(X(bending2, 1), X(bending2, 2), 'kd', 'MarkerFaceColor', 'y', ...
    'MarkerSize', 7);
else
    plot(X(bending2, 1), X(bending2, 2), 'kd', 'MarkerFaceColor', [0.5 0.5
0.5], ...
    'MarkerSize', 7);
end
if (a==3)||(a==0)
```

```
    plot(X(cycling, 1), X(cycling, 2), 'ko', 'MarkerFaceColor', 'g', ...
    'MarkerSize', 7);
else
    plot(X(cycling, 1), X(cycling, 2), 'ko', 'MarkerFaceColor', [0.5 0.5
0.5], ...
    'MarkerSize', 7);
end

if (a==4)||(a==0)
    plot(X(lying, 1), X(lying, 2), 'kd', 'MarkerFaceColor', 'r', ...
    'MarkerSize', 7);
else
    plot(X(lying, 1), X(lying, 2), 'kd', 'MarkerFaceColor', [0.5 0.5 0.5],
...
    'MarkerSize', 7);
end

if (a==5)||(a==0)
    plot(X(sitting, 1), X(sitting, 2), 'kx', 'MarkerFaceColor', 'r', ...
    'MarkerSize', 7);
else
    plot(X(sitting, 1), X(sitting, 2), 'kx', 'MarkerFaceColor', [0.5 0.5
0.5], ...
    'MarkerSize', 7);
end
if (a==6)||(a==0)
    plot(X(standing, 1), X(standing, 2), 'k*', 'MarkerFaceColor', 'g', ...
    'MarkerSize', 7);
else
    plot(X(standing, 1), X(standing, 2), 'k*', 'MarkerFaceColor', [0.5 0.5
0.5], ...
    'MarkerSize', 7);
end
if (a==7)||(a==0)
    plot(X(walking, 1), X(walking, 2), 'ks', 'MarkerFaceColor', 'g', ...
    'MarkerSize', 7);
else
    plot(X(walking, 1), X(walking, 2), 'ks', 'MarkerFaceColor', [0.5 0.5
0.5], ...
    'MarkerSize', 7);
end

ylabel('var_rss12');                % Set the y-axis label
xlabel('avg_rss12');                 % Set the X-axis label
legend('bending1', 'bending2', 'cycling',
'lying','sitting','standing','walking');


hold off;

end
```

plotDecisionBoundary_team15

```
function plotDecisionBoundary_team15(all_theta, x, Y)
%Plot the data with the aditional decision boundary find whit the weight
%vector w.
 %REDUCTION DE NOMBRE D'instance pour le plot
P = 0.30 ;
dataC=[x Y];
[m,n] = size(dataC);
idx = randperm(m);
```

```
Train = dataC(idx(1:round(P*m)),:) ;
X=Train(:,1:6);
y=Train(:,7);

m = size(X, 1);
num_labels = size(all_theta,1);
X = [ones(m, 1) X];

figure('position', [100, 100, 1200, 300])
for k = 1:num_labels
    %subplot(2,4,k)
    subplot(1,num_labels,k)
    plotDataA(X(:,2:3), y, k);%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    axis([2 40 0.5 7.5])
    hold on

    % Only need 2 points to define a line, so choose two endpoints
    plot_x = [min(X(:,2))-2,  max(X(:,2))+2];

    % Calculate the decision boundary line
    plot_y = (-1./all_theta(k,3)).*(all_theta(k,2).*plot_x +
all_theta(k,1));

    % Plot, and adjust axes for better viewing
    plot(plot_x, plot_y);
    title(sprintf('Decision Boundary %d',k));
    hold off
end

end
```

predictOneVsAll_team15

```
function p = predictOneVsAll_team15(all_theta, X)

m = size(X, 1);
num_labels = size(all_theta, 1);


% Add ones to the X data matrix
X = [ones(m, 1) X];
```

sigmoid

```
A=sigmoid(all_theta*X');
[maxVal maxInd] = max(A);
p=maxInd';

function g = sigmoid_team15(z)
%SIGMOID Compute sigmoid functoon
%   J = SIGMOID(z) computes the sigmoid of z.

g = 1 ./ (1 + exp(-z));

end


end
```

c. Decision Tree

```
%Mettre les X et les résultats Y dans des matrices séparées
X = Training(:,1:6);
Y = Training(:,7);

%Appel de fonction prédifinie et récupérér les résultats dans la variable M
M =fitctree(X,Y);

%Affichage de l'arbre
view(M,'mode','graph');

%Affichage de résultats finales
view(M);

%Calculs de feature le plus important
imp = predictorImportance(M);

%Affichage de feature le plus important pas histogramme
figure;
bar(imp);
title('Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = M.PredictorNames;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
```

Validation

```
%Test de Decision Tree
Correct = 0;
Incorrect = 0;
for i=1:size(Testing,1)
   result = predict(M,Testing(i,1:6));
   if(result==Testing(i,7))
      Correct = Correct+1;
   else
      Incorrect = Incorrect+1;
   end
end
%Calcul de prédiction
Predection = Correct * 100 / size(Testing,1);

fprintf('Prédiction correcte = ');
disp(Correct);
fprintf('\nPrédiction incorrecte = ');
disp(Incorrect);
%Affichage de % de prédiction
disp('Prédiction : %');
disp(Predection);
```

d. Neural Network

1. 1 Hidden layer

Main

```
%-----------------Réseau de neurones - 1 couche cachée---------------------
--------------------------------------
Nbr_result=7;
Nbr_Hidden_Units = int8((size(Training,2)-1)*3/4);
```

```
%tableau de la couche caché
Nbr_Hidden_Layer = 1;
Hidden_Layers = zeros(Nbr_Hidden_Units,Nbr_Hidden_Layer);

%Matrices des resultats (Output))
Y = zeros(size(Training,1),Nbr_result);
%Mettre les resultats à 0 et 1
for i=1:size(Y,1)
    Y(i,Training(i,7))=1;
end

X = Training(:,1:6);

Nbr_Feature = size(Training,2)-1;

%Valeur d'epsilon - a ne pas depasser par theta lors d'initialisation
Init_Epsilon =sqrt(2/(7+5));


%les poids relative à la couche input
WeightInput = rand(Nbr_Hidden_Units,Nbr_Feature+1)*Init_Epsilon;

%les poids relative à les couches finale
WeightFinal = rand(Nbr_result,Nbr_Hidden_Units+1)*Init_Epsilon;


%Nombre d'itération max
Nbr_Iteration = 200;
Alpha = 0.00001;
Lambda = 0.01;
[Costvect,Result,WeightInput,WeightFinal] =
NeuralNetwork_Team15(X,Y,WeightInput,WeightFinal,Hidden_Layers,Nbr_Iteratio
n,Lambda,Alpha);

figure;
%Affichage de la fonction J
plot(Costvect)
%Mettre une clé pour le graphe (le nom de la courbe)
legend('Cost function');
%Mettre un nom pour les deux axes
xlabel('Iteration');
ylabel('J(Theta)');
%Mettre un titre pour le graphe
title('Cost function');
```

NeuralNetwork

```
function [Costvect,Result,WeightInput,WeightFinal] =
NeuralNetwork_Team15(X,Y,WeightInput,WeightFinal,Hidden_Layers,Nbr_Iteratio
n,Lambda,Alpha)
    Costvect=[];
    DeltaD = zeros(size(Y,2),size(Hidden_Layers,1)+1);
    DeltaF = zeros(size(Hidden_Layers,1),size(X,2)+1);
    %Repétition jusqu'au un nombre max d'itération
    for i=1:Nbr_Iteration
        %Parcourir tous les echantillons
        for z=1:size(X,1)
            %ForwardPropagation
            Input = [1,X(z,:)];
```

```
[Hidden_Layers(:,1)]=ForwardPropagation_Team15(Input',WeightInput);

            Input = [1;Hidden_Layers(:,1)];
            [Result]=ForwardPropagation_Team15(Input,WeightFinal);

            %BackPropagation

[DeltaD,DeltaF]=BackPropagation_Team15(X(z,:),Y(z,:),Result,WeightFinal,Hid
den_Layers,DeltaD,DeltaF);

        end

        %D de derniere couche
        DD=(DeltaD/size(X,1));
        for k=1:size(DD,1)
            for j=2:size(DD,2)
                DD(k,j) = DD(k,j)+Lambda*WeightFinal(k,j) ;
            end
        end
        %D des inputs
        DF=(DeltaF)/size(X,1);
        for k=1:size(DF,1)
            for j=2:size(DF,2)
                DF(k,j) = DF(k,j)+Lambda*WeightInput(k,j) ;
            end
        end

        WI=WeightInput-Alpha*DF;
        WeightInput= WI;
        WF=WeightFinal-Alpha*DD;
        WeightFinal = WF;

        %Cost

[j]=Cost_Team15(X,Y,Result,WeightInput,WeightFinal,Lambda,Hidden_Layers);
        Costvect=[Costvect;j];
        disp(i);
    end
```

ForwardPropagation

```
function [Result] = ForwardPropagation_Team15 (X, WeightInput)
    z = WeightInput*X;
    Result= Sigmoid_Team15(z);
end
```

BackPropagation

```
function
[DeltaD,DeltaF]=BackPropagation_Team15(X,Y,Result,WeightFinal,Hidden_Layers
,DeltaD,DeltaF)
    ErrorF= Result-Y';
    Input = [1;Hidden_Layers(:,1)];
    Error(:,1) = (WeightFinal'*(ErrorF)).*(DerivSigmoid_Team15(Input));
    DeltaD=DeltaD+ErrorF*Input';

    Err = Error(2:size(Error,1),1);
    Input = [1,X];
    DeltaF=DeltaF+Err*Input;
end
```

Sigmoid

```
function S=Sigmoid_Team15(X)
    for i=1:size(X,1)
        S(i,1)=1/(1+exp(-X(i,1)));
    end
end
```

DerivSigmoid

```
function S=DerivSigmoid_Team15(X)
    S = X.*(1-X);
end
```

Cost

```
function
[j]=Cost_Team15(X,Y,Output,WeightInput,WeightFinal,Lambda,Hidden_Layers)
    j=0;
    for i=1:size(X,1)
        %ForwardPropagation
        Input = [1,X(i,:)];
        [Hidden_Layers(:,1)]=ForwardPropagation_Team15(Input',WeightInput);

        Input = [1;Hidden_Layers(:,1)];
        [Output]=ForwardPropagation_Team15(Input,WeightFinal);

        for k=1:size(Y,2)
            j=j+(Y(i,k)*log(Output(k,1)))+((1-Y(i,k))*log(1-
(Output(k,1))));
        end
    end
    j = (-1/size(X,1))*j;
    %Regularisation
    Regul =0;
    for i=1:size(WeightFinal,1)
        for j=2:size(WeightFinal,2)
            Regul = Regul +(WeightFinal(i,j)^2);
        end
    end
    j=j+(Regul*(Lambda/(2*size(X,1))));
end
```

Test

```
%Test de Decision Tree
Correct = 0
Incorrect = 0
for i=1:size(Testing,1)
    XTest= Training(i,1:6);
    Input = [1,XTest];
    [Hidden_Layers(:,1)]=ForwardPropagation(Input',WeightInput);
    Input = [1;Hidden_Layers(:,1)];
    [Result]=ForwardPropagation(Input,WeightFinal);
    [Max,I]=max(Result);
    if(I==Testing(i,7))
      Correct = Correct+1;
    else
      Incorrect = Incorrect+1;
    end
end
%Calcul de prédiction
Predection = Correct * 100 / size(Testing,1);

fprintf('Prédiction correcte = ');
```

```
disp(Correct);
fprintf('\nPrédiction incorrecte = ');
disp(Incorrect);
%Affichage de % de prédiction
disp('Prédiction : %');
disp(Predection);
```

2. 2 Hidden layers

Main

```
Nbr_result=7;
Nbr_Hidden_Units = int8((size(Training,2)-1)*3/4);

%tableau de la couche caché
Nbr_Hidden_Layer = 2;
Hidden_Layers = zeros(Nbr_Hidden_Units,Nbr_Hidden_Layer);

%Matrices des resultats (Output))
Y = zeros(size(Training,1),Nbr_result);
%Mettre les resultats à 0 et 1
for i=1:size(Y,1)
    Y(i,Training(i,7))=1;
end

X = Training(:,1:6);

Nbr_Feature = size(Training,2)-1;

%Valeur d'epsilon - a ne pas depasser par theta lors d'initialisation
Init_Epsilon =sqrt(2/(7+5));

%les poids relative à la couche input
WeightInput = rand(Nbr_Hidden_Units,Nbr_Feature+1)*Init_Epsilon;

%les poids relative à les couches cachées
Weight = rand(Nbr_Hidden_Units,Nbr_Hidden_Units+1)*Init_Epsilon;

%les poids relative à les couches finale
WeightFinal = rand(Nbr_result,Nbr_Hidden_Units+1)*Init_Epsilon;


%Nombre d'itération max
Nbr_Iteration = 200;
Alpha = 0.00001;
Lambda = 0.01;
[Costvect,Result,WeightInput,Weight,WeightFinal] =
NeuralNetwork_Team15(X,Y,WeightInput,Weight,WeightFinal,Hidden_Layers,Nbr_I
teration,Lambda,Alpha);

figure;
%Affichage de la fonction J
plot(Costvect)
%Mettre une clé pour le graphe (le nom de la courbe)
legend('Cost function');
%Mettre un nom pour les deux axes
xlabel('Iteration');
ylabel('J(Theta)');
%Mettre un titre pour le graphe
title('Cost function');
```

NeuralNetwork

```
function [Costvect,Result,WeightInput,Weight,WeightFinal] =
NeuralNetwork_Team15(X,Y,WeightInput,Weight,WeightFinal,Hidden_Layers,Nbr_I
teration,Lambda,Alpha)
    Costvect=[];
    Delta = zeros(size(Hidden_Layers,1),size(Hidden_Layers,1)+1);
    DeltaD = zeros(size(Y,2),size(Hidden_Layers,1)+1);
    DeltaF = zeros(size(Hidden_Layers,1),size(X,2)+1);
    %Repétition jusqu'au un nombre max d'itération
    for i=1:Nbr_Iteration
        %Parcourir tous les echantillons
        for z=1:size(X,1)
            %ForwardPropagation
            Input = [1,X(z,:)];

[Hidden_Layers(:,1)]=ForwardPropagation_Team15(Input',WeightInput);

            Input = [1;Hidden_Layers(:,1)];
            [Hidden_Layers(:,2)]=ForwardPropagation_Team15(Input,Weight);

            Input = [1;Hidden_Layers(:,2)];
            [Result]=ForwardPropagation_Team15(Input,WeightFinal);

            %BackPropagation

[Delta,DeltaD,DeltaF]=BackPropagation_Team15(X(z,:),Y(z,:),Result,Weight,We
ightFinal,Hidden_Layers,Delta,DeltaD,DeltaF);
        end
        %D de derniere couche
        DD=(DeltaD/size(X,1))+ Lambda*WeightFinal;
        %D des hidden layers
        D=(Delta/size(X,1))+Lambda * Weight;
        %D des inputs
        DF=(DeltaF)/size(X,1);


        %D de derniere couche
        DD=(DeltaD/size(X,1));
        for k=1:size(DD,1)
            for j=2:size(DD,2)
                DD(k,j) = DD(k,j)+Lambda*WeightFinal(k,j) ;
            end
        end
        %D des hidden layers
        D=(Delta)/size(X,1);
        for k=1:size(D,1)
            for j=2:size(D,2)
                D(k,j) = D(k,j)+Lambda*Weight(k,j) ;
            end
        end
        %D des inputs
        DF=(DeltaF)/size(X,1);
        for k=1:size(DF,1)
            for j=2:size(DF,2)
                DF(k,j) = DF(k,j)+Lambda*WeightInput(k,j) ;
            end
        end

        WI=WeightInput-Alpha*DF;
```

```
        WeightInput= WI;
        W=Weight-Alpha*D;
        Weight = W;
        WF=WeightFinal-Alpha*DD;
        WeightFinal = WF;

        %Cost
        [j]=Cost_Team15(X,Y,Result,Weight,WeightInput,WeightFinal,Lambda);

        Costvect=[Costvect;j];
        disp(i);
    end
```

ForwardPropagation

```
function [Result] = ForwardPropagation_Team15 (X, WeightInput)
    z = WeightInput*X;
    Result= Sigmoid_Team15(z);
end
```

BackPropagation

```
function
[Delta,DeltaD,DeltaF]=BackPropagation_Team15(X,Y,Result,Weight,WeightFinal,
Hidden_Layers,Delta,DeltaD,DeltaF)
    ErrorF= Result-Y';
    Input = [1;Hidden_Layers(:,2)];
    Error(:,1) = (WeightFinal'*(ErrorF)).*(DerivSigmoid_Team15(Input));
    DeltaD=DeltaD+ErrorF*Input';
    j=2;

    Input = [1;Hidden_Layers(:,1)];
    Err = Error(2:size(Error,1),j-1);
    Error(:,j) = (Weight'*Err).*DerivSigmoid_Team15(Input);
    Delta(:,:,1)=Delta(:,:,1)+Err*Input';
    j=j+1;

    Err = Error(2:size(Error,1),j-1);
    Input = [1,X];
    DeltaF=DeltaF+Err*Input;
end
```

Sigmoid

```
function S=Sigmoid_Team15(X)
    for i=1:size(X,1)
        S(i,1)=1/(1+exp(-X(i,1)));
    end
end
```

DerivSigmoid

```
function S=DerivSigmoid_Team15(X)
    S=X.*(ones(size(X,1),size(X,2))-X);
end
```

Cost

```
function [j]=Cost_Team15(X,Y,Output,Weight,WeightInput,WeightFinal,Lambda)
    j=0;
    for i=1:size(X,1)
        Input = [1,X(i,:)];
        [Hidden_Layers(:,1)]=ForwardPropagation_Team15(Input',WeightInput);

        Input = [1;Hidden_Layers(:,1)];
        [Hidden_Layers(:,2)]=ForwardPropagation_Team15(Input,Weight);
```

```
        Input = [1;Hidden_Layers(:,2)];
        [Output]=ForwardPropagation_Team15(Input,WeightFinal);
        for k=1:size(Y,2)
            j=j+(Y(i,k)*log((Output(k,1))))+((1-Y(i,k))*log(1-
(Output(k,1))));
        end
    end
    j = (-1/size(X,1))*j;
    %Regularisation
    Regul =0;
    for j=2:size(Weight,2)
        for k=1:size(Weight,1)
            Regul = Regul +(Weight(k,j)^2);
        end
    end
    for i=1:size(WeightFinal,1)
        for j=2:size(WeightFinal,2)
            Regul = Regul +(WeightFinal(i,j)^2);
        end
    end
    j=j+(Regul*(Lambda/(2*size(X,1))));
end
```

Test

```
%Test de Decision Tree
Correct = 0
Incorrect = 0
for i=1:size(Testing,1)
    XTest= Training(i,1:6);
    Input = [1,XTest];
    [Hidden_Layers(:,1)]=ForwardPropagation(Input',WeightInput);

    Input = [1;Hidden_Layers(:,1)];
    [Hidden_Layers(:,2)]=ForwardPropagation(Input,Weight);

    Input = [1;Hidden_Layers(:,2)];
    [Result]=ForwardPropagation(Input,WeightFinal);
    [Max,I]=max(Result);
    if(I==Testing(i,7))
      Correct = Correct+1;
    else
      Incorrect = Incorrect+1;
    end
end
%Calcul de prédiction
Predection = Correct * 100 / size(Testing,1);

fprintf('Prédiction correcte = ');
disp(Correct);
fprintf('\nPrédiction incorrecte = ');
disp(Incorrect);
%Affichage de % de prédiction
disp('Prédiction : %');
disp(Predection);
```

e. SVM

```
clear;
clc;
make;
load x_train.txt;
load y_train.txt;
load x_test.txt;
load y_test.txt;
load x_valid.txt;
load y_valid.txt;


model = svmtrain(y_train, x_train, '-s ---SVC -t --linear -c 1 -g  0.07-v 5
-b -h -d 4');
[predicted_validation] = svmpredict(y_valid, x_valid,model);
[predicted_test] = svmpredict(y_test, x_test, model);
```

ANNEXE B:

a. Decision Tree :

1. Les résultats de l'affichage de l'arbre de décision ont été les suivants :

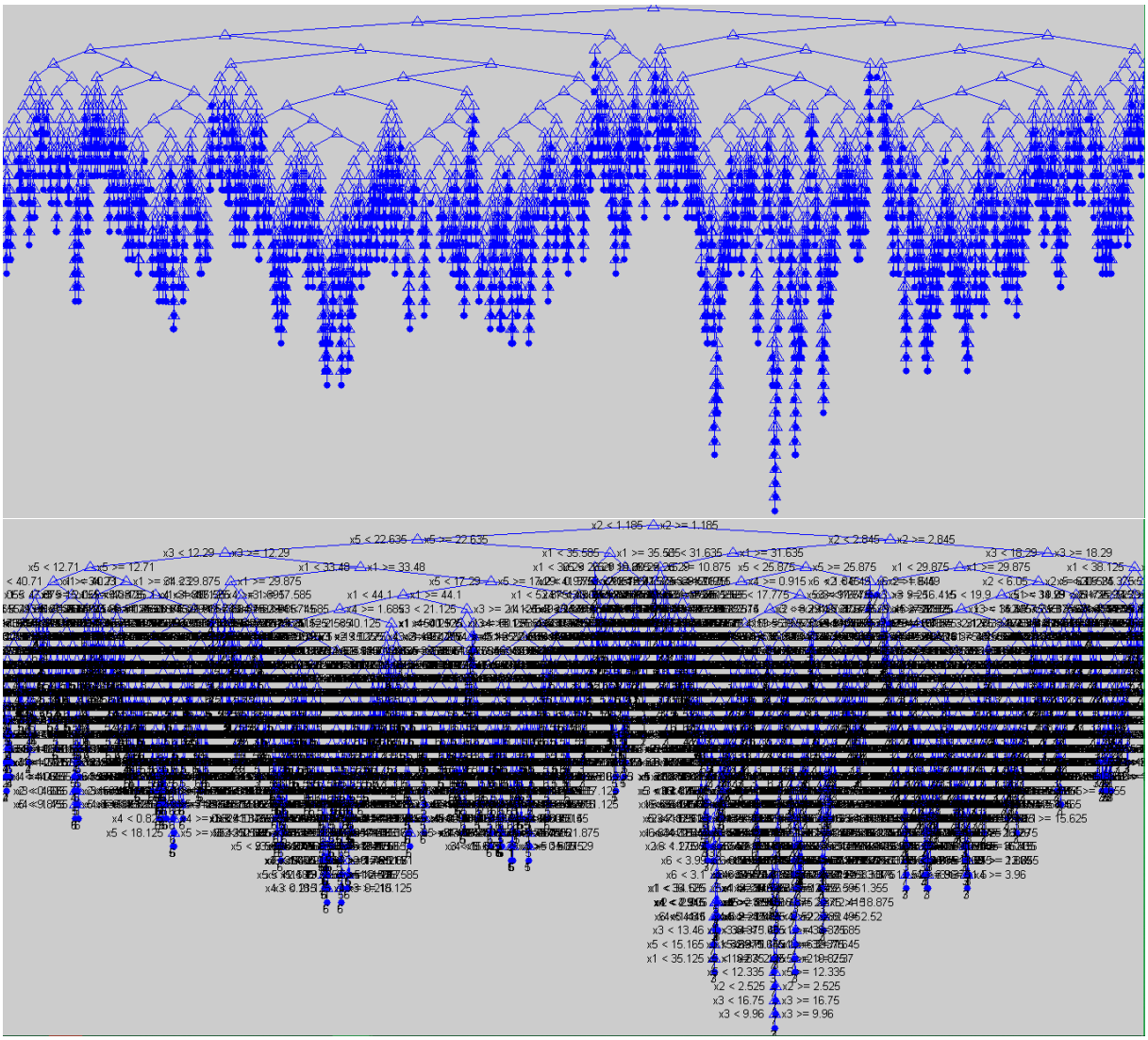


Figure X : Arbre de décision final

2. Fonction qui générer les nœuds de l'arbre de décision (non utilisée à cause de problème de (Matlab))

Main

```
Nodes = zeros(1,1);
Decision = ones(1,2);
[Nodes,Decision] = DecisionTree(Data,Decision,Nodes,0);
```

Fonction principale :

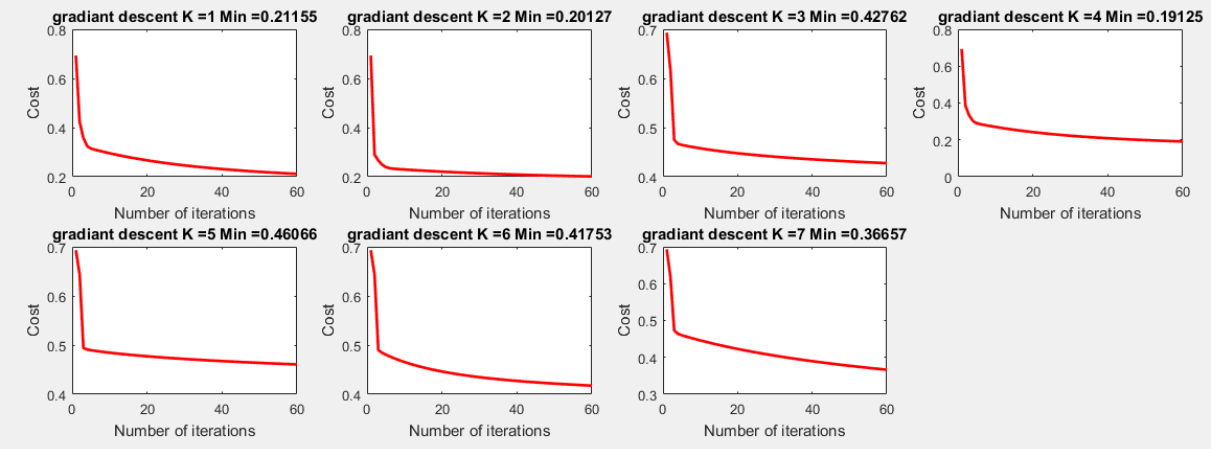
```
function [Nodes,Decision] = DecisionTree(Data,Decision,Nodes,NumNode)
if(size(Data,1)==0)
   disp(Data)
   return;
end
NbrFeature=size(Data,2);
class = Data(1,NbrFeature);
for i=2:size(Data,1)
```

```
    if(Data(i,NbrFeature)~=class)
        Fin = true;
        break;
    end
end
if(Fin==true)
   Cross = ones(1,7);
   N=size(Data,1);
   for i=1:NbrFeature
       Ligne = [];
       for j =1:N
           found = 0;
           for k=1:size(Ligne,1)
               if(Data(j,i)==Ligne(k,1))
                  found = k;
                  Colonne = Data(j,7);
                  break;
               end
           end
           if(found ==0)
               Ligne = [Ligne;[Data(j,i),1,0,0,0,0,0,0,0,0,0]];
           else
               Ligne(found,2) = Ligne(found,2)+1;
               Ligne(found,Colonne+2) = Ligne(found,Colonne+2)+1;
           end
       end
       CrossEntropy = 0;
       Tmp = 0;
       for k=1:size(Ligne,1)
           if(Ligne(k,3)~=0)
              Tmp = Tmp + (-
Ligne(k,3)/Ligne(k,2))*(log2(Ligne(k,3)/Ligne(k,2)));
           elseif(Ligne(k,4)~=0)
              Tmp = Tmp + (-
Ligne(k,4)/Ligne(k,2))*(log2(Ligne(k,4)/Ligne(k,2)));
           elseif(Ligne(k,5)~=0)
              Tmp = Tmp + (-
Ligne(k,5)/Ligne(k,2))*(log2(Ligne(k,5)/Ligne(k,2)));
           elseif(Ligne(k,6)~=0)
              Tmp = Tmp + (-
Ligne(k,6)/Ligne(k,2))*(log2(Ligne(k,6)/Ligne(k,2)));
           elseif(Ligne(k,7)~=0)
              Tmp = Tmp + (-
Ligne(k,7)/Ligne(k,2))*(log2(Ligne(k,7)/Ligne(k,2)));
           elseif(Ligne(k,8)~=0)
              Tmp = Tmp + (-
Ligne(k,8)/Ligne(k,2))*(log2(Ligne(k,8)/Ligne(k,2)));
           elseif(Ligne(k,9)~=0)
              Tmp = Tmp + (-
Ligne(k,9)/Ligne(k,2))*(log2(Ligne(k,9)/Ligne(k,2)));
           end
           CrossEntropy = CrossEntropy + Tmp;
       end
       Cross(i) = CrossEntropy;
   end
   MinFeature = 1;
   Min = Cross(1);
   for i=2:size(Cross)
       if(Min>Cross(i))
           MinFeature = i;
           Min = Cross(i);
```

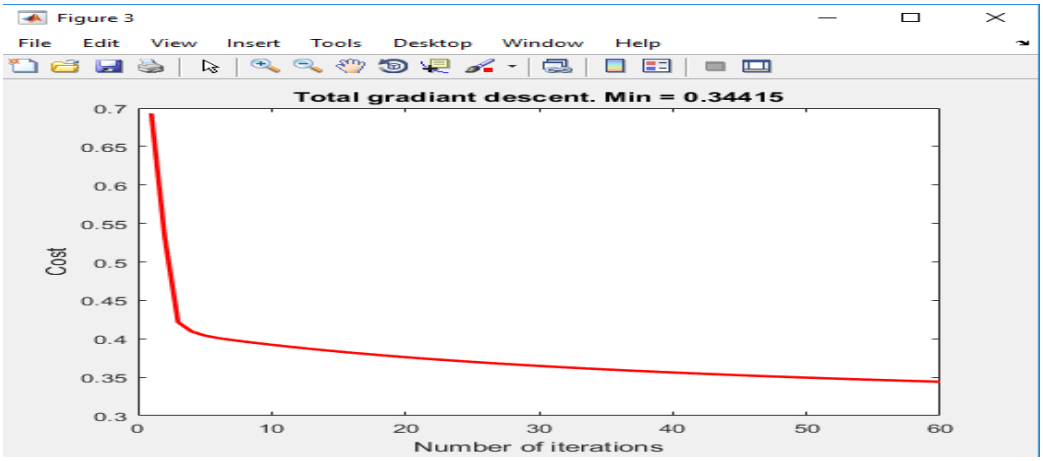
```
        end
    end
    Median = median(Data);
    disp(Median );
    disp('--------------------');
    disp(MinFeature);
    Nodes = [Nodes,NumNode];
    Decision = [Decision;MinFeature,Median(MinFeature)];
    DataLeft = ones(1,7);
    DataRight = ones(1,7);
    for i=1:N
        if(Median(MinFeature)>Data(i,MinFeature))
            DataLeft = [DataLeft;Data(i,:)];
        end
    end
    for i=1:N
        if(Median(MinFeature)<=Data(i,MinFeature))
            DataRight = [DataRight;Data(i,:)];
        end
    end
    DataLeft(1,:) = [];
    DataRight(1,:) = [];
    [Nodes,Decision] = DecisionTree(DataLeft,Decision,Nodes,size(Nodes));
    [Nodes,Decision] = DecisionTree(DataRight,Decision,Nodes,size(Nodes));
    return;
else
    return;
end
```

a. Regression logistique régularisé non optimisé

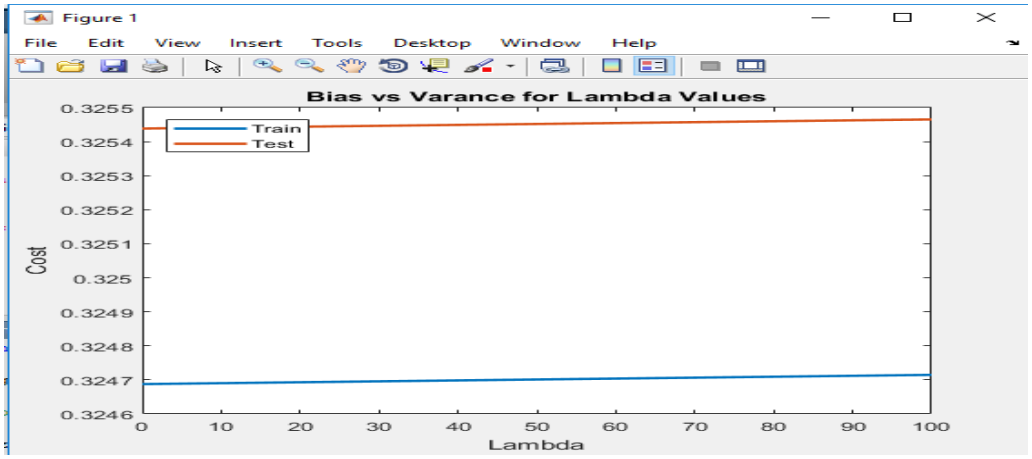
1 résultat d'affichage de la fonction cost relative a chaque classe



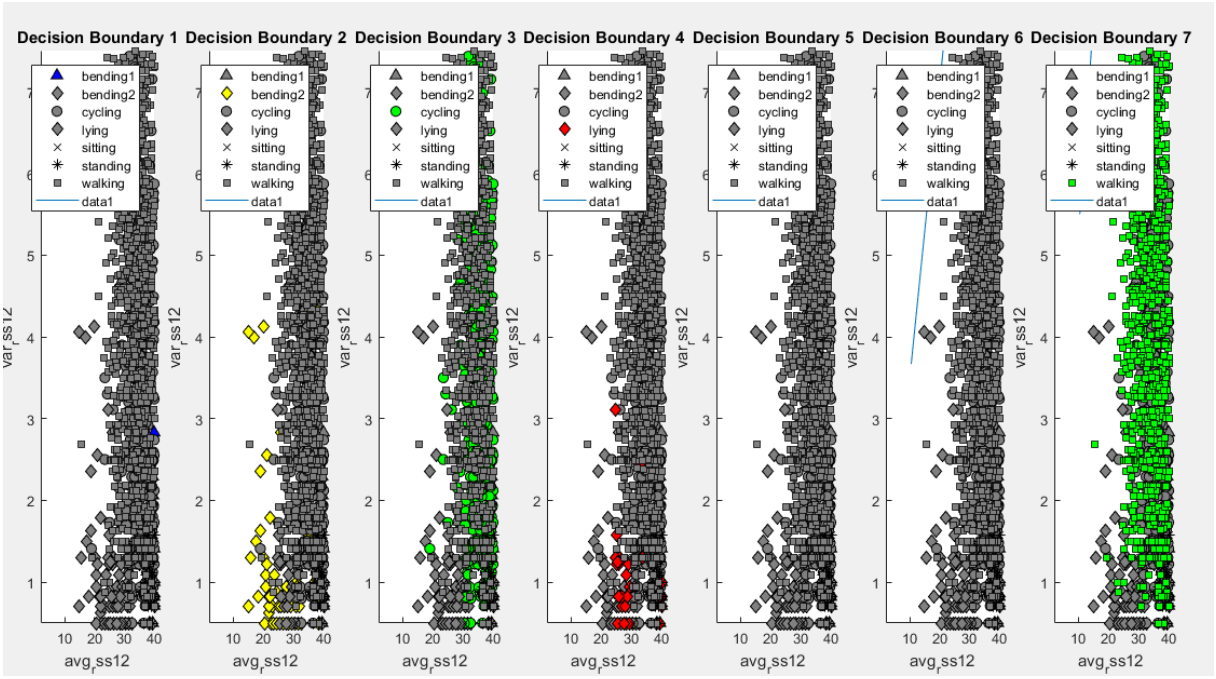
1 résultat d'affichage de la fonction cost global



3 plot bias vs variance lambda : dans le but de trouvé la meilleur valeur de lambda



4-plot decision boundary



Bibliographies:

[i] Classification trees in Matlab | Quantdare (visited 08.01.2019)
http://cedric.cnam.fr/vertigo/Cours/ml2/coursArbresDecision.html
[ii] Classification trees in Matlab | Quantdare (visited 12.01.2019) https://quantdare.com/classification-trees-in-matlab/
[iii] Fit binary classification decision tree for multiclass classification - MATLAB fitctree | Quantdare (visited 15.01.2019) https://www.mathworks.com/help/stats/fitctree.html
[iv] Cour Apprentissage Automatique, Université d'Alger 1, Spécialité ISII, 2018-2019
[v] Étape 2 : déterminer le nombre de neurones par couches cachées (visited 17.01.2019)
http://mapdata.science/~stpierre/reseaux-neuronaux/node18.html
[vi] Deep Learning Best Practices (1) — Weight Initialization| Quantdare (visited 17.01.2019)
https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94

1-https://github.com/cjlin1/libsvm : reference svm
2-https://github.com/ch-ms/iris-logistic-regression/blob/master/ : LOGISTIQUE REGRESSION