

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Benyoucef BENKHEDDA - Alger1

Faculté des Sciences

Département **Mathématiques et Informatique**



MINI-PROJET

Partie-B

Régression : Bike-Sahring Data-Set

Classification : Site Web Phishing Data-Set

Réalisé par

Benbaba Rym Amina

Walid Kebbab

Chiheb Ibrahim Amine

Chikh Abderrahmane

2018/2019

Table des matières

Who Did What.....	3
Site Web Phishing Data Set :.....	3
PARTIE-a :.....	3
Rappel :	3
Le jeu des données :.....	3
PARTIE-b :	4
Régression logistique :	4
Réseaux de neurones	7
Support Vector Machine (SVM).....	8
Les Arbres de décision	9
Comparaison.....	11
Bike Sharing Dataset :	11
PARTIE-a :	11
Rappel :	11
Le jeu des données :.....	12
PARTIE-b :	13
Régression linéaire :	13
Implémentation	13
Réseau de neurone	14
Bibliographie	17

Who Did What...

Afin de bien réaliser ce mini-projet, nous avons d'abord divisé les tâches entre nous (soit par sujet ou par algorithme de plus de la rédaction de rapport) afin de réaliser un travail de niveau d'un côté et de gagner du temps de l'autre.

Linear Regression et Logistic Regression par Chikh Abderrahmane.

Neural Networks par Benbaba Rym Amina.

Support Vector Machin par Chiheb Ibrahim Amine.

Decision Trees par Walid Kebbab.

Avant de commencer notre travail en détail nous avons voulu mentionner que notre but n'était pas vraiment de fournir des algorithmes exactes ou correctes à 100% mais plutôt de se pratiquer et essayer de pratiquer les mathématiques derrière chaque algorithme.

Ainsi que nous avons utilisé plusieurs code sources trouvés dans le net comme des sources pour mieux comprendre et même utilisé certaines fonctions de ces codes avec des modifications bien-sûr.

Site Web Phishing Data Set :

PARTIE-a :

Rappel :

En résumé Le phishing est une sorte d'attaque dans laquelle des criminels utilisent des courriels et des sites Web frauduleux pour tromper les organisations financières et leurs clients. Toute en attirant les utilisateurs en ligne en les persuadant de révéler le nom d'utilisateur, les mots de passe, le numéro de carte de crédit et de mettre à jour les informations du compte ou de renseigner les informations de facturation. [1]

Et pour cela nous envisageons à classer les sites web en deux catégories (phishing ou légitime) tout en comparons les algorithmes de l'apprentissage automatique utilisé et vu en cours pour déterminer l'algorithme le plus fiable qui donne un meilleur résultat par rapport à notre problème.

Le jeu des données :

Nous avons examiné les données de notre base de données qui contient 1353 instances et 10 variables,

On a constaté que les valeurs sont de valeurs (-1, 0 ou 1), Les attributs sont :

SFH (Server Form Handler) : si le contenu d'une chaîne est vide ou « about : blank » → **suspect 0**, sinon si le nom du Domain SFH est le même du nom du domaine alors **1 légitime**, sinon **-1 phishing**.

Pop-up Window : si la fenêtre contextuelle demande des informations personnelles alors **phishing -1**, sinon **légitime 1**.

SSL state : si un site utilise un certificat SSL cela veut dire **légitime 1**, **sinon -1**, mais dans notre temps actuel avec le développement inquiétant tout le site peut être **suspect 0**.

Request URL : si les objet externe contenu dans le site web (image, vidéo, son ...) on le même nom de domaine cela veut dire **1 (légitime)** sinon si le nom du domaine des objets et sites web sont défèrent alors **-1 (phishing)**.

URL Anchor : C'est l'élément définit par la balise <a> cela prend la valeur

1 si %de URL ANCHOR <31% → légitime.

0 si 31%<=%de URL ANCHOR<=67% →suspect

-1 sinon.

Web traffic : si le site web est classée dans la base de données Alexa cela veut dire **légitime 1**, sinon s'il est classe parmi les 100000 premiers qui ne sont pas dans la base de données Alexa alors il est **suspect 0**, sinon il est forcément **phishing -1**.

URL Length : si la longueur du l URL est inférieure ou égal 54 caractères cela veut dire **1 légitime**, et si elle est entre 54 et ses environ **0 suspect**, sinon **-1 phishing**.

Domain Age : Si l'Age d'un nom du domaine est plus qu'un an il est surement **légitime 1**, sinon si son âge est entre 1an et 6mois →0 « Suspect », -1 sinon.

IP : Si on trouve une adresse IP (même parfois en hexadécimal) alors **-1 phishing** sinon **1 légitime**.

Résultat : le résultat de classification des site web 1→ légitime, 0→suspect ou -1→phishing.

Afin de faciliter la tâche de nettoyage nous avons utilisé le fichier Excel pour supprimer les doublons dans le but d'améliorer la qualité de notre BDD ce qui a permet d'éliminer 701 tuples et cela a rendu la BDD plus légère.

PARTIE-b :

Régression logistique :

Définition

La régression logistique est un **algorithme supervisé** de classification, populaire en **Machine Learning** très utilisée car elle permet de modéliser des variables binaires ou des sommes de variables binaires. [2]

Avantages

- Les sorties ont une interprétation probabiliste intéressante et l'algorithme peut être régularisé pour éviter les sur ajustements. Les modèles logistiques peuvent être facilement mis à jour avec de nouvelles données en utilisant une descente de gradient stochastique.

Inconvénients

- La régression logistique a tendance à sous-performer en présence de limites de décision multiples ou non linéaires. Ils ne sont pas assez souples pour capturer naturellement des relations plus complexes. [3]

Notre dataset est composé de 561 lignes et 10 colonnes, nous avons utilisé 70% de cette première pour l'apprentissage qui fait 392 lignes et les 30 % restants seront utilisé pour les tests.

Nous avons initialisé un vecteur labels qui contient les valeurs uniques de notre dataset et qui représente les classes de classification 0, -1, 1.

Ensuite nous avons divisé notre dataset comme nous avons mentionné avant en 2 parties 70% Pour l'apprentissage en stockant les valeurs d'entrées dans **Xtrain** et les valeurs de sorties dans **Ytrain**, 30% pour le test en stockant les valeurs d'entrées dans **Xtest** et les valeurs de sorties dans **Ytest**.

Après ça on a fait la normalisation des données à l'aide de la fonction `normaliser`.

Nous avons initialisé les valeurs `alpha`, `iter`, `lambda` qui représentent respectivement le pas de convergence utilisé pour calculer le gradient descent, le nombre d'itérations de la même fonction, et le paramètre de régularisation.

Après on a appelé la fonction **prediction** qui a en entrée **Xtrain**, **Ytrain**, **Xtest**, **labels**, **alpha**, **iter** et **lambda** qu'on a déjà initialisé et nous donne en retour la meilleure valeur **theta**, les prédictions, et le **cost**.

Après l'initialisation de la taille de la partie d'apprentissage, nous avons créé une matrice **all_theta** qui contiendra les valeurs de **theta** retourné par le **gradient Descent** pour chaque classe, donc elle contient un nombre de ligne égale au nombre de labels.

On a aussi initialisé le vecteur prédiction qui contiendra les prédictions ainsi que d'autres initialisations nécessaires, ensuite nous avons calculé le **theta** pour chaque classe `c` en la mettant à 1 et les autres classes à 0 pour enfin les enregistrer dans la matrice **all_theta**.

Et pour cela nous avons utilisé la fonction **gradientDescent** dans la figure (1) :

```
function [theta , vect_cost] = gradientDescent(X, y, theta, alpha, iteration, lambda)
    m = length(y);
    vect_cost = zeros(iteration, 1);
    n = length(theta);
    temp = theta;
    for it = 1 : iteration
        erreur = (hypothese(X * theta) - y);
        for j=1 : n
            temp(j,1) = sum(erreur.*X(:,j));
        end
        theta = theta - (alpha/m) * temp + (lambda/m) * temp;
        vect_cost(it,1) = cost(X,y,theta,lambda);
    end
end
```

Figure 1

Afin d'éviter les '**overfitting**' et les '**underfitting**' nous avons utilisé le paramètre de régularisation '**lambda**' qui a affecté les calculs du **cost** et du **gradientDescent**.

Retournant à la classe prédiction, on a fait le calcul des probabilités de chaque classe et pour cela nous avons utilisé la fonction `hypothese` qui contient un simple calcul mathématique déjà vu en cours, (voir la figure 2)

```
function [h]=hypothese(e)
    % Calcule de l'hypothese
    h = (1 ./ (1 + exp(-e)));
end
```

Figure 2

On a utilisé ces probabilités pour trouver la prédiction qui est l'indice de la ligne ayant l'hypothèse la plus élevée, cette prédiction nous a permis de calculer la précision de notre apprentissage.

Malgré qu'on a fait la normalisation des données et qu'on a modifié dans notre BDD afin de réduire les données bruitées, et on a varié notre lambda et le nombre d'itération on a eu une précision de 53.25%

```
% Calcul de la précision
fprintf('Précision : %f \n', mean(double(Ytest == predictions))*100);
```

Enfin on affiche le graphe de la fonction **cost** qui donne le résultat suivant (figure 3) :

```
% Plot de la fonction cost
plot(1:iter,cost,'-b');
```

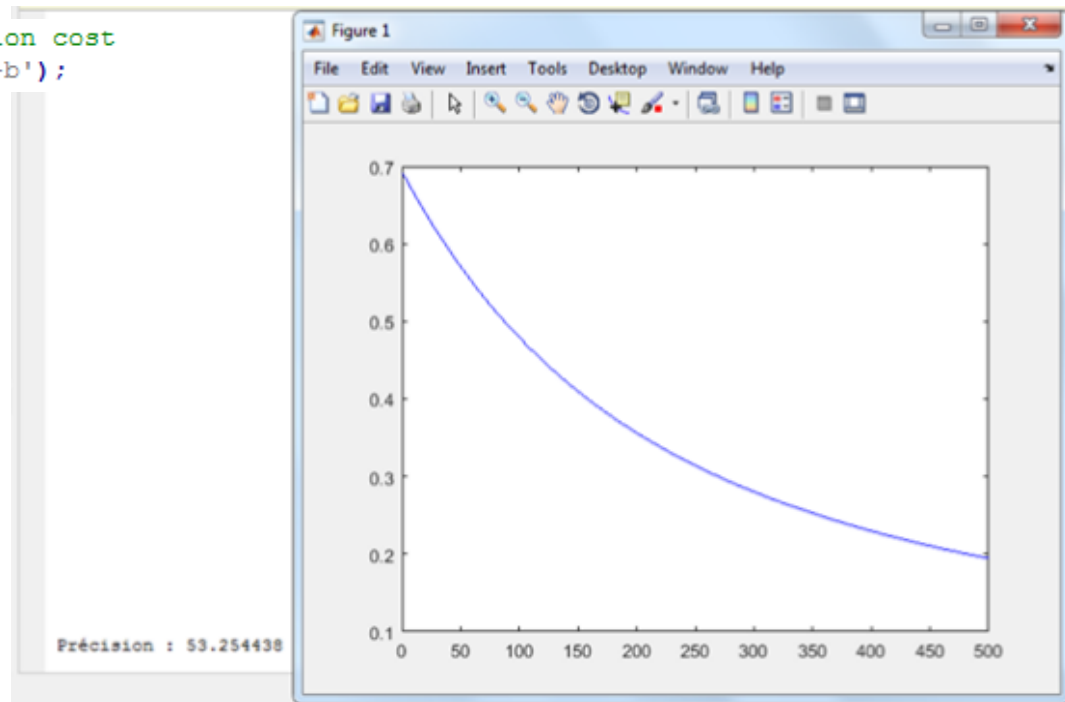


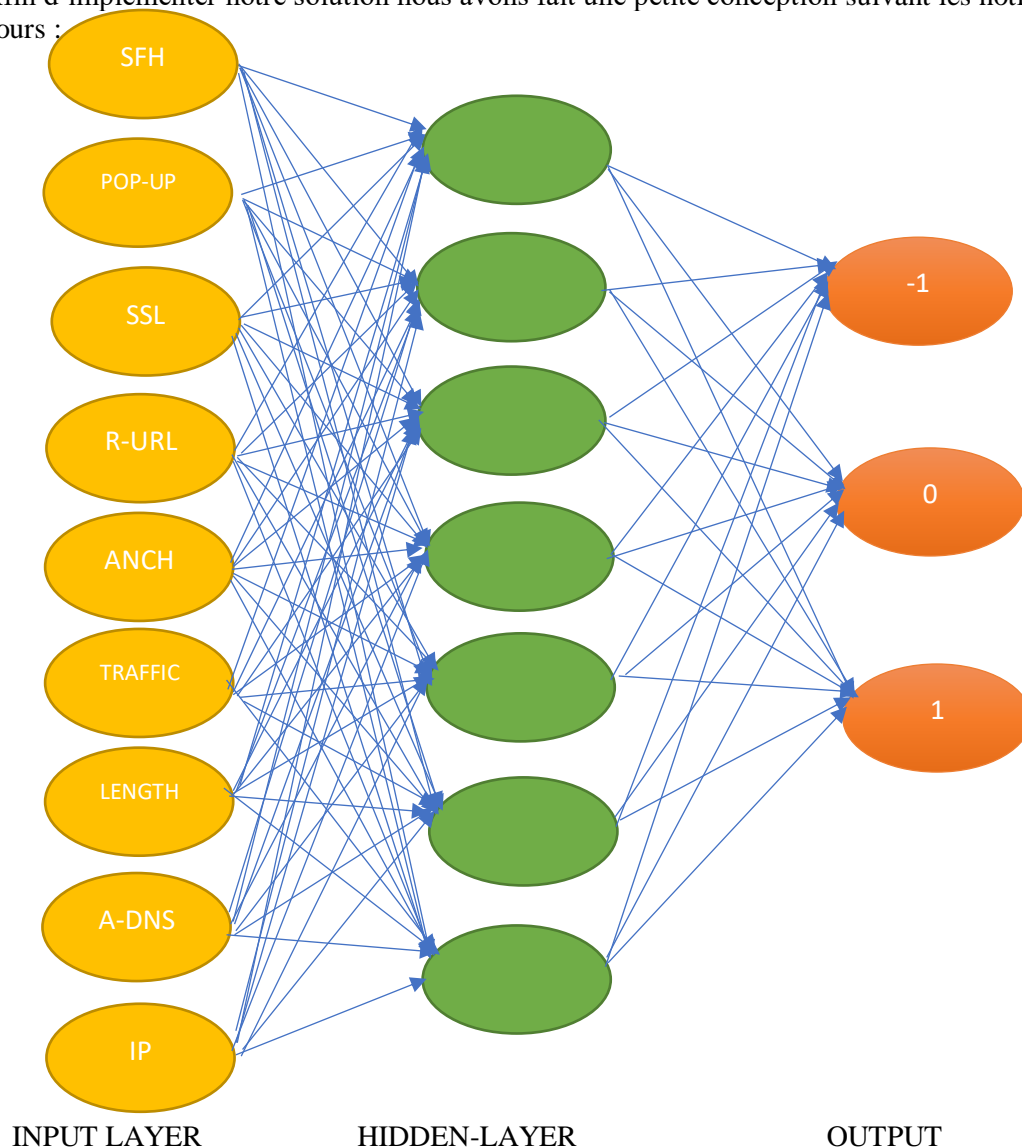
Figure 3

Réseaux de neurones

Le réseau neuronal lui-même n'est pas un algorithme, mais plutôt un cadre permettant à de nombreux algorithmes d'apprentissage automatique de travailler ensemble et de traiter des entrées de données complexes. [4]

Le réseau neuronal artificiel consiste en un ensemble d'éléments de traitement qui sont très interconnectés et transforment un ensemble d'entrées à un ensemble de sorties souhaitées. Le résultat de la transformation est déterminé par les caractéristiques des éléments et les poids associés aux interconnexions parmi eux. Un réseau de neurones effectue une analyse de l'information et fournit une estimation de probabilité qu'il correspond aux données qu'il a été formé à reconnaître. [5]

Afin d'implémenter notre solution nous avons fait une petite conception suivant les notions vues en cours :



Tout d'abord nous avons importé les données depuis la BDD déjà améliorée, nous avons séparé les inputs avec une sélection des colonnes et on a fait la même chose avec la colonne des résultats Y, par la suite nous avons défini le nombre des input (9 dans notre cas), et le nombre de hidden layer par 7 (suivant la règle de 75%), et enfin nous avons fixé le nombre de class à 3 (-1 0 et 1).

Par la suite nous avons initialisé θ_1 et θ_2 avec la fonction (randWeight_Team02) qui prend en entrée les Layer input et output et en sortie elle initialise les poids de layer input d'une façon aléatoire.

Après nous avons transformé les matrices de θ en un vecteur plat.

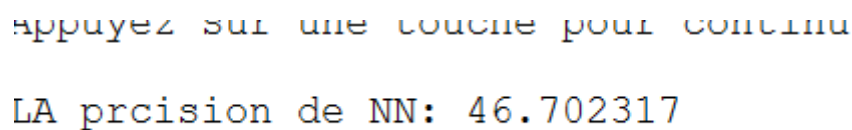
La fonction cost va tout d'abord remodeler la variable `nn_params` dans les paramètres de θ_1 et θ_2 , Feedforward le réseau de neurones et renvoyer le cout dans la variable `J`, fait la régularisation et l'implémente avec gradient et cost.

La fonction cost fait aussi l'appel à la fonction sigmoïde vu déjà en cours.

Nous avons calculé le cost pour des 50 itérations, et nous avons constaté que le cost est en train de se diminuer ce qui est bon.

Nous avons aussi utilisé la fonction `Fmincng` qui effectue une descente de gradient.

Enfin nous avons récupéré les valeurs de θ_1 et θ_2 de `nn_params` pour les utiliser dans le calcul de précision avec la fonction `prédit` et nous avons obtenu une précision égale à 46,702317 comme le montre la figure (4).



```
Appuyez sur une touche pour continuer
LA precision de NN: 46.702317
```

Figure 4

Support Vector Machine (SVM)

Définition

Les machines à vecteurs de support (SVM) utilisent un mécanisme appelé noyaux, qui calculent essentiellement la distance entre deux observations. L'algorithme SVM trouve ensuite une limite de décision qui maximise la distance entre les membres les plus proches des classes séparées.

Par exemple, une SVM avec un noyau linéaire est similaire à la régression logistique. Par conséquent, en pratique, l'avantage des SVM provient généralement de l'utilisation de noyaux non linéaires pour modéliser les limites de décision non linéaires.

Avantages

- Les SVM peuvent modéliser des limites de décision non linéaires, et il existe de nombreux noyaux parmi lesquels choisir. Ils sont également assez robustes contre les surajustements, en particulier dans les espaces de grandes dimensions.

Inconvénients

- Cependant, les SVM nécessitent beaucoup de mémoire, sont plus difficiles à régler en raison de l'importance de choisir le bon noyau et ne s'adaptent pas bien à des ensembles de données plus volumineux. Actuellement dans l'industrie, les forêts aléatoires sont généralement préférées aux SVM. [3]

Implémentation

Dans une première étape on a fait le prétraitement de données par l'importation de ces derniers en mettant les données entrantes dans la variable `X` et celles de sortantes en `Y`, ensuite on a décomposé notre « dataset » en deux parties, 70% pour l'apprentissage de nos données et 30% pour le test.

Ensuite, on a appelé la fonction « fitcecoc » et stocker le résultat de cette fonction qui est de type « ECOC » dans la variable model et qui est formé de l'ensemble de données prédicats de X et le Y.

Après l'enregistrement de modèle on a fait appel à la fonction predict qui va prédire l'ensemble de données qui est dans model et celui dans X_test pour les enregistrer dans y_prediction.

Enfin, on a affiché la précision de données en calculant la similarité entre les anciennes valeurs de Y_test et les nouvelles valeurs prédicats obtenus dans la ligne précédente.

Après avoir exécuté le code on a obtenu une précision de 73,39% comme le montre la figure (5) suivante :

```
16 % entraînement
17 - model = fitcecoc(X_train, y_train);
18 % prediction
19 - y_prediction = predict(model, X_test);
20 % Afficher la précision (la similarité entre y_restant et y qu'on a prédicé)
```

Command Window

New to MATLAB? See resources for [Getting Started](#).

Précision : 73.394495


 >>

Figure 5

Les Arbres de décision

Définition

Les arbres de décision sont un type d'apprentissage automatique supervisé, L'arbre peut être expliqué par deux ou plusieurs entités, à savoir les nœuds de décision et les feuilles. Les feuilles sont les décisions ou les résultats finaux dans notre cas ce sont les trois classes (légitime [1], suspect [0] et phishing [0]), Et les nœuds de décision sont ceux où les données sont stockées. [6]

Avantage

- Simple, c'est un modèle qui simple à expliquer une situation est cela à l'aide de la logique booléen
- On peut l'utiliser même sans faire normaliser notre dataset ou bien la modifier
- La validation se fait à l'aide d'un test statique, pour voir la fiabilité d'un model

Inconvénients

- On peut avoir un arbre très complexe (relation avec les attributs)
- On ne peut pas tous exprimée à l'aide des arbres de décision (XOR)

Implémentation

Comme tous les autre modèle on commencer importer notre dataset (juste 70% pour l'apprentissage) , notre X contient les 9 attribut et le Y contient les classes résultat (1,0,-1)

On suit on utilision la fonction predefinie fitctree qui prend en entrée les attribut X et Y les resultat de ces dernier pour construire et la fonction view afficher notre arbre

La fonction fitctree :

```
tree = fitctree(x,y);
```

La fonction view :

```
view (tree,'mode','graph');
```

Le resultat est présenté dans la figure (6).

Dans le schémat on remarque que le feilles sont notre classe est les noued sont les attribut

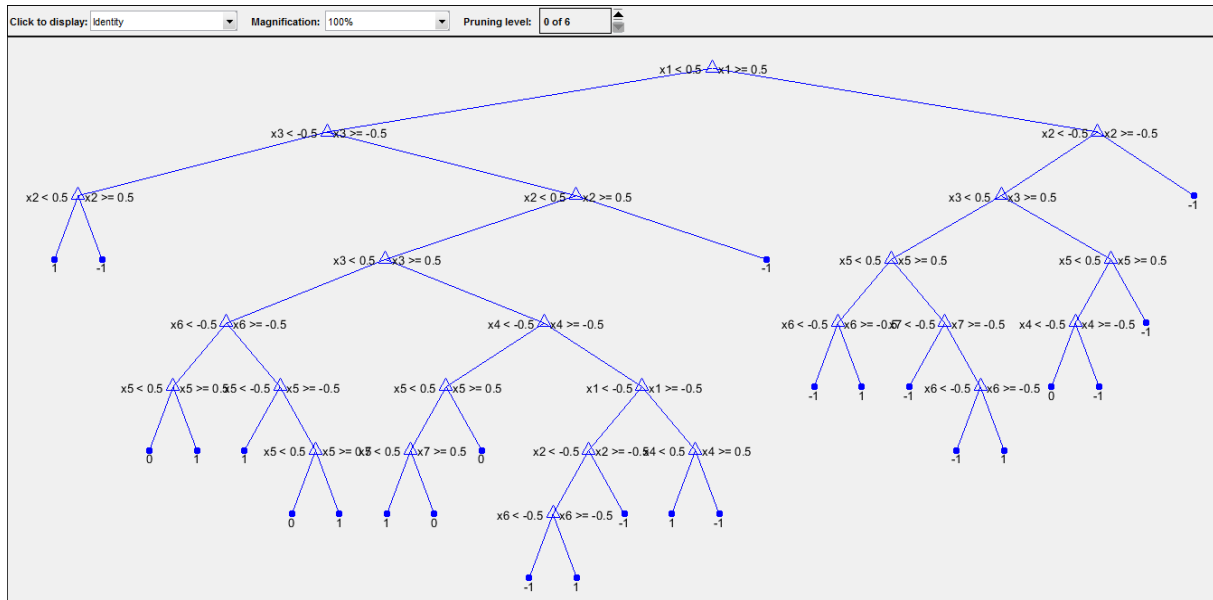


Figure 6

Est pour tester notre arbre on peut utiliser des test statique en utilisant la fonction predict qui prend en entree notre arbre , et un tuple de notre choix (a partir des 30% dernier dedier pour les test)

```
test = predict (tree,[1,1,-1,-1,1,-1,0,1,0]);
```

Pour la prediction de notre taux apprentissage de notre modèle d'arbre de desision on a fait plusieurs test statique et on a rarement tomber sur des conflusion entre classe par exemple les test suivant

Test 1 : les X [1 1 -1 -1 1 -1 0 1 0] et le resultat d'apres notre dataset et -1

Le resultat d'apres notre modèle est dans la figure (7)

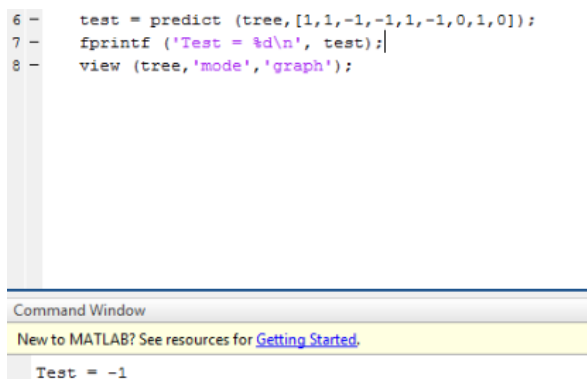
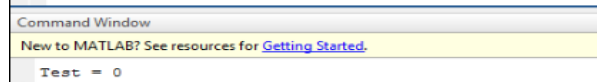


Figure 7

Test 2 : les X [-1 -1 1 -1 1 1 -1 -1 0] et le resultat d'apres notre dataset et 0

Le resultat d'apres notre modèle est (figure 8):

```
6 - test = predict (tree, [-1,-1,1,-1,1,1,-1,-1,0]);
7 - fprintf ('Test = %d\n', test);
8 - view (tree,'mode','graph');|
```



Command Window

New to MATLAB? See resources for [Getting Started](#).

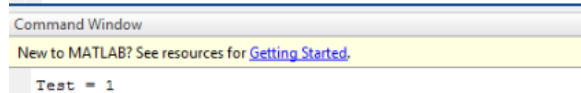
Test = 0

Figure 8

Test 3 : les X [-1 0 -1 -1 -1 0 1 -1 0] et le resultat d'apres notre dataset est 1

Le resultat d'apres notre modèle est dans la figure (9)

```
6 - test = predict (tree, [-1,0,-1,-1,-1,0,1,-1,0]);
7 - fprintf ('Test = %d\n', test);
8 - view (tree,'mode','graph');|
```



Command Window

New to MATLAB? See resources for [Getting Started](#).

Test = 1

Figure 9

Comparaison

Après avoir exécuter les différents algorithmes de l'apprentissage automatique sur notre problème de classification intitulé « site web phishing » et d'après les résultats que nous avons obtenu surtout par rapport la précision nous avons remarqué que Logistic Regression a atteint une précision de 53,25%, la précision de Neural Networks est de 46,70%, 73,39% pour SVM, et enfin pour les arbres de décision on a obtenu des résultat correct à 100% (pas vraiment) pour les tests que nous avons fait et pour cela nous avons déduit que les arbres de décisions ont fournit des résultats meilleurs par rapport aux autres algorithmes dans notre cas, cela peut être parce que nous avons utilisé des fonctions prédéfinies ... Le SVM est dans l 2eme place pour les mêmes raisons.

Bike Sharing Dataset :

PARTIE-a :

Rappel :

Les systèmes de partage de vélos sont un moyen de louer des vélos de façon automatique, dont le nombre des vélos et de stations est limité.

La station est dite :

- Équilibrée s'il dispose de suffisamment de vélos et de places disponible à chaque station à chaque visite d'un utilisateur
- Pénurie s'il y'a pas assez de vélos dans la station.
- Débordement si le nombre de vélos dans la station peut atteindre 100%. [7]

Et donc les questions qui se posent sont :

- Quels sont les facteurs et les conditions qui affectent le trafic en vélos partagés ?
- Comment résoudre le problème de l'équilibrage des systèmes de partage de vélos ?

Nous envisageons à réduire la frustration des clients et assurer que la station ne soit jamais vide, tout en utilisant les différentes algorithmes et techniques de l'apprentissage automatique.

Le jeu des données :

Après avoir examiné notre base de données day.csv, les données couvrent une période de 2ans allant de 01/01/2011 à 31/12/2012.

L'ensemble de données résultant que nous utiliserons contient 17389 instances et 17 variables.

Les attributs sont de type entier et réel et sont les suivants :

- **instant** : index d'enregistrement.
- **dteday** : date de jour format DD/M/Y
- **saison** : contenant les entiers 1 à 4 (1: printemps, 2: été, 3: automne, 4: hiver).
- **yr** : année (0→2011, 1→2012).
- **mnth** : mois (1 à 12).
- **hr** : heure (0 à 23).
- **holiday** : (1→ jour férié et 0→ jour non férié)
- **weekday** : jour de la semaine (1 à 7)
- **workingday** : si le jour n'est ni le week-end ni les jours fériés, la valeur 1, sinon, 0.
- **weathersit** : contenant les entiers 1 à 4 représentant quatre listes différentes de conditions météorologiques :
 1. Clair, Peu de nuages.
 2. Nuages fragmentés, Brouillard.
 3. Faible neige, faible pluie + orage.
 4. Fortes pluies + palettes de glace + brouillard, neige + brouillard
- **temp** : température normalisée en Celsius
- **atemp** : contenant les valeurs de la température de sensation au moment donné, normalisée en degrés Celsius.
- **hum** : humidité normalisée contenant les valeurs du niveau d'humidité relative au moment donné.
- **windspeed** : contenant les valeurs de la vitesse du vent normalisée.
- **casual** : nombre d'utilisateurs occasionnel (non enregistrés).
- **registered** : nombre d'utilisateurs enregistrés.
- **cnt** : contenant nombre total de vélos loués, par des utilisateurs occasionnels ou enregistrés .

« cnt » sera utilisé comme variable de réponse ici, et tous les autres comme prédicteur. (s.d.)

Comme déjà mentionné dans la Partie-A livrée nous avons fait un nettoyage dont on a annulé l'attribut « atemp » car elle est répétitive à l'attribut « cnt », on a aussi supprimé « casual » et « registred » car elles totalisent « cnt ».

PARTIE-b :

Régression linéaire :

Définition

La régression linéaire est une modélisation linéaire qui permet d'établir des estimations dans le futur à partir d'informations provenant du passé. [8]

Avantages

- La régression linéaire est excellente lorsque la relation entre la variable d'entrée et la variable de réponse est connue pour être linéaire.
- La régression linéaire est simple à comprendre et peut être régularisée pour éviter un sur ajustement. De plus, les modèles linéaires peuvent être facilement mis à jour avec de nouvelles données en utilisant une descente de gradient stochastique.

Inconvénients

- La régression linéaire fonctionne mal lorsqu'il existe des relations non linéaires. Ils ne sont pas naturellement suffisamment flexibles pour capturer des régularités plus complexes, et l'ajout des termes d'interaction ou des polynômes appropriés peut s'avérer délicat et prendre du temps. [3]

Implémentation

Notre dataset est composé de 731 lignes et 16 colonnes, nous avons utilisé 70% de cette première pour l'apprentissage qui fait 511 lignes et les 30 % restants seront utilisés pour les tests.

Nous avons commencé par importer notre dataset en utilisant la fonction load et de stocker les valeurs entrantes dans X et les valeurs sortants dans Y qui fait.

Après l'importation des données il faut normaliser ces derniers et pour cela nous avons utilisé la fonction normaliser qui en entrées X et en sortie X, muu et sigma qui représente respectivement les données entrantes après la normalisation, la moyenne de ces derniers et l'écart type.

Ensuite nous avons réinitialiser notre jeu de données X en le concaténons avec une colonne qui contiens seulement des valeur égale à 1 qui représente le X0 utilisant la fonction prédéfinis ones qui a en entrées « m » qui est le nombre de lignes.

Maintenant nous allons initialiser les valeurs de alpha qui représente le pat de convergence utilisé pour calculer le gradient **descent**, **iters** qui représente le nombre d'itérations de la même fonction. **theta** qui est une colonne de 15 ligne contenant des 0.

On fait l'appelle de la fonction gradientDescent qui prend en entré X,y,theta,alpha,iters qui sont déjà initialisés et qui retourne les meilleurs valeurs de theta ainsi que le J représentant les taux d'erreurs .

Dans la fonction gradientDescent nous avons initialisé notre J avec une colonne qui contiens des 0, ensuite en nombre d'itération, nous avons remplis notre j en utilisant la fonction computeCost qui prend en entré le X, le y ainsi que le theta.

Le calcul de la fonction Cost ce fait par un calcul mathématique déjà vu dans le cours.

Une fois les calculs sont faits, on affiche le graphe de la cost fonction utilisant la fonction plot_Cost

On stock par la suit la valeur minimale de J dans un minj et on l'affiche dans la console,

Voici les résultats dans la figure (10) :

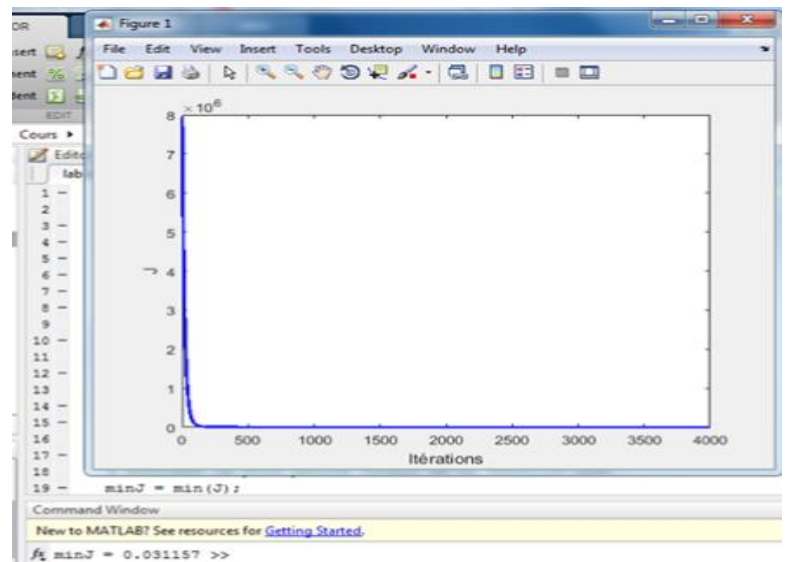


Figure 10

Maintenant on stock les 30% des données de la dataset restants dans les variables X_restant et Y_restant , après avoir faire la normalisation sur ces 2 derniers on calcule les données sortantes prédicats Y_new_predict pour calculer le taux d'erreur de chaque donnée

Le résultat est le suivant (voir la figure 11) :

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-0.1150												
2	-0.0462												
3	0.4382												
4	-0.0407												
5	-0.3393												
6	-0.2593												
7	-0.1384												
8	0.1224												
9	-0.0361												
10	0.1680												
11	0.0297												
12	0.0162												
13	-0.2428												
14	-0.4188												

Figure 11

On remarque que La distance entre Y_new_predict et y_restant converge vers le 0 , ce que implique le résultat est bon .

Réseau de neurone

Pour les Réseaux de neurones du problème de régression nous avons suivi presque le même principe que la classification, alors après l'importation de la bdd et la sélection des X et Y , nous avons défini un tableau de hidden-layer avec 13 couches, par la suite nous avons initialisé les poids pour layer1 et 2 tout en ajoutant le bias.

Défini le nombre d'itération égale a 200 avec un échantillon de 700, on a après calculer l'hypothese avec les poids précédents et modifier ces poids en fonction des outputs.

Enfin nous avons fait un test pour vérifier le résultat en comparant le y obtenu avec le y qu'on a déjà dans la bdd.

La Figure 12 décrit le résultat obtenu et le résultat réel dans la colonne 100

```
output = 0.43960
ans = 0.35891
```

Figure 12

LA Figure 13 décrit le résultat obtenu et le résultat réel dans la colonne 50

```
output = 0.14857
ans = 0.15880
```

Figure 13

On Remarque que pour les deux algorithmes la distance entre Y_new_predict et y_restant converge vers le 0

ANNEX

Régression_Logistique

```
* lab_Team02.m
1 clear ; close all; clc;
2 data = load('PiishingData.txt');
3 X = data(:, [1: 9]);
4 y = data(:, 10);
5 [X muu sigma] = normaliser_Team02(X); % Normalisation des données
6 % les labels des classes
7 % 0 : Suspect
8 % -1 : Phishing
9 % 1 : Legitime
10 labels = size(unique(y),1);
11 % Division des données en deux parties 70% Pour l'apprentissage et 30% pour le test
12 Xtrain = data(1:392, [1: 9]);
13 Ytrain = data(1:392, 10);
14 Xtest = data(393:561, [1: 9]);
15 Ytest = data(393:561, 10);
16 alpha = 0.002; % Initialisation de taux d'apprentissage
17 iter = 500; % Initialisation de nombre d'itérations
18 % Initialisation de paramètre de régularisation
19 lambda = 0.001; % Calcul de theta, les predictions, et cost
20 [theta,predictions, cost] = prediction_Team02(Xtrain,Ytrain,Xtest,labels,alpha,iter,lambda);
21 % Plot de la fonction cost
22 plot(1:iter,cost,'-b');
23 fprintf('Précision : %f \n', mean(double(Ytest == predictions))*100); % Calcul de la précision
24
```

```
* cost_Team02.m
1 function [j] = cost_Team02(X, y , theta,lambda)
2 m = size(X,1);
3 j = (-1/m) * sum(y.*log(hypothese_Team02(X * theta)) +
4 (1 - y).*log(1 - hypothese_Team02(X * theta))) + (lambda/(2*m)) * sum(theta.^2);
5 end
6
```

```
gradientDescent_Team02.m
1 function [theta , vect_cost] = gradientDescent_Team02(X, y, theta, alpha, iteration,lambda)
2 m = length(y);
3 vect_cost = zeros(iteration, 1);
4 n = length(theta);
5 temp = theta;
6 for it = 1 : iteration
7 erreur = (hypothese_Team02(X * theta) - y);
8 for j=1 : n
9 temp(j,1) = sum(erreur.*X(:,j));
10 end
11 theta = theta - (alpha/m) * temp + (lambda/m) * temp;
12 vect_cost(it,1) = cost_Team02(X,y,theta,lambda);
13 end
14 end
```

```
hypothese_Team02.m
1 function [h]=hypothese_Team02(e)
2 % Calcule de l'hypothese
3 h = (1 ./ (1 + exp(-e)));
4 end
5
```

```

normaliser_Team02.m
1 function [X_normalise, m, sigma] = normaliser_Team02(X)
2 X_normalise = X;
3 m = zeros(1, size(X, 2));
4 sigma = zeros(1, size(X, 2));
5 m = mean(X_normalise);
6 sigma = std(X_normalise);
7 tfmu = X_normalise - repmat(m, length(X_normalise), 1);
8 tfstd = repmat(sigma, length(X_normalise), 1);
9 X_normalise = tfmu ./ tfstd;
10 end

prediction_Team02.m
1 function [theta, prediction, vect_cost] = prediction_Team02(X, Y, Xtest, labels, alpha, iter)
2 %Taille de tuples
3 m = size(X, 1);
4 %Nombre de colonnes
5 n = size(X, 2);
6 % Initialisation d'une matrice qui contient les valeurs des théta pour
7 % chaque classe , donc elle contient un nombre de lignes est égale à
8 % nombre de labels et un nombre d'attributs de nos données
9 all_theta = zeros(labels, n + 1);
10 %Initialisation du vecteur qui contiendra les predictions
11 prediction = zeros(size(X, 1), 1);
12 % Ajout des uns a la matrice de données
13 X = [ones(m, 1) X];
14 %Initialisation de la matrice theta
15 theta_initial = zeros(n + 1, 1);
16 vecteur = zeros(iter, labels);
17 for c = 1 : labels
18     %Calcul du theta pour chaque classe c en la mettant à 1 et les autres classes à 0
19     [theta, vecteur(:, c)] = gradientDescent_Team02(X, (Y == c), theta_initial, alpha, iter);
20     % L'enregistrement des thétas de chaque classe dans la matrice all_theta
21     all_theta(c, :) = theta';
22 end
23 vect_cost = min(vecteur');
24 vect_cost(:);
25 %Ajout des uns a la matrice des tests
26 Xtest = [ones(size(Xtest, 1), 1) Xtest];
27 %Calcul des probabilités de chaque classe, l'indice de la ligne ayant l'hypothese la
28 [probability indices] = max(hypothese_Team02(all_theta * Xtest));
29 prediction = indices';
30 end

```

Réseaux de neurones

[illegible]

```

43
44 costFunction = @(p) nnCostFunction_Team02(p, ...
45                                     input_layer_size, ...
46                                     hidden_layer_size, ...
47                                     num_labels, X, y, lambda);
48
49 % costFunction est une fonction qui prend un seul argument
50 [nn_params, cost] = fmincg_Team02(costFunction, initial_nn_params, options);
51
52 % Obtenir Theta1 et Theta2 depuis nn_params
53 Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
54                 hidden_layer_size, (input_layer_size + 1));
55
56 Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end),
57                 num_labels, (hidden_layer_size + 1));
58
59 fprintf('\nAppuyez sur une touche pour continue.\n');
60 pause;
61
62 pred = predict_Team02(Theta1, Theta2, X);
63
64 fprintf('\nLA précision de NN: %f\n', mean(double(pred == y)) * 100);
65 pause;

```

nnCostFunction_Team02.m

```

1 function [J grad] = nnCostFunction_Team02(nn_params, ...
2                                     input_layer_size, ...
3                                     hidden_layer_size, ...
4                                     num_labels, ...
5                                     X, y, lambda)
6
7
8 % Remodelez nn_params dans les paramètres Theta1 et Theta2, les matrices de poids
9 % pour notre réseau de neurones à 2 couches
10 Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
11                 hidden_layer_size, (input_layer_size + 1));
12
13 Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), .
14                 num_labels, (hidden_layer_size + 1));
15
16
17 m = size(X, 1);
18 fprintf('%d\n', size(Theta1));
19 fprintf('%d\n', size(Theta2));
20
21 J = 0;
22 Theta1_grad = zeros(size(Theta1));
23 Theta2_grad = zeros(size(Theta2));
24 %Feedforward le réseau de neurones et renvoyer le coût dans la variable J.

```

nnCostFunction_Team02.m

```

26 K = num_labels;
27 X = [ones(m,1) X];
28
29 for i = 1:m
30     X_i = X(i,:);
31     h_of_Xi = sigmoid_Team02( [1 sigmoid_Team02(X_i * Theta1')] * Theta2' );
32
33
34     y_i = zeros(1,K);
35     y_i(X(y(i))) = 1;
36     J = J + sum( y_i .* log(h_of_Xi) + (1 - y_i) .* log(1 - h_of_Xi));
37 end;
38
39 J = (-1 / m) * J;
40 Theta1s=Theta1(:,2:end);
41 Theta2s=Theta2(:,2:end);
42 % la regularisation
43
44 J = J + (lambda / (2 * m) * (sum(sum(Theta1s.^2)) + sum(sum(Theta2s.^2))));
45
46 delta_accum_1 = zeros(size(Theta1));
47 delta_accum_2 = zeros(size(Theta2));
48
49 for t = 1:m

```

nnCostFunction_Team02.m

```

50     a1 = X(t,:);
51     z2 = a1 * Theta1';
52     a2 = [1 sigmoid_Team02(z2)];
53     z3 = a2 * Theta2';
54     a3 = sigmoid_Team02(z3);
55     yi = zeros(1,K);
56     yi(X(y(t))) = 1;
57
58     delta_3 = a3 - yi;
59     delta_2 = delta_3 * Theta2 .* sigmoidGradient_Team02([1 z2]);
60
61     delta_accum_1 = delta_accum_1 + delta_2(2:end)' * a1;
62     delta_accum_2 = delta_accum_2 + delta_3' * a2;
63 end;
64
65 Theta1_grad = delta_accum_1 / m;
66 Theta2_grad = delta_accum_2 / m;
67
68
69 % Implement regularization avec cost function et gradients.
70
71
72 Theta1_grad(:, 2:input_layer_size+1) = Theta1_grad(:, 2:input_layer_size+1) + lambda /
73 Theta2_grad(:, 2:hidden_layer_size+1) = Theta2_grad(:, 2:hidden_layer_size+1) + lambda /

```

```

68
69 % Implement regularization avec cost function et gradients.
70
71
72 Theta1_grad(:, 2:input_layer_size+1) = Theta1_grad(:, 2:input_layer_size+1) + lambda /
73 Theta2_grad(:, 2:hidden_layer_size+1) = Theta2_grad(:, 2:hidden_layer_size+1) + lambda /
74
75
76 grad = [Theta1_grad(:) ; Theta2_grad(:)];
77
78
79 end
80

```

```

1 function [X, fX, i] = fmincg(f, X, options, P1, P2, P3, P4, P5)
2
3 if exist('options', 'var') && ~isempty(options) && isfield(options, 'MaxIter')
4     length = options.MaxIter;
5 else
6     length = 100;
7 end
8
9
10 RHO = 0.01; % a bunch of constants for line searches
11 SIG = 0.5; % RHO and SIG are the constants in the Wolfe-Powell conditions
12 INT = 0.1; % don't reevaluate within 0.1 of the limit of the current bracket
13 EXT = 3.0; % extrapolate maximum 3 times the current bracket
14 MAX = 20; % max 20 function evaluations per line search
15 RATIO = 100; % maximum allowed slope ratio
16
17 argstr = ['feval(f, X)']; % compose string used to call function
18 for i = 1:(nargin - 3)
19     argstr = [argstr, 'P', int2str(i)];
20 end
21 argstr = [argstr, ' '];
22
23 if max(size(length)) == 2, red=length(2); length=length(1); else red=1; end
24 S=['Iteration '];
25
26 S=['Iteration '];
27
28 i = 0; % zero the run length counter
29 ls_failed = 0; % no previous line search has failed
30 fX = [];
31 [f1 df1] = eval(argstr); % get function value and gradient
32 i = i + (length<0); % count epochs?!
33 s = -df1; % search direction is steepest
34 d1 = -s'*s; % this is the slope
35 z1 = red/(1-d1); % initial step is red/(|s|+1)
36
37 while i < abs(length) % while not finished
38     i = i + (length>0); % count iterations?!
39
40 X0 = X; f0 = f1; df0 = df1; % make a copy of current values
41 X = X + z1*s; % begin line search
42 [f2 df2] = eval(argstr);
43 i = i + (length<0); % count epochs?!
44 d2 = df2'*s;
45 f3 = f1; d3 = d1; z3 = -z1; % initialize point 3 equal to point 1
46 if length>0, M = MAX; else M = min(MAX, -length-i); end
47 success = 0; limit = -1; % initialize quantities
48 while 1
49     while ((f2 > f1+z1*RHO*d1) || (d2 > -SIG*d1)) && (M > 0)
50         limit = z1; % tighten the bracket

```

```

48     limit = z1; % tighten the bracket
49     if f2 > f1
50         z2 = z3 - (0.5*d3*z3*z3)/(d3*z3+f2-f3); % quadratic fit
51     else
52         A = 6*(f2-f3)/z3+3*(d2+d3); % cubic fit
53         B = 3*(f3-f2)-z3*(d3+2*d2);
54         z2 = (sqrt(B*B-A*d2*z3*z3)-B)/A; % numerical error possible - ok!
55     end
56     if isnan(z2) || isinf(z2)
57         z2 = z3/2; % if we had a numerical problem then bisect
58     end
59     z2 = max(min(z2, INT*z3), (1-INT)*z3); % don't accept too close to limits
60     z1 = z1 + z2; % update the step
61     X = X + z2*s;
62     [f2 df2] = eval(argstr);
63     M = M - 1; i = i + (length<0); % count epochs?!
64     d2 = df2*s;
65     z3 = z3-z2; % z3 is now relative to the location of z2
66 end
67 if f2 > f1+z1*RHO*d1 || d2 > -SIG*d1
68     break; % this is a failure
69 elseif d2 > SIG*d1
70     success = 1; break; % success
71 elseif M == 0
72     break; % failure

```

```

72     break; % failure
73 end
74 A = 6*(f2-f3)/z3+3*(d2+d3); % make cubic extrapolation
75 B = 3*(f3-f2)-z3*(d3+2*d2);
76 z2 = -d2*z3*z3/(B+sqrt(B*B-A*d2*z3*z3)); % num. error possible - ok!
77 if ~isreal(z2) || isnan(z2) || isinf(z2) || z2 < 0 % num prob or wrong sign?
78     if limit < -0.5 % if we have no upper limit
79         z2 = z1 * (EXT-1); % the extrapolate the maximum amount
80     else
81         z2 = (limit-z1)/2; % otherwise bisect
82     end
83 elseif (limit > -0.5) && (z2+z1 > limit) % extrapolation beyond max?
84     z2 = (limit-z1)/2; % bisect
85 elseif (limit < -0.5) && (z2+z1 > z1*EXT) % extrapolation beyond limit
86     z2 = z1*(EXT-1.0); % set to extrapolation limit
87 elseif z2 < -z3*INT
88     z2 = -z3*INT;
89 elseif (limit > -0.5) && (z2 < (limit-z1)*(1.0-INT)) % too close to limit?
90     z2 = (limit-z1)*(1.0-INT);
91 end
92 f3 = f2; d3 = d2; z3 = -z2; % set point 3 equal to point 2
93 z1 = z1 + z2; X = X + z2*s; % update current estimates
94 [f2 df2] = eval(argstr);
95 M = M - 1; i = i + (length<0); % count epochs?!
96 d2 = df2*s;

```

```

95     M = M - 1; i = i + (length<0); % count epochs?!
96     d2 = df2'*s;
97     end % end of line search
98
99     if success % if line search succeeded
100         f1 = f2; fx = [fx' f1]';
101         fprintf('%s %4i | Cost: %4.6e\r', S, i, f1);
102         s = (df2'*df2-df1'*df2)/(df1'*df1)*s - df2; % Polack-Ribiere direction
103         tmp = df1; df1 = df2; df2 = tmp; % swap derivatives
104         d2 = df1'*s;
105         if d2 > 0 % new slope must be negative
106             s = -df1; % otherwise use steepest direction
107             d2 = -s'*s;
108         end
109         z1 = z1 * min(RATIO, d1/(d2-realmmin)); % slope ratio but max RATIO
110         d1 = d2;
111         ls_failed = 0; % this line search did not fail
112     else
113         X = X0; f1 = f0; df1 = df0; % restore point from before failed line search
114         if ls_failed || i > abs(length) % line search failed twice in a row
115             break; % or we ran out of time, so we give up
116         end
117         tmp = df1; df1 = df2; df2 = tmp; % swap derivatives
118         s = -df1; % try steepest
119         d1 = -s'*s;
120
121     else
122         X = X0; f1 = f0; df1 = df0; % restore point from before failed line search
123         if ls_failed || i > abs(length) % line search failed twice in a row
124             break; % or we ran out of time, so we give up
125         end
126         tmp = df1; df1 = df2; df2 = tmp; % swap derivatives
127         s = -df1; % try steepest
128         d1 = -s'*s;
129         z1 = 1/(1-d1);
130         ls_failed = 1; % this line search failed
131     end
132
133     if exist('OCTAVE_VERSION')
134         fflush(stdout);
135     end
136 end
137 fprintf('\n');
138

```

predict_Team02.m

```

1 function p = predict_Team02(Theta1, Theta2, X)
2 %Prédire le libellé d'une entrée à partir d'un réseau de neurones formé
3
4 m = size(X, 1);
5 num_labels = size(Theta2, 1);
6
7 p = zeros(size(X, 1), 1);
8
9 h1 = sigmoid_Team02([ones(m, 1) X] * Theta1');
10 h2 = sigmoid_Team02([ones(m, 1) h1] * Theta2');
11 [dummy, p] = max(h2, [], 2);
12 end
13

```

randWeight_Team02.m

```

1 function W = randInitializeWeights(L_in, L_out)
2 %RANDINITIALIZEWEIGHTS Randomly initialize the weights of a layer with L_in
3 W = zeros(L_out, 1 + L_in);
4 EPSILON=sqrt(6)./(sqrt(L_in)+sqrt(L_out+1));
5 W = rand(L_out, 1 + L_in) * 2 * EPSILON-EPSILON;
6
7 end
8

```




```
sigmoid_Team02.m
1 function g = sigmoid_Team02(z)
2 %SIGMOID Compte sigmoid functoon
3
4
5 g = 1.0 ./ (1.0 + exp(-z));
6 end
7

sigmoidGradient_Team02.m
1 function g = sigmoidGradient(z)
2 %SIGMOIDGRADIENT retourne le gradient de le fonction sigmoid (z)
3 g = zeros(size(z));
4 g=sigmoid_Team02(z).*(1-sigmoid_Team02(z));
5
6 end
7
```


SVM

```
lab_svm_Team02.m
1 clear; close all; clc;
2 % preparation du dataset
3 data = load('PiiishingData.txt');
4 X = data(:,1:9);
5 y = data(:,10);
6
7 % Diviser nos données sur 2 (70% apprentissage et 30% test)
8 % données d'apprentissage
9 X_train = X(1:506,:);
10 y_train = y(1:506,:);
11
12 % données de test
13 X_test = X(507:724,:);
14 y_test = y(507:724,:);
15
16 % entraînement
17 model = fitcecoc(X_train, y_train);
18 % prediction
19 y_prediction = predict(model, X_test);
20 % Afficher la précision (la similarité entre y_restant et y qu'on a prédité)
21 fprintf('Précision : %f \n', mean(double(y_test == y_prediction))*100);
22
```

Arbres de décisions

```
lab_Team02.m 
1 clear; close all; clc;
2 data = load ('PiiishingData.txt');
3 x= data(1:392,1:9);
4 y = data(1:392,10);
5 tree = fitctree(x,y);
6 test = predict (tree,[-1,0,-1,-1,-1,0,1,-1,0]);
7 fprintf ('Test = %d\n', test);
8 view (tree,'mode','graph');
```

Régressin Linéaire

```
lab_Team02.m 
1 clear ; close all; clc
2 %% Load Data
3 data = load('day.txt');
4 X = data(1:511, 2:15);
5 y = data(1:511, 16);
6 m = length(y);
7 [X muu sigma] = normaliser_Team02(X);
8 X = [ones(m, 1) X];
9 % Choisir la valeur d'alpha
10 alpha = 0.025;
11 % Choisir le nombre d'itérations
12 iters = 4000;
13 % Init Theta and Run Gradient Descent
14 theta = zeros(15, 1);
15 [theta, J] = gradientDescent_Team02(X, y, theta, alpha, iters);
16 % Afficher le graphe de Cost Function
17 plot_Cost_Team02(J);
18 % Afficher la plus petite valeur de la fonction cost
19 minJ = min(J);
20 fprintf ('minJ = %f ', minJ);
21 Y_predict = X * theta;
22 X_restant = data(512:731, 2:15);
23 y_restant = data(512:731, 16);
24 % normalisation des valeurs restantes
25 mtemp = repmat(muu, 220,1);
26 sigmatemp = repmat(sigma,220,1);
27 X_restant = (X_restant - mtemp)./sigmatemp;
28 X_restant = [ones(size(X_restant,1), 1), X_restant(:,:)];
29 % Pour voir les données sortantes prédicats
30 Y_new_predict = X_restant * theta;
31 % Pour voir le taux d'erreur pour chaque donnée
32 diff = Y_new_predict - y_restant;
```

ligne: 1 col: 1 encodage: SYSTEM fin de ligne: CRLF

computeCost_Team02.m

```
1 function J = computeCost_Team02(X, y, theta)
2 m = length(y);
3 J = 0;
4 J = (1/(2*m))*sum(power((X*theta - y),2));
5 end
```

gradientDescent_Team02.m

```
1 function [theta, J] = gradientDescent_Team02(X, y, theta, alpha, iters)
2 m = length(y);
3 J = zeros(iters, 1);
4 for iter = 1:iters
5     delta = (1/m)*sum(X.*repmat((X*theta - y), 1, size(X,2)));
6     theta = (theta' - (alpha * delta))';
7     J(iter) = computeCost_Team02(X, y, theta);
8 end
9 end
```

normaliser_Team02.m

```
1 function [X_normalise, m, sigma] = normaliser_Team02(X)
2 X_normalise = X;
3 m = zeros(1, size(X, 2));
4 sigma = zeros(1, size(X, 2));
5 m = mean(X_normalise);
6 sigma = std(X_normalise);
7 tfmu = X_normalise - repmat(m,length(X_normalise),1);
8 tfstd = repmat(sigma,length(X_normalise),1);
9 X_normalise = tfmu ./ tfstd;
10 end
```

plot_Cost_Team02.m

```
1 function plot_Cost_Team02 (J_history)
2 figure;
3 plot(1:numel(J_history), J_history, '-b', 'LineWidth', 2);
4 xlabel('Itérations');
5 ylabel('J');
6 end
```

RNN

mainTeam02.m

```

1 clear; close all; clc;
2 file=importdata('day.csv',';');
3 x=file(:,2:15);%colonne 1 contient le ID et sans cosédere y
4
5 y=zeros(size(x,1),1);#output
6 nb_couche_cache=13;
7 layer1 = zeros(1,nb_couche_cache); #hidden layer array
8
9 for i=2:size(y,1)
10     y(i,1)=x(i,14);
11 end
12
13 %x(:,13:14) = [];#enlever la colonne de y de la matrice des input
14
15 for k=1:size(x,2)
16     x(:,k)=(x(:,k) - min(x(:,k))) / ( max(x(:,k)) - min(x(:,k)) );
17 end
18
19 y(:)=(y(:) - min(y(:))) / ( max(y(:)) - min(y(:)) );
20 %random weight pour layer1 et 2
21 weight1= rand(size(x,2)+1,nb_couche_cache);
22 weight2 = rand(nb_couche_cache+1,1);
23 coste=[];
24 end
25
26 y(:)=(y(:) - min(y(:))) / ( max(y(:)) - min(y(:)) );
27 %random weight pour layer1 et 2
28 weight1= rand(size(x,2)+1,nb_couche_cache);
29 weight2 = rand(nb_couche_cache+1,1);
30 coste=[];
31 x=[ones(size(x,1),1),x];
32
33 for k=1:200 #nombre iteration
34     c=[];
35     for j=1:700 #nombre dechantillon a parcourir
36         [layer1,output]= feedforward_Team02(x(j,:),weight1,weight2); #'hypothese' est calcul
37         [weight1,weight2]=backProbagation_Team02(x(j,:), layer1,weight1,weight2,y(j,:),output);
38         c= [c; (sum(y(j,:)-output)^2)];
39     end
40     coste=[coste;sum(c)];
41 end
42 coste=coste/(size(x,1)*2)

```

sigmoid_Team02.m

```

1 function s = sigmoid_Team02(x)
2
3     for i=1:size(x,2)
4         s(1,i)=1/(1+exp(-x(1,i)));
5     end
6 endfunction
7

```

* sigmoid_deriv_Team02.m

```
1 function [s] = sigmoid_deriv_Team02(x)
2     for i=1:size(x,2)
3         s(1,i)=exp(-x(1,i))/(1+exp(-x(1,i)))^2;
4     end
5
6 endfunction
7
```

feedforward_Team02.m

```
1 function [layer1,output] = feedforward_Team02(input, weight1,weight2)
2     layer1 = sigmoid_Team02(input*weight1);
3     output= sigmoid_Team02([1,layer1]*weight2);
4
5 endfunction
6
```

backProbagation_Team02.m

```
1 function [weight21,weight22] = backProbagation_Team02(input, layer1,weight1,weight2,y,output)
2     d_weight2=[1,layer1]'*(2*(y(1,:)-output).*sigmoid_deriv_Team02(output)) ;
3     d_weight1=input'*((2*(y(1,:)-output).*sigmoid_deriv_Team02(output))
4     *weight2').*sigmoid_deriv_Team02([1,layer1]);
5
6
7     weight22=weight2+d_weight2*0.01;
8     weight21=weight1+d_weight1(:,2:size(d_weight1,2))*0.01;
9 endfunction
10
```

test_Team02.m

```
1 [xx,output]= feedforward_Team02(x(50,:),weight1,weight2);
2 output
3 %pour notre reel y
4 y(50,:)
```

Bibliographie

- [1] Manickam, S. (2017, 11 05). Récupéré sur Acamedia:
<https://www.academia.edu/RegisterToDownload#RelatedPapers>
- [2] *mrmint*. (2019). Récupéré sur mrmint: <https://mrmint.fr/logistic-regression-machine-learning-introduction-simple>
- [3]. Récupéré sur <https://elitedatascience.com/machine-learning-algorithms>
- [4]. Récupéré sur github: <https://github.com/trekhleb/machine-learning-octave>
- [5]. Récupéré sur
https://www.researchgate.net/publication/226420039_Detection_of_Phishing_Attacks_A_Machine_Learning_Approach
- [6]. Récupéré sur <https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html>
- [7]. Récupéré sur BIKESHARING DATASER:
<https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset?fbclid=IwAR0DF0gjbDG4fos8AvprOxqVmyZKC1WU1hbHZn0p8Ngww-R-cg6XmpfCFgo>
- [8]. Récupéré sur <https://www.andlil.com/definition-de-regression-lineaire-132481.html>