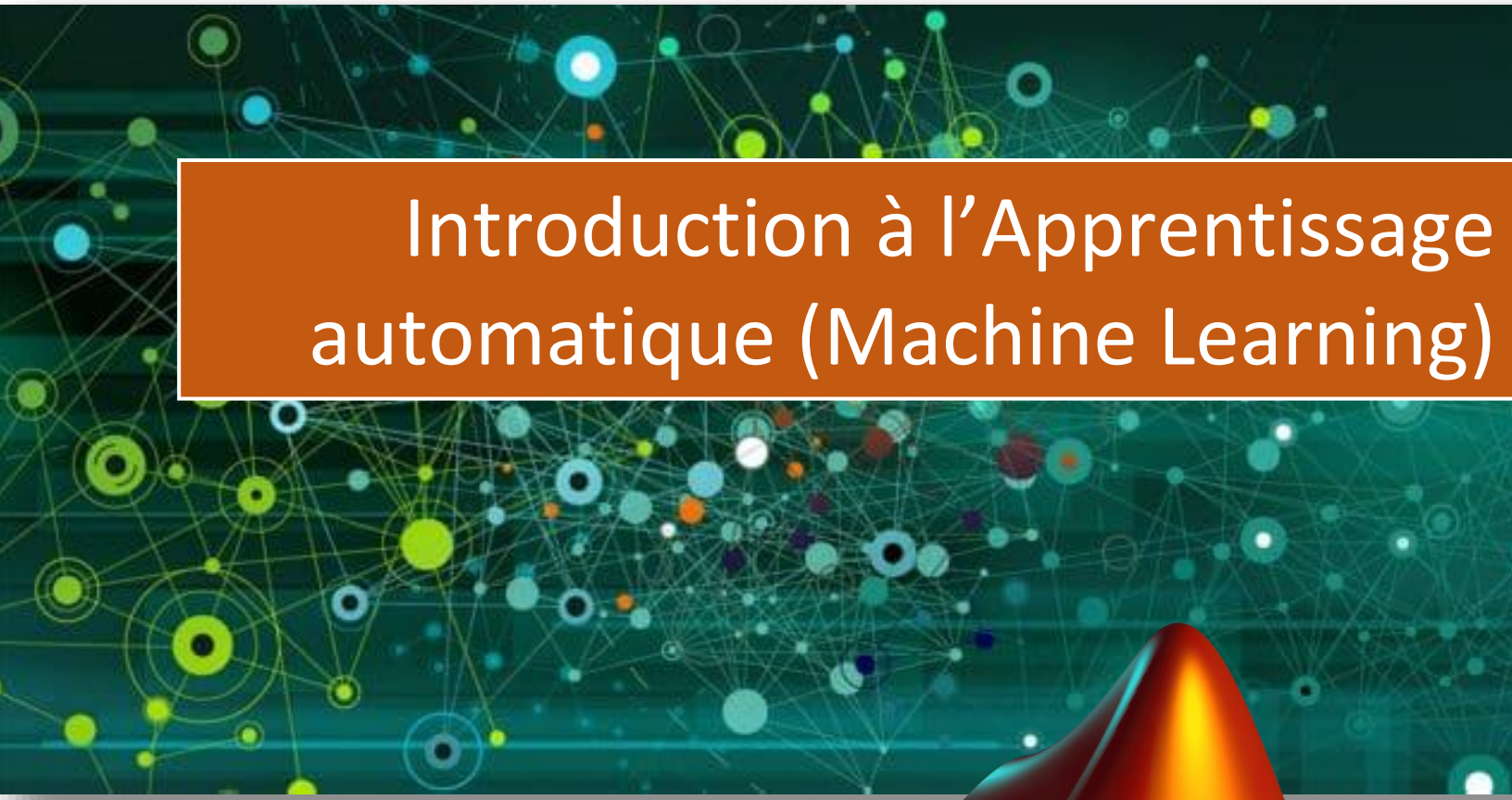
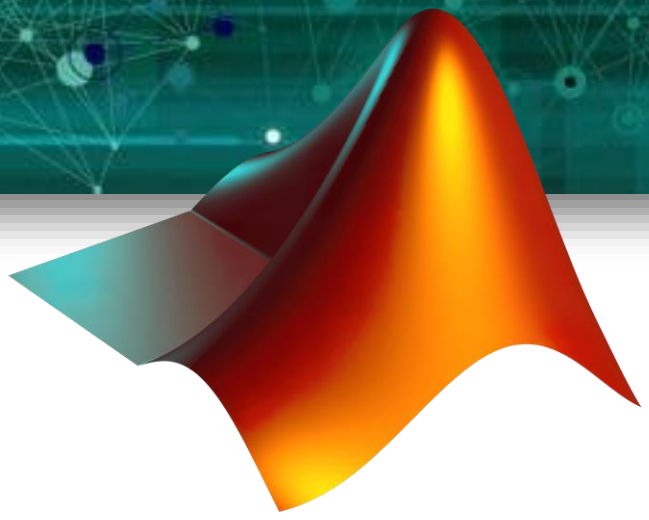
Sciences Industrielles de l'Ingénieur



Introduction à l'Apprentissage automatique (Machine Learning)



Cahier de cours dirigé interactif

Philippe Hautcoeur
PSI1
Lycée Clemenceau Nantes
philippe.hautcoeur@ac-nantes.fr

Le 11 décembre 2007, à l'occasion des 40 ans de l'Inria (Institut national de recherche en informatique et en automatique), Michel Serres a donné une conférence sur la révolution culturelle et cognitive engendrée par les nouvelles technologies. Le célèbre académicien y explicite comment la révolution informatique change notre rapport au monde. Tout comme avant elle, l'écriture, puis l'imprimerie, ont profondément transformé nos modes de vie. Une conséquence inévitable de toute révolution.

https://bit.ly/ConfMichelSerres

Table des matières

Une brève histoire de l’Intelligence Artificielle…

YouTube FR

https://bit.ly/HistoireIA

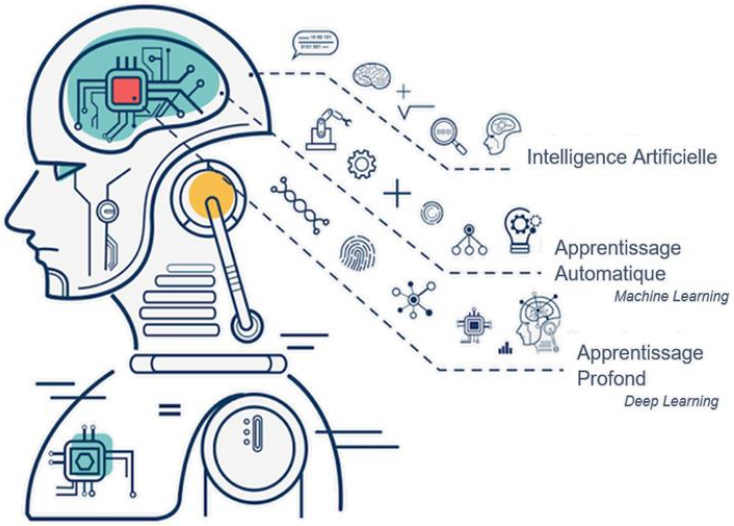
Cette vidéo est éditée par le CEA (Commissariat à l'énergie atomique et aux énergies alternatives).

Résumé…

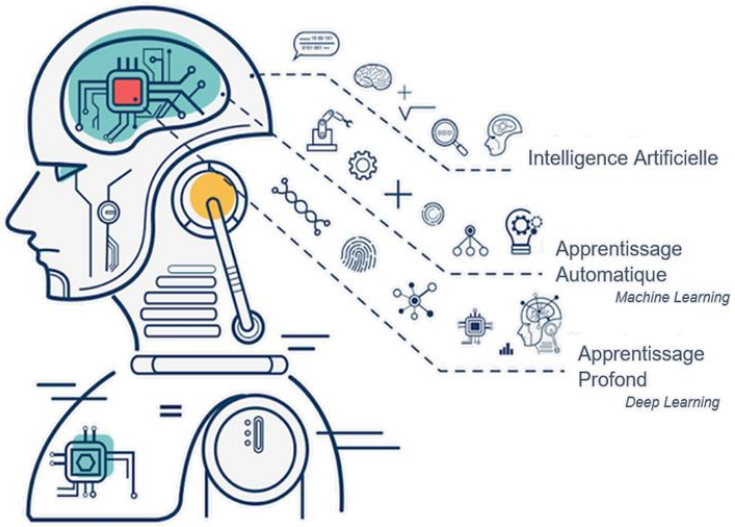
Les premières machines pensantes apparaissent dans les histoires de science-fiction dès les années 1920. En 1950, c’est au tour des scientifiques d’imaginer des machines pensantes. Le mathématicien britannique, Alan Turing, publie un article intitulé « l’ordinateur et l’intelligence », dans lequel il décrit comment savoir si une machine s'approche d'une intelligence humaine. Il appelle cette méthode « le jeu de l'imitation », plus connu aujourd'hui sous le nom de test de Turing. Voilà les prémices de l'intelligence artificielle ! Depuis, les recherches sur l'IA ont connu successivement des périodes d'essor et de gel. Dans les années 80, les investissements sur le développement de l'intelligence artificielle reprennent. Au cours de la dernière décennie, l'intelligence artificielle prend un nouveau tournant ! La puissance de calcul des ordinateurs, la capacité de stockage et l'accumulation des données augmentent de façon extraordinaire. Les améliorations techniques développent ainsi la performance des algorithmes. C'est l’ère du deep learning. Aujourd'hui, le champ des applications de l'IA est immense, mais les questions éthiques évoquées, dès les années 1970, sont plus que jamais d'actualité et doivent nous faire réfléchir au statut des futures IA. Découvrez en animation-vidéo les principaux faits marquants associés au développement des IA au fil du temps.

Chapitre 1

Introduction à l'Apprentissage Automatique



Introduction à l'Apprentissage Automatique en Sciences Industrielles de l'Ingénieur



Ce livret accompagne les différents *live scripts* construits sous Matlab dans le but de se familiariser avec les éléments de base de l'apprentissage automatique conformément au programme de Sciences de l'Ingénieur de PSI. Certains liens de ce document renvoient vers des ressources complémentaires...

Le livret comporte trois chapitres :

- **Chapitre 1 : Introduction à l'apprentissage automatique.**
- **Chapitre 2 : Introduction aux problèmes de régression.**
- **Chapitre 3 : Introduction aux problèmes de classification**

Plan du chapitre 1

1- Qu'est-ce que l'apprentissage automatique (ou le machine learning) ?

Commençons par définir l'intelligence artificielle (IA). En fait il n'existe pas vraiment de définition exacte de l'IA. Néanmoins Peter Novig et Stuart Russel dans leur ouvrage "Artificial Intelligence : A Modern Approach " Edition Pearson - 4ème édition 2020, propose de décrire l'IA comme l'étude et la conception d'agents intelligents. Un agent est une entité autonome capable de percevoir son environnement grâce à des capteurs, d'agir sur celui-ci grâce à des effecteurs. L'agent est qualifié d'intelligent s'il est en mesure d'analyser, d'apprendre, et de prendre des décisions. Chatbot, voiture autonome, système de diagnostic médical, système de reconnaissance d'images, robot humanoïde etc.…sont des agents intelligents.

Faire "apprendre" à une machine est un processus qui s'appelle « machine learning » en anglais, que nous traduisons en français par apprentissage automatique. Les méthodes employées pour l'apprentissage font appel à des sous-domaines de la science des données (Data Science).

Lien vers la 3ème édition de l'ouvrage de Peter Novig et Stuart Russel en anglais, lien vers des extraits en français.

L'apprentissage automatique est donc un domaine de l'intelligence artificielle qui fait référence à un ensemble de méthodes algorithmiques qui "apprennent" à partir de données.

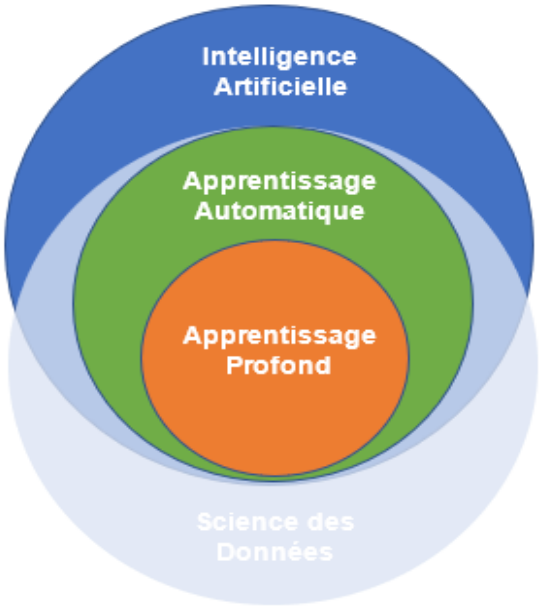


Figure 1 - L'apprentissage automatique

Enfin, comme le montre la figure précédente, l'apprentissage profond (deep learning) est un sous domaine de l'apprentissage automatique. La plupart des algorithmes d'apprentissage profond sont basés sur les réseaux de neurones artificiels.

Il existe deux grandes familles d'apprentissage automatique : l'apprentissage supervisé et l'apprentissage non-supervisé. Les algorithmes d'apprentissage supervisé nécessitent la présence de données étiquetées (données de sortie). L'apprentissage consiste alors à modéliser la relation qui existe entre les données d'entrées (les caractéristiques ou features en anglais) et les données

de sortie (étiquettes ou labels en anglais). Le modèle sert ensuite à prédire la sortie pour de nouvelles entrées, de nouveaux exemples. La régression et la classification font appel à des méthodes d'apprentissage supervisé.

En matière d'apprentissage il existe trois principaux types de problèmes traités par les méthodes suivantes :

- La régression permet la prédiction de réponses continues,
- La classification permet de classer des éléments dans des catégories discrètes connues,
- Le clustering permet d'identifier des groupes, des associations dans les jeux de données.

Le clustering n'est pas au programme de PSI.

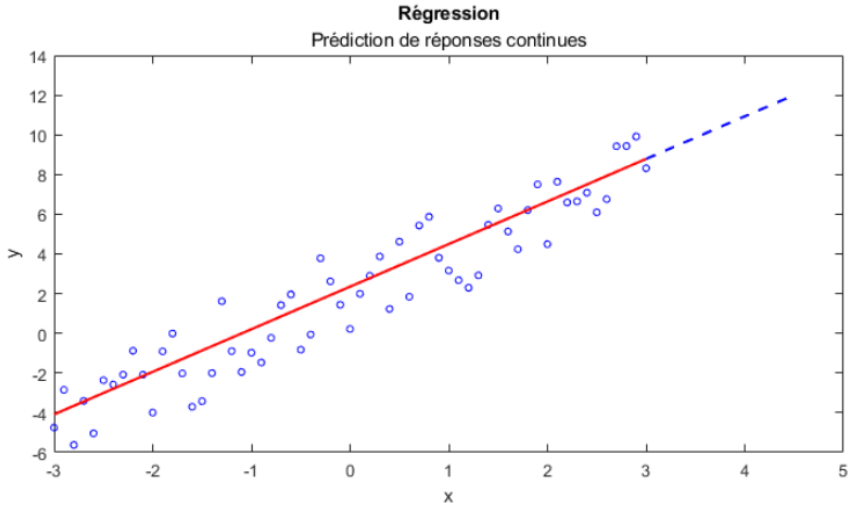


Figure 2 - La régression linéaire.

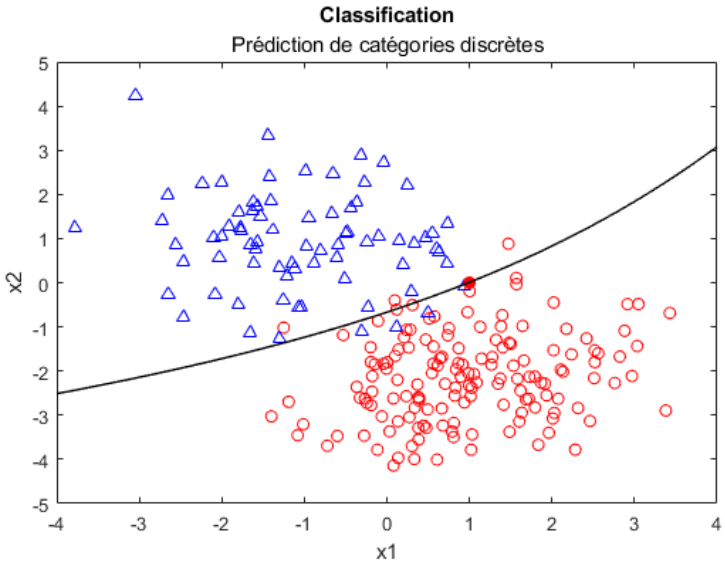


Figure 3 - La classification.

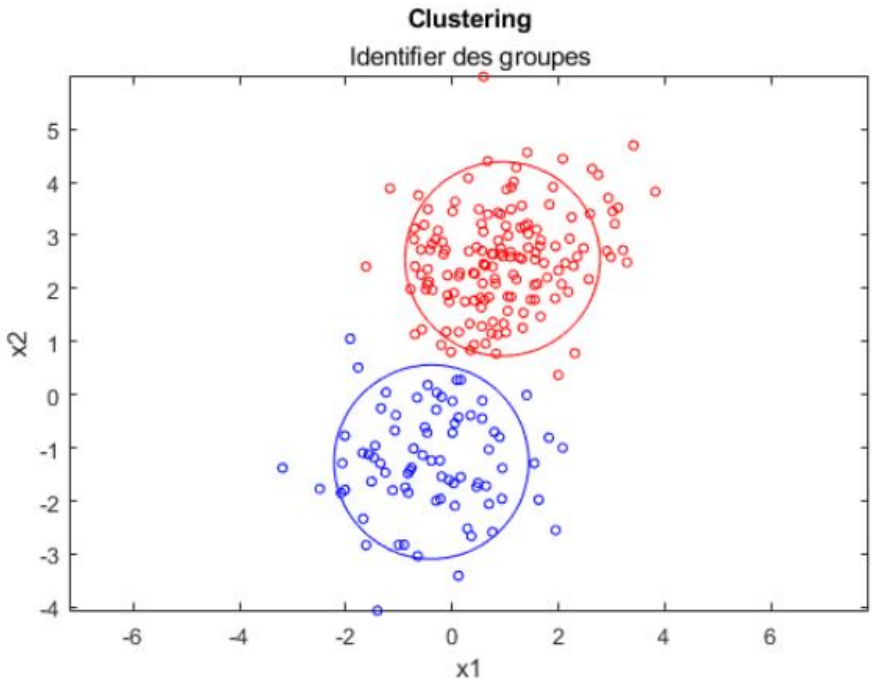


Figure 4 - Le clustering.

Les méthodes d'apprentissage non-supervisé ne nécessitent pas d'exemples étiquetés, ces méthodes sont utilisées pour le clustering. À l'intersection de ces deux grandes familles prennent naissance des méthodes d'apprentissage semi-supervisé qui traitent des données partiellement étiquetées.

L'apprentissage non-supervisé et l'apprentissage semi-supervisé ne sont pas au programme de PSI.

2- Apprentissage automatique : démarche mise en œuvre pour apporter une solution IA à un problème donné

Le synoptique qui suit précise la démarche générale mise en œuvre pour apporter une solution IA à un problème donné :

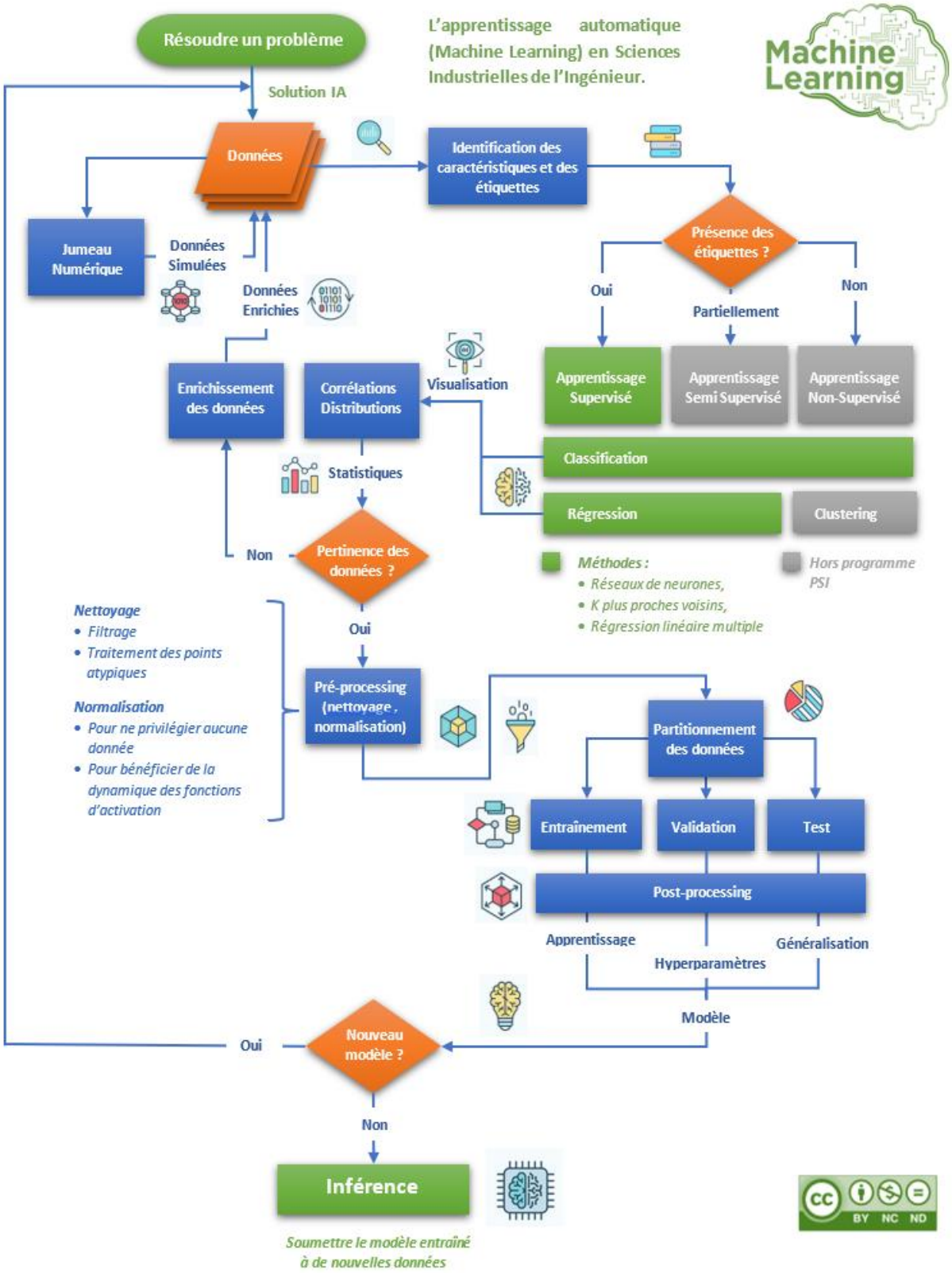


Figure 5 – L'apprentissage automatique.

3- Éléments du programme de PSI

ANALYSER les principes de l'intelligence artificielle

- Régression et classification
- Apprentissage supervisé et non supervisé
- Phases d'apprentissage et d'inférence
- Modèle linéaire monovariable ou multivariable
- Réseaux de neurones (couches d'entrée, cachée et de sortie, neurones, biais, poids et fonction d'activation)

CHOISIR une démarche de résolution d'un problème d'ingénierie numérique ou d'intelligence artificielle

- Décomposition d'un problème complexe en sous-problèmes simples. Choix des algorithmes (réseaux de neurones, k plus proches voisins et régression linéaire multiple)

RESOUDRE un problème en utilisant une solution d'intelligence artificielle

- Apprentissage supervisé
- Choix des données d'apprentissage
- Mise en œuvre d'algorithmes (réseaux de neurones, k plus proches voisins, régression linéaire multiple)
- Phases d'apprentissage et inférence

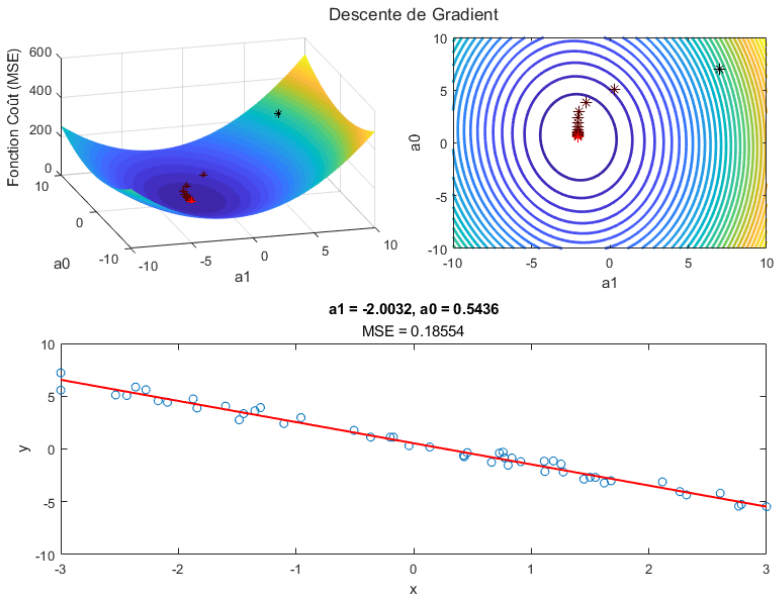
4- Pour aller plus loin

Voici quelques liens qui pointent vers des ressources Matlab sur l'apprentissage automatique :

- Synthèse des modèles de régression et de classification les plus courants.
- Machine Learning Onramp est un tutoriel gratuit d'environ 2 heures qui propose une introduction interactive aux méthodes de classification.
- Introduction à l'apprentissage automatique cette page contient 4 vidéos en anglais qui présentent les fondamentaux de l'apprentissage automatique (machine learning).
- Machine Learning with MATLAB est un document de synthèse en anglais édité par MathWorks qui reprend pas à pas les bases de l'apprentissage automatique.

Chapitre 2

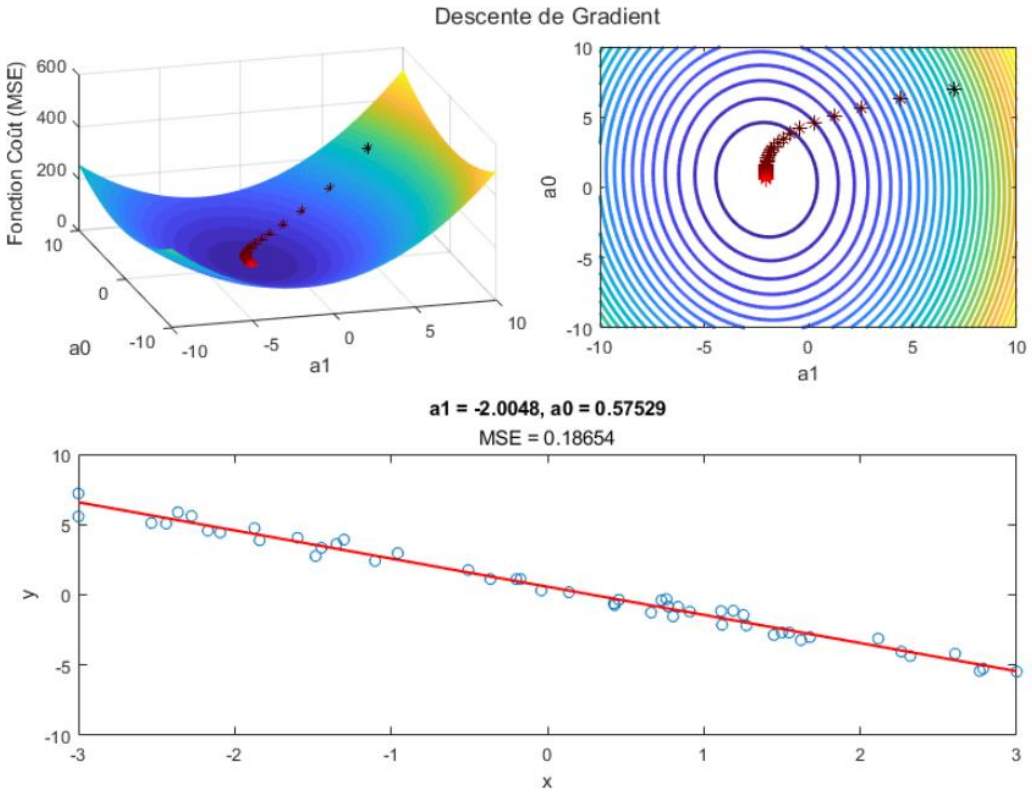
Introduction aux problèmes de régression



Introduction aux problèmes de régression

Approche analytique par la méthode des moindres carrés ordinaires.

Approche algorithmique par la méthode de la descente de gradients.



Plan du Chapitre 2

Dans ce document deux types d'activités sont proposées :

 **Interaction** : vous interagissez avec Matlab en saisissant des éléments ou encore en agissant sur des éléments de contôle.

 **Observation - Réflexion** : vous analysez les résultats obtenus suite aux différentes interactions.

L'objectif de ce *live script* est de se familiariser avec la notion de régression rencontrée en statistiques ou dans le cadre de l'apprentissage automatique.

Les éléments de contrôle des *live scripts* n'apparaissent pas sur ce document

1- Qu'est-ce qu'une régression ?

En sciences statistiques ou encore en apprentissage automatique, le terme régression fait référence aux techniques d'estimation de la relation entre des variables quantitatives : une variable dépendante ou expliquée et une ou plusieurs variables indépendantes ou explicatives. C'est donc une méthode de modélisation d'une réponse, la variable dépendante ou expliquée, en fonction de prédicteurs, les variables indépendantes.

La régression est très largement utilisée pour :

- Expliquer ou quantifier une relation entre des variables,
- Réaliser des prédictions de réponses continues à partir de données.

En apprentissage automatique, la méthode de régression linéaire est considérée comme une méthode d'apprentissage supervisé utilisée pour prédire une variable quantitative.

Il existe différents types de régression : la régression linéaire et non linéaire, la régression simple et multivariables.

Pour plus de précisions sur la régression vous pouvez consulter cette page dont s'inspire ce document.

2- La régression linéaire simple

Prenons un exemple pour illustrer le propos.

L'incubateur de votre école vous a permis de développer votre start up dans le domaine de l'IA. Une levée de fonds inattendue vous permet désormais de vous lancer à plus grande échelle. Pour cela vous devez disposer d'une infrastructure.

Supposez que vous vouliez estimer le prix d'achat d'un plateau pour installer votre entreprise. Vous décidez de vous installer dans une petite ville de province. A priori le prix du bien dépendra de la ville choisie, du quartier et de la surface du plateau.

Pour une ville et un quartier donné on dispose des données suivantes : le prix de vente du plateau et la superficie du plateau.

On va chercher à modéliser la relation qui existe entre ces deux variables dans le but d'estimer le budget nécessaire à cet investissement.

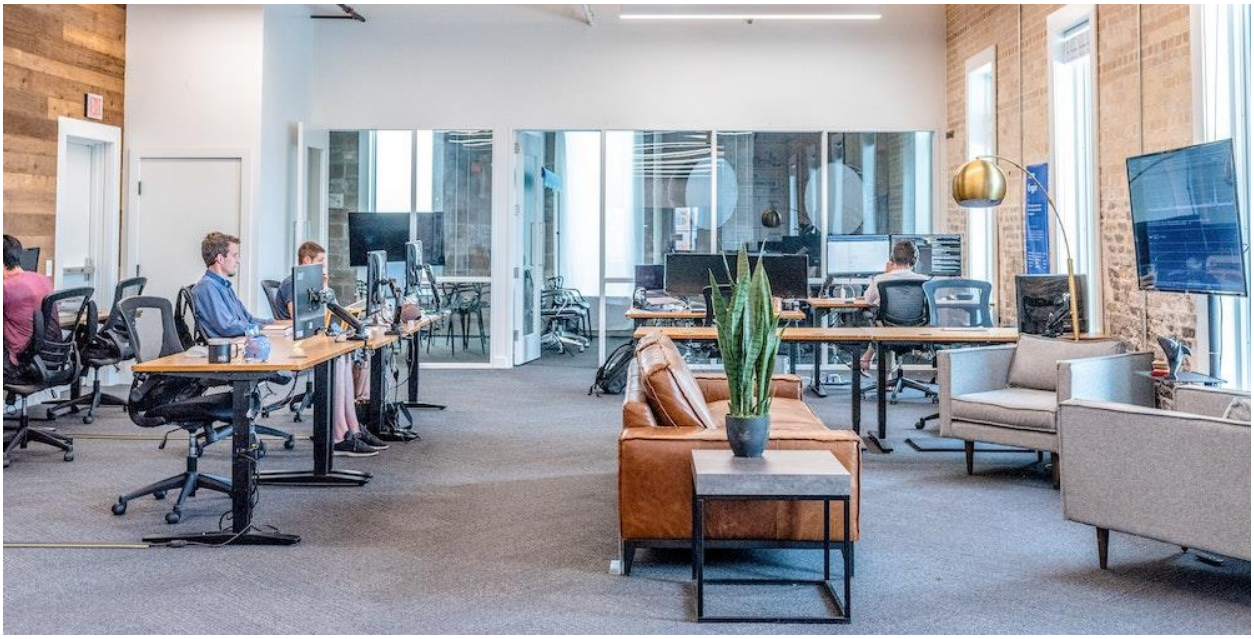


Figure 6 – Exemple de plateau aménagé.

Interaction

- *Cliquez sur le bouton* ***Charger et visualiser les données***

```
load linearData.mat x y
figure
scatter(x,y)
xlabel("Superficie (m²)")
ylabel("Prix de vente (€)")
title("Prix de vente vs Superficie")
```

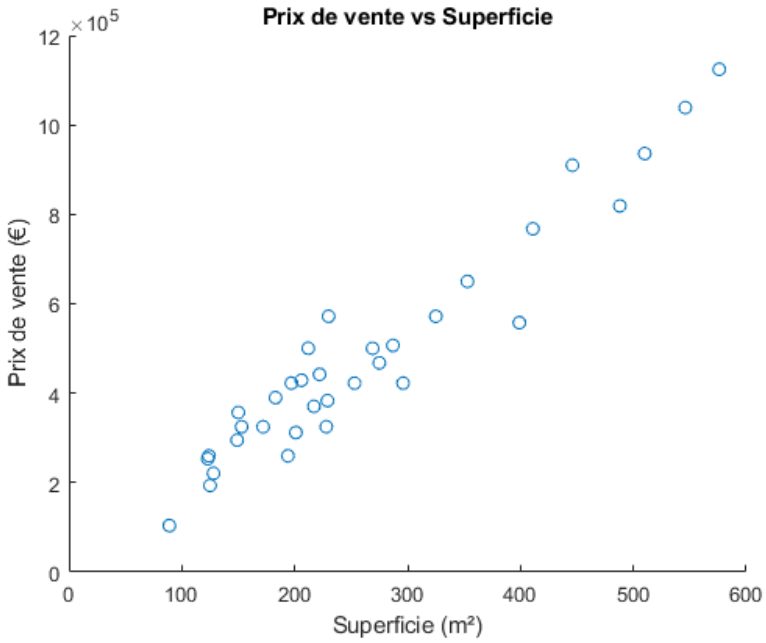


Figure 7 - Visualisation des données.

Observation - Réflexion

- *À partir de ce graphique, voyez-vous se dégager une tendance lorsqu'on affiche le prix de vente en fonction de la superficie ?*

Dans cet exemple, on dispose d'une variable dépendante (le prix) et d'une variable indépendante (la superficie) qui semblent être liées linéairement.

Nous allons traiter un problème de régression linéaire simple qui a pour objectif de faire passer une droite entre ces points. La relation entre la variable dépendante y (le prix) et la variable indépendante x (la superficie) sera modélisée par une droite d'équation : $y = a_0 + a_1 x$

L'objectif est donc la détermination des coefficients de cette expression pour que le modèle s'ajuste au mieux au nuage de points.

Interaction

- *Cliquez sur le bouton **Tracer** pour tracer une droite de pente a_1 et d'ordonnée à l'origine a_0 qui se superpose aux données :*

```
load linearData.mat x y
a1 =1850;
a0 =6900;
plotFit(x,y,[a1 a0],0);
```

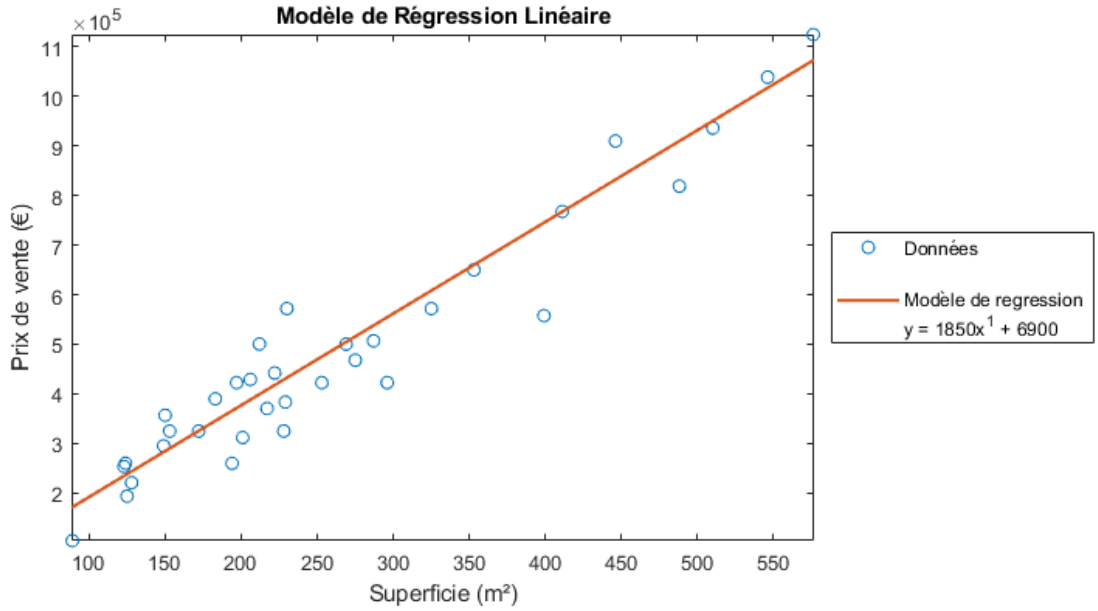


Figure 8 – Modèle de régression linéaire.

Interaction

- *Utiliser les curseurs pour adapter la pente a_1 et l'ordonnée à l'origine a_0*
- *Trouver les valeurs de a_1 et a_0 qui vous paraissent ajuster au mieux le modèle aux données.*

Observation - Réflexion

- *L'ajustement de la droite vous semble-t-il aisé ?*
- *Quelle stratégie peut-on mettre en œuvre pour obtenir ce qui pourrait être le meilleur modèle de régression ?*

Interaction

- *Vous pouvez comparer votre modèle au modèle calculé par Matlab en cliquant sur le bouton **Solution.***
- *Pouvez-vous estimé le prix d'un plateau de 200 m²*

```
f = openfig('linearSoln.fig');
f2=figure("Position",[0 0 800 400]);
set(f.Children(2),'Parent',f2);
```

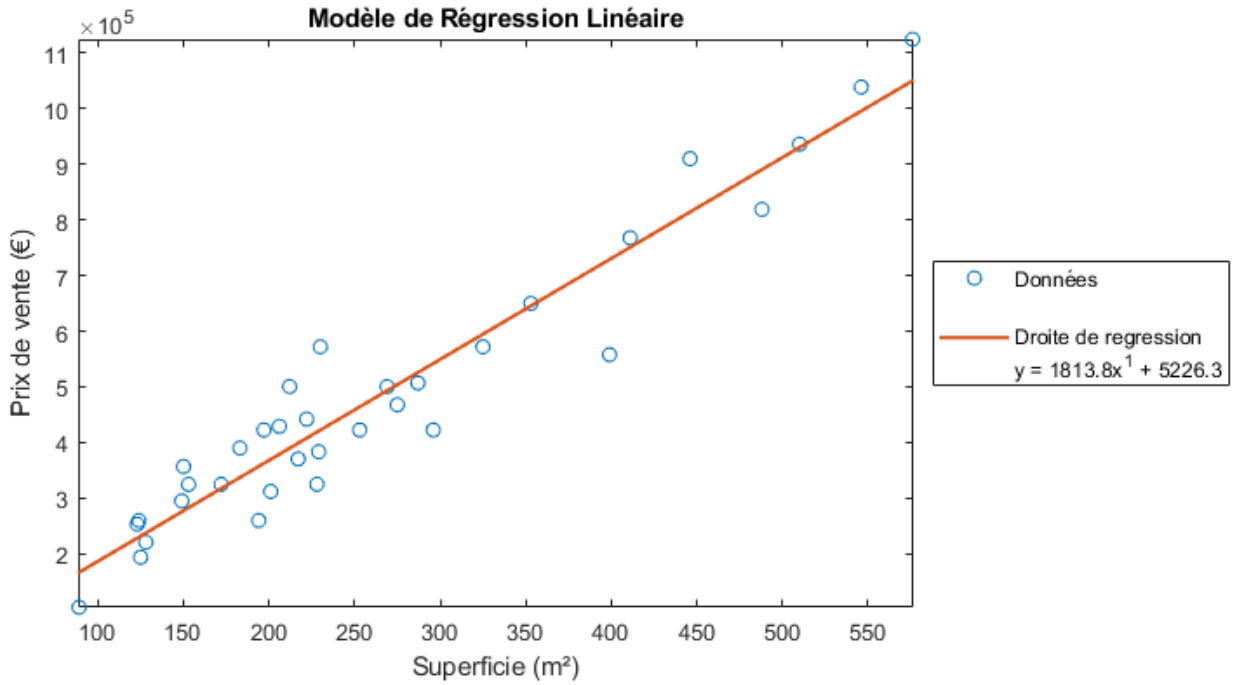


Figure 9 - Modèle de régression optimisé.

```
newSurf=200
```

```
newSurf = 200
```

```
prix=1813.8*newSurf+5226.2
```

```
prix = 3.6799e+05
```

Observation - Réflexion

- *Votre stratégie d'ajustement est-elle la bonne ?*
- *Si non, quelles idées vous permettraient de l'améliorer ?*

3- La méthode des moindres carrés ordinaires

L'objectif de la régression linéaire est de trouver le meilleur modèle qui décrit la variable dépendante en fonction de la variable ou des variables indépendantes.

Une méthode pour ajuster au mieux le modèle est de calculer la somme des erreurs quadratiques, c'est à dire la somme des erreurs entre les valeurs courantes et prédites de la variable dépendante, et d'élever cette somme au carré (Sum of Squared Errors: SSE en anglais) :

$$\mathrm{SSE} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

y_i est la valeur courante de la variable dépendante y correspondant au $i^{\text{ème}}$ élement sur n

$\hat{y}_i$ est la valeur prédite de la variable dépendante y correspondant au $i^{\text{ème}}$ élement sur n

Interaction

- *Cliquez sur le bouton **Afficher les erreurs** pour afficher les barres d'erreur et la somme des erreurs au carré.*

```
load linearData.mat x y
a1 = 1813.8;
a0 = 5226.3;
plotFit(x,y,[a1 a0],1);
```

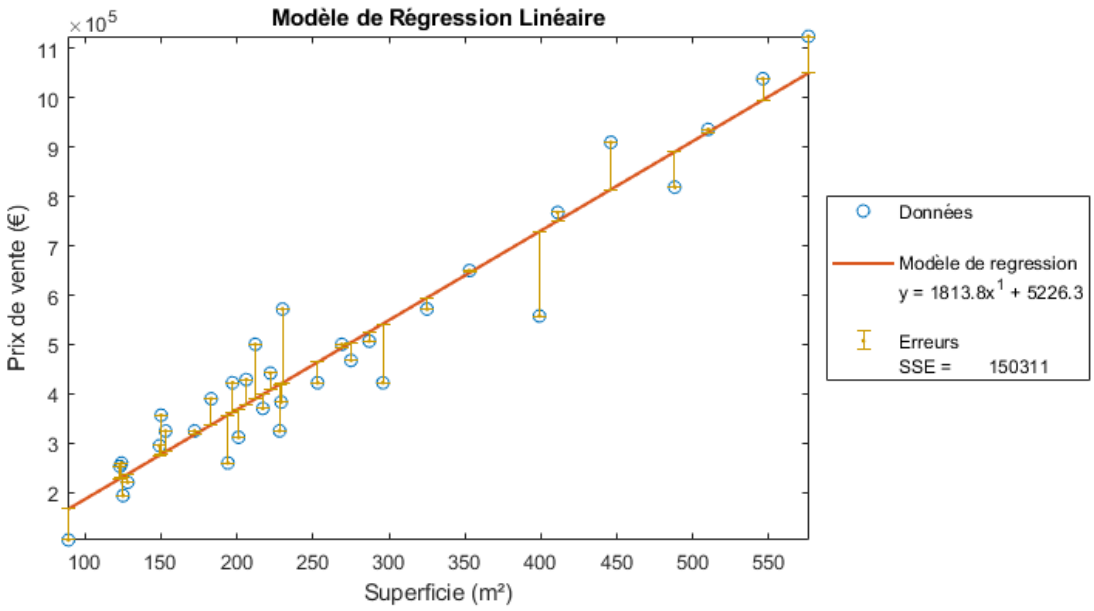


Figure 10 - Modèle de régression et erreurs.

En remplaçant $\hat{y}_i$ par son expression, on va chercher à minimiser la somme des erreurs quadratiques, c'est la méthode des moindres carrés ordinaires :

$$\min_{a_0,a_1} \sum_{i=1}^{n} (y_i - (a_0 + a_1 x_i))^2$$

Nous pouvons exprimer a_1 et a_0 en remarquant que les minima sont obtenus en écrivant que la dérivée de la somme des erreurs par rapport à chacun des paramètres est nulle :

$$\frac{\partial \sum_{i=1}^{n} (y_i - (a_0 + a_1 x_i))^2}{\partial a_0} = 0 \quad \text{et} \quad \frac{\partial \sum_{i=1}^{n} (y_i - (a_0 + a_1 x_i))^2}{\partial a_1} = 0$$

on en déduit alors :

$$a_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$$a_0 = \bar{y} - a_1 \bar{x}$$

où

- $\bar{x}$ est la moyenne des x_i
- $\bar{y}$ est la moyenne des y_i
- i est l'indice de l' $i^{\text{ème}}$ élément parmi n

4- Qualité de la prédiction : coefficient de détermination et coefficient de corrélation

Il existe différents critères pour définir la qualité de la prédiction.

On introduit les éléments suivants :

- $\mathrm{SSE} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$, la somme des carrés des erreurs (de prédiction) ou somme résiduelle des carrés (Sum of Squared Errors)
- $\mathrm{SSR} = \sum_{i=1}^{n} \left(\hat{y}_i - \overline{y}\right)^2$, la somme de régression des carrés, ou somme des carrés due à la régression, ou somme expliquée des carrés (Sum of Squares due to Regression).
- $\mathrm{SST} = \mathrm{SSE} + \mathrm{SSR} = \sum_{i=1}^{n} (y_i - \overline{y})^2$, somme totale des carrés (Sum of Squares Total).

Le but de la régression est de construire un modèle de prédiction affine $y = a_0 + a_1 x$.

Pour apprécier la qualité de ce modèle, il convient de construire $x = a'_0 + a'_1 y$. En effet la prédiction sera parfaite si les deux modèles sont l'inverse l'un de l'autre.

Ce qui donnerait un produit des pentes unitaire : $a_1 . a'_1 = 1$.

On appelle ce produit *coefficient de détermination*, noté R^2. Ce coefficient varie entre 0 et 1. Plus la valeur est proche de 1, meilleure est la qualité de la prédiction.

Aussi, on montre que :

$$R^2 = \frac{\mathrm{SSE}}{\mathrm{SST}} = \frac{\mathrm{SST} - \mathrm{SSR}}{\mathrm{SST}} = \frac{\sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2} = 1 - \frac{\sum_{i=1}^{n} \left(\hat{y}_i - \overline{y}\right)^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2}$$

Il existe un autre critère d'appréciation de la qualité de la prédiction : $R = \sqrt{R^2}$, appelé coefficient de corrélation.

Interaction

- *Retrouver les coefficients a_1, a_0 et R^2 en saisissant le code correspondant aux expressions précédentes dans le champ ci-dessous. Conclure.*

```
xbar = mean(x)
```

```
xbar = 266.2571
```

```
ybar = mean(y)
```

```
ybar = 4.8815e+05
```

```
ypred=a0+a1.*x;

a1=sum((x-xbar).*(y-ybar))./sum((x-xbar).^2)
```

```
a1 = 1.8138e+03
```

```
a0=ybar-a1*xbar
```

```
a0 = 5.2263e+03
```

```
R2=1-(sum((y-ypred).^2)/sum((y-ybar).^2))
```

```
R2 = 0.9242
```

Interaction

- *Retrouver les coefficients a_1, a_0 en utilisant la fonction `polyfit` de matlab. Saisir la fonction dans le champ ci-dessous.*
- *Retrouver les caractéristiques du modèle en utilisant la fonction `fitlm` de matlab. Saisir la fonction dans le champ ci-dessous. Cliquer sur le bouton **Lancer le calcul.***

```
p=polyfit(x,y,1)
```

```
p = 1×2
10^3 ×
    1.8138    5.2263
```

```
mdl=fitlm(x,y)
```

```
mdl =
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate      SE       tStat      pValue
                   ________    ______    _______    _________

    (Intercept)     5226.3      26698    0.19575        0.846
    x1              1813.8     90.418      20.06    4.672e-20


Number of observations: 35, Error degrees of freedom: 33
Root Mean Squared Error: 6.83e+04
```

```
R-squared: 0.924,  Adjusted R-Squared: 0.922
   F-statistic vs. constant model: 402, p-value = 4.67e-20
```

5- La régression linéaire multiple

Nous allons enrichir l'exemple précédent en ajoutant une autre variable explicative, en complément de la superficie du bien. Par exemple, nous pouvons retenir le nombre de caractéristiques additionnelles qui améliorent le confort des habitants (nombre de pièces, sanitaires, climatisation, ascenseur, terrasse, etc…)

Interaction

- *Cliquer sur le bouton **Charger et Afficher** les données.*

```
load multivariateData.mat x1 x2 y
figure;
scatter3(x1,x2,y,"o")
xlabel("Nombre de caractéristiques additionnelles (#)")
ylabel("Superficie (sqft)")
zlabel("Prix de vente (Livres)")
title("Prix vs Superficie vs Caractéristiques")
view([-150 20])
```

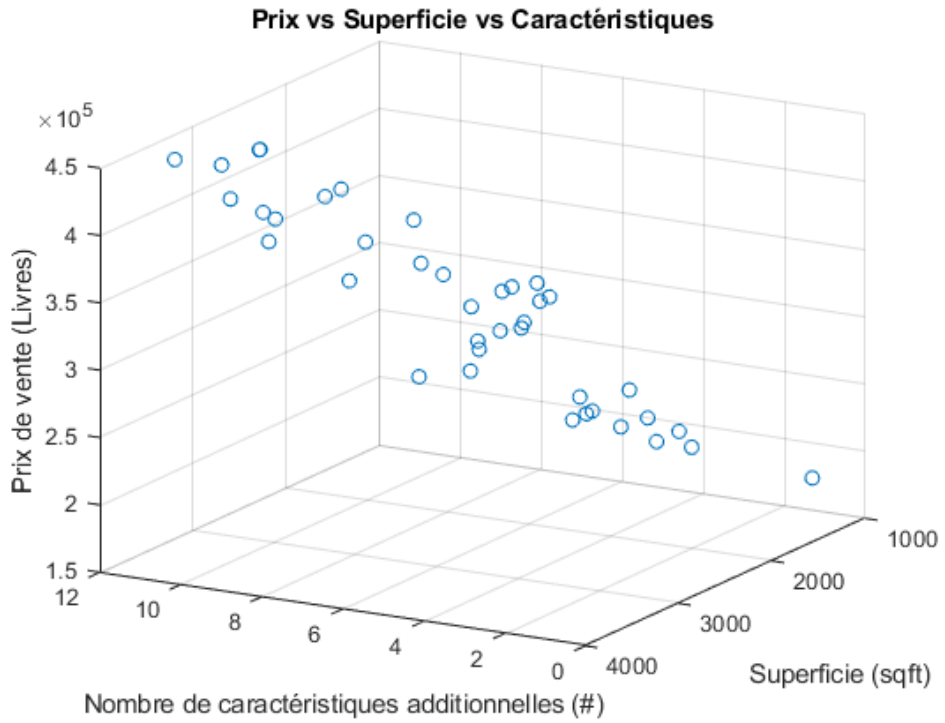


Figure 11 - Visualisation des données.

Observation - Réflexion

- *Constatez-vous une tendance ? Utilisez les commandes* ***Rotate*** *et* ***Pan*** *qui apparaissent en haut à gauche de la figure.*

Une manière d'observer les tendances est de tracer chacune des variables en fonction des autres variables.

Interaction

- *Construire la matrice X qui synthétise les prédicteurs* x_1 *et* x_2
- *Tracer les graphes donnant* x_1 *et* x_2 *en fonction de y. On utilisera la fonction* `plotmatrix` *de Matlab.*

```
X=[x1 x2];
plotmatrix(X,y)
```

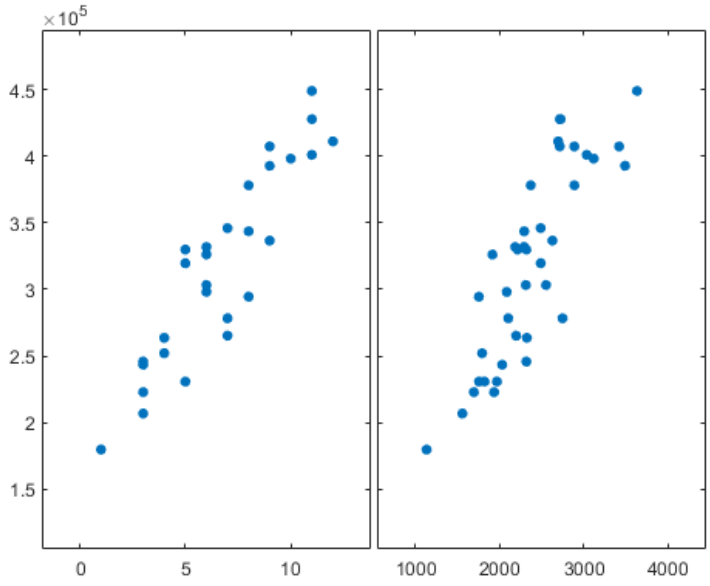


Figure 12 – Tracé de chacune des variables en fonction d'une autre.

Observation - Réflexion

- *Constatez-vous une tendance ?*

Dans cet exemple, nous avons une variable dépendante et deux variables indépendantes. Pour construire notre modèle de prédiction, nous pouvons appliquer un modèle de régression linéaire multiple.

Le modèle aura alors l'expression suivante :

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \dots$$

La normalisation des données

Il s'agit d'un prétraitement (ou preprocessing) des données. Nous constatons ici que les échelles de grandeurs de ces données sont très différentes les unes des autres. De manière générale, ces différences peuvent nuire à la convergence des algorithmes utilisés en apprentissage automatique. Il faut donc procéder à une mise à l'échelle des données, les faire varier de 0 à 1 ou de -1 à 1.

Par exemple :

- La normalisation minMax : $x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{Max}} - x_{\text{min}}}$ dans ce cas x_{norm} varie entre 0 et 1.

La distribution des données est aussi un facteur important à considérer. En effet les lois statistiques classiques sont fondées sur la distribution normale. Par conséquent la qualité des résultats donnés par les algorithmes dépendra de la distribution des données traitées. Par exemple :

- La normalisation standard (ou standardisation) : $x_{\text{norm}} = \frac{x - \bar{x}}{\sigma}$ dans ce cas x_{norm} à une moyenne centrée en 0 et un écart type de 1 (on retrouve ici l'expression du z-score vu dans votre cours de physique sur les mesures et incertitudes)

Par conséquent normaliser des données c'est les mettre à l'échelle et/ ou les centrer.

Après traitement des données normalisées, il faudra procéder à l'opération inverse pour une bonne interprétation des résultats. C'est l'étape de post-traitement ou postprocessing.

Dans notre exemple, pour faciliter le calcul des différents coefficients, nous allons procéder à une normalisation standard des différentes variables.

Pour ce faire on utilise la fonction `normalize`. Les variables ont alors une moyenne centrée en 0 et un écart type de 1.

Interaction

- *Charger les données. En utilisant la fonction* `normalize` *procéder à la normalisation des variables x2 and y en complétant les champs suivants.*
- *Comparer les distributions des variables non normalisées et normalisées en utilisant la fonction* `histogram`

```
load multivariateData.mat x1 x2 y
figure("Position",[0 0 400 300])
x1norm =normalize(x1,'range',[0 1]);
x2norm =normalize(x2,'range',[0 1]);
ynorm = normalize(y,'range',[0 1]);
histogram(x2norm)
```

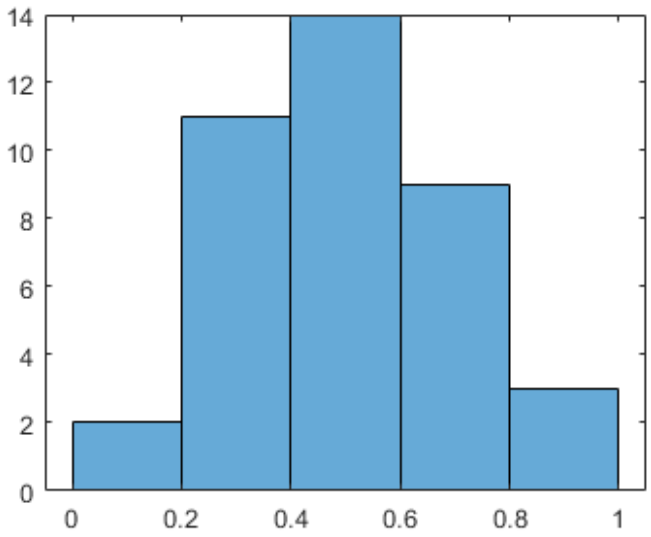


Figure 13 - Données normalisées.

Interaction

- *Construire la matrice Xnorm qui synthétise les prédicteurs* $x_{1\text{norm}}$ *et* $x_{2\text{norm}}$
- *Tracer les graphes donnant* $x_{1\text{norm}}$ *et* $x_{2\text{norm}}$ *en fonction de* y_{norm} *. On utilisera la fonction* `plotmatrix` *de matlab.*

```
Xnorm=[x1norm  x2norm];
plotmatrix(Xnorm,ynorm)
```

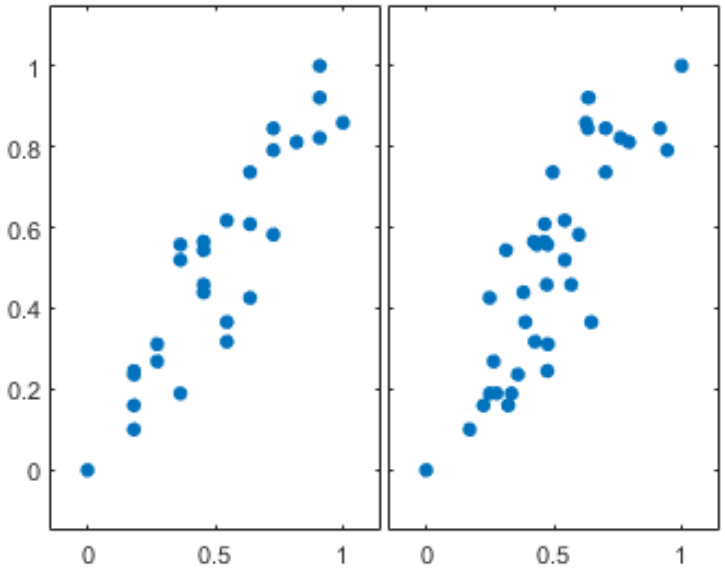


Figure 14 – Graphes donnant $x_{1\text{norm}}$ et $x_{2\text{norm}}$ en fonction de y_{norm}

Interaction

- *Cliquer sur le bouton **Tracer** pour générer un plan avec les coefficients a_0, a_1, a_2 donnés ci-dessous.*
- *Utiliser les curseurs pour régler les coefficients a_0, a_1, a_2 pour que le plan de régression s'ajuste au mieux aux données. Utilisez les commandes **Rotate** et **Pan** qui apparaissent en haut à gauche de la figure.*

```
a0 =0.028;
a1 =0.46;
a2 =0.37;
figure("Position",[0 0 800 400]);
plotMultiFit(x1norm,x2norm,ynorm,[a0 a1 a2]);
```

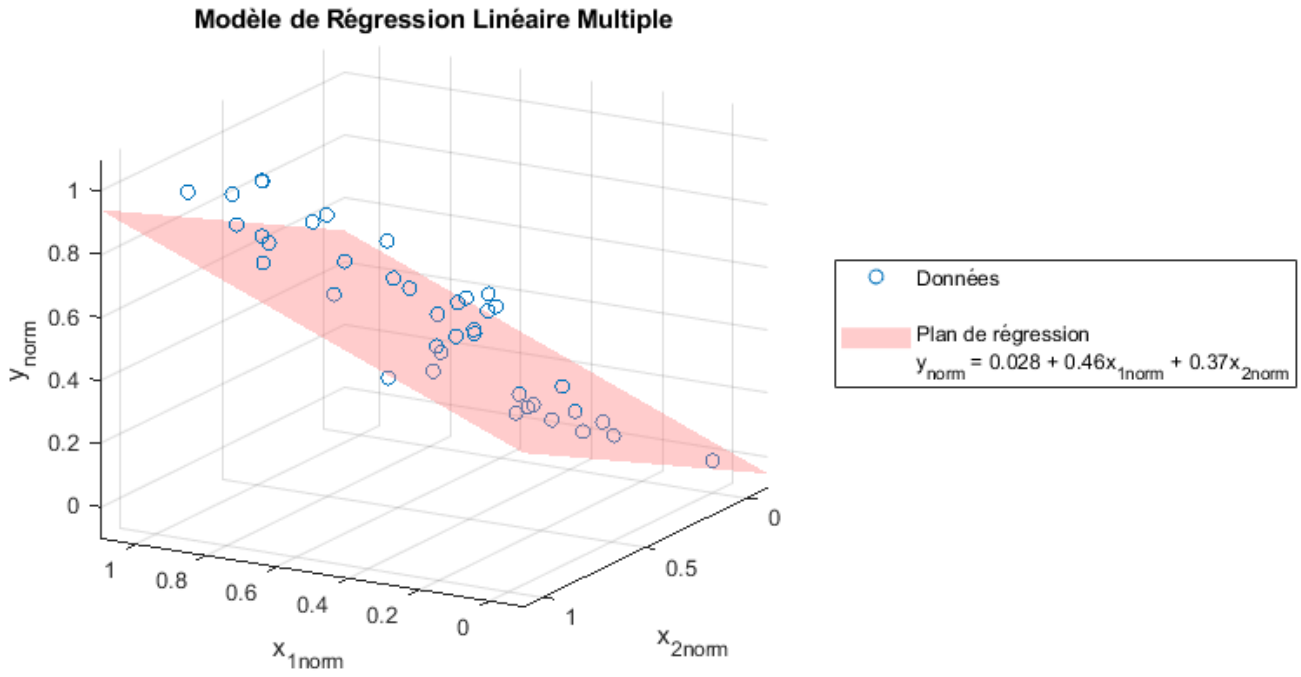


Figure 15 - Plan de régression.

Observation - Réflexion

- *L'ajustement a-t-il été aisé ?*
- *Comment avez-vous procédé pour déterminer les meilleurs coefficients ?*

Les coefficients a_0, a_1, a_2 peuvent être calculés pour que le plan s'ajuste au mieux aux données normalisées grâce à la fonction *fitlm* vue précédemment.

Interaction

- *Construire la matrice* `Xnorm` *qui synthétise les prédicteurs* $x_{1\text{norm}}$ *et* $x_{2\text{norm}}$
- *Déterminer les coefficients* a_0, a_1, a_2*. Cliquer sur* ***Lancer le calcul*** *et conclure sur la qualité du modèle.*

```
Xnorm=[x1norm x2norm]
```

```
Xnorm = 39×2
           0           0
      0.1818      0.1703
      0.1818      0.3210
      0.1818      0.2246
      0.3636      0.3343
      0.3636      0.2752
      0.3636      0.2495
      0.1818      0.3594
      0.1818      0.4743
      0.2727      0.2641
        ⋮
```

```
mdl=fitlm(Xnorm,ynorm)
```

```
mdl =
Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate        SE        tStat       pValue
                   ________    ________    _______    __________

    (Intercept)    -0.056945    0.042086    -1.3531       0.18447
    x1               0.66596      0.1058     6.2946    2.8126e-07
    x2               0.46031     0.12064     3.8155    0.00051462


Number of observations: 39, Error degrees of freedom: 36
Root Mean Squared Error: 0.102
R-squared: 0.864,  Adjusted R-Squared: 0.857
   F-statistic vs. constant model: 114, p-value = 2.51e-16
```

Interaction

- *Ajuster le plan compte tenu des résultats*

```
a0 =-0.057;
a1 =0.66;
a2 =0.46;
figure("Position",[0 0 800 400]);
plotMultiFit(x1norm,x2norm,ynorm,[a0 a1 a2]);
```

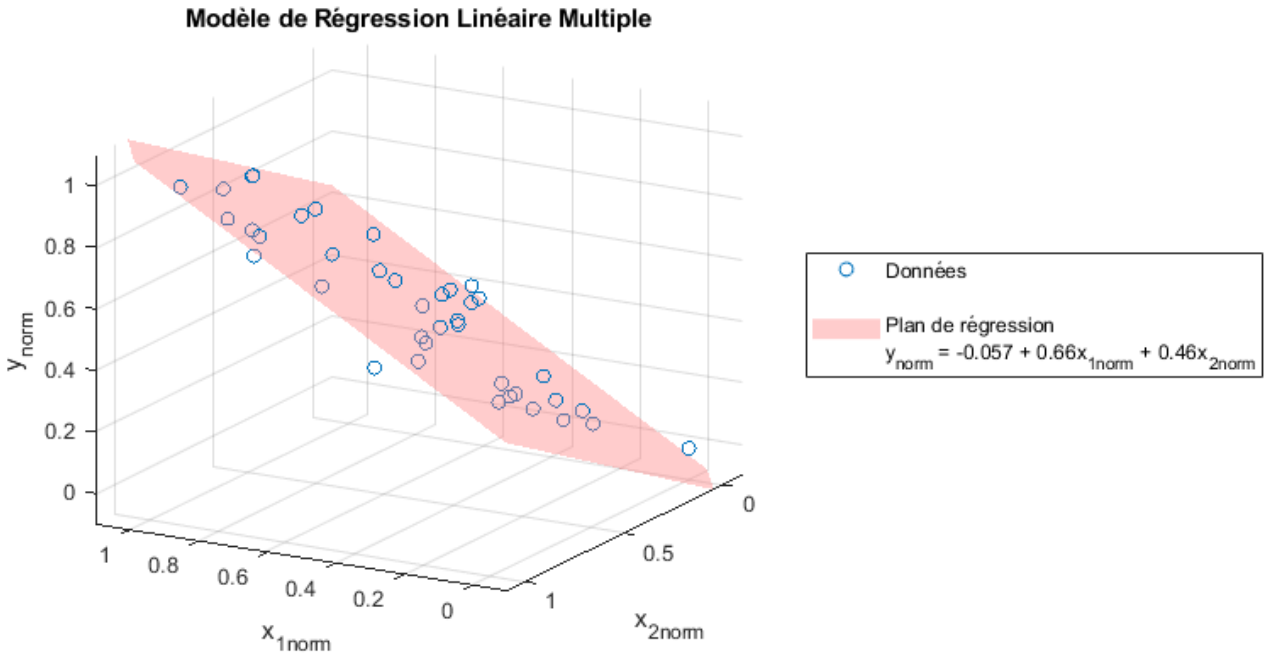


Figure 16 - Plan de régression ajusté.

Observation - Réflexion

- *L'estimation d'un bien est-elle possible à ce stade ?*
- *Quelle démarche doit-on mettre en œuvre pour estimer un bien ?*

Interaction

- *Calculer la moyenne et l'écart type de chaque variable en cliquant sur le bouton ci-dessous:*

```
load multivariateData.mat x1 x2 y
x1bar= mean(x1)
```

```
x1bar = 6.6923
```

```
x2bar= mean(x2)
```

```
x2bar = 2.3753e+03
```

```
ybar= mean(y)
```

```
ybar = 3.1887e+05
```

```
sigmax1= std(x1)
```

```
sigmax1 = 2.7256
```

```
sigmax2= std(x2)
```

```
sigmax2 = 541.9297
```

```
sigmay= std(y)
```

```
sigmay = 7.2641e+04
```

Observation - Réflexion

- *Estimer le prix d'un bien de 100 m² qui dispose de 6 éléments de confort.*

6- Cas particulier de la régression polynômiale

Comme nous l'avons vu précédemment, le modèle linéaire de régression multiple a pour expression:

$$y = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + \dots$$

Les variables explicatives sont x_1, x_2, x_3, ...

Dans le cas de la régression polynomiale, le modèle est donc un polynôme :

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots = a_0 + a_1X_1 + a_2X_2 + a_3X_3 + \dots$$

La forme de cette expression est donc identique à la précédente où on peut considérer x, x^2, x^3, ... comme les variables explicatives. C'est en ce sens qu'on considèrera que la régression polynômiale est un cas particulier de la régression linéaire multiple.

Dans cette expression les coefficients a_i sont les paramètres du modèle. Le degré du polynôme choisi comme modèle est un hyperparamètre. De façon générale en apprentissage automatique, les hyperparamètres sont fixés avant le traitement (régression, classification) alors que les paramètres s'ajustent au cours du traitement.

Prenons un exemple. On souhaite analyser l'impact de la salinité du sol sur le rendement de la culture du blé dans une zone agricole définie. On considère alors la salinité du sol comme la variable indépendante et le rendement comme la variable dépendante. La salinité s'exprime en déci Siemens par mètre (dS/m) elle est évaluée en mesurant la conductivité électrique du sol. Le rendement s'exprime en grammes par mètre au carré (g/m^2). Il correspond à la masse du blé recueilli au mètre carré.

L'objectif est donc de modéliser la relation qui existe entre ces deux variables.

Figure 17 – Rendement d'un champ de blé en fonction de la salinité du sol.

Interaction

- *Visualiser les données en cliquant sur le bouton **Charger et Afficher***

```
load nonlinearData.mat x y
figure;
scatter(x,y)
xlabel("Salinité (dS/m)")
ylabel("Rendement (g/m^{2})")
title("Rendement vs Salinité")
```

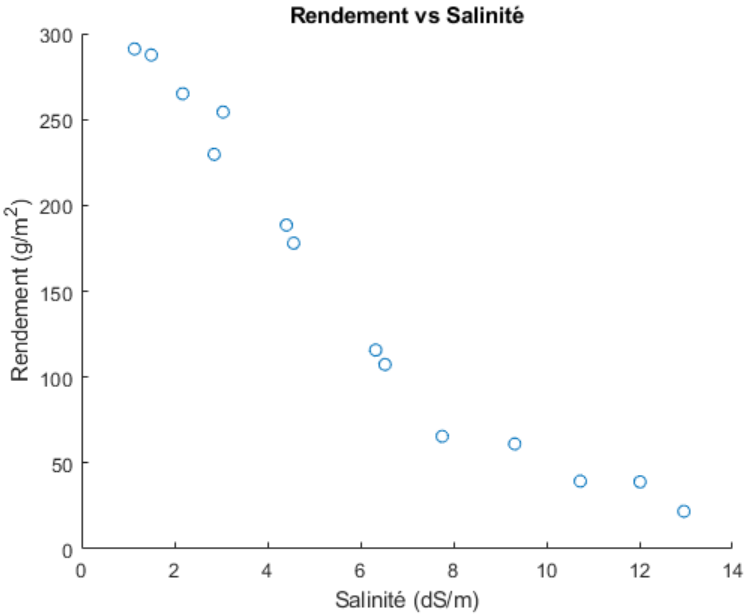


Figure 18 - Visualisation des données.

Observation - Réflexion

- *Quelle tendance se dégage de ce graphique ?*

Sur ce graphique apparait une certaine courbure que nous souhaitons prendre en compte dans le modèle. Pour ce faire nous allons considérer un modèle polynômial dont l'expression est la suivante

$$y = a_0 + a_1 x + a_2 x^2$$

Le modèle retenu est un polynôme de degré 2 (hyperparamètre). Les coefficients a_0, a_1, *et* a_2 sont les paramètres que nous allons ajuster.

Interaction

- *Cliquer sur le bouton **Tracer** pour ajuster un modèle polynômial du second degré aux données.*
- *Utiliser les curseurs pour ajuster les coefficients* a_0, a_1, *et* a_2 .

```
load nonlinearData.mat x y
a0 = 361.6;
a1 =-49.8;
a2 =1.84;

figure("Position",[0 0 800 400])
plotFit(x,y,[a2 a1 a0],0);
xlabel("Salinité (dS/m)")
ylabel("Rendement (g/m^{2})")
title("Rendement vs Salinité")
```

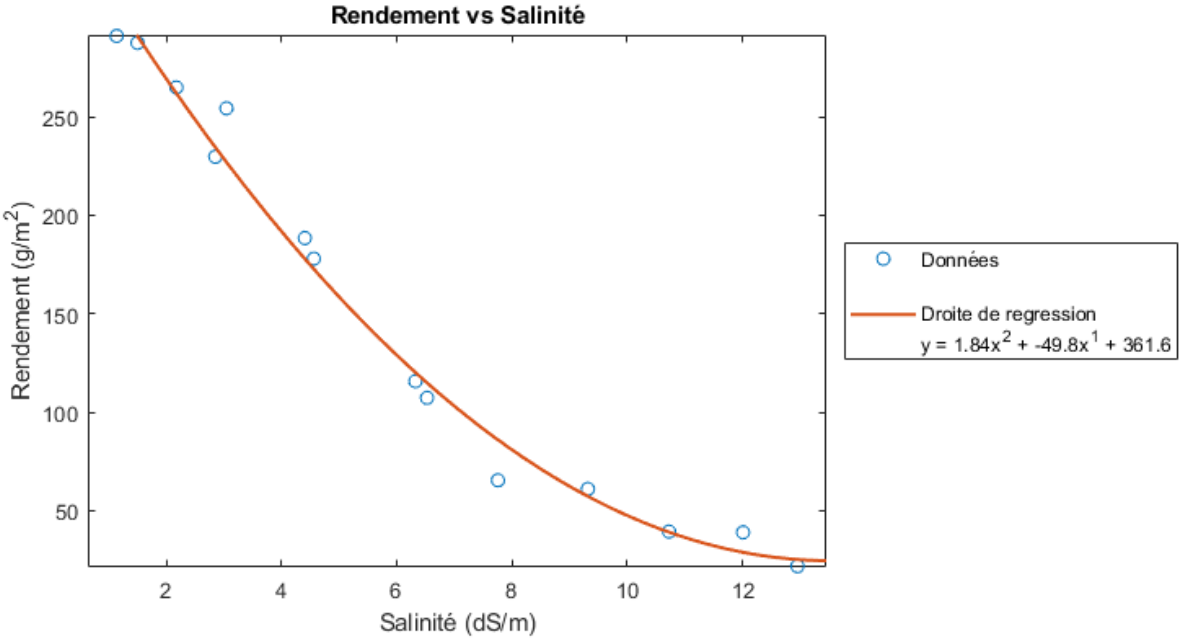


Figure 19 - Modèle polynômial du second degré.

Interaction

- *Saisir la fonction retenue pour le modèle (modelFun), affecter la valeur initiale 1 aux coefficients (beta0), afficher les caractéristiques du modèle en utilisant la fonction fitnlm de Matlab. Cliquer sur le bouton* ***Calculer.***
- *Relever les coefficients* a_0, a_1, *et* a_2 *ajustés. Tester sur la figure précédente.*
- *Que dire de la qualité du modèle retenu ?*

```
modelFun=@(a,x)a(1)+a(2)*x+a(3)*x.^2;
beta0=[1 1 1];
mdl=fitnlm(x,y,modelFun,beta0)
```

```
mdl =
Nonlinear regression model:
    y ~ a1 + a2*x + a3*x^2

Estimated Coefficients:
          Estimate      SE        tStat       pValue
          ________    _______    _______    __________

    a1      361.6      11.178      32.35    2.9338e-12
    a2     -49.807      3.992    -12.477    7.7854e-08
    a3      1.8412     0.28263     6.5145    4.3406e-05


Number of observations: 14, Error degrees of freedom: 11
Root Mean Squared Error: 12.9
R-Squared: 0.986,  Adjusted R-Squared 0.984
   F-statistic vs. constant model: 389, p-value = 6.23e-11
```

Observation - Réflexion

- *Pouvez-vous prédire le rendement d'un terrain dont la salinité est de 5 dS/m?*

- *Est-ce que l'expression qui suit* $y = a_0 + a_1\sin(x) + a_2x^2 + a_3\log(x)$ *décrit un modèle de regression linéaire entre les variables* x *et* y*? Pourquoi ?*

7- La régression non linéaire

Il existe une multitude de modèles qui pourraient s'ajuster aux données précédentes. Dans ce paragraphe, nous allons proposer une autre modélisation pour estimer le rendement d'un terrain en fonction de sa salinité. Nous allons considérer un modèle qui décrit une courbe en S inversé :

$$y = \frac{Y_{\max}}{1 + \left(\frac{x}{a}\right)^b}$$

$Y_{\max}$ est le rendement maximum pour lequel la salinité du terrain n'a aucun effet.

Ce modèle n'est pas linéaire compte tenu des définitions vues précédemment.

Pour une meilleure interprétation des résultats nous allons considérer le rendement relatif :

$$y_{\text{rel}} = \frac{y}{Y_{\max}} = \frac{1}{1 + \left(\frac{x}{a}\right)^b}$$

Le rendement relatif varie de 0 à 100%.

Interaction

- *Cliquer sur le bouton* ***Charger les données****.*

```
load nonlinearData.mat x y
```

Interaction

- *Saisir l'expression du modèle retenu dans le champ ci-dessous.*
- *Utiliser les curseurs pour ajuster les valeurs des paramètres* a et b.
- *Cliquer sur le bouton* ***Tracer*** *pour afficher les données et le modèle en S inversé.*

```
yRel = y./max(y);
mdl = @(x,a,b) 1./(1+(x./a).^b);
```

```
a = 5.4;
b = 2.7;

figure("Position",[0 0 800 400])
plotNlinFit(x,yRel,a,b,mdl);
```

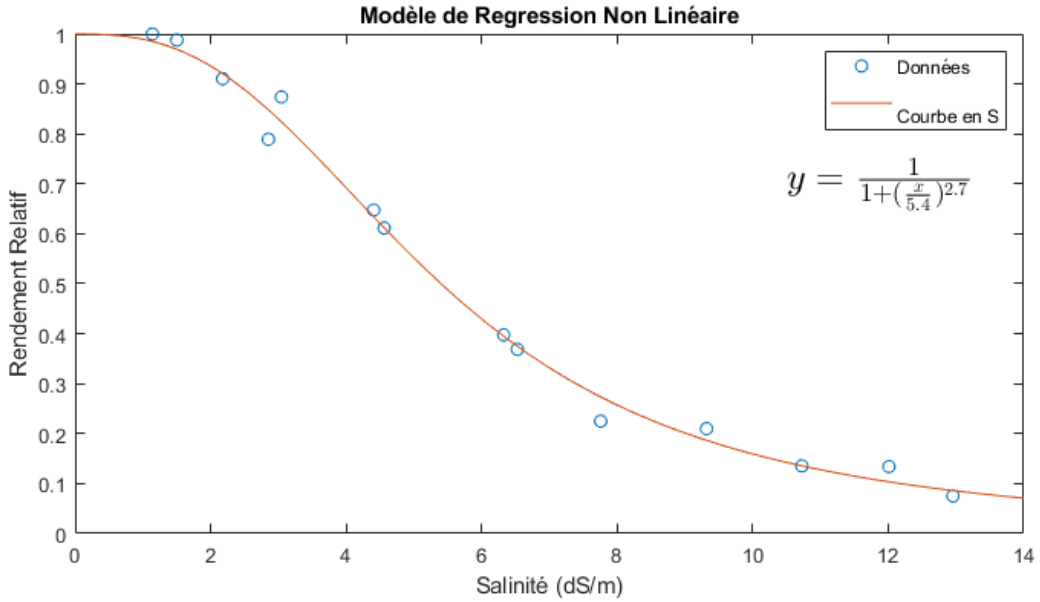


Figure 20 - Rendement relatif en fonction de la salinité du sol.

Interaction

- *En vous inspirant des activités précédentes caractériser le modèle et analyser sa qualité.*

```
modelFun=@(a,x)1./(1+(x./a(1)).^a(2));
beta0=[1 1 ];
mdl=fitnlm(x,yRel,modelFun,beta0)
```

```
mdl =
Nonlinear regression model:
    y ~ 1/(1 + (x/a1)^a2)

Estimated Coefficients:
          Estimate      SE       tStat       pValue
          ________    _______    ______    __________

    a1     5.3887     0.10086    53.427    1.2165e-15
    a2     2.7052     0.12807    21.122    7.3663e-11


Number of observations: 14, Error degrees of freedom: 12
Root Mean Squared Error: 0.03
R-Squared: 0.993,  Adjusted R-Squared 0.992
   F-statistic vs. zero model: 3.01e+03, p-value = 6.17e-17
```

Observation - Réflexion

- *Pouvez-vous prédire le rendement d'un terrain dont la salinité est de 5 dS/m? Comparer cette prédiction à celle obtenue avec un modèle polynômial de degré 2.*

8- La fonction coût

Dans les activités précédentes vous avez utilisé des curseurs pour ajuster les valeurs des paramètres des modèles retenus. Aussi nous avons utilisé des fonctions Matlab qui calculent ces paramètres pour ajuster au mieux les modèles aux données. Pour calculer ces paramètres, il convient de minimiser une fonction qu'on appelle fonction objectif ou encore fonction coût. La régression est donc un problème d'optimisation dont le but sera de minimiser une fonction coût pour déterminer les paramètres du modèle retenu.

Précédemment, nous avons introduit la somme des carrés des erreurs (SSE) pour évaluer la qualité de l'ajustement du modèle. Un autre indicateur est la moyenne de la somme des carrés des erreurs (MSE: Mean Squared Error) :

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$$

- y_i est la valeur courante de la variable dépendante y correspondant au $i^{\text{ème}}$ élément sur n
- $\hat{y}_i$ est la valeur prédite de la variable dépendante y correspondant au $i^{\text{ème}}$ élément sur n

Dans le cadre de la régression, nous définirons la moyenne de la somme des carrés des erreurs comme la fonction coût.

La fonction coût s'écrit :

$$J(\theta_1, \theta_2, \ldots) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(x_i, \theta_1, \theta_2, \ldots)\right)^2$$

- $\theta_1, \theta_2, \ldots$ sont les paramètres du modèle
- f est la fonction qui décrit le modèle, qui relie les variables dépendantes et indépendantes.

L'objectif est de minimiser la fonction coût :

$$\min_{\theta_1,\theta_2,\cdots} \sum_{i=1}^{n} J(\theta_1, \theta_2, \ldots)$$

Dans la méthode des moindres carrés ordinaires nous cherchions à minimiser la somme des carrés des erreurs (SSE) pour trouver les paramètres. Cette approche analytique n'est pas toujours possible pour des modèles plus complexes que ceux que nous venons de traiter.

Le but de l'activité qui suit est de mettre en œuvre une démarche itérative pour déterminer les coefficients du modèle.

Interaction

- *Cliquer sur le bouton **Charger les données***

```
load linearData2.mat x y
```

- *Construire un modèle de régression linéaire pour estimer y en ajustant les valeurs des coefficients. Vous devez vous servir de la valeur de la fonction coût pour adapter votre modèle de manière itérative. Cliquer sur le bouton **Tracer**.*

```
a1 =-2;
a0 =0.5;
figure("Position",[0 0 800 400])
yhat = plotFit(x,y,[a1 a0],0);
MSE = mean((y-yhat).^2);
xlabel("x")
ylabel("y")
text(-1,6,"MSE = " + MSE,"FontSize",14)
```

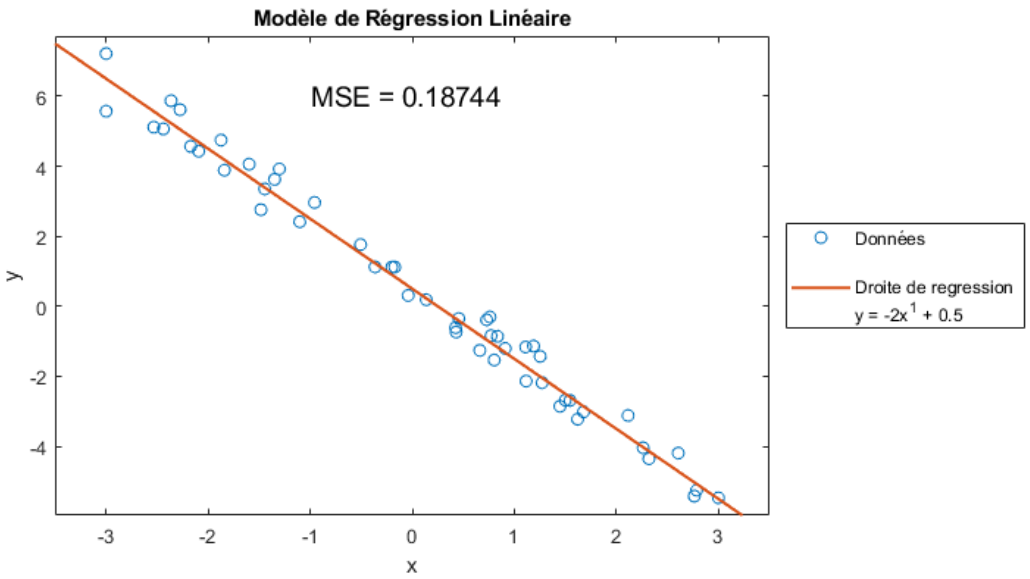


Figure 21 - Construction du modèle de régression linéaire.

Observation - Réflexion

- *Quelle a été votre stratégie ?*

- *Vérifier la qualité du modèle que vous venez de construire*

```
mdl=fitlm(x,y)
```

```
mdl =
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE        tStat       pValue
                   ________    ________    _______    __________

    (Intercept)     0.5436     0.062258     8.7315    1.7677e-11
    x1             -2.0032     0.037154    -53.917    1.3144e-44


Number of observations: 50, Error degrees of freedom: 48
Root Mean Squared Error: 0.44
R-squared: 0.984,  Adjusted R-Squared: 0.983
   F-statistic vs. constant model: 2.91e+03, p-value = 1.31e-44
```

9- La descente de gradient

La descente de gradient est une méthode algorithmique qui minimise de manière itérative la fonction coût afin de déterminer les paramètres d'un modèle de régression. Comme son nom l'indique la méthode s'appuie sur les gradients de la fonction coût, c'est à dire les dérivées partielles de la fonction par rapport à chaque paramètre du modèle. Les paramètres sont ajustés à chaque itération en soustrayant les valeurs des dérivées partielles respectives de leur valeur courante.

Dans notre cas en considérant MSE comme fonction coût et un modèle de régression linéaire simple, si on trace MSE en fonction d'un paramètre, la courbe ainsi obtenue est convexe et admet un seul minimum. Ce minimum sera obtenu lorsque la dérivée de la fonction coût par rapport au paramètre s'annule (voir l'animation qui suit)

Les équations des gradients sont les suivantes :

$$\frac{\partial(f)}{\partial a_1} = \frac{2\alpha}{n}\sum_{i=1}^{n} x_i(\hat{y}_i - y_i) \qquad \frac{\partial(f)}{\partial a_0} = \frac{2\alpha}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)$$

Où α représente le taux d'apprentissage (learning rate)

Interaction

- *Cliquer sur le bouton **Charger les données***

```
load linearData2.mat x y
```

- *Calculer les coefficients en utilisant la fonction Matlab `polyfit` ainsi que MSE*

```
p=polyfit(x,y,1);
a1=p(1)
a0=p(2)
yhat =a0 + a1.*x;
MSE = mean((y-yhat).^2)
```

Observons l'évolution de l'algorithme de descente de gradient :

- *Cliquer sur le bouton **Animer** pour observer l'évolution de la descente de gradient en fonction du taux d'apprentissage et du nombre de pas de calculs.*

```
tauxApp = 0.13;
nbrPas = 50;

figure("Position",[0 0 800 600])
[a1,a0,mse] = gradientDescent(x,y,nbrPas,tauxApp);
```

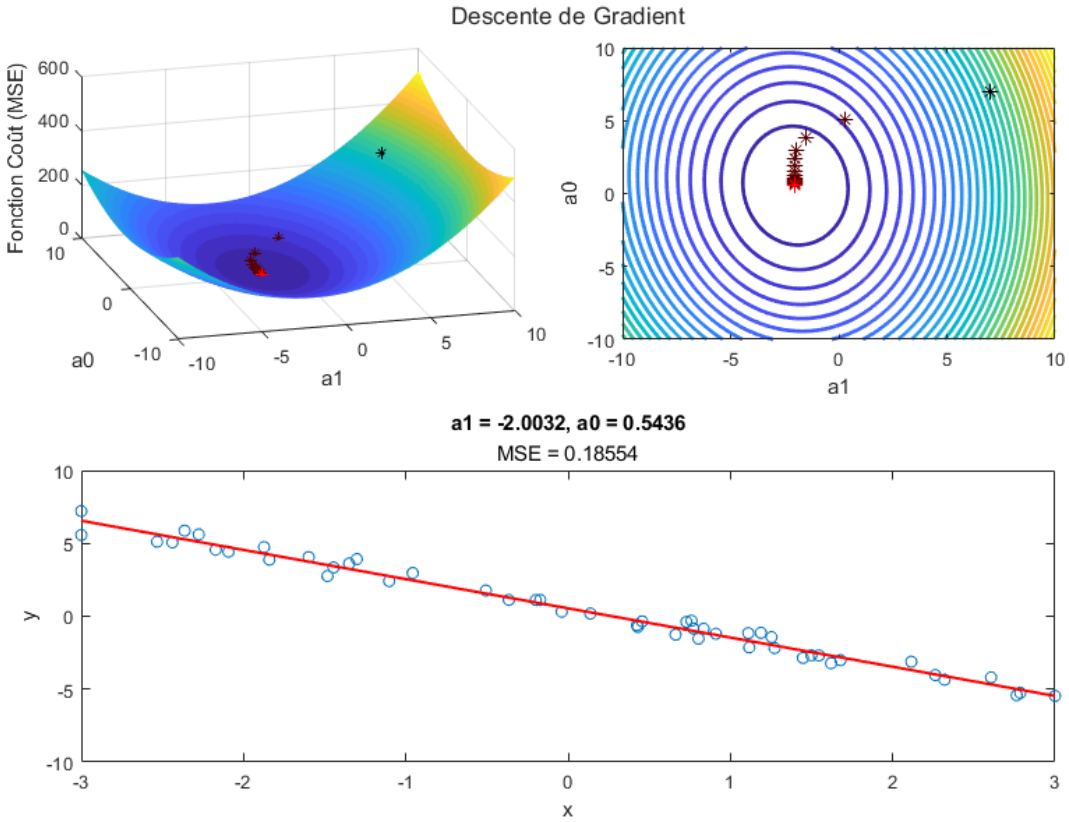


Figure 22 - Descente de gradient.

Ici nous considérons un cas simple avec seulement deux paramètres à calculer. On remarquera que les variables sont normalisées (mises à l'échelle et centrées en 0)

La représentation tridimensionnelle de la fonction coût en fonction des deux paramètres ressemble à une "vallée" avec un minimum (point rouge) à atteindre, le fond de la "vallée". Rappelez-vous, l'objectif est d'atteindre ce point.

En commençant à un point initial (à partir d'un des coins supérieurs par exemple) à préciser, l'algorithme de descente de gradient calcule la direction à prendre pour descendre la "vallée" et atteindre le minimum. Ce calcul est mené en plusieurs pas correspondant à autant d'itérations. La taille du pas est appelé taux d'apprentissage.

Interaction

- *À partir de la figure précédente, utiliser les curseurs pour tester différentes valeurs du taux d'apprentissage et du nombre de pas.*
- *Régler le taux d'apprentissage à 0.01.*
- *Pour contrer un taux d'apprentissage faible, augmenter le nombre de pas à 80.*
- *Régler le taux d'apprentissage à 0.25 et le nombre de pas à 30.*
- *Augmenter le taux d'apprentissage à 0.35 et le nombre de pas à 50.*

Observation - Réflexion

- *Que se passe-t-il lorsque le taux d'apprentissage est faible ?*
- *Que se passe-t-il lorsque le taux d'apprentissage est élevé ?*
- *Décrire l'influence du nombre de pas sur le résultat du calcul ?*

10- Conclusion

Au cours de cette séance nous avons passé en revue les éléments de base de la régression. Il est important de distinguer les régressions linéaire simple, linéaire multiple et non linéaire. Nous avons vu que la régression pouvait se définir comme un problème d'optimisation dont l'objectif est de minimiser une fonction coût. Pour résoudre ce problème, deux approches ont été présentées. La méthode des moindres carrés ordinaires qui est une méthode analytique, et la méthode de descente de gradient qui est une méthode algorithmique.

Ces approches nécessitent la connaissance du modèle, par exemple un polynôme du second degré. Dans ce cas le degré du polynôme retenu constitue un hyperparamètre, et les coefficients les paramètres du modèle.

Le choix du modèle, a priori, est souvent difficile. En apprentissage automatique, nous verrons que les réseaux de neurones peuvent être d'excellents candidats pour construire un modèle de régression a priori inconnu.

11- Annexe : fonctions utiles au live script

Fonction qui permet de tracer le modèle de régression à une variable de degré n

```
function yhat = plotFit(x,y,A,errorFlag)

% Prepare data and compute the model predictions
powers = length(A)-1:-1:0;
X = x'.^powers;
yhat = A*X';

% Plot
figure("Position",[0 0 800 400]);
plot(x,y,"o");
dispLine = sprintf("\nModèle de regression \n" + "y = " + join(A +
["x^{"+string(powers(1:end-1))+"}",""] ," + "),"Interpreter","tex");
xlim([min(x)-0.5,max(x)+0.5]);
ylim([min(y)-0.5,max(y)+0.5]);
hold on
fplot(@(x)A*(x'.^powers)',"LineWidth",1.5);
legend("Données",dispLine,"Location","eastoutside")
hold off
xlabel("Superficie (m²)")
ylabel("Prix de vente (€)")
title("Modèle de Régression Linéaire")

if errorFlag
    hold on
    dispSSE = sprintf("\nErreurs \nSSE = %12g",sum(yhat-y).^2);
    errorbar(x,y,min(yhat-y,0),max(yhat-
y,0),".","vertical","DisplayName",dispSSE,"Color",[0.8 0.6 0]);
    hold off
end

end
```

Fonction qui permet de tracer un plan de régression

```
function yhat = plotMultiFit(x1,x2,y,coeffs)

% Split the coefficients into individual variables and estimate y.
[a0,a1,a2] = deal(coeffs(1),coeffs(2),coeffs(3));
yhat =a0 + a1.*x1 + a2.*x2 ;
```

```
 % Plot
 scatter3(x1,x2,y);
 xlim([min(x1)-0.1,max(x1)+0.1]);
 ylim([min(x2)-0.1,max(x2)+0.1]);
 zlim([min(y)-0.1,max(y)+0.1]);
 [X1,X2] = meshgrid(xlim,ylim);
 hold on
 surf(X1,X2,a0 + a1.*X1 + a2.*X2 ,"FaceAlpha", 0.4,"FaceColor",[1 0.5
0.5],"EdgeColor","none");
 hold off
 % Annotate
 title("Modèle de Régression Linéaire Multiple")
 dispPlane = sprintf("\nPlan de régression"+"\ny_{norm} = "+ a0 +" + " + a1 +
"x_{1norm} + " + a2 + "x_{2norm}" );
 legend("Données",dispPlane,"Location","eastoutside");
 view([-150 20])
 xlabel('x_{1norm}'), ylabel('x_{2norm}'), zlabel("y_{norm}")

 end
```

Fonction qui permet de tracer le modèle de régression non linéaire à deux paramètres a et b

```
 function plotNlinFit(x,yRel,a,b,mdl)
 plot(x,yRel,"o");                                    % Plot the data samples
 hold on
 fplot(@(x)mdl(x,a,b))                                % Plot the fitted model
 hold off
 title("Modèle de Regression Non Linéaire")                          % Add
title
 xlabel("Salinité (dS/m)")              % Label the X axis
 ylabel("Rendement Relatif")                          % Label the Y axis
 legend("Données",newline + "Courbe en S")
 str = compose("$y = \\frac {1}{1+(\\frac {x}{%2g})^{%2g}}$",a,b);
 text(10.5,0.7,str,"Interpreter","latex","FontSize",18);
 end
```

Fonction qui permet de tracer la descente de gradient

```
 function [a1,a0,mse] = gradientDescent(x,y,nbrPas,tauxApp)

 % Prepare variables for visualizing the surface plot of cost function
 [M,B] = meshgrid(-10:0.1:10,-10:0.1:10);
 cost = arrayfun(@(m,b)costFunc(m,b,x,y),M,B);
```

```
% Initialize loop
a1 = 7;                         % initial slope value
a0 = 7;                         % initial intercept value
n = length(x);                  % total number of data points
mse = costFunc(a1,a0,x,y);      % compute initial MSE
r = logspace(-0.5,0,nbrPas);  % red color intensities for the plots

% Plot three different visualizations
T = tiledlayout(2,2,"TileSpacing","compact","Padding","compact");

% Plot 1: surface
nexttile(T,1)
surf(M,B,cost,"EdgeColor","none")
hold on
plot3(a1,a0,mse,"*","Color",[0 0 0]);
hold off
colormap(parula(30))
view([-16 33])
xlabel("a1"), ylabel("a0"), zlabel("Fonction Coût (MSE)")

% Plot 2: contour
nexttile(T,2)
contour(M,B,cost,30,"LineWidth",2)
hold on
plot(a1,a0,"*","Color",[0 0 0],"MarkerSize",8)
hold off
xlabel("a1"), ylabel("a0")

% Plot 3: line fit
title(T,"Descente de Gradient")
nexttile(T,[1,2])
plot(x,y,"o")
hold on
fitLine = plot(xlim,a1.*xlim+a0,"-","LineWidth",1.5,"Color",[0 0 0]);
hold off
ylim([-10,10]), xlabel("x"), ylabel("y")


for step=1:nbrPas

    % Compute gradient descent
    pause(0.02)
    yhat = a1*x + a0;
    diff = yhat - y;
    da1 = tauxApp * sum(diff .* x) * 2 / n;
    da0 = tauxApp * sum(diff) * 2 / n;
```

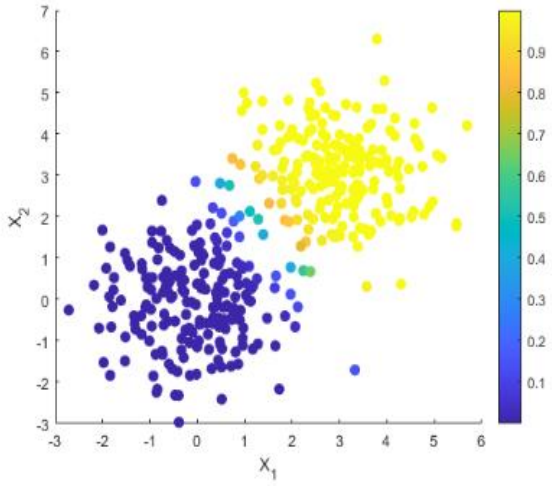
```
    a1 = a1 - da1;
    a0 = a0 - da0;
    mse = costFunc(a1,a0,x,y);

    % Plot new iteration data over the existing plots
    nexttile(T,1)
    hold on
    plot3(a1,a0,mse,"*","Color",[r(step) 0 0]);
    hold off

    nexttile(T,2)
    hold on
    plot(a1,a0,"*","Color",[r(step) 0 0],"MarkerSize",8)
    hold off

    nexttile(T,3)
    fitLine.YData = a1*xlim+a0;
    fitLine.Color = [r(step) 0 0];
    title("a1 = "+a1 + ", a0 = "+a0, "MSE = "+mse)

end
end
```

Fonction qui permet le calcul de la fonction coût

```
function mse = costFunc(m,b,x,y)
yhat = m*x+b;
mse = sum((yhat-y).^2)/length(x);
end
```

Chapitre 3

Introduction aux problèmes de classification

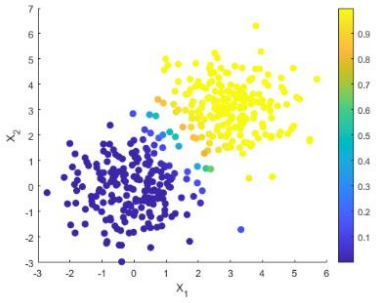


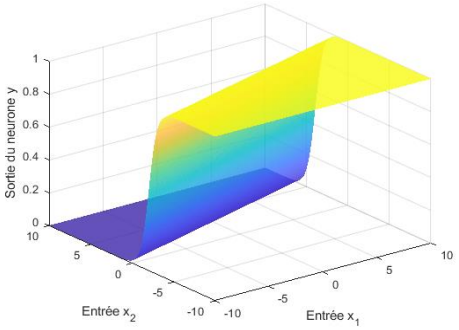
Introduction aux problèmes de classification

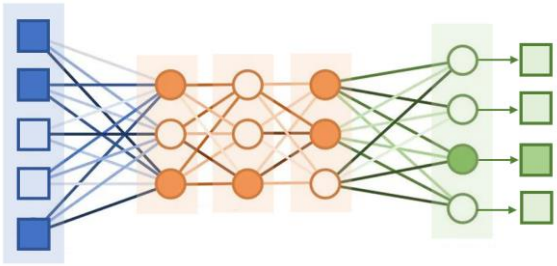
Deux classifieurs linéaires : la régression logistique et le perceptron

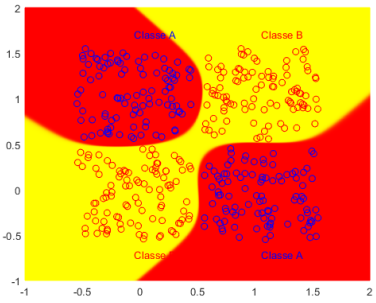
Du perceptron simple au perceptron multicouches

Les k plus proches voisins (kNN)









Plan du chapitre 3

Dans ce document deux types d'activités sont proposées :

 **Interaction** : vous interagissez avec Matlab en saisissant des éléments ou encore en agissant sur des éléments de contrôle.

 **Observation - Réflexion** : vous analysez les résultats obtenus suite aux différentes interactions.

1- Qu'est-ce que la classification ?

Dans un problème de classification, l'objectif est de classer un objet en cherchant à prédire la valeur d'une *variable discrète qualitative ou quantitative*. Par exemple on cherchera à savoir :

- si un risque cardiovasculaire y est possible ($y = 1$) ou pas ($y = 0$) en fonction de l'âge (X_1) et du poids (X_2) d'un patient, y est la variable expliquée (ici binaire) et X_1 puis X_2 sont les variables explicatives.
- si il y a risque de rupture ($y = 1$)ou pas ($y = 0$) d'une pièce mécanique en fonction de défauts mesurés,($X_1, X_2, \ldots X_n$)
- si il y a risque de panne pour un moteur en fonction de grandeurs mesurées,
- Si un courriel reçu est un spam ou pas,
- ...

Mais on pourra aussi prédire la classe d'appartenance d'un objet en fonction de ses caractéristiques :

- S'agit-il de la photo d'un chat ou d'un chien ?
- Cet iris est-il un iris setosa, virginica ou encore versicolor compte tenu de ses caractéristiques ?
- Cette personne marche-t-elle ? Courre-t-elle ? Monte-t-elle des escaliers ? Se repose-t-elle ? Compte tenu des informations extraites de son téléphone portable glissé dans sa poche.
- …

L'objectif de ce travail est de s'approprier des méthodes de classification usuelles.

2 - La régression logistique binaire : un classifieur linéaire

Dans certains cas la classification d'un objet peut se faire au moyen d'une régression particulière appelée régression logistique. Ce modèle de régression fait partie de la famille des *modèles linéaires généralisés* tout comme la régression linéaire.

La régression logistique est aussi une technique prédictive. Elle vise à construire un modèle permettant de prédire les valeurs prises par une variable cible qualitative le plus souvent binaire, dans ce cas on parle de *régression logistique binaire.*

En apprentissage automatique, le rôle d'un classifieur est de classer dans des classes les données possédant des propriétés similaires, mesurées sur des observations. Un classifieur linéaire est un type particulier de classifieur, dont la décision s'obtient par combinaison linéaire des échantillons.

Pour illustrer le propos, nous traiterons le cas de jeux de données d'abord avec une seule variable explicative, puis avec deux variables explicatives.

2.1 - Cas d'un jeu de données avec une seule variable explicative et une variable expliquée binaire

Notations :

- X est une matrice
- X_i est un vecteur
- x_i est un scalaire

Le vecteur $X = (x_1, \ldots, x_n)^T$ correspond à la variable explicative (ou encore la variable d'entrée) du jeu de données et Y le vecteur correspondant à la variable expliquée (ou encore la variable de sortie). Cette dernière est binaire : $Y = (0, 1)^T$.

On définit alors deux classes, la première pour laquelle les sorties valent 0 et la seconde pour laquelle les sorties valent 1.

L'objectif est de proposer un modèle de classification au moyen d'une régression logistique binaire, c'est un dire un modèle qui puisse prédire la classe d'appartenance d'un échantillon.

Interaction

```
K =60;
offset = 0.8;
X=[rand(1,K/2) offset+rand(1,K/2)];
Y = [zeros(1,K/2) ones(1,K/2)];

figure
gscatter(X,Y,Y);
grid on
xlabel('X')
ylabel('Y')
title('Observations')
```

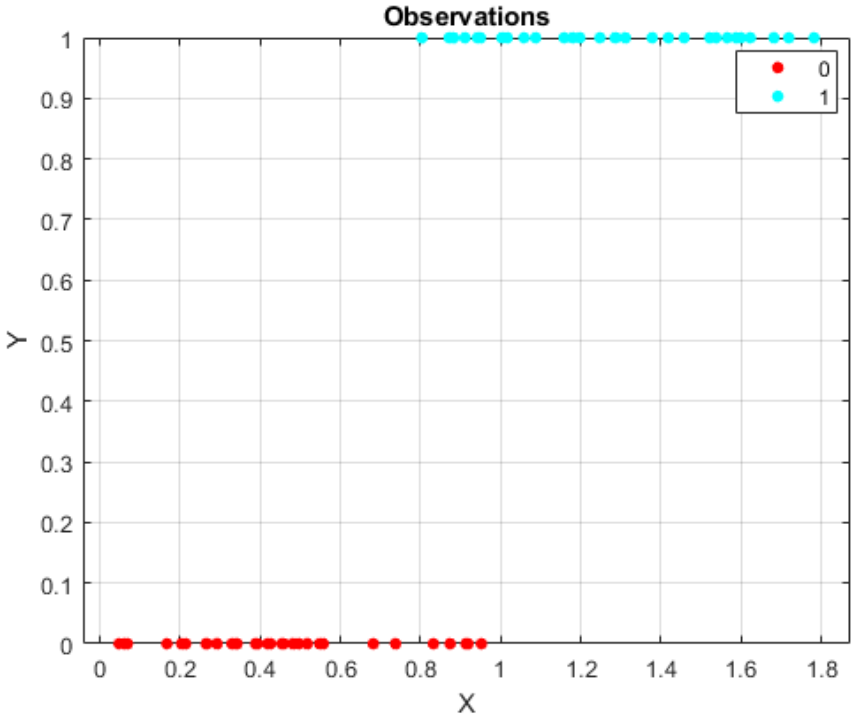


Figure 23- Exemple de données binaires

Observation - Réflexion

- *Un modèle de régression linéaire est-il adapté dans ce cas ? Pourquoi ?*

2.1.1 Une approche probabiliste

Dans le cas d'une régression linéaire, la sortie estimée par le modèle est une combinaison linéaire des entrées. Dans le cas qui nous intéresse cette hypothèse n'est plus applicable comme nous venons de le voir. Cependant une approche probabiliste montre que (transformation logit) :

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) = w_0 + w_1 X_1 + \ldots + w_n X_n$$

Les coefficients $w_{i\,(i\neq 0)}$ sont appelés *poids* et w_0 est appelé *biais.*

Les vecteurs $X_1, \ldots, X_n$ sont les vecteurs associés aux variables d'entrée

p est une probabilité conditionnelle. Elle sera toujours comprise entre 0 et 1, et donc toute tentative pour ajuster un nuage de probabilité par une droite sera invalidée par le fait que la droite n'est pas bornée comme nous venons de le voir. La transformation de p en $p/(1-p)$ permet de travailler sur des valeurs variant de 0 à $+\infty$, puis le passage au logarithme permet de travailler sur un nuage de points dont les valeurs varient entre $-\infty$ et $+\infty$, ce qui rend ainsi possible l'approximation par une droite.

En manipulant l'expression précédente on peut exprimer p :

$$p = P(Y=1|X) = \frac{1}{1+e^{-(w_0+w_1X_1+\ldots+w_nX_n)}}$$

L'expression de la probabilité $P(Y=1|X)$ est de la forme : $f(z) = \frac{1}{1+e^{-z}}$

On reconnaît là une *fonction sigmoïde qui évolue entre 0 et 1.*

2.1.2 La fonction sigmoïde comme modèle de régression logistique

La fonction sigmoïde est une *fonction logistique* dont l'expression est la suivante : $f(z) = \frac{1}{1+e^{-z}}$

Interaction

```
z=-10:0.1:10;
f=1./(1+exp(-z));
plot (z,f)
grid on
xlabel('z')
ylabel('$f(z)$',"Interpreter","latex")
title('Fonction sigmoïde')
legend('$f(z)=\frac{1}{1+e^{-
z}}$',"Interpreter","latex","FontSize",18,'Location','best')
```

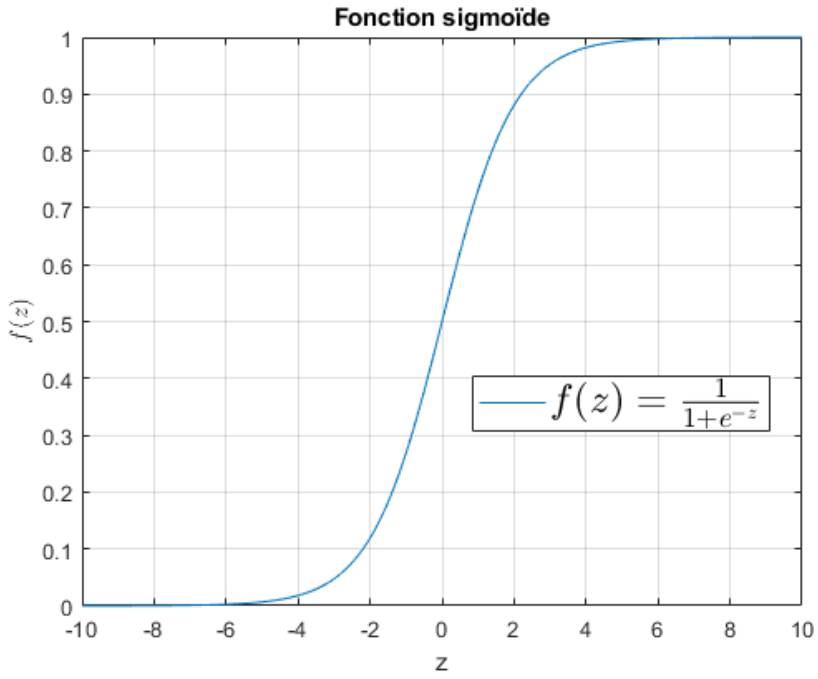


Figure 24- Fonction sigmoïde

Observation - Réflexion

- *Quelles propriétés peut-on relever sur cette représentation graphique ?*

2.1.3 Les paramètres du modèle de régression

Si nous reprenons l'expression de p précédente ou encore $P(Y=1|X)$, les poids et le biais qui interviennent dans celle-ci, constituent les paramètres du modèle qu'il conviendra d'ajuster.

Pour notre premier exemple qui ne comporte qu'une seule variable d'entrée : $z = w_0 + w_1 X_1$

Avec : X_1 =($x_1, \ldots x_m$).

On en déduit $f(X_1) = \frac{1}{1+e^{-(w_0+w_1X_1)}} = \frac{1}{1+e^{-w_1\left(\frac{w_0}{w_1}+X_1\right)}}$

Soit pour un échantillon x_i : $f(x_i) = \frac{1}{1+e^{-(w_0+w_1x_i)}}$

Interaction

Vous allez observer l'influence des paramètres w_0 et w_1 sur la fonction sigmoïde :

```
Xg=0:0.01:2;
w0=-20;
w1=24;
f=1./(1+exp(-w1*((w0/w1)+Xg)));
plot (Xg,f)
hold on
gscatter(X,Y,Y);
plot(-w0/w1,0.5,'r+')
hold off
grid on
xlabel('X1')
ylabel('$P(Y_1=1|X_1)$',"Interpreter","latex")
title('$P(Y_1=1|X_1)=\frac{1}{1+e^{-
w_{1}(\frac{w_{0}}{w_{1}}+X_{_{1}})}}$',"Interpreter","latex","FontSize",18)
legend('','Y=0','Y=1','frontière de décision : seuil')
```

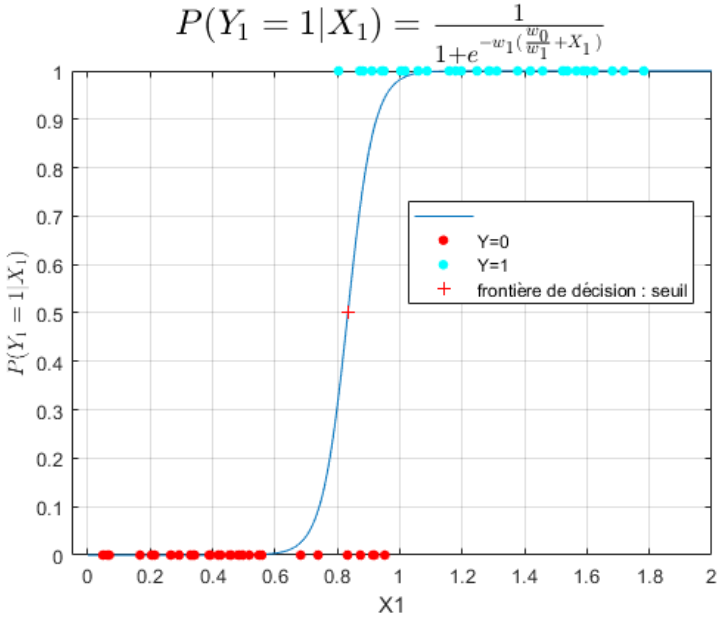


Figure 25- Superposition de la fonction sigmoïde et des données

Observation - Réflexion

- *Quelle est l'influence des paramètres* w_0 et w_1 *sur l'allure de la sigmoïde ?*

Ici, le résultat du calcul de $f(x_i)$, étant interprété pour chaque échantillon comme une probabilité d'appartenance à la classe 1, on estimera que l'échantillon est classé 1 si $f(x_i) > 0.5$, et classé 0 si $f(x_i) \leq 0.5$.

Le seuil de 0.5 constitue alors la frontière de décision du classifieur.

À ce propos, on montre que pour un espace des variables d'entrée de dimension n, la frontière de décision est un hyperplan de dimension $n-1$.

Dans le cas d'une régression logistique linéaire, l'équation de l'hyperplan est donnée par l'équation :

$$w_0 + w_1 X_1 + \ldots + w_n X_n = 0.$$

Ainsi, pour un espace des variables d'entrée de dimension 3 la frontière de décision est une surface (dimension 2), pour un espace des variables d'entrée de dimension 2 la frontière de décision est une droite (dimension 1), pour un espace des variables d'entrée de dimension 1 la frontière de décision est un point (dimension 0).

2.2 - Cas d'un jeu de données avec deux variables explicatives et une variable expliquée binaire.

Interaction

```
K = 200;
offset = 3;
X = [randn(2,K) randn(2,K)+offset];
Y = [zeros(1,K) ones(1,K)];
X1=X(1,:);
X2=X(2,:);
```

Affichage des données à classifier :

```
figure
gscatter(X1,X2,Y);
grid on
hold on
```

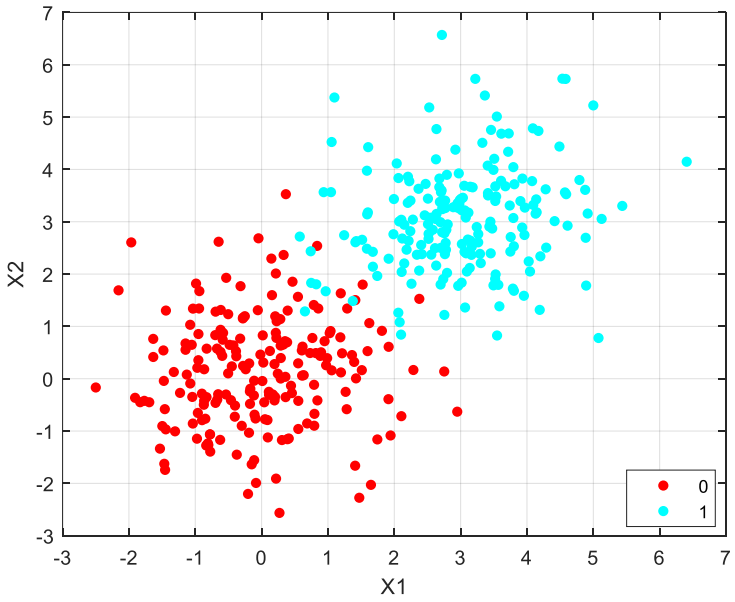


Figure 26- Jeux de données à classifier

Nous allons utiliser la fonction Matlab `glmfit` (generalized linear model regression) pour évaluer les paramètres du modèle (biais et poids) :

```
w = glmfit(X',Y','binomial','link','logit');
w0=w(1)
```

```
w0 = -8.7653
```

```
w1=w(2)
```

```
w1 = 2.6772
```

```
w2=w(3)
```

```
w2 = 2.7205
```

La *vraisemblance* décrit la plausibilité d'une valeur des paramètres du modèle compte tenu des observations. L'ajustement du modèle, c'est à dire la détermination des paramètres du modèle se fait alors en maximisant la vraisemblance. La mise en œuvre de l'algorithme de descente de gradient permet la détermination du maximum de la fonction de vraisemblance dont on montre qu'elle est concave.

Observation - Réflexion

- *Donner l'équation de la droite de décision dans le plan des variables d'entrée.*

Interaction

- *Tracer la droite de décision :*

```
figure
gscatter(X1,X2,Y);
grid on
hold on
plot(X1,(-w(1)-w(2)*X1)/w(3))
legend('y=0','y=1','Frontière de décision')
```

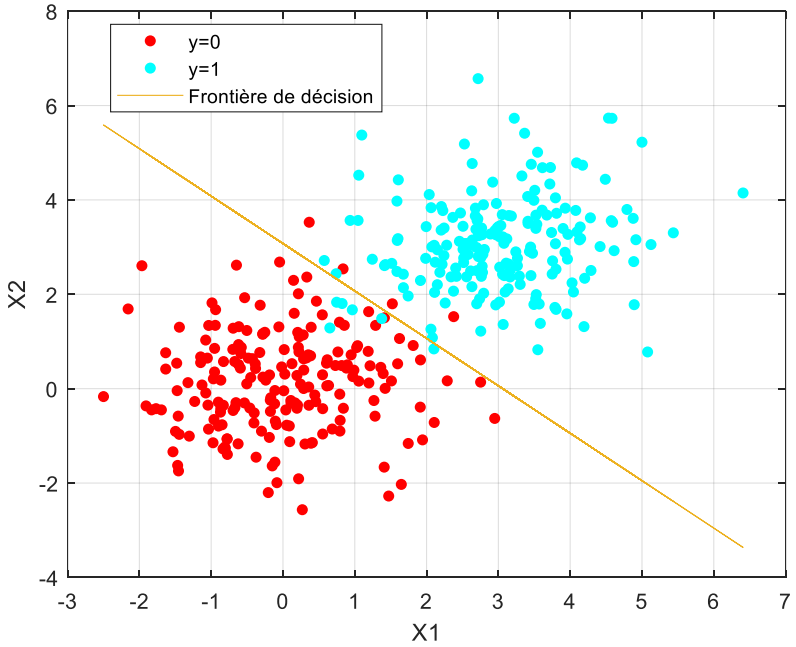


Figure 27- Frontière de décision.

- Tracer un graphique donnant la probabilité p = P(Y=1|X) :

```
p=1./(1+exp(-w0-w1*X1-w2*X2));

figure
scatter(X1,X2,50,p,'filled')
xlabel('X_1')
ylabel('X_2')
colorbar
```

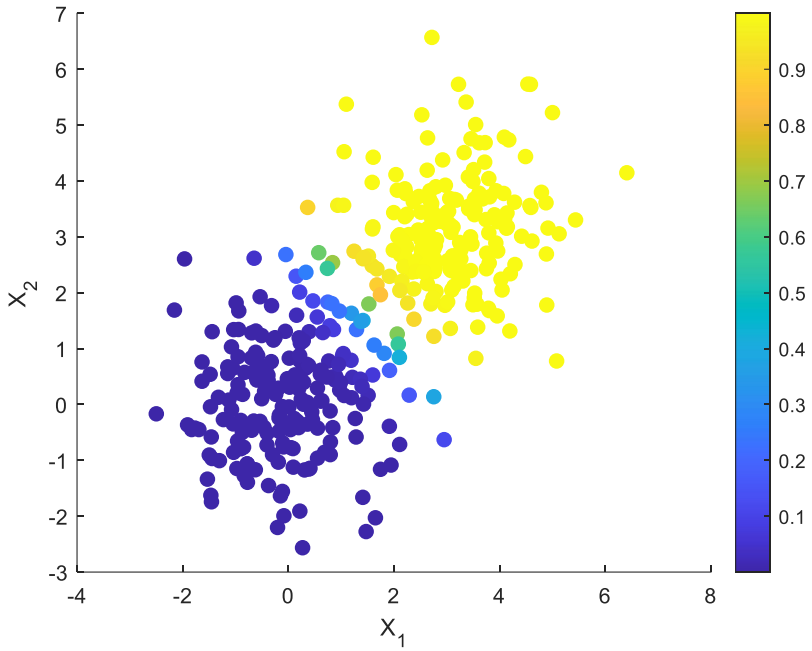


Figure 28- Représentation de p=P(Y=1 | X)

2.3 Pour conclure sur la régression logistique...

Nous venons de voir un premier classifieur. La régression logistique est donc une technique qu'on utilisera pour des études ayant pour but de vérifier si des variables indépendantes peuvent prédire une variable dépendante dichotomique.

3 - Le perceptron : un autre classifieur linéaire !

Le neurone artificiel est parfois appelé *neurone formel* ou encore perceptron. Il s'agit d'une unité élémentaire de traitement j d'un *réseau neuronal*, ayant plusieurs entrées x_i et une sortie y_j, dont la valeur est une fonction non linéaire φ d'une combinaison linéaire des valeurs d'entrée. Les cœfficients de pondération ou poids w_{ji} de la combinaison étant ajustables. Les poids sont des paramètres qu'il faudra ajuster en fonction de la sortie souhaitée. Le coefficient w_{j0} est aussi appelé biais. Parfois, on considère le biais unitaire et on lui associe un poids.

Cette unité peut être représentée de la manière suivante :

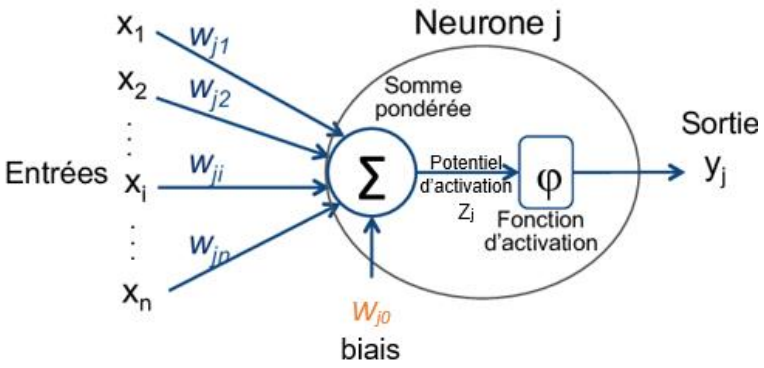


Figure 29- Représentation graphique d'un perceptron.

La fonction non linéaire est communément appelée *fonction d'activation* ou encore *fonction de transfert* du neurone. Il en existe un grand nombre, mais on peut citer les plus courantes, disponibles dans Matlab :

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin

Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \le n \le 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \le n$		poslin

Figure 30- Les fonctions d'activation proposées par Matlab.

La relation entre la sortie et les entrées du neurone, qui traduit son comportement, est appelée *équation de propagation* :

$$y_j = \varphi(z_j) = \varphi(x_i \,.\, w_{ji} + b_j)$$

4 - Analyse du comportement d'un perceptron

Interaction

Choix de la fonction d'activation :

```
f_activation='logsig';
```

Ajustement des poids :

```
w1=-0.1;
w2= 1.3;
```

Ajustement du biais :

```
b = 1.9;
```

Représentation graphique de l'évolution de la sortie du neurone en fonction des entrées, des poids et du biais :

```
[x1,x2] = meshgrid(-10:0.1:10);
w = [w1 w2];
z = feval(f_activation, [x1(:) x2(:)]*w'+b );
z = reshape(z,length(x1),length(x2));
figure
surf(x1,x2,z,'EdgeColor','none','FaceAlpha',0.8)
colorbar
grid on
xlabel('Entrée x_1')
ylabel('Entrée x_2')
zlabel('Sortie du neurone y')
title('Sortie du neurone en fonction des entrées et de la fonction
d''activation')
```

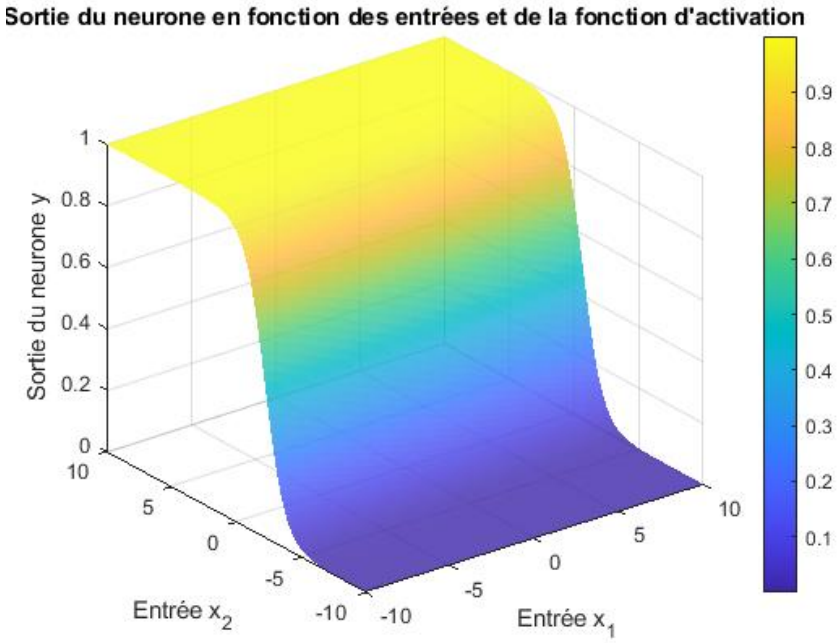
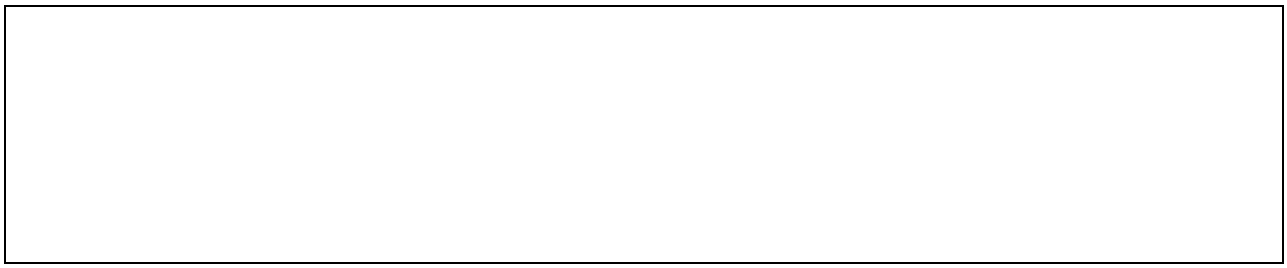


Figure 31- Sortie d'un neurone.

Observation - Réflexion

- *Quelle influence ont les poids et le biais sur la sortie du neurone ?*

- *Quelle influence a la fonction d'activation sur la sortie du neurone ?*



Interaction

- *Saisir le vecteur d'entrée X*
- *Sélectionner une fonction d'activation de Heaviside.*
- *Ajuster les poids à 0.5.*
- *Donner une valeur du biais permettant de construire une fonction logique ET, et une fonction logique OU.*
- *Ajuster le biais à 0.5.*
- *Donner des valeurs des poids permettant la construction des deux fonctions logiques précédentes.*

On rappelle les tables de vérité :

Table de vérité de ET		
a	b	a ET b
0	0	0
0	1	0
1	0	0
1	1	1

Table de vérité de OU		
a	b	a OU b
0	0	0
0	1	1
1	0	1
1	1	1

Figure 32- Tables de vérité.

Saisie du vecteur d'entrée

```
X = [0 0 ;0 1 ;1 0 ; 1 1 ]
```

```
X = 4×2
     0     0
     0     1
     1     0
     1     1
```

Choix de la fonction d'activation

```
f_activation='hardlim';
```

Ajustement des poids et du biais

```
w1=0.5;
w2= 0.5;
b = -0.4;
```

```
[x1,x2] = meshgrid(0:0.001:1);
w = [w1 w2];
z = feval(f_activation, [x1(:) x2(:)]*w'+b );
z = reshape(z,length(x1),length(x2));
Z= feval(f_activation, [X(:,1) X(:,2)]*w'+b );

figure
subplot(2,1,1)
surf(x1,x2,z,'EdgeColor','none','FaceAlpha',0.4)
hold on
plot(X(:,1),X(:,2),'ro','MarkerSize',7,'MarkerFaceColor','red')
plot3(X(:,1),X(:,2),Z,'go','MarkerSize',10)
hold off
grid on
xlabel('Entrée x_1')
ylabel('Entrée x_2')
zlabel('Sortie du neurone y')
title('Sortie du neurone en fonction des entrées')
view([65 35])
subplot (2,1,2)
surf(x1,x2,z,'EdgeColor','none','FaceAlpha',0.4)
hold on
plot(X(:,1),X(:,2),'ro','MarkerSize',7,'MarkerFaceColor','red')
plot3(X(:,1),X(:,2),Z,'go','MarkerSize',10)
hold off
grid on
xlabel('Entrée x_1')
ylabel('Entrée x_2')
zlabel('Sortie du neurone y')
title('Sortie du neurone (vue de dessus)')
view([90,90])
```

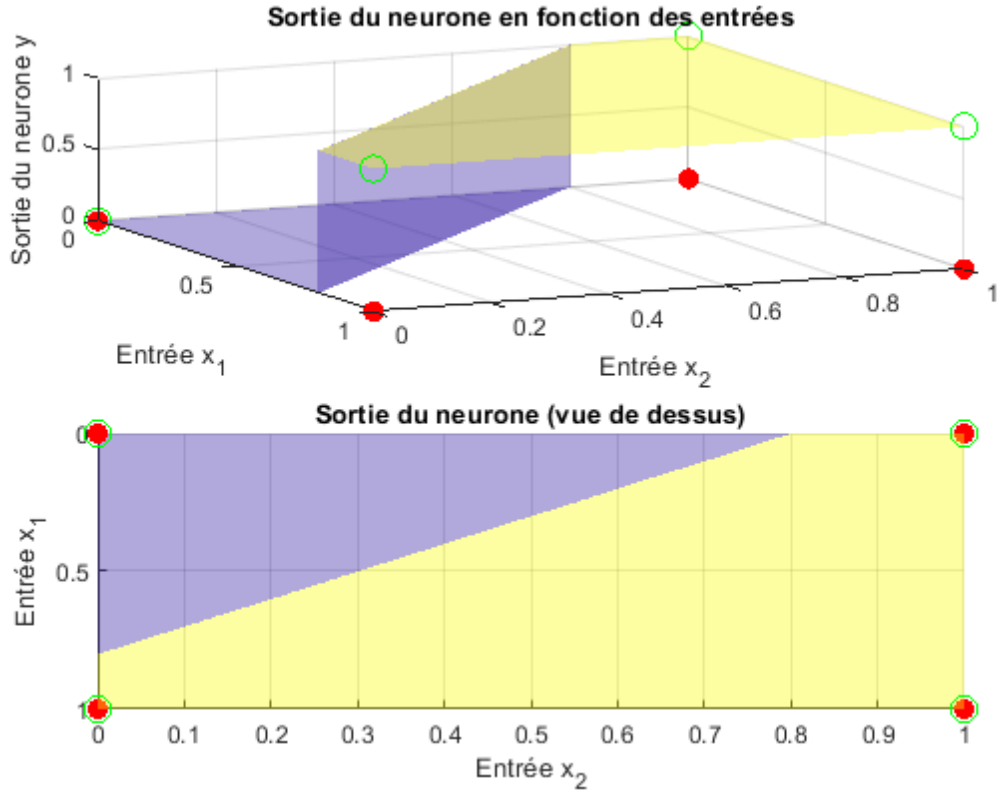


Figure 33- Sortie d'un neurone.

```
display(X)
```

```
X = 4×2
     0     0
     0     1
     1     0
     1     1
```

```
potentiel_activation = X*w'+b
```

```
potentiel_activation = 4×1
   -0.4000
    0.1000
    0.1000
    0.6000
```

```
sortie_neurone = feval(f_activation, potentiel_activation)
```

```
sortie_neurone = 4×1
     0
     1
     1
     1
```

Observation - Réflexion

- *Quelle que soit la fonction logique réalisée, la combinaison des entrées, des poids et du biais est-elle unique ?*
- *Pourquoi peut-on qualifier le perceptron de classifieur linéaire ?*
- *Donner l'équation du plan (appelé hyperplan de décision) qui sépare les deux classes (0 ou 1) de la sortie du perceptron.*
- *Donner l'équation de la droite de décision, intersection de l'hyperplan de décision et du plan des variables d'entrée.*
- *Peut-on réaliser une fonction OU Exclusif (XOR) avec un perceptron ?*

5 - L'apprentissage automatique d'un perceptron simple

Dans les manipulations précédentes vous avez ajusté les paramètres du perceptron, c'est à dire les différents poids, pour que la sortie soit celle attendue. L'apprentissage automatique consiste à ajuster ces paramètres au moyen d'un algorithme de descente de gradient qui minimisera la fonction coût (MSE). La démarche mise en œuvre est identique à celle que nous avons vue pour la régression. A chaque itération de l'algorithme les poids sont réajustés.

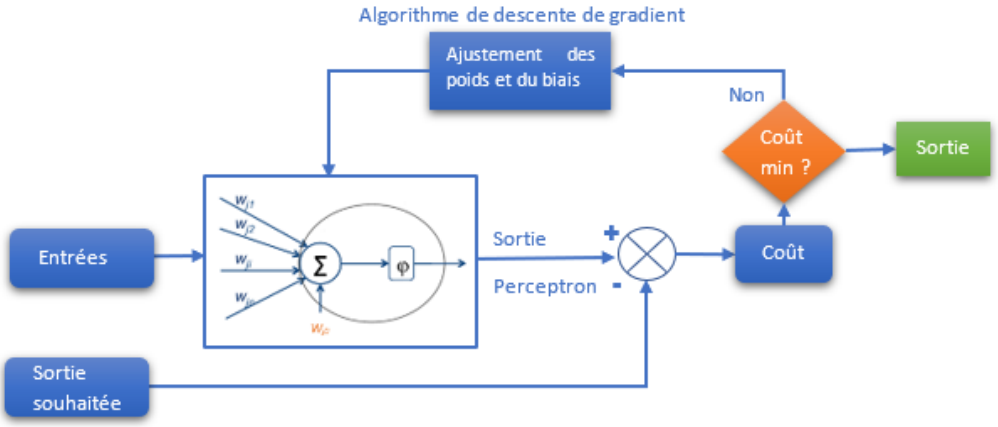


Figure 34- Apprentissage d'un perceptron simple.

L'activité qui suit a pour objectif de proposer une classification grâce à l'apprentissage d'un perceptron :

Interaction

Saisie du vecteur d'entrée (en lignes)

```
X = [0 0 1 1; 0 1 0 1];
```

Saisie de la sortie attendue

```
Y = [0 0 0 1];
```

Paramétrage du perceptron (hyperparamètres) :

- choix de la structure
- choix de la fonction coût
- choix de la fonction d'activation

```
net = perceptron;
net.performFcn = 'mse';
net.layers{1}.transferFcn = 'hardlim';
```

Apprentissage

```
net = train(net,X,Y);
```

Test de l'apprentissage : on soumet le modèle aux entrées synthétisées dans le vecteur x :

```
x=[0 0 1 1; 0 1 0 1];
y = net(x);
table_verite=[x',y']
```

```
table_verite = 4×3
     0     0     0
     0     1     0
     1     0     0
     1     1     1
```

```
poids = net.IW{1}
```

```
poids = 1×2
     2     1
```

```
biais = net.b{1}
```

```
biais = -3
```

Interaction

- *Vous pouvez reprendre les valeurs des poids et du biais pour les tester avec la simulation précédente.*

Observation - Réflexion

- *Combien d'itérations sont nécessaires dans le cas de l'usage de la fonction d'activation de Heaviside ? De la sigmoïde logistique ?*
- *Comparer les temps d'apprentissage.*
- *Comparer les performances.*
- *Conclure sur le choix de la fonction d'activation.*

Le but de l'activité qui suit est de séparer deux nuages de points par une droite. Pour ce faire on utilise un perceptron, dont on a vu l'aptitude à classifier de manière binaire des données linéairement séparables. Bien sûr ici une régression logistique serait tout aussi bien adaptée.

Interaction

Création des deux nuages de points à classifier :

```
close all, clear all, clc,
K = 50;
q = 0.5;

A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)-q];
```

Affichage des données à classifier :

```
figure
plot(A(1,:),A(2,:),'bo')
hold on
grid on
plot(B(1,:),B(2,:),'ro')

text(0.7,1.3,'Classe A','color','blue')
text(0.1,-0.3,'Classe B','color','red')
title('2 classes : A et B')
xlabel('X_1')
ylabel('X_2')
```

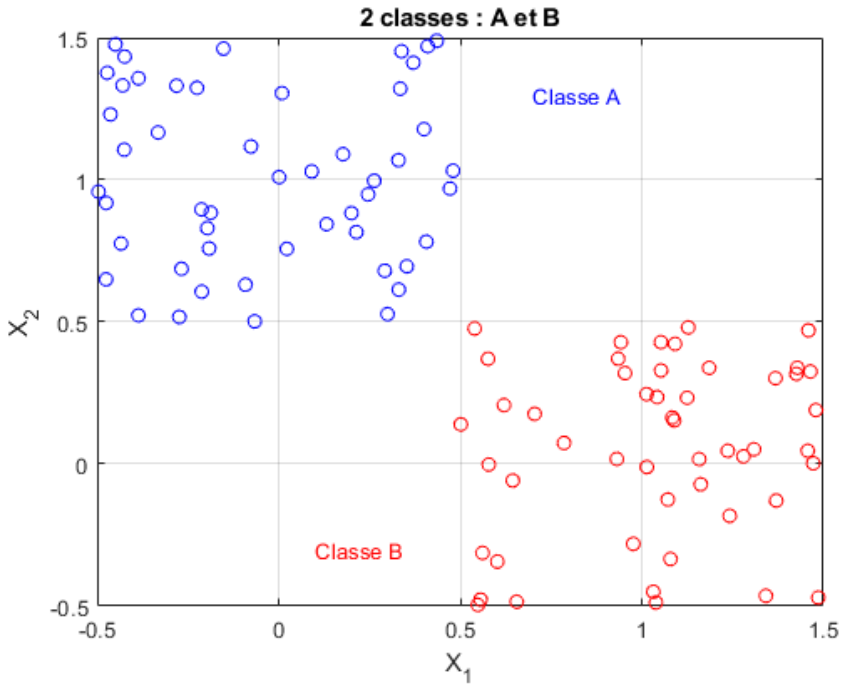


Figure 35- Deux classes à classifier.

Codage des sorties pour la classification :

```
a = 0;
b = 1;
```

Création des entrées du perceptron :

```
X = [A B];
```

Création des sorties :

```
Y = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

Création de la structure du perceptron et entraînement :

```
net = perceptron;
net.performFcn = 'mse';
net.layers{1}.transferFcn = 'hardlim';
net.divideFcn = 'dividetrain';
```

Apprentissage du perceptron :

```
net = train(net,X,Y);
```

Affichage des poids et des biais

```
poids=net.IW{1}
```

```
poids = 1×2
    4.8627   -3.2103
```

```
biais=net.b{1}
```

```
biais = -1
```

Observation - Réflexion

- *Donner l'équation de la droite de décision dans le plan des variables d'entrée.*

Tracé de la droite de décision :

```
plotpc(net.IW{1},net.b{1});
hold on
grid on
```

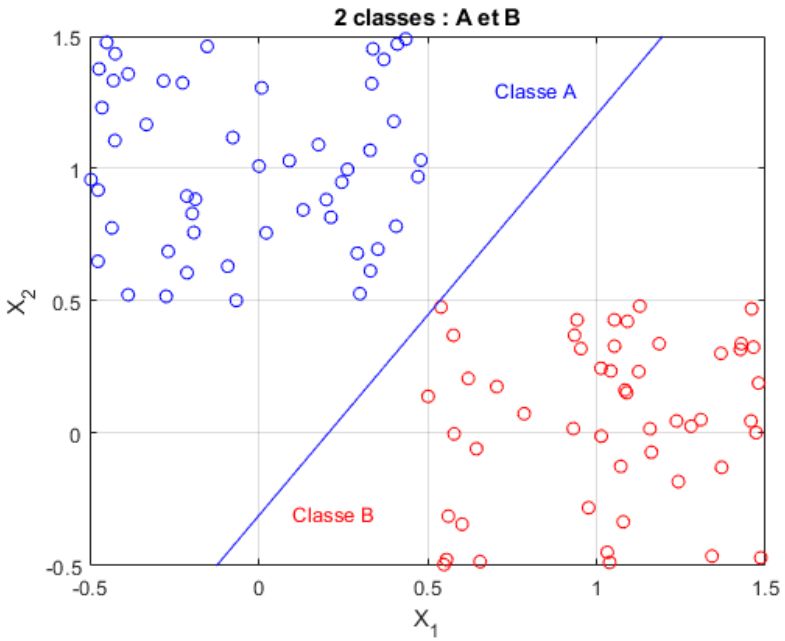


Figure 36- Frontière de décision.

Observation - Réflexion

- *Le tracé est-il conforme au résultat attendu ?*

On soumet le modèle à une entrée quelconque :

```
test=[1 1.2];
p1=plot(test(1),test(2),'k*');
drawnow
```

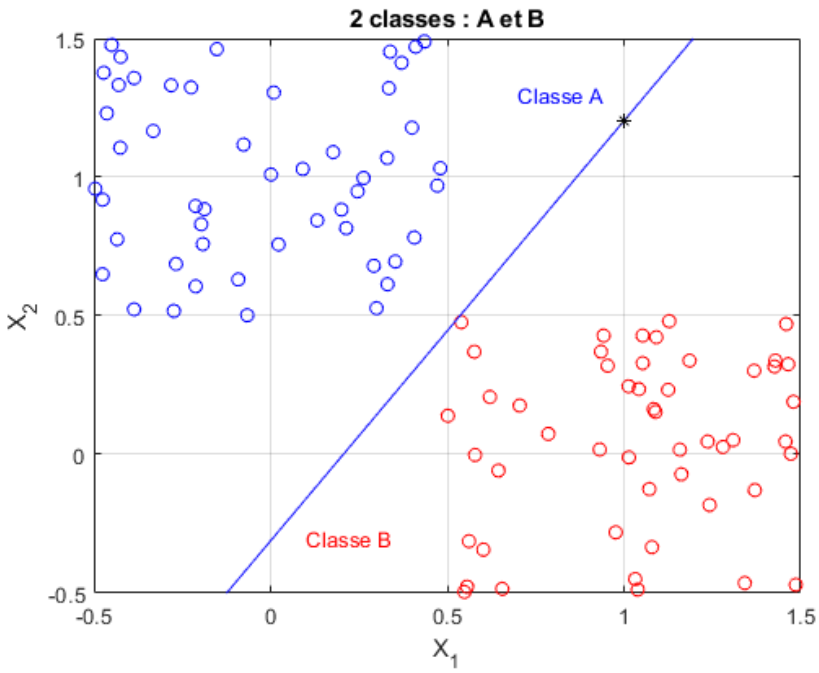


Figure 37- A quelle classe appartient ce nouveau point ?

Inférence :

```
sortie_test=net(test')
```

```
sortie_test = 1
```

```
if sortie_test == 0
```

```
    color = "blue";
    style = "o";
else
    color = "red";
    style = "o";
end
set(p1,'Color',color,'Marker',style,'MarkerFaceColor',color);
drawnow
```

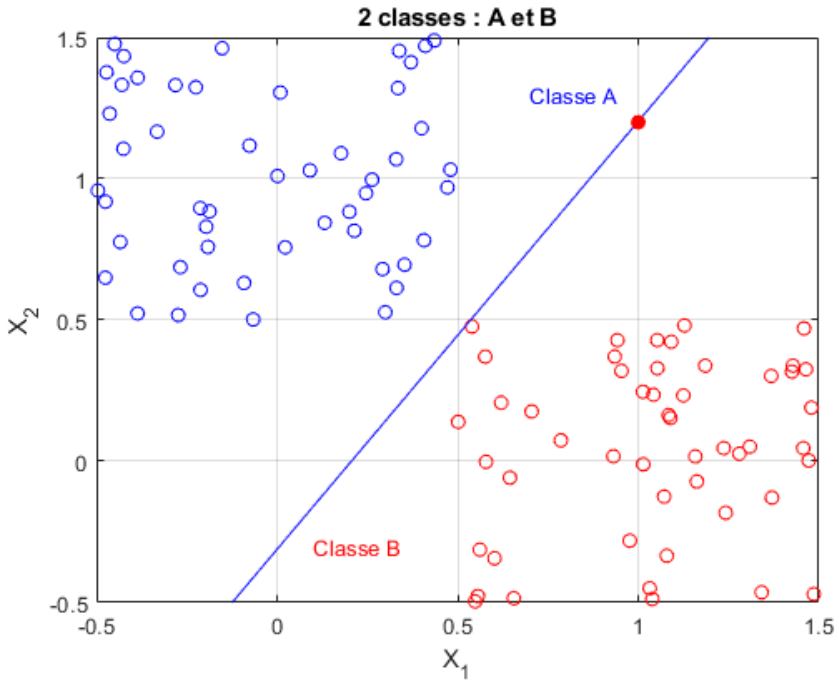


Figure 38- Classification du nouveau point.

Nous venons de voir que le perceptron permettait la classification de données linéairement séparables en deux classes. Nous pouvons alors légitimement nous demander si l'association de deux perceptrons pourrait classifier quatre classes avec deux droites de décision. C'est l'objet de l'exemple qui suit.

La structure que nous allons mettre en place est celle d'un perceptron simple couche composé de deux neurones :

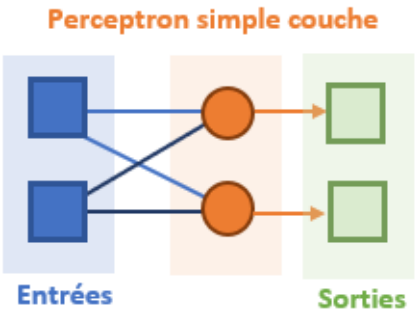


Figure 39- Perceptron simple couche

Interaction

Création des quatre nuages de points à classifier :

```
close all, clear all, clc,

K = 50;% nombre de points par nuage (2 mini)
q = .54; % offset

A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
```

Affichage des données :

```
figure(1)
plot(A(1,:),A(2,:),'bo')
hold on
grid on
plot(B(1,:),B(2,:),'ro')
plot(C(1,:),C(2,:),'go')
plot(D(1,:),D(2,:),'mo')
title('4 classes : A B C D')
xlabel('X_1')
ylabel('X_2')

text(.5-q,.5+2.2*q,'Classe A','color','blue')
text(.5+q,.5+2.2*q,'Classe B','color','red')
text(.5+q,.5-2.2*q,'Classe C','color','green')
text(.5-q,.5-2.2*q,'Classe D','color','magenta')
```

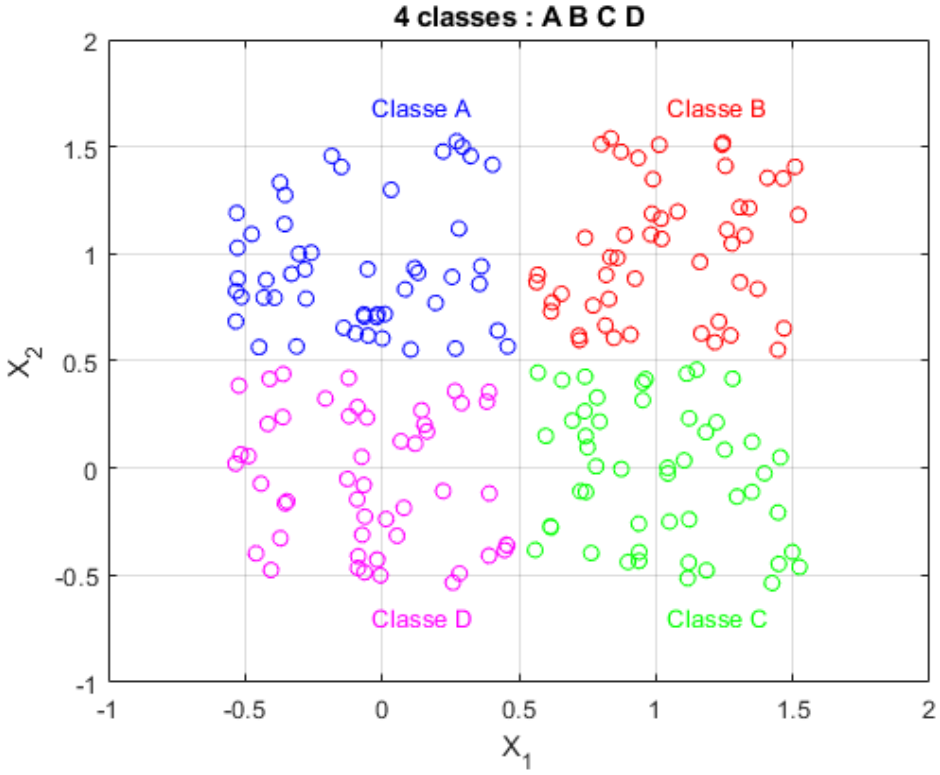


Figure 40- Quatre classes à classifier.

Codage des sorties pour la classification :

```
a = [0 1]';
b = [1 1]'; % a | b
c = [1 0]'; % -----
d = [0 0]'; % d | c
```

Création des entrées du perceptron :

```
X = [A B C D];
```

Création des sorties :

```
Y = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
repmat(c,1,length(C)) repmat(d,1,length(D)) ];
```

Création de la structure du perceptron et entraînement :

```
net = perceptron;
net.divideFcn = 'dividetrain';
net.performFcn = 'mse';
net = train(net,X,Y);
```

Affichage des poids et des biais

```
poids=net.IW{1}
```

```
poids = 2×2
    5.7213    0.4298
    0.3585    9.6029
```

```
biais=net.b{1}
```

```
biais = 2×1
    -3
    -5
```

Observation - Réflexion

- *Commenter la structure du perceptron entraîné.*
- *Tracer la structure du perceptron en indiquant les poids et les biais sur le schéma.*
- *Donner les équations des frontières de décision.*

Tracé des frontières de décision :

```
figure(1)
plotpc(net.IW{1},net.b{1});
title('Frontières de décision')
xlabel('X_1')
ylabel('X_2')
grid on
hold on
```

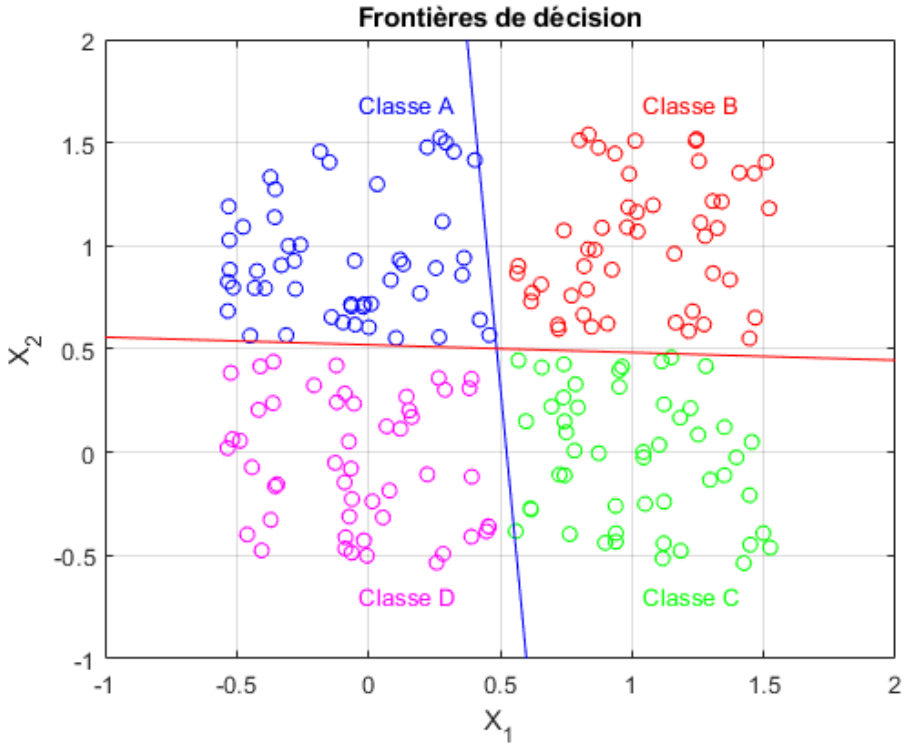


Figure 41- Frontières de décision.

Observation - Réflexion

- *Le tracé est-il conforme au résultat attendu ?*

On soumet le modèle à une entrée quelconque :

```
figure(1)
test=[0 0];
p1=plot(test(1),test(2),'k*');
hold on
```

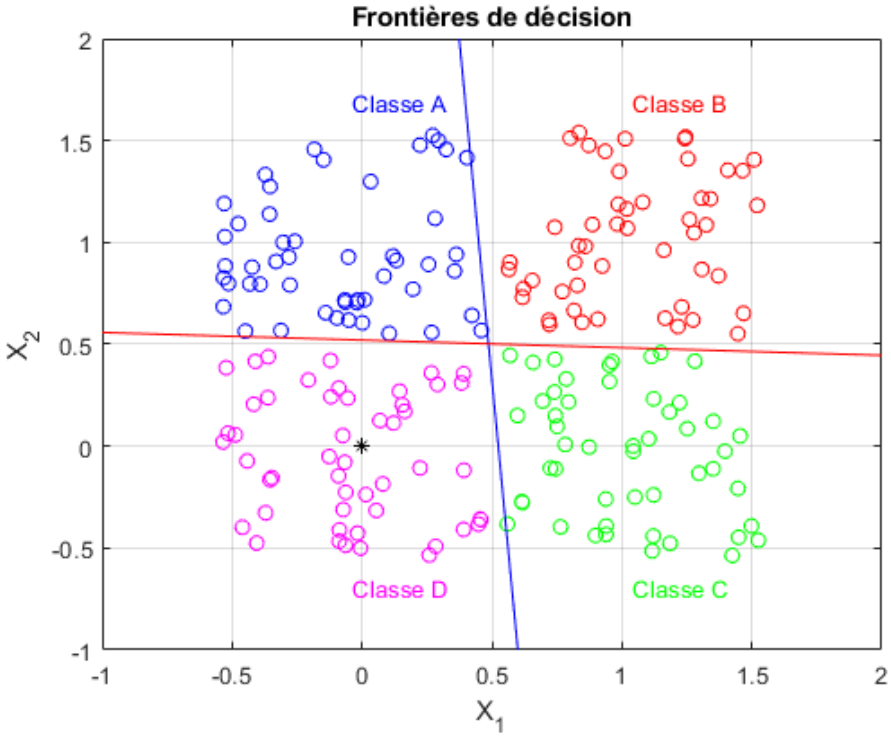


Figure 42- A quelle classe appartient le nouveau point ?

Interaction

Inférence :

```
sortie_test=net(test');

if sortie_test ==a
    color = "blue";
    style = "o";
elseif sortie_test ==b
    color = "red";
    style = "o";
 elseif sortie_test ==c
    color = "green";
    style = "o";
else
    color = "magenta";
```

```
    style = "o";
end
set(p1,'Color',color,'Marker',style,'MarkerFaceColor',color);
drawnow
```

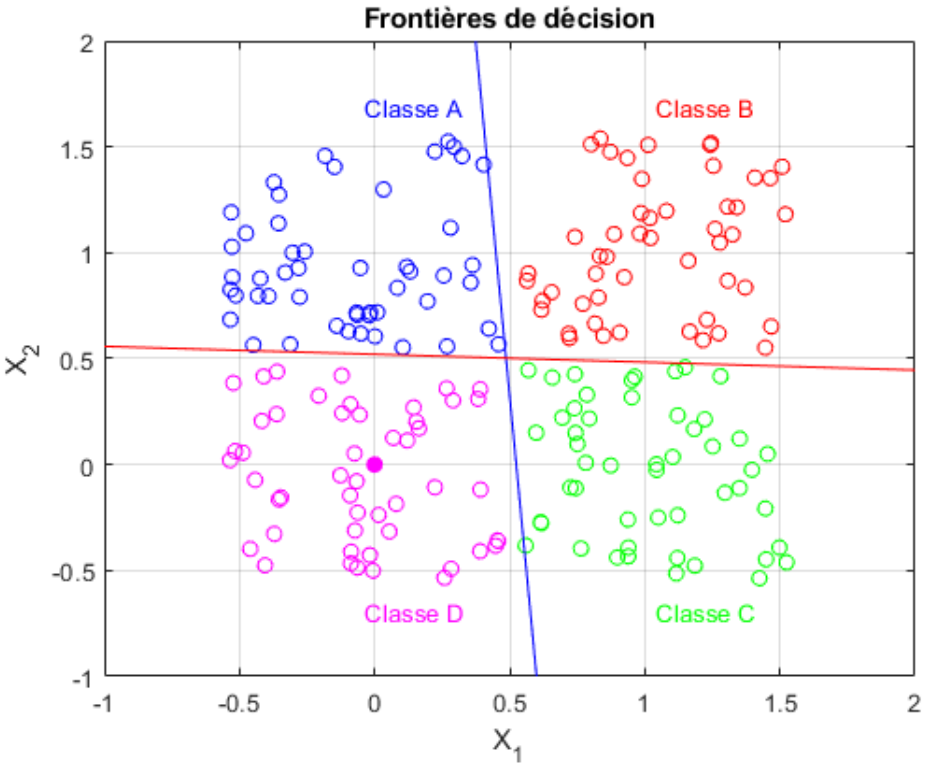


Figure 43- Classification du nouveau point.

6 - Les réseaux de neurones : le perceptron multicouches

Un perceptron simple ou simple couche ne peut traiter que les problèmes linéairement séparables comme nous venons de le voir. Cependant, une combinaison de séparateurs linéaires permet de produire un séparateur global non-linéaire, comme nous allons le voir.

Le traitement des problèmes non-linéaires devient donc possible en associant plusieurs neurones entre eux. Les *réseaux neuronaux* ou *réseaux de neurones*, sont alors constitués de neurones ou de perceptrons interconnectés.

Cette interconnexion se fait entre *couches* successives. Une couche pour les entrées, une autre couche pour les sorties, et entre ces deux couches, des couches intermédiaires qu'on appelle *couches cachées.*

La couche d'entrée est une simple copie des entrées représentées par des carrés, elle n'est pas constituée de neurones à proprement parlé. Les couches cachées peuvent avoir un nombre de neurones différent du nombre d'entrées. La couche de sortie comporte les neurones qui génèrent les sorties.

Les connexions neuronales s'établissent entre les neurones de deux couches adjacentes. Les neurones d'une même couche ne sont pas connectés entre eux.

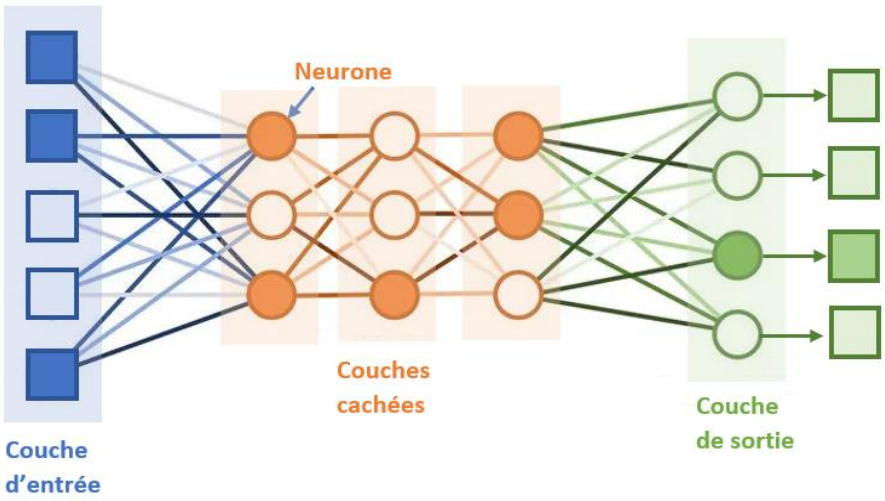


Figure 44- Perceptron multicouches.

Ce type de réseau à propagation directe ou avant (*feedforward*) est aussi appelé perceptron multicouches. Il ne contient aucun cycle contrairement aux réseaux récurrents (*feedback*) pour lesquels les neurones d'une même couche peuvent aussi être connectés.

Les réseaux possédant une seule couche cachée (perceptron historique) sont appelés *réseaux peu profond (shallow network)*. S'ils sont constitués de plusieurs couches cachées, on parle alors de *réseaux profonds (deep network)*.

Interaction

Le problème de la fonction OU exclusif (XOR) a bloqué le développement de l'IA dans les années 1970. Cette période a été baptisée par les spécialistes « the AI winter ».

Le problème peut être traité de deux manières :

- Soit en modifiant les données en ajoutant une caractéristique ce qui augmente la dimension de l'espace des variables d'entrée et donc celle de l'hyperplan de décision,
- Soit en connectant plusieurs perceptrons entre eux.

1. On ajoute une caractéristique aux données :

X1	X2	X1.X2	OU exclusif (XOR)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 45- Création d'une troisième variable.

Saisie du vecteur d'entrée (en lignes)

```
X = [0 0 1 1; 0 1 0 1;0 0 0 1];
```

Saisie de la sortie attendue

```
Y = [0 1 1 0];
```

Paramétrage du perceptron:

```
net = perceptron;
net.performFcn = 'mse';
net.layers{1}.transferFcn = 'hardlim';
```

Apprentissage

```
net = train(net,X,Y);
```

Test de l'apprentissage : on soumet le modèle aux entrées synthétisées dans le vecteur x :

```
x=[0 0 1 1; 0 1 0 1 ; 0 0 0 1];
y = net(x);
```

```
disp('la table de vérité s"écrit :')
```

```
la table de vérité s"écrit :
```

```
disp('     x1   x2   x1.x2  Xor')
```

```
     x1   x2   x1.x2  Xor
```

```
tab=[x',y'];
disp(tab)
```

```
     0     0     0     0
     0     1     0     1
     1     0     0     1
     1     1     1     0
```

```
disp('les paramètres du perceptron sont :')
```

```
les paramètres du perceptron sont :
```

```
poids = net.IW{1}
```

```
poids = 1×3
     1     1    -3
```

```
biais = net.b{1}
```

```
biais = -1
```

2. On construit un perceptron multicouche avec 2 couches cachées et une couche de sortie.

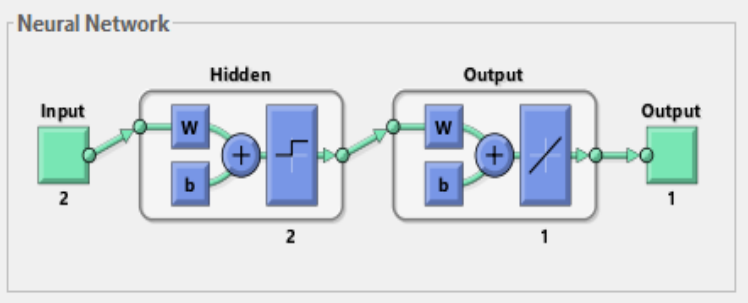


Figure 46- Structure du réseau.

On saisit les entrées et la sortie souhaitée :

```
X = [0 0 1 1 ; 0 1 0 1 ];
Y = [0 1 1 0 ];

net=feedforwardnet(2);
net.layers{1}.transferFcn = 'hardlim';
net.divideFcn = 'dividetrain';
net.performFcn = 'mse';
```

On initialise les poids et biais du perceptron multicouche :

```
net =init(net);
net = train(net,X,Y);
x=[0 0 1 1; 0 1 0 1];
y = net(x);
disp('la table de vérité s"écrit :')
```

```
 la table de vérité s"écrit :
```

```
disp('          x1       x2       Xor')
```

```
          x1       x2       Xor
```

```
tab=[x',y'];
disp(tab)
```

```
         0         0    0.0000
         0    1.0000    1.0000
    1.0000         0    1.0000
    1.0000    1.0000    0.0000
```

```
disp('les paramètres du perceptron sont :')
```

```
 les paramètres du perceptron sont :
```

```
poids_couche1 = net.IW{1,1}
```

```
poids_couche1 = 2×2
   -0.8116   -0.5668
    0.1541    0.9779
```

```
biais_couche1 = net.b{1}
```

```
biais_couche1 = 2×1
    0.9899
    0.9899
```

```
poids_couche2 = net.IW{2,1}
```

```
poids_couche2 =
     []
```

```
biais_couche2 = net.b{2}
```

```
   biais_couche2 = -3.0000
```

Nous allons traiter un autre exemple de problème de classification XOR appliqué à un jeu de données plus conséquent. Le but de la prochaine activité est de séparer deux classes de données non-linéairement séparables.

Interaction

```
close all, clear all, clc
```

Création et tracé des données :

```
% nombre de points pour chaque nuage
K = 100;
% Création des 4 nuages de points
q = 0.55; % offset des classes
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];



figure(1)
plot(A(1,:),A(2,:),'bo')
hold on
grid on
plot(B(1,:),B(2,:),'ro')
plot(C(1,:),C(2,:),'bo')
plot(D(1,:),D(2,:),'ro')

text(.5-q,.5+2.2*q,'Classe A','color','blue')
text(.5+q,.5+2.2*q,'Classe B','color','red')
text(.5+q,.5-2.2*q,'Classe A','color','blue')
text(.5-q,.5-2.2*q,'Classe B','color','red')
```

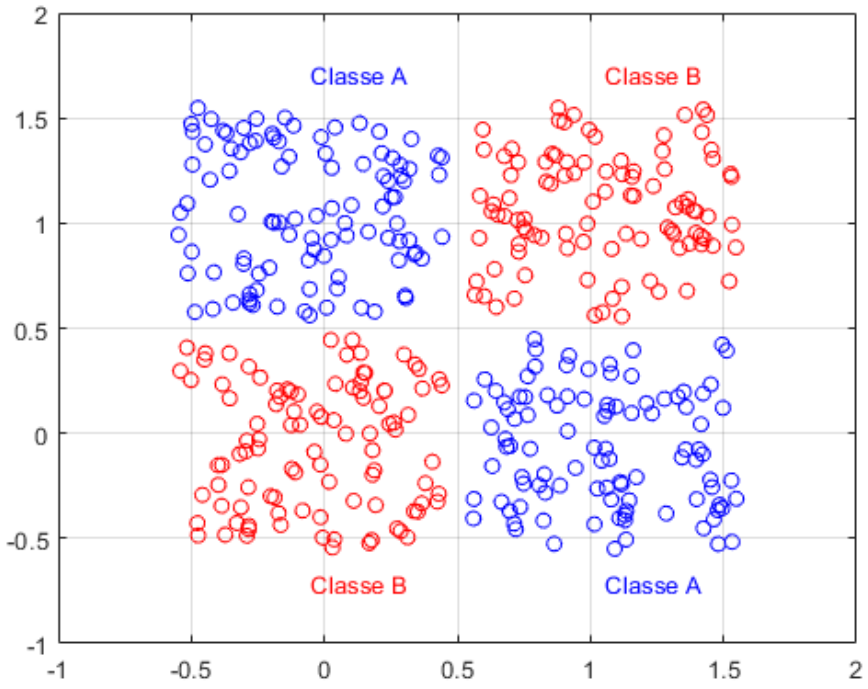


Figure 47- Quatre nuages de points et deux classes.

Codage des nuages de points en deux classes :

```
 % codage des nuages de points a et c en une classe, et b et d en une autre
classe
 a = -1; %  a | b
 c = -1; % -------
 b = 1;  %  d | c
 d = 1;  %
```

Création des exemples :

```
 X = [A B C D];
 Y = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
 repmat(c,1,length(C)) repmat(d,1,length(D)) ];
```

Interaction

Création du réseau de neurones : construire un réseau à deux couches cachées. La première possédant 5 neurones et la seconde possédant 3 neurones

```
net = feedforwardnet([4 2]);
net.divideFcn = 'dividetrain';
```

Entraînement du réseau

```
[net,tr,Ynet] = train(net,X,Y);
```

Observation - Réflexion

- *En observant la structure du réseau qui apparaît au moment de l'apprentissage expliquez pourquoi le nombre d'entrées vaut 2.*

Performance du réseau : on trace les sorties et les sorties estimées pour chaque échantillon :

```
figure
plot(Y','linewidth',2)
hold on
plot(Ynet','r--')
hold off
ylim([-1.25 1.25])
grid on
title('Performance du réseau')
legend('Sorties','Sorties estimées','location','best')
```

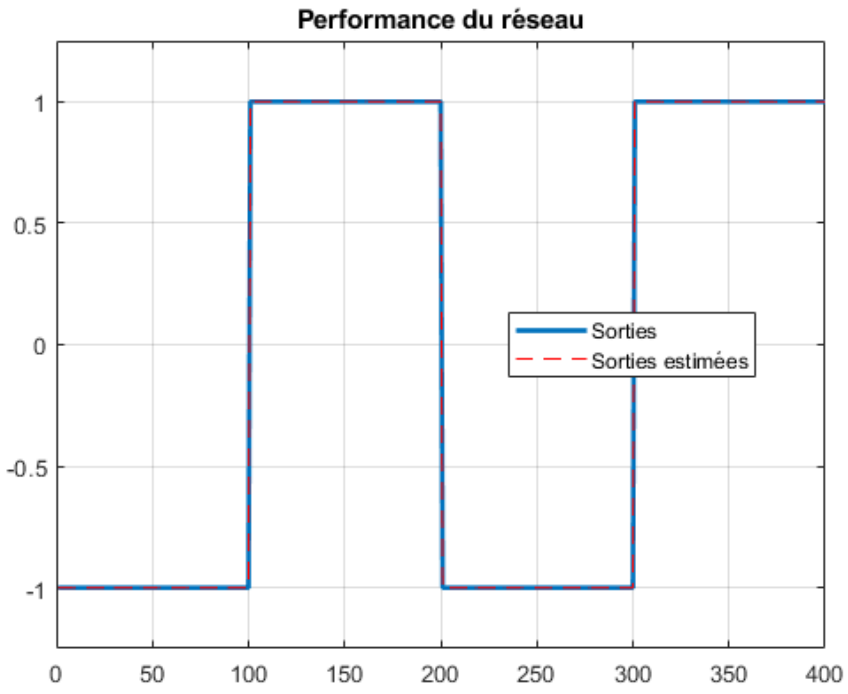


Figure 48- Performance du réseau.

Observation - Réflexion

- *Commenter les courbes obtenues.*
- *Quel doit être le nombre minimal de neurones de la première couche cachée pour que le modèle soit satisfaisant ?*

Génération d'une grille et simulation pour tous les points de la grille :

```
echelle = -1:.005:2;
[Xg,Yg] = meshgrid(echelle,echelle);
XYg = [Xg(:) Yg(:)]';
XYgnet = net(XYg);
```

Tracé des zones de décision :

```
figure(1)
mesh(Xg,Yg,reshape(XYgnet,length(echelle),length(echelle))-5);
colormap autumn
```

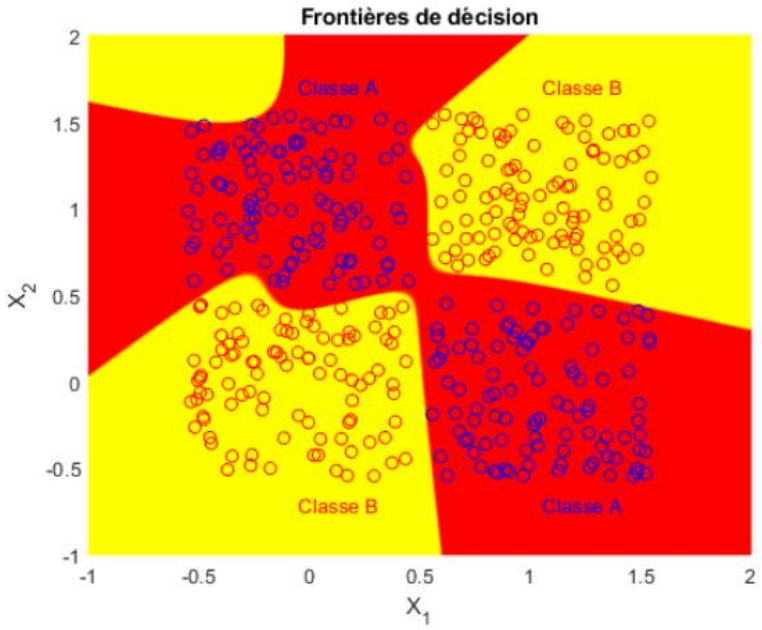


Figure 49- Frontières de décision.

Observation - Réflexion

- *Tester le modèle pour différentes entrées.*

```
Xtest=[1 0]'
Ytest_estime =net(Xtest)
if Ytest_estime ==a|c
    disp('le point appartient à la classe B')
else
    disp('le point appartient à la classe A')
end
```

7- Un autre algorithme de classification : les k plus proches voisins

La méthode des k plus proches voisins (*k Nearest Neighbors* en anglais) est une méthode prédictive non paramétrique. Pour estimer la sortie associée à une nouvelle entrée, la méthode des *k* plus proches voisins consiste à considérer de façon identique les *k* échantillons dont l'entrée est la plus proche de la nouvelle entrée, selon une distance à définir.
Ainsi, dans un problème de classification, on retiendra la classe la plus représentée parmi les *k* sorties associées aux *k* entrées les plus proches de la nouvelle entrée. Un objet d'entrée est classifié selon le résultat majoritaire des statistiques de classes d'appartenance de ses k plus proches voisins, (*k* est un nombre entier positif généralement petit). Si *k* = 1, alors l'objet est affecté à la classe d'appartenance de son plus proche voisin.

7.1- Le jeu de données Iris de Fisher

Ce jeu de données multivariées a servi en 1936 au biologiste anglais Ronald Fisher pour développer de nouvelles méthodes statistiques appliquées à la génétique.
Le jeu de données comprend 50 échantillons de chacune des trois espèces d'iris (*Iris setosa*, *Iris virginica* et *Iris versicolor*). Quatre caractéristiques ont été mesurées à partir de chaque échantillon : la longueur et la largeur des sépales et des pétales, en centimètres. Sur la base de la combinaison de ces quatre variables, Fisher a élaboré un modèle d'analyse discriminante linéaire permettant de distinguer les espèces les unes des autres.
quelques rappels botanniques :

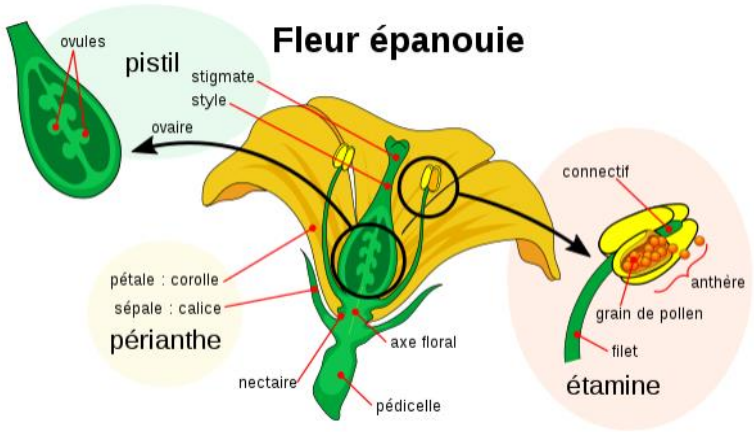


Figure 50 - Pétales et sépales.

(1)Iris Setosa

(2) Iris Versicolor

(3) Iris Virginica

Figure 51 - Différents types d'iris

Nous allons travailler sur ce jeu de données "historique" pour illustrer la méthode des k plus proches voisins.

Afficher les données :

```
close all, clear all, clc
load fisheriris
x = meas(:,3:4);
gscatter(x(:,1),x(:,2),species)
xlabel('Longueur des pétales [cm]')
ylabel('Largeur des pétales [cm]')
title ('Caractéristiques des pétales de l"iris')
legend('Location','best')
grid on
hold on
```

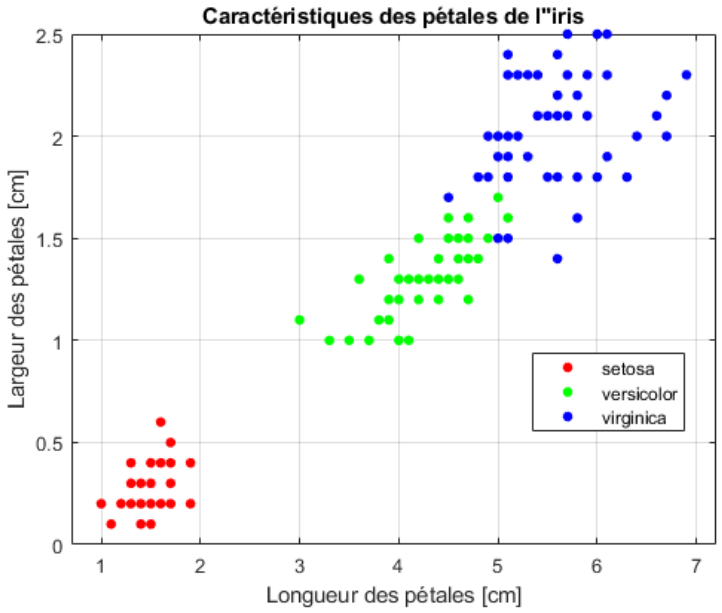


Figure 52 - Caractéristiques des pétales de l'iris.

7.2 Prédiction sur un premier jeu de données

Le premier jeu donne l'espèce en fonction de la largeur et de la longueur des pétales.

Interaction

Saisir les caractéristiques d'un nouvel iris

```
newpoint = [2.5 0.8];
plot(newpoint(1),newpoint(2),'marker','x','color','m',...
   'markersize',10,'linewidth',2)
legend('setosa','versicolor','virginica','nouvel iris')
```

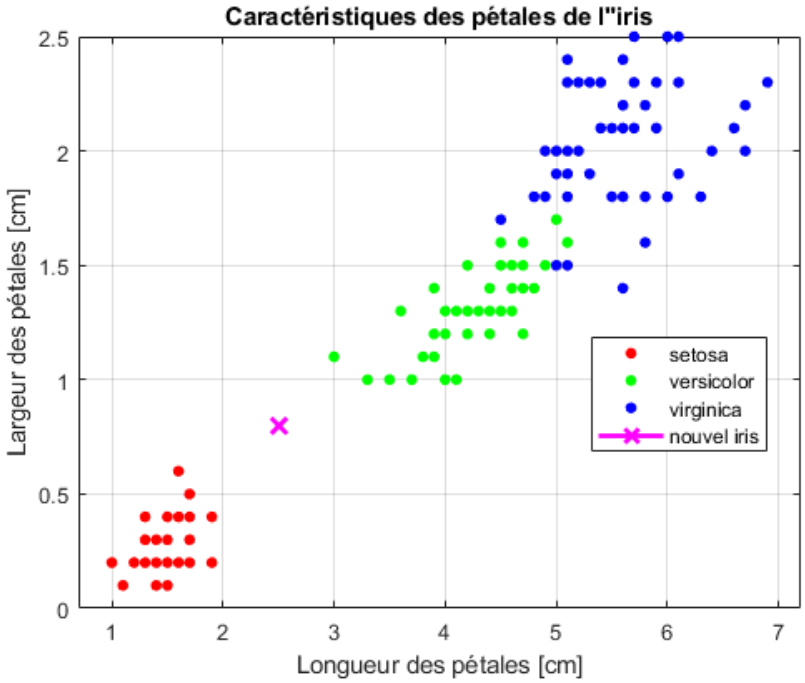


Figure 53 – Tracé d'un nouveau point

```
kPPV=5;
Mdl = KDTreeSearcher(x);
[n,d] = knnsearch(Mdl,newpoint,'k',kPPV);
line(x(n,1),x(n,2),'color',[.5 .5 .5],'marker','o',...
    'linestyle','none','markersize',10)
legend('setosa','versicolor','virginica','nouvel iris','kPPV')
```

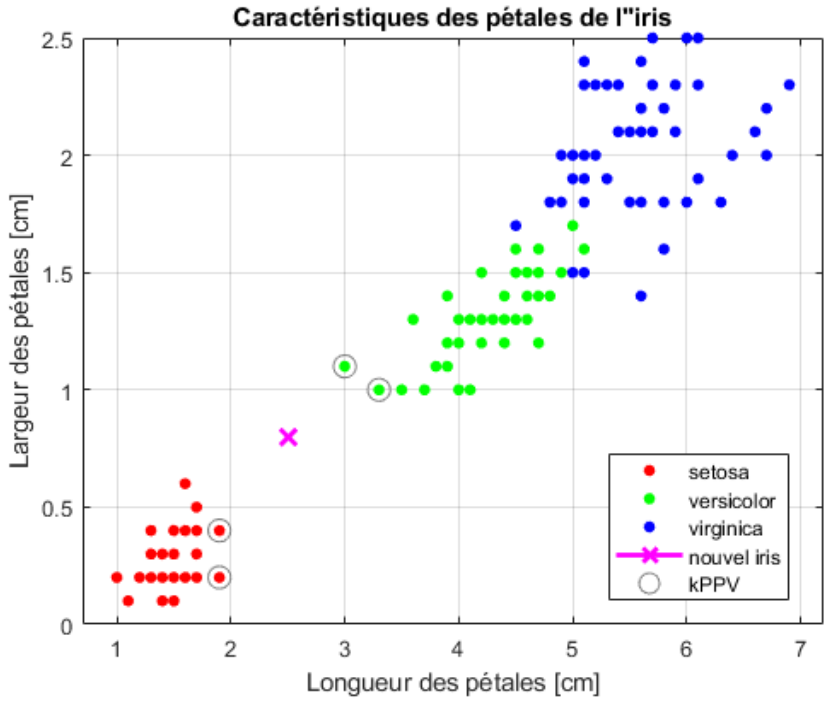


Figure 54 - Tracé des k plus proches voisins.

```
x(n,:)
```

```
ans = 5×2
    3.0000    1.1000
    1.9000    0.4000
    3.3000    1.0000
    3.3000    1.0000
    1.9000    0.2000
```

Observation - Réflexion

- Dans le cas où k=5, 4 plus proches voisins s'affichent. Pourquoi ?

```
tabulate(species(n));
```

```
      Value    Count   Percent
versicolor        3     60.00%
    setosa        2     40.00%
```

Observation - Réflexion

- Après plusieurs essais conclure sur la qualité de la prédiction.

7.3 Prédiction sur un second jeu de données

Le second jeu de donne l'espèce en fonction de la largeur et de la longueur des sépales.

Interaction

```
close all, clear all, clc
load fisheriris
X = meas(:,1:2);
y = categorical(species);
labels = categories(y);
gscatter(X(:,1),X(:,2),species);
xlabel('Longueur des sépales [cm]');
ylabel('Largeur des sépales [cm]');
title ('Caractéristiques des sépales de l"iris')
grid on
hold on
```

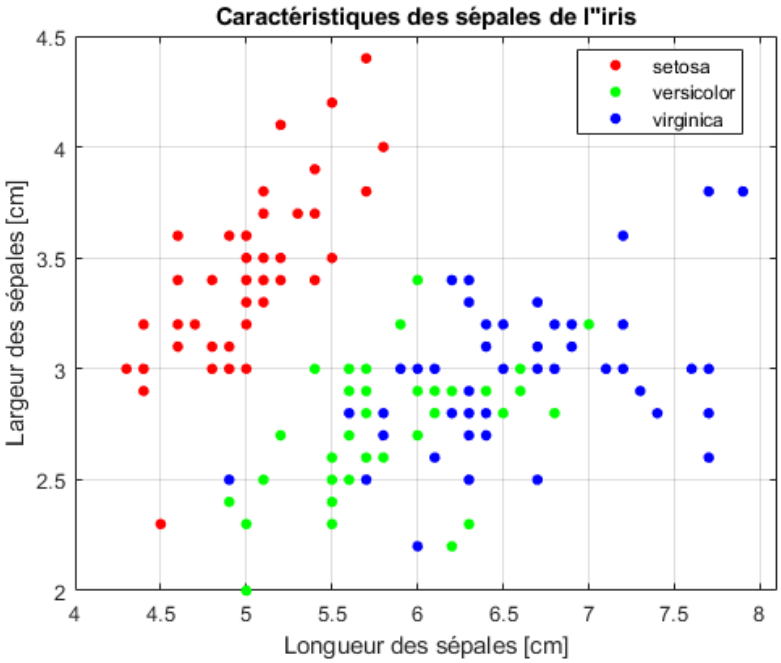


Figure 55 - Espèce en fonction des caractéristiques des sépales.

Saisir les caractéristiques d'un nouvel iris

```
newpoint = [6.2 3];
plot(newpoint(1),newpoint(2),'marker','x','color','m',...
   'markersize',10,'linewidth',2)
legend('setosa','versicolor','virginica','nouvel iris')
```

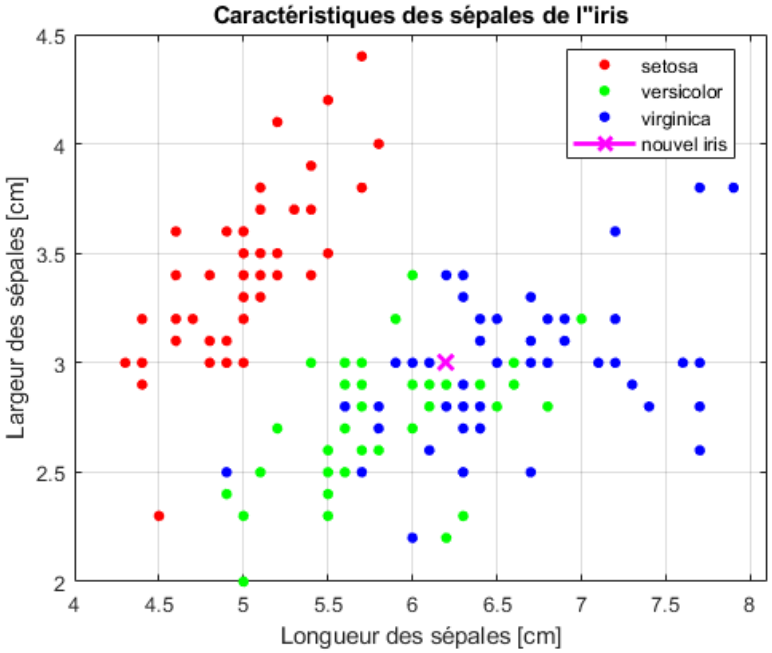


Figure 56 - Tracé d'un nouveau point.

```
kPPV=5;
Mdl = KDTreeSearcher(X);
[n,d] = knnsearch(Mdl,newpoint,'k',kPPV);
line(X(n,1),X(n,2),'color',[.5 .5 .5],'marker','o',...
    'linestyle','none','markersize',10)
legend('setosa','versicolor','virginica','nouvel iris','kPPV')
```

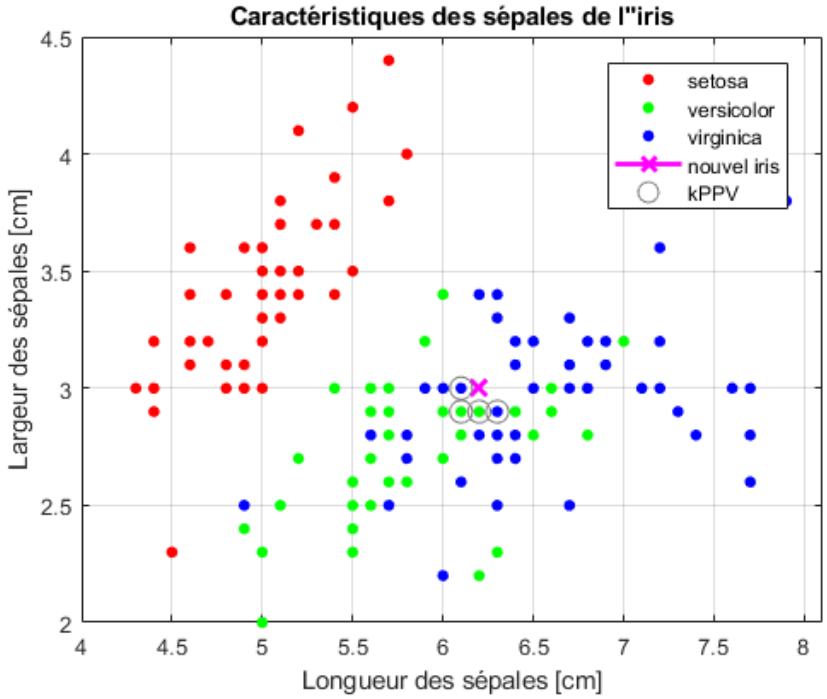


Figure 57 - Tracé des k plus proches voisins.

```
X(n,:)
```

```
ans = 5×2
    6.2000    2.9000
    6.1000    3.0000
    6.1000    3.0000
    6.3000    2.9000
    6.1000    2.9000
```

```
tabulate(species(n));
```

```
      Value    Count   Percent
 versicolor        3    60.00%
  virginica        2    40.00%
```

Observation - Réflexion

- Après plusieurs essais conclure sur la qualité de la prédiction.

Afficher les frontières de décision

```
% newpoint = [6.2 3];
kPPV=1;
classifieur= fitcknn(X,y,'NumNeighbors',kPPV);



gscatter(X(:,1),X(:,2),species);
hold on

x1range = min(X(:,1)):.01:max(X(:,1));
x2range = min(X(:,2)):.01:max(X(:,2));
[xx1, xx2] = meshgrid(x1range,x2range);
XGrid = [xx1(:) xx2(:)];

especePredite = predict(classifieur,XGrid);
gscatter(xx1(:), xx2(:), especePredite,'rgb');
hold on
plot(newpoint(1),newpoint(2),'marker','x','color','m',...
   'markersize',10,'linewidth',2)
title('Frontières de décision')
legend('setosa','versicolor','virginica','nouvel iris')
hold off
```

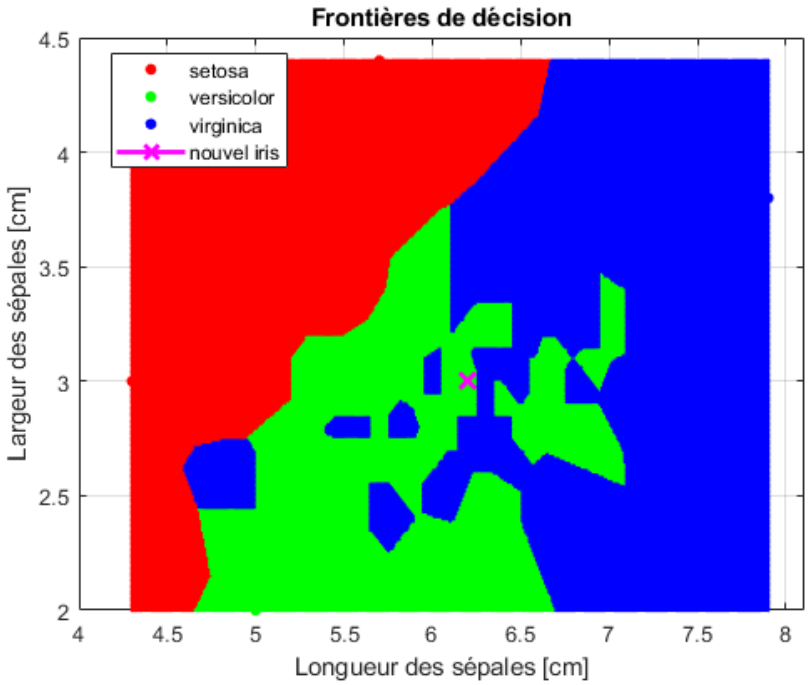


Figure 58 - Frontières de décision.

Observation - Réflexion

- En prenant k=1 et k=149, interpréter alors la construction des frontières de décision.

8- Conclusion

Nous venons de passer en revue plusieurs modèles qui permettent le traitement des problèmes de classification : la régression logistique, le perceptron et les k plus proches voisins. Chaque algorithme possède ses avantages et inconvénients.

La *régression logistique* est un modèle qui peut prédire la probabilité d'une réponse binaire d'appartenir à une classe ou à l'autre. Parce que sa mise en œuvre est relativement aisée, la régression logistique est couramment utilisée pour une première approche d'un problème de classification binaire.

Un *réseau de neurones* est constitué de neurones interconnectés reliant la sortie attendue aux entrées pour un apprentissage supervisé. Le modèle est entrainé de manière itérative en ajustant les paramètres (poids et biais) pour que la sortie du réseau s'ajuste au mieux à la sortie attendue. La minimisation de l'erreur entre sortie attendue et sortie estimée par le modèle s'obtient par un algorithme de descente de gradient. Toute fonction, linéaire ou pas, peut être approximée par un réseau de neurones. Ce modèle peut donc être aussi utilisé pour traiter un problème de régression, en particulier un problème de régression non-linéaire. La principale difficulté rencontrée lorsqu'on souhaite mettre en œuvre un réseau de neurones est le choix des hyperparamètres : nombre de neurones, nombre de couches, choix des fonctions d'activation…

Enfin, le modèle des *k plus proches voisins* classe un objet en fonction de la classe d'appartenance de ses plus proches voisins. La décision s'obtient par la majorité. Pour trouver les plus proches voisins d'un objet l'algorithme calcule une distance. La distance euclidienne est la plus courante mais il en existe d'autres adaptées aux différents problèmes à résoudre. Le calcul de la distance par rapport à l'ensemble des données du jeu de référence peut nuire aux performances de l'algorithme. Il convient alors de partitionner les données en utilisant un arbre binaire. Le modèle des *k plus proches voisins* peut être utilisé pour traiter un problème de régression. Dans ce cas la valeur estimée par le modèle est la moyenne des valeurs des plus proches voisins.

Pour finir, l'ensemble des outils présentés dans ce document de synthèse, va permettre l'analyse de problématiques faisant appel à des solutions d'apprentissage automatique et donc d'intelligence artificielle.

Par exemple, nous travaillerons sur des problématiques de robotique avec le robot alphai développé par la société française Learning Robots. Ce robot est équipé d'un certain nombre de capteurs (camera, ultrasons, infrarouge…) délivrant des signaux dont le traitement par un réseau de neurones lui permettra d'apprendre un comportement…

Figure 59 - Robot alphai développé par Learning Robots

La prochaine révolution de l'intelligence artificielle

« …dans le domaine des sciences et des technologies, l'artefact a toujours précédé la science qu'on a construite autour…Quel est l'équivalent pour l'intelligence ? »

https://bit.ly/ConfYannLeCun

Pour sa seconde intervention à USI (Unexpected Sources of Inspiration), Yann Le Cun nous parle de l'état de l'Intelligence Artificielle, son histoire mais aussi les défis des prochaines décennies à venir en matière d'Intelligence Artificielle.