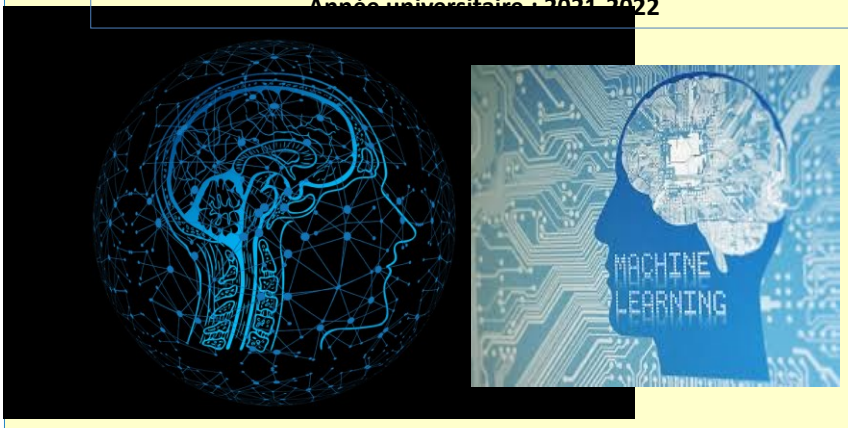
Machine Learning & Text Mining

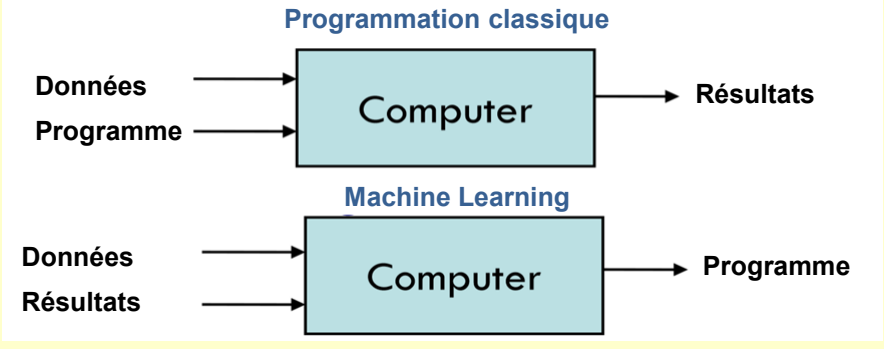
Année universitaire : 2021-2022



Pr. Abdelali ZAKRANI
Professeur Habilité à ENSAM de Casablanca
Docteur en informatique de l'ENSIAS

Introduction à l'apprentissage automatique

Définition: Machine Learning (apprentissage automatique) concerne l'étude, la conception et le développement d'algorithmes qui permettent aux ordinateurs d'apprendre sans être explicitement programmés (Arthur Samuel).



2

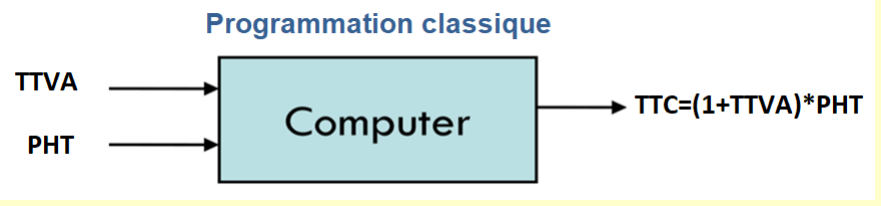
Introduction à l'apprentissage automatique

Programmation classique:

- Développement de programmes pour la gestion de stock
- Administration d'une entreprise, ….
- Gestion d'un cabinet de formations, …..

On peut trouver un algorithme qui permet de trouver les résultats à partir des données saisies par l'utilisateur.

Exemple:

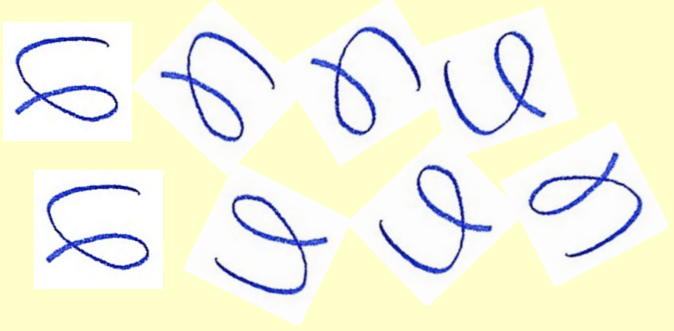


3

Introduction à l'apprentissage automatique

Machine Learning: On ne dispose pas d'une formule ou d'une méthode analytique qui permet de trouver les résultats à partir des données.

Connaissez vous une méthode analytique qui permet de reconnaitre l'une des images suivantes ? S'agit-il du chiffre 6 ou 9 ou la lettre v ?



4

Quelques applications de Machine Learning

Exemples:

- **Prédiction des prix:** Estimer le prix d'une maison en fonction de sa superficie, sa localisation, possibilité de Parking ou non etc… Ces estimations sont faites en observant d'autres produits similaires pour en tirer des conclusions.

- **Diagnostique médical:** En se basant sur les données médicales d'un patient, l'algorithme peut diagnostiquer si le sujet est atteint d'une maladie donnée. Parfois, ces algorithmes peuvent alerter d'un incident grave de santé avant que cela n'arrive, notamment pour les crises cardiaques.

Quelques applications de Machine Learning

https://www.01net.com/actualites/ia-ces-chercheurs-ont-decouvert-un-puissant-antibiotique-grace-a-l-apprentissage-automatique-1862232.html?fbclid=IwAR0sdwCFrxJdMg2d75jAtGZ1EF_JH21ouRfr98v2hx2cEHpzYcH0R9QSTvM

Le 21 avril 2020, Des chercheurs du MIT (Massachussetts Institute of Technology) ont créé un algorithme capable de trouver automatiquement de nouveaux antibiotiques, ce qui devrait ravir l'industrie pharmaceutique en s'appuyant sur des techniques d'apprentissage automatique.
Cette découverte est d'autant plus importante que les bactéries ont tendance à devenir de plus en résistantes. Cette nouvelle méthode fondée sur l'intelligence artificielle pourrait donc sauver les vies des millions de personnes.

Quelques applications de Machine Learning

- **Recommandation de produits:** Ce type de système se base sur les historiques d'achats, les recherches faites en ligne (Tracking Web) par un internaute pour lui recommander des produits qui pourront l'intéresser. Pour Amazon, cette fonctionnalité est critique car elle est au cœur de l'augmentation des volumes de vente et par conséquent des gains de la société.
- **Regroupement d'items:** Ce type de technique sert notamment pour l'application Iphoto d'Apple pour regrouper les images en fonction des gens qui s'y retrouvent. Généralement, les données n'ont pas d'étiquettes, et l'algorithme tentera de retrouver des items similaires et les regroupera dans un même groupe.

7

Quelques applications de Machine Learning

- **Conduite autonome:** En apprenant le comportement de conduite des humains, les algorithmes de Deep Learning avec l'apprentissage par renforcement (reinforcement learning) permettent d'apprendre des tâches complexes comme la conduite.
- **Banque et Assurance:**

  Prédire qu'un client va quitter sa banque.
  Définir ceux qui veulent changer leur assurance vie.
  Améliorer la satisfaction client en proposant les offres les plus pertinentes possibles.

  Prise de décision pour donner un crédit à un client ou non, prédire les risques, ….

8

Quelques applications de Machine Learning

Une banque reçoit quotidiennement plusieurs demandes d'approbation de crédit. Elle veut automatiser leurs processus d'évaluation. La banque n'a pas de formule magique pour savoir quand il faut accorder un crédit, mais il a beaucoup de données.

L'existence de données nous ramène à l'approche d'apprentissage des données. Donc, la banque utilise l'historique des documents des anciens clients pour trouver la bonne formule d'approbation de crédit.

Application du Machine Learning

1- Existence d'un modèle à apprendre: Il existe une corrélation entre les variables d'entrées et de sorties. On sait qu'un modèle existe même si on le connait pas.

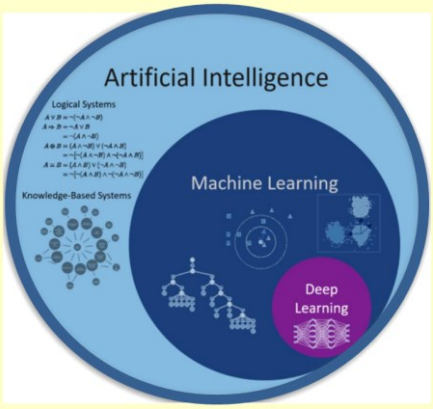
2-Modélisation mathématique est impossible: On ne peut pas résoudre le modèle mathématiquement (pas de solution analytique).

3- Existence des données: Il existe des données qui représentent le modèle.

Datasets: exemples
https://archive.ics.uci.edu/ml/index.php
https://www.kaggle.com/datasets

L'apprentissage automatique est une branche de l'intelligence artificielle

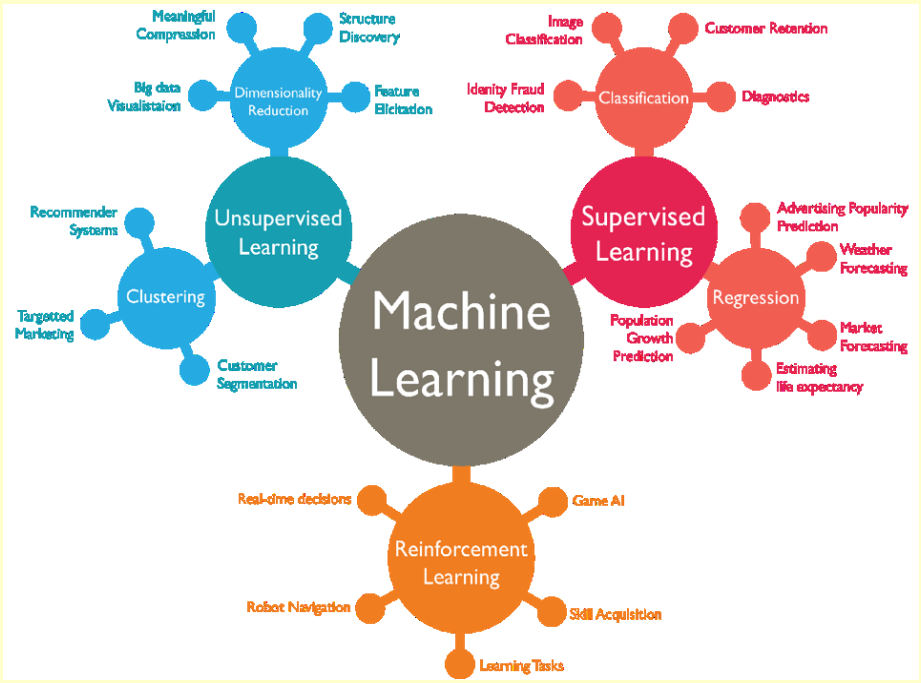
Le **Machine Learning** ou **Apprentissage automatique** est une branche de l'intelligence artificielle qui concerne la conception, l'analyse, le développement et l'implémentation de modèles, permettant à une machine d'apprendre à partir des données par un processus systématique afin de remplir une tâche.



11

Les types d'apprentissage automatique

- Apprentissage supervisé (supervised learning)
- Non supervisé (unsupervised learning)
- Semi supervisé (semi-supervised learning)
- Par renforcement(reinforcement learning)



Apprentissage supervisé (supervised learning)

Des données (entrées) annotées de leurs sorties pour entraîner le modèle, c'est-à-dire que à chaque **entrée est** associée à une classe **cible (sortie)**, une fois entraîné, le modèle (l'algorithme de ML) devient capable de prédire (éventuellement avec un pourcentage d'erreur) la cible sur de **nouvelles données** non annotées.

Exemple:

Les données d'entrée représentent des images et la cible (ou *target* en anglais) représente la catégorie de photos.

Apprentissage supervisé (supervised learning)

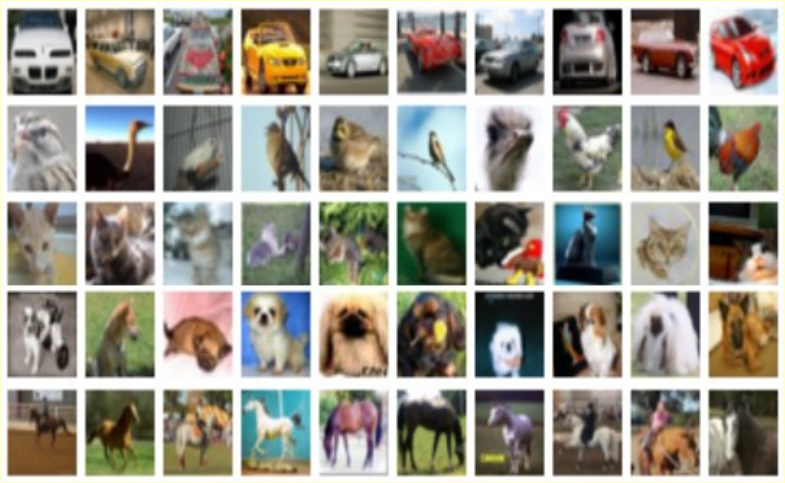
Voiture

Oiseau

Chat

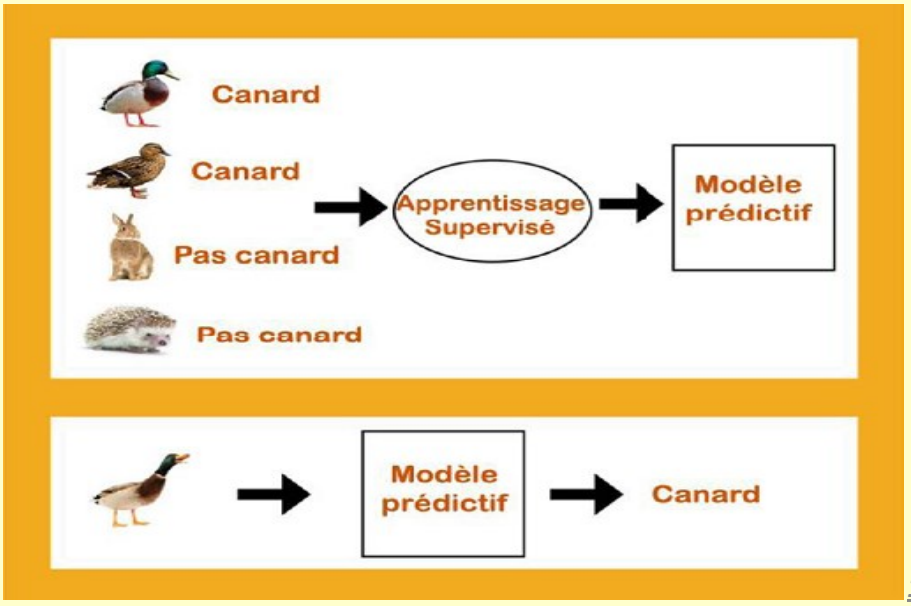
Chien

cheval



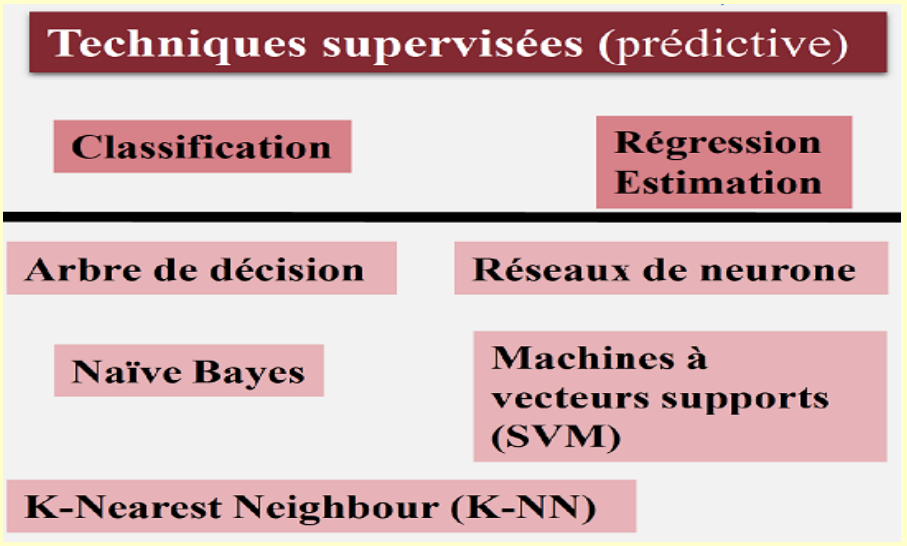
15

Apprentissage supervisé (supervised learning)



16

Apprentissage supervisé (supervised learning)



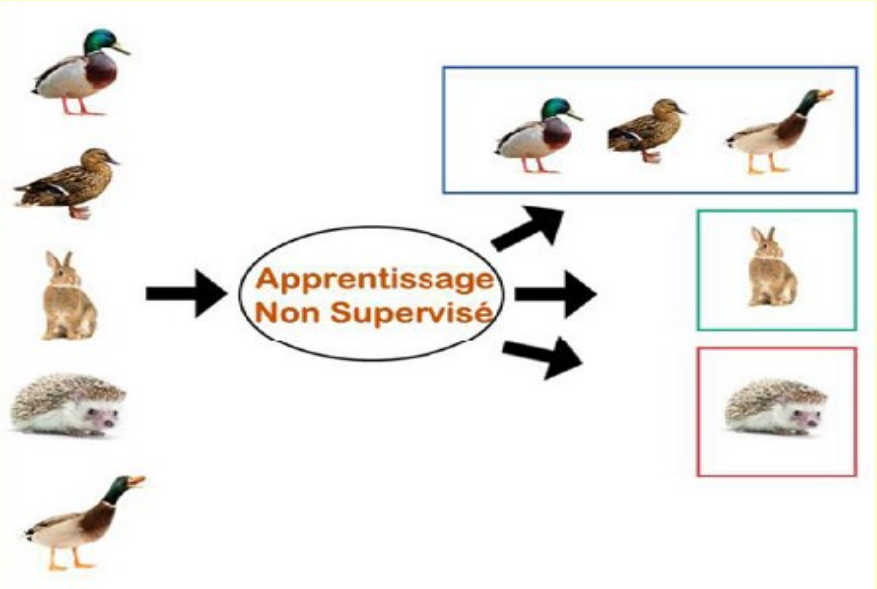
17

Apprentissage non supervisé (unsupervised learning)

En apprentissage non supervisé, les données d'entrées ne sont pas annotées. L'algorithme d'entraînement s'applique dans ce cas à trouver les similarités et distinctions au sein de ces données, et à regrouper ensemble celles qui partagent des caractéristiques communes. Dans notre exemple, les photos similaires seraient ainsi regroupées automatiquement au sein d'une même catégorie.

18

Apprentissage non supervisé (unsupervised learning)



19

Apprentissage non supervisé (unsupervised learning)

Techniques non supervisées (descriptive)

Clustering	Associations
K-Means K-moyennes	Apriori Eclat, FP-growth RElim, SaM, JIM
Expectation Maximisation (EM)	

20

Apprentissage sem-supervisé (semi-supervised learning)

- L'**apprentissage semi-supervisé** est une classe de techniques d'apprentissage automatique qui utilise un ensemble de données étiquetées et non étiquetées. Il se situe ainsi entre l'apprentissage supervisé qui n'utilise que des données étiquetées et l'apprentissage non supervisé qui n'utilise que des données non étiquetées. Il a été démontré que l'utilisation de données non étiquetées, en combinaison avec des données étiquetées, permet d'améliorer significativement la qualité de l'apprentissage.
- Un autre intérêt provient du fait que l'étiquetage de données nécessite souvent l'intervention d'un utilisateur humain. Lorsque les jeux de données deviennent très grands, cette opération peut s'avérer fastidieuse. Dans ce cas, l'apprentissage semi-supervisé, qui ne nécessite que quelques étiquettes, revêt un intérêt pratique évident.

Apprentissage par renforcement(reinforcement learning)

L'**apprentissage par renforcement** consiste, pour un agent autonome (robot, etc.), à apprendre les actions à prendre, à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps. L'agent est plongé au sein d'un environnement, et prend ses décisions en fonction de son état courant. En retour, l'environnement procure à l'agent une récompense, qui peut être positive ou négative. L'agent cherche, au travers d'expériences itérées, un comportement décisionnel (appelé *stratégie* ou *politique*, et qui est une fonction associant à l'état courant l'action à exécuter) optimal, en ce sens qu'il maximise la somme des récompenses au cours du temps.

Les 7 étapes de l'apprentissage automatique

L'apprentissage automatique ne se résume pas à un ensemble d'algorithmes mais suit une succession d'étapes:

1) L'acquisition de données : l'algorithme se nourrissant des données en entrée, c'est une étape importante. Il en va de la réussite du projet, de récolter des données pertinentes et en quantité suffisante.

2) La préparation et le nettoyage de la donnée : les données recueillies doivent être retouchées avant utilisation. En effet, certains attributs sont inutiles, d'autre doivent être modifiés afin d'être compris par l'algorithme, et certains éléments sont inutilisables car leurs données sont incomplètes.

23

Les 7 étapes de l'apprentissage automatique

3) Choix du modèle.

4) Formation et apprentissage du modèle

5) L'évaluation : une fois l'algorithme d'apprentissage automatique entraîné sur un premier jeu de donnée, on l'évalue sur un deuxième ensemble de données afin de vérifier que le modèle ne fasse pas de surapprentissage.

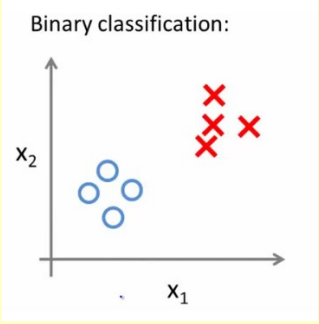
6) Réglage des paramètres: Ajuster les paramètres pour de meilleures performances

7) Le déploiement : le modèle est déployé en production pour faire des prédictions, et potentiellement utiliser les nouvelles données en entrée pour se ré-entraîner et être amélioré.

24

Classification binaire

- La classification est une tâche très répandue en Machine Learning. Dans ce genre de problématique, on cherche à mettre une étiquette (un label) sur une observation : une tumeur est-elle maligne ou non, une transaction est- elle frauduleuse ou non… ces deux cas sont des exemples de classification.
- Quand on a deux choix d'étiquettes possibles (tumeur maligne ou non), on parle de Binary Classification (classification binaire). Par ailleurs, l'étiquette Y aura deux valeurs possibles 0 ou 1. En d'autres termes $Y \in \{0,1\}$



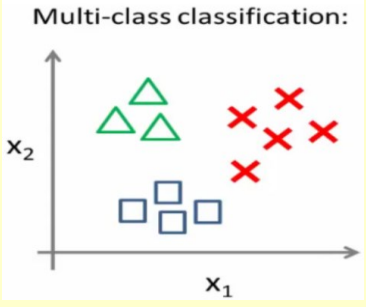
25

Classification binaire et multiclasses

Le but du jeu c'est qu'on trouve une ligne (Boundary Decision) séparant les deux groupes (les cercles et les carrés).

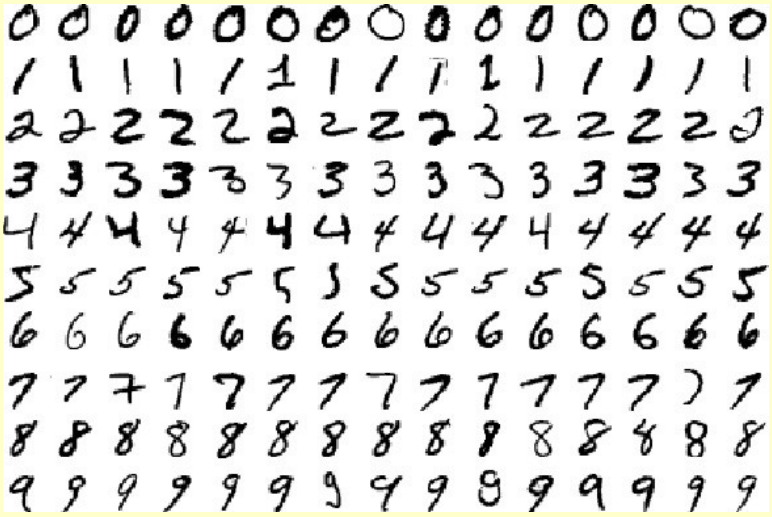
Note : Pour l'exemple de la tumeur, on peut attribuer arbitrairement la classe (étiquette) 1 pour dire qu'il s'agit d'une tumeur maligne et la valeur 0 pour les tumeurs bénignes.

Quand notre problème a plusieurs étiquettes possibles (par exemple classifier un article dans une catégorie ("sport", "politique", "High-Tech")…), on parle de Multi-class classification. Dans ce cas on peut attribuer arbitrairement les numéros des classes aux observations du Training Set. Dans l'exemple vu en TP en haut pour la reconnaissance des chiffres : dix classes: 0,1,2,3,4,5,6,7,8,9



26

Application de Machine Learning sur la reconnaissance des nombres manuscrits



27

Application de Machine Learning sur la reconnaissance des nombres manuscrits

Prenons des images en niveau de gris : chaque pixel est codé sur 8 bits
Pixel est représenté par un nombre compris entre 0 et 255.
Valeur du pixel proche de 0: noir
Valeur du pixel proche de 1: blanc

28

Application de Machine Learning sur la reconnaissance des nombres manuscrits

```
import numpy as np
import matplotlib.pyplot as plt
import pandas
from sklearn.tree import DecisionTreeClassifier
data=pandas.read_csv("d:\\train.csv").as_matrix()
clf=DecisionTreeClassifier()
x=data[0:21000,1:]
label=data[0:21000,0]
clf.fit(x,label)
xtest=data[21000:,1:]
actual_label=data[21000:,0]
p=clf.predict(xtest)
```

29

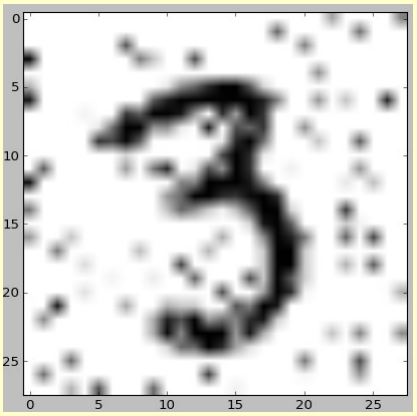
Application de Machine Learning sur la reconnaissance des nombres manuscrits

```
p=clf.predict(xtest)
count=0
for i in range(0,21000):
count+=1 if p[i]==actual_label[i] else 0
print("Accuracy=",(count/21000)*100)
d=xtest[5]
Nombre_de_pixels_errones=100
for i in range(Nombre_de_pixels_errones):
    position=np.random.randint(0,784,1)[0]
    bruit=np.random.randint(-200,200,1)[0]
    d[position]+=bruit
    d[position]=d[position]%255
print(clf.predict([d]))
d.shape=(28,28)
plt.imshow(255-d,cmap='gray')
plt.show()
```

30

Application de Machine Learning sur la reconnaissance des nombres manuscrits

Avec un Nombre_de_pixels_errones=100 on obtient l'image suivante:

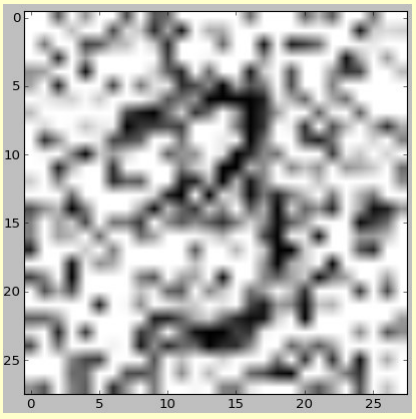


L'instruction print(clf.predict([d])) a donné la valeur 3 donc une reconnaissance réussie

31

Application de Machine Learning sur la reconnaissance des nombres manuscrits

Avec un Nombre_de_pixels_errones=500 on obtient l'image suivante:

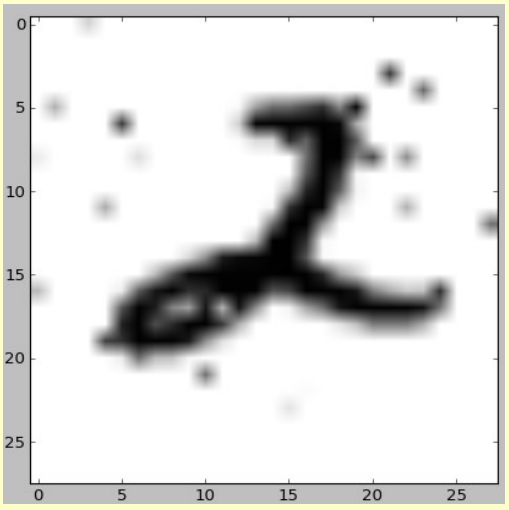


L'instruction print(clf.predict([d])) a donné la valeur 5 donc une reconnaissance non réussie à cause du bruit très fort.

32

Application de Machine Learning sur la reconnaissance des nombres manuscrits

Le programme donné en haut a pu reconnaitre l'image suivante: chiffre 2



33

Application de Machine Learning sur la reconnaissance des nombres manuscrits

Refaire le même TP avec un modèle d'apprentissage KNN

34

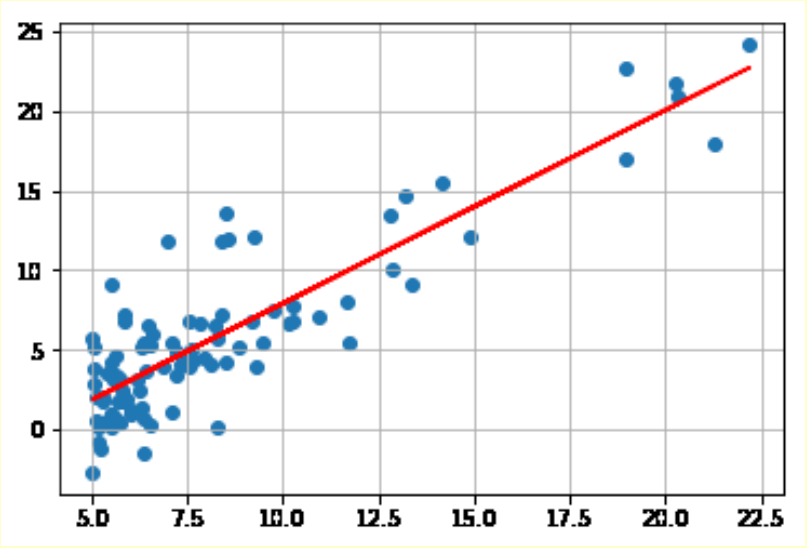
Supervised (inductive) learning

Régression linéaire

Les algorithmes de régression linéaire modélisent la relation entre des variables prédictives et une variable cible. La relation est modélisée par une fonction de prédiction. Le cas le plus simple est la régression linéaire univariée. Elle va trouver une fonction sous forme de droite pour estimer la relation.

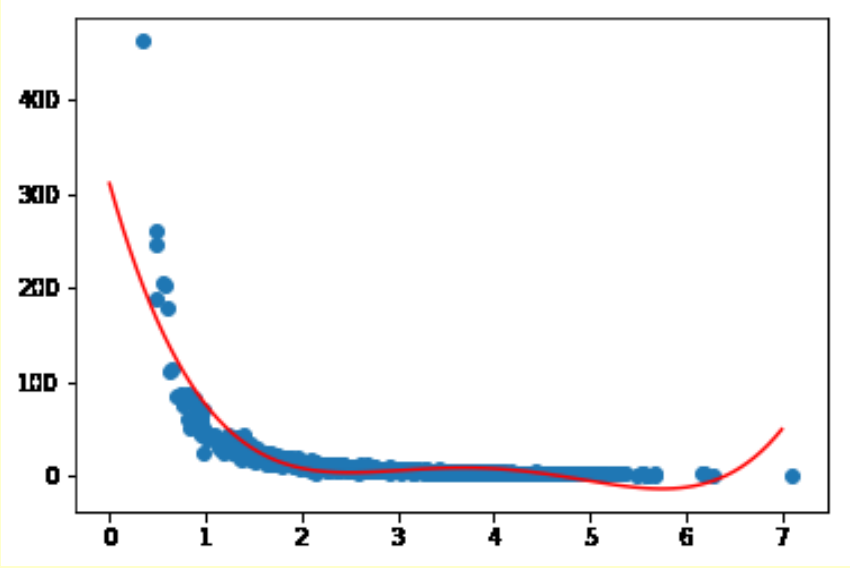
La régression polynomiale permet de modéliser des relations complexes qui ne sont pas forcément linéaires.

Relation linéaire entre des variables prédictives et la variable cible



37

Relation non linéaire entre des variables prédictives et la variable cible



38

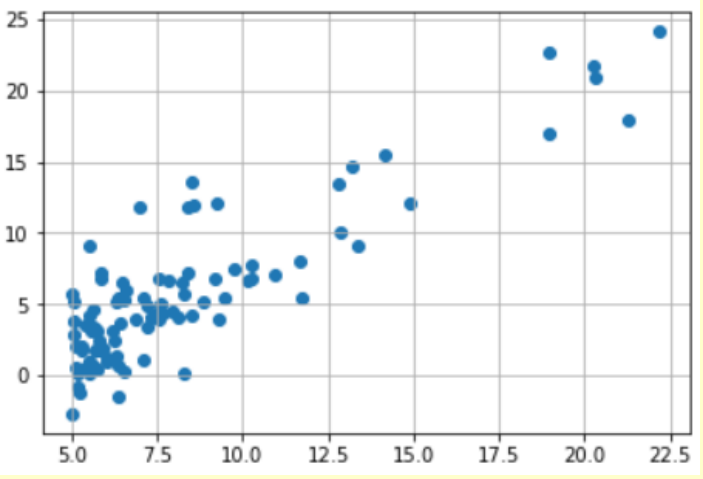
Régression linéaire

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
df = pd.read_csv("d:\\univariate_linear_regression_dataset.csv")
X =  df.iloc[:,0]
Y =  df.iloc[:,1]
axes = plt.axes()
axes.grid()
plt.scatter(X,Y)
plt.show()
SL=stats.linregress(X, Y)
slope=SL.slope; intercept=SL.intercept; coef_correlation=SL.rvalue
def predict(x):
    return slope * x + intercept
axes = plt.axes()
axes.grid()
plt.scatter(X,Y)
fitLine = predict(X)
plt.plot(X, fitLine, c='r')
plt.show()
```

39

Régression linéaire

```
axes = plt.axes()
axes.grid()
plt.scatter(X,Y)
plt.show()
```



40

Régression linéaire

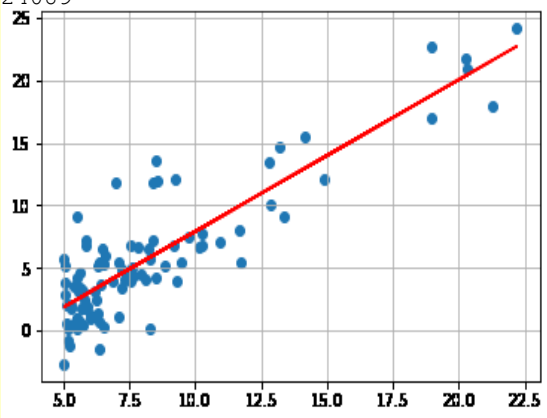
- Le fonction de prédiction pour une régression linéaire univariée est comme suit :

$$H(x)=intercept+slope*x$$

- avec : $slope$: représente la "pente" de la ligne de prédiction et $intercept$ représente le point d'intersection avec l'axe des ordonnées

SL= stats.linregress(X, Y)

```
SL.slope est 1.2135472539083585
SL.intercept est -4.211504005424089
SL.rvalue:
0.8721572919685905
r= X.corr(Y)
print(r)
0.8721572919685905
```



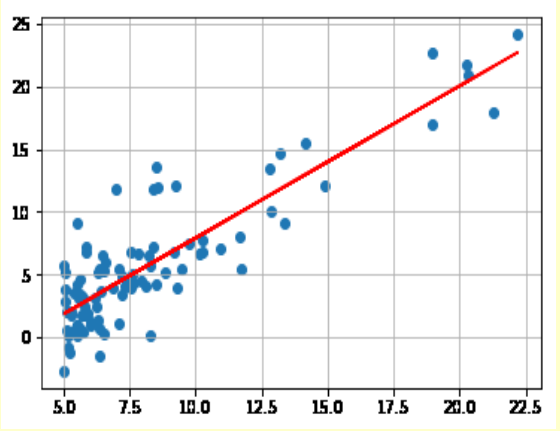
Régression linéaire

- **Prédiction d'une nouvelle observation:**

  On voit que pour la valeur x = 22.5, la valeur de y est environ 25.

  Utilisons la fonction $predict$ pour trouver une estimation de $H(x$=22.5):

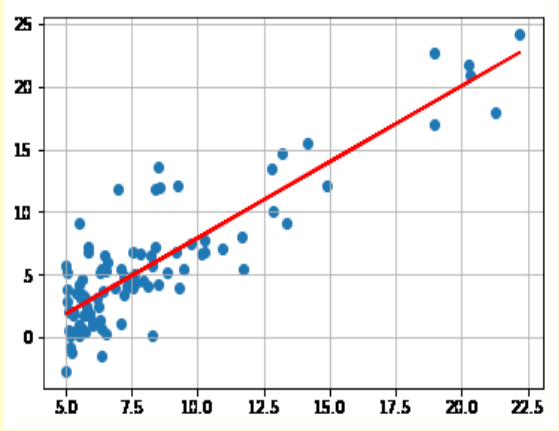
  predict(22.5) ➔ 23.093309207513975



42

Régression linéaire avec l'algorithme de descente de gradient

Dans la pratique, les Data Scientists utilisent le package **sklearn**, qui permet d'écrire un tel code en 4 lignes, mais ici nous écrirons chaque fonction mathématique de façon explicite, ce qui est un **bon exercice** pour la compréhension du modèle.



43

Régression linéaire avec l'algorithme de descente de gradient

$$X = \begin{pmatrix} x & 1 \end{pmatrix} \quad ; \quad \theta = \begin{pmatrix} a \\ b \end{pmatrix} \quad et \quad f(x) = \theta.X = \begin{pmatrix} x & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = a.x + b$$

Objectif: trouver θ qui donne une bonne prédiction d'une entrée x.

Pour développer un modèle linéaire avec la descente de gradient, il faut implémenter les 4 fonctions clefs suivantes :

- la fonction de notre modèle : $f(x) = X.\theta$
- la fonction Cout : $J(\theta) = \frac{1}{2m}\sum(X.\theta - Y)^2$
- le gradient : $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}X^T.(X.\theta - Y)$
- la descente de gradient : $\theta = \theta - \alpha \times \frac{\partial J(\theta)}{\partial \theta}$

α: est appelé taux d'apprentissage: learning_rate

m: est la taille de l'ensemble d'apprentissage

Objectif: trouver θ qui minimise J(θ)

44

Régression linéaire avec l'algorithme de descente de gradient

- la fonction de notre modèle : $f(x) = X.\theta$
- la fonction Cout : $J(\theta) = \frac{1}{2m}\sum(X.\theta - Y)^2$
- le gradient : $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}X^T.(X.\theta - Y)$
- la descente de gradient : $\theta = \theta - \alpha \times \frac{\partial J(\theta)}{\partial \theta}$

```
def f(X, theta):
    return X.dot(theta)
def erreur_somme_des_distances(X, y, theta):
    m = len(y)
    return (1/(2*m))*np.sum((f(X, theta)-y)**2)
def gradient(X, y, theta):
    m = len(y)
    return 1/m * X.T.dot(f(X, theta) - y)
```

45

Régression linéaire avec l'algorithme de descente de gradient

- la fonction de notre modèle : $f(x) = X.\theta$
- la fonction Cout : $J(\theta) = \frac{1}{2m}\sum(X.\theta - Y)^2$
- le gradient : $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}X^T.(X.\theta - Y)$
- la descente de gradient : $\theta = \theta - \alpha \times \frac{\partial J(\theta)}{\partial \theta}$

```
def gradient_descent(X, y, theta, learning_rate, n_iterations):
    # création d'un tableau de stockage pour enregistrer l'évolution des er
    historique_des_erreurs = np.zeros(n_iterations)
    for i in range(0, n_iterations):
        theta = theta - learning_rate * gradient(X, y, theta) # mise a jour
        historique_des_erreurs[i] = erreur_somme_des_distances(X, y, theta)
    return theta, historique_des_erreurs
```

46

Régression linéaire avec l'algorithme de descente de gradient

```
import  numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("d:\\univariate_linear_regression_dataset.csv")
x=  df.iloc[:,0] ; n_samples = len(x); x=np.array(x).reshape((n_samples, 1)) #96
y=  df.iloc[:,1] ; y=np.array(y).reshape((n_samples, 1))
x=np.array(x).reshape((n_samples, 1))
plt.scatter(x, y) # afficher les résultats. X en abscisse et y en ordonnée
plt.show()
# ajout de la colonne de biais a X
X = np.hstack((x, np.ones(x.shape)))
# création d'un vecteur parametre theta
theta = np.random.randn(2, 1)
print(theta)
```

47

Régression linéaire avec l'algorithme de descente de gradient

```
def f(X, theta):
    return X.dot(theta)
def erreur_somme_des_distances(X, y, theta):
    m = len(y)
    return (1/(2*m))*np.sum((f(X, theta)-y)**2)
def gradient(X, y, theta):
    m = len(y)
    return 1/m * X.T.dot(f(X, theta) - y)
def gradient_descent(X, y, theta, learning_rate, n_iterations):
    # création d'un tableau de stockage pour enregistrer l'évolution des erreurs
    historique_des_erreurs = np.zeros(n_iterations)
    for i in range(0, n_iterations):
        theta = theta - learning_rate * gradient(X, y, theta) # mise a jour du theta
        historique_des_erreurs[i] = erreur_somme_des_distances(X, y, theta)
    return theta, historique_des_erreurs
```

48

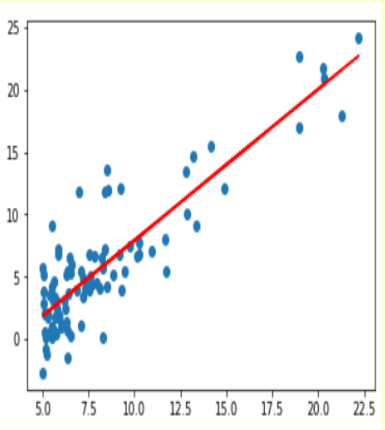
Régression linéaire avec l'algorithme de descente de gradient

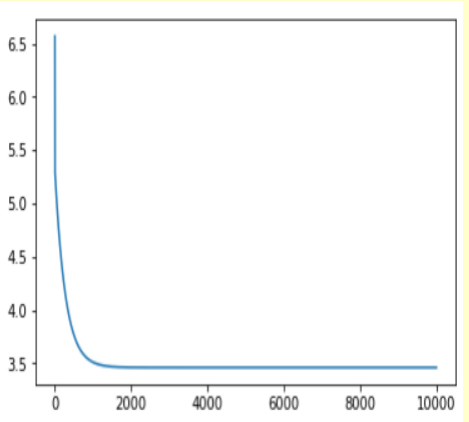
```
# Example de test :
n_iterations = 10000
learning_rate = 0.01
theta_final, historique_des_erreurs=gradient_descent(X, y, theta,
    learning_rate, n_iterations)
print(theta_final) # theta une fois que la machine a  été entrainée
 # création d'un vecteur prédictions qui contient les prédictions de notre modele final
predictions = f(X, theta_final)
 # Affiche les résultats de prédictions (en rouge) par rapport a notre Dataset (en bleu)
plt.scatter(x, y)
plt.plot(x, predictions, c='r')
plt.show()
plt.figure()
plt.plot(range(n_iterations), historique_des_erreurs)
plt.show()
print(f(np.array([22.5,1]), theta_final))
```

49

Régression linéaire avec l'algorithme de descente de gradient

f(np.array([22.5,1]), theta_final) ➔ donne: array([23.09330913])





50

Régression linéaire avec l'algorithme de descente de gradient

$f(x) = x^2 - x + 1 \Rightarrow \nabla f(x) = \frac{\partial f(x)}{\partial x} = f'(x) = 2x - 1$

Minimum au point x: f(x)=0
X=0.5

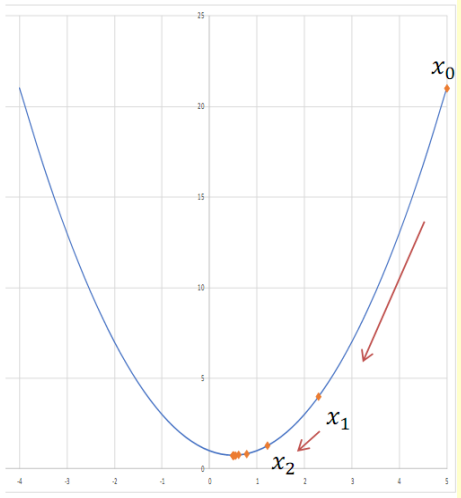
$\eta = 0.3$

eta 0.3

$x_0 = 5$

x	f'(x_t)	f(x)
5.0000		21.0000
2.3000	9.0000	3.9900
1.2200	3.6000	1.2684
0.7880	1.4400	0.8329
0.6152	0.5760	0.7633
0.5461	0.2304	0.7521
0.5184	0.0922	0.7503
0.5074	0.0369	0.7501
0.5029	0.0147	0.7500
0.5012	0.0059	0.7500
0.5005	0.0024	0.7500
0.5002	0.0009	0.7500
0.5001	0.0004	0.7500
0.5000	0.0002	0.7500

$x_{t+1} = x_t - \eta \times \nabla f(x_t)$



Régression linéaire avec l'algorithme de descente de gradient

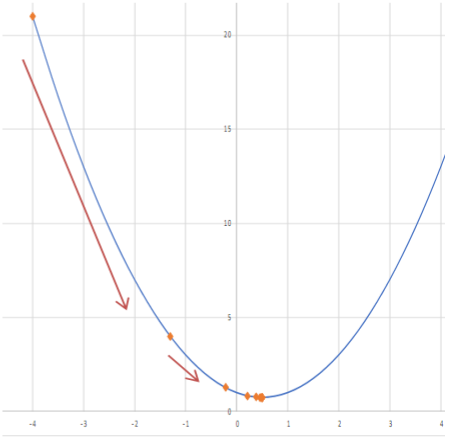
$f(x) = x^2 - x + 1 \Rightarrow \nabla f(x) = \frac{\partial f(x)}{\partial x} = f'(x) = 2x - 1$

Minimum au point x: f(x)=0
X=0.5

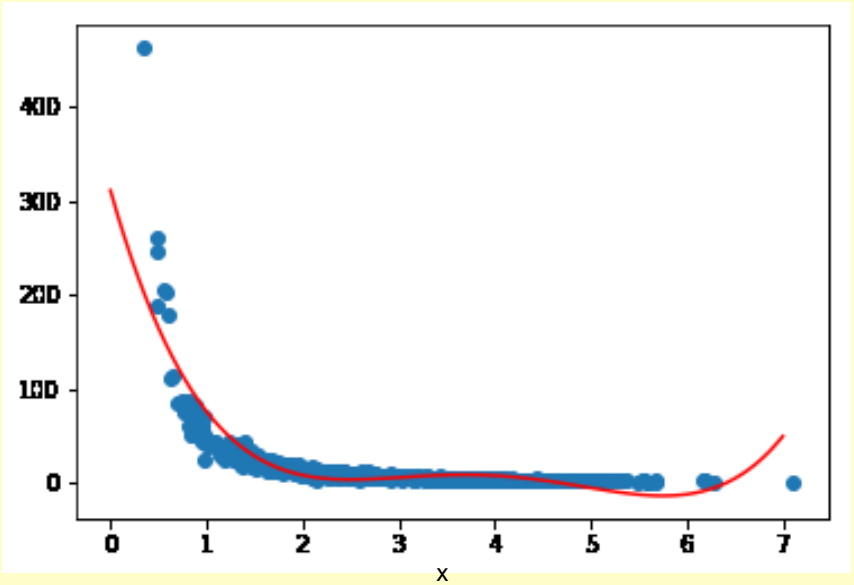
eta 0.3

x	f'(x_t)	f(x)
-4.0000		21.0000
-1.3000	-9.0000	3.9900
-0.2200	-3.6000	1.2684
0.2120	-1.4400	0.8329
0.3848	-0.5760	0.7633
0.4539	-0.2304	0.7521
0.4816	-0.0922	0.7503
0.4926	-0.0369	0.7501
0.4971	-0.0147	0.7500
0.4988	-0.0059	0.7500
0.4995	-0.0024	0.7500
0.4998	-0.0009	0.7500
0.4999	-0.0004	0.7500
0.5000	-0.0002	0.7500

$x_{t+1} = x_t - \eta \times \nabla f(x_t)$

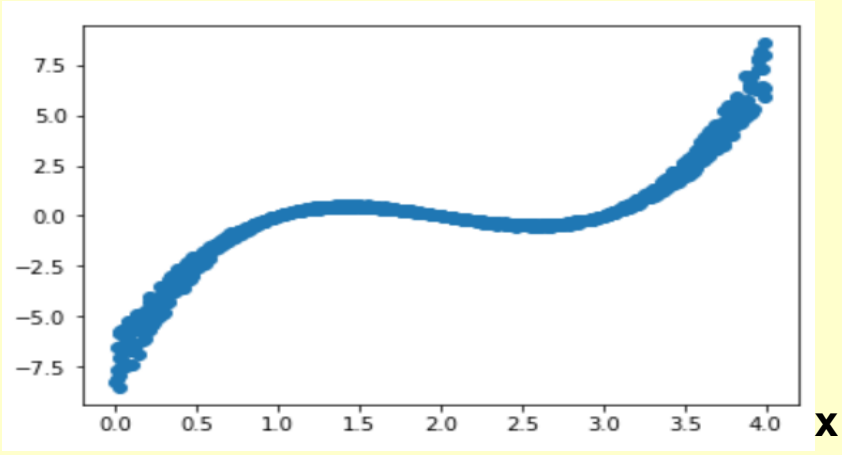


Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible



53

Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible



L'objectif ici c'est de trouver un polynôme P:P(X)≈y

Pour une régression polynomiale, la puissance de certaines variables indépendantes est supérieure à 1: $Y = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_1 \cdot x^1 + a_0 \cdot x^0$

54

Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible
Régression polynomiale

```
def afficher_pol(LPol):
    L=LPol[:]; L.reverse()
    L=[round(e,2) for e in L]
    for i in range(len(L)):
        print('('+str(L[i])+'*X^'+str(i)+')',end='+')
def Evaluer_pol(LPol,x):
    s=0
    L=LPol[:]
    L.reverse()
    for i in range(len(LPol)):
        s+=L[i]*(x**i)
    return(s)
```

```
P=[-4.04264497e-03, 1.26341942e+00, -7.46569108e+00,
1.36046931e+01, -7.39512820e+00]
```

Afficher_pol(P)➔(-7.4*X^0)+(13.6*X^1)+(-7.47*X^2)+(1.26*X^3)+(-0.0*X^4)

55

Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible

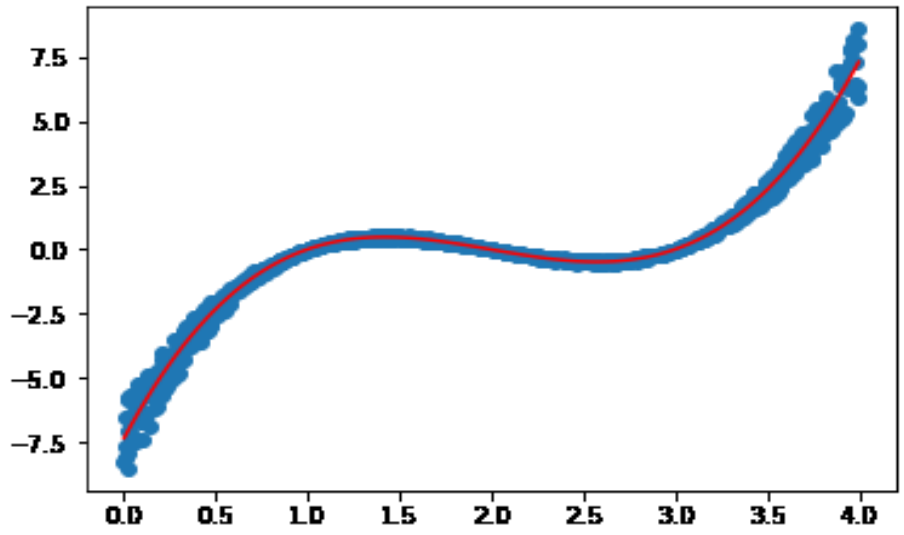
```
import numpy as np
import matplotlib.pyplot as plt
import pandas
data=pandas.read_csv("d:\\dataset_polynomial3.csv").as_matrix()
x=data[1:1001,0]
y=data[1:1001,1]
degre_polynome=5
p= np.poly1d(np.polyfit(x, y,degre_polynome))
LPol=list(p)
afficher_pol(LPol)
les_x= np.linspace(min(x), max(x), 100)
plt.scatter(x, y)
plt.plot(les_x, p(les_x), c='r')
plt.show()
```

P(x)=(-7.36*X^0)+(13.35*X^1)+(-7.01*X^2)+(0.96*X^3)+(0.08*X^4)+(-0.01*X^5)

56

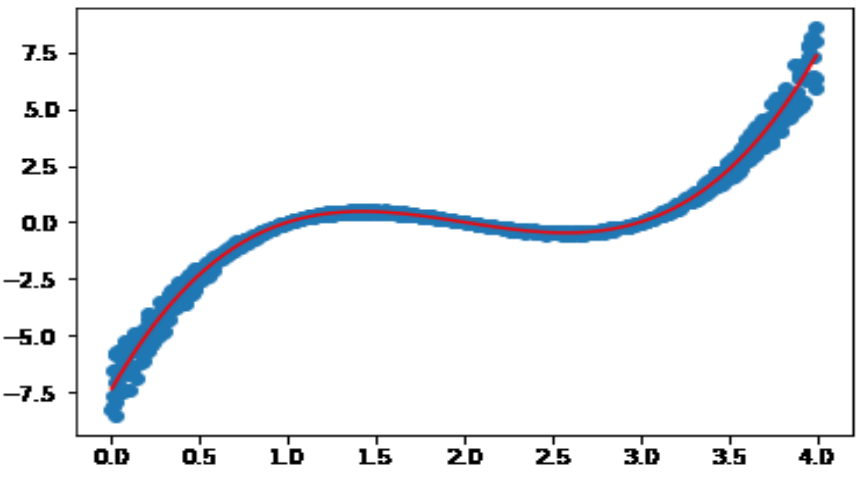
Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible

Polynôme de degré 5:
P(x)=(-7.36*X^0)+(13.35*X^1)+(-7.01*X^2)+(0.96*X^3)+(0.08*X^4)+(-0.01*X^5)



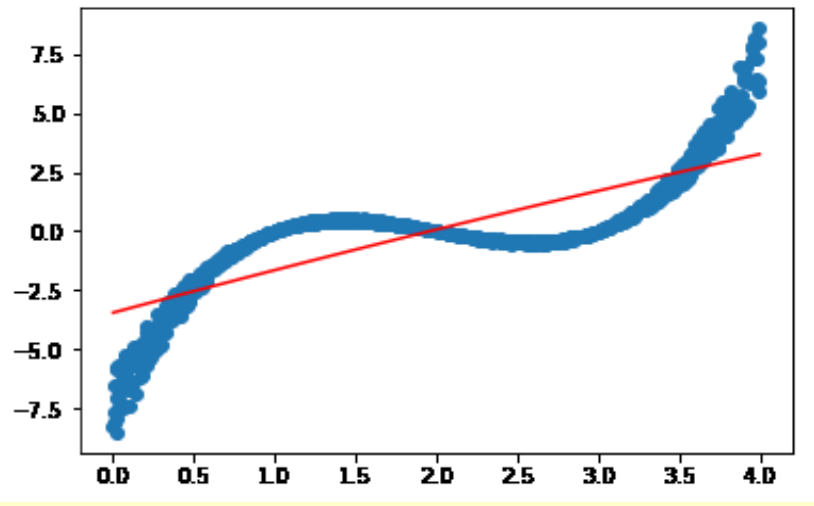
Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible

Polynôme de degré 3:
P(x)=(-7.38*X^0)+(13.53*X^1)+(-7.38*X^2)+(1.23*X^3)



Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible

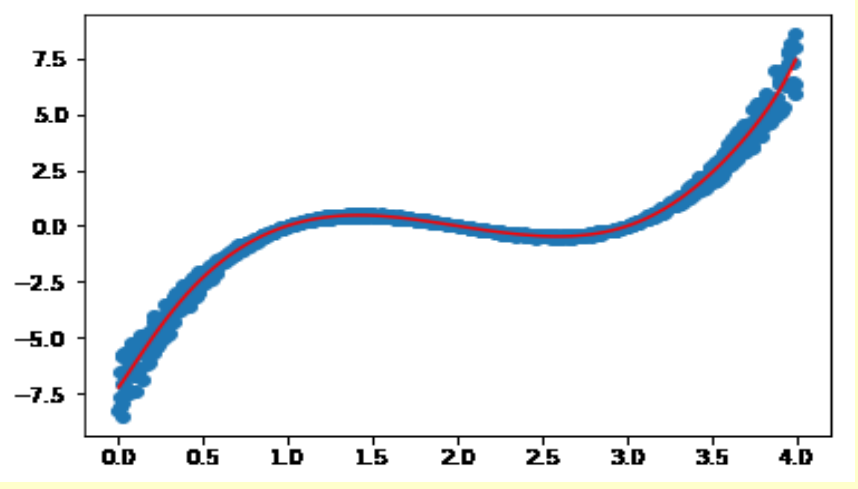
Polynôme de degré 2:
P(x)=(-3.45*X^0)+(1.84*X^1)+(-0.04*X^2)



59

Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible

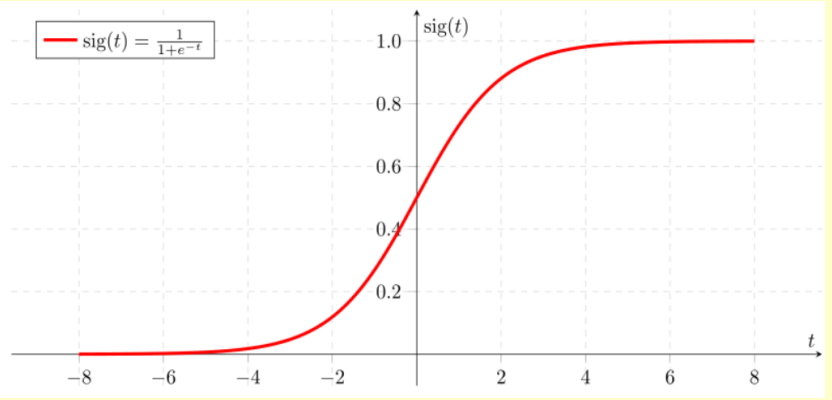
Polynôme de degré 10:
P(x)=(-7.24*X^0)+(10.79*X^1)+(6.73*X^2)+(-32.93*X^3)+(47.14*X^4)+
(-40.23*X^5)+(21.92*X^6)+(-7.62*X^7)+(1.63*X^8)+(-0.2*X^9)+(0.01*X^10



60

Sigmoid Function : La fonction pour la régression Logistique

la fonction sigmoid (**Sigmoid Function)** produit des valeurs comprises entre 0 et 1.



$$sigmoide(x) = \frac{1}{1+e^{-x}}$$

61

Sigmoid Function : La fonction pour la régression Logistique

- La régression logistique est utilisée pour trouver la probabilité d'un événement. On veut déterminer le succès ou l'échec d'un événement.
- Largement utilisé pour les problèmes de classification
- Ne nécessite pas de relation linéaire entre les variables dépendantes et indépendantes.
- La régression logistique permettra de répondre à des problèmes comme :

  Est-ce que le client est solvable pour lui accorder un crédit ou non?

  Est-ce que la tumeur diagnostiquée est bénigne ou maligne ?

  Est-ce qu'un message est spam ou non ?

62

Sigmoid Function : La fonction pour la régression Logistique

- La fonction hypothèse:

$$S(X^{(i)}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_n x_n$$

- $X^{(i)}$: une observation (du *Training Set* ou du *Test Set*), cette variable est un vecteur contenant: $[x_1, x_2, x_3, ...x_n]$
- x_i : est une variable prédictive (feature) qui servira dans le calcul du modèle prédictif
- θ_i : est un poids/paramètre de la fonction hypothèse. Ce sont ces θ_i qu'on cherche à calculer pour obtenir notre fonction de prédiction.
- θ_0 : est une constante nommée le **bias (biais)**

63

Sigmoid Function : La fonction pour la régression Logistique

- On peut réécrire $\theta_0 = \theta_0 * x_0$: avec $x_0=1$, Cela nous permet de réécrire notre fonction S(X) de façon plus compacte comme suit : $S(X) = \sum_{i=0}^{n+1} \theta_i * x_i$

On appelle cette fonction hypothèse : **la fonction score.** L'idée est de trouver des coefficients: $[\theta_0, \theta_1, \theta_2, \theta_3, ...\theta_n]$

de sorte que :

- $S(X^{(i)}) > 0$: quand la classe (étiquette) vaut 1
- $S(X^{(i)}) < 0$: quand la classe (étiquette) vaut 0

$$\Pr obabilité(X \in Classe"1") = Sigmoide(\theta.X) = \frac{1}{1+\exp(-S(X))} = \frac{1}{1+\exp(-\theta.X)}$$

Le résultat obtenu par la fonction **sigmoid (θ.X)** est interprété comme la **probabilité que l'observation X soit d'un label (étiquette) 1.**

64

la régression Logistique implémenté dans sklearn

```
import numpy as np  # Chargement de numpy
import matplotlib.pyplot as plt # import de Matplotlib
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
iris = datasets.load_iris()
X = iris.data[:, :2] # avoir un problème de classification binaire.
# re-etiquettage des fleurs pour se retrouver avec deux classes au lieu de trois
y = []
for e in iris.target:
    if e==0:        y+=[0]
    else:       y+=[1]
y=np.array(y)
```

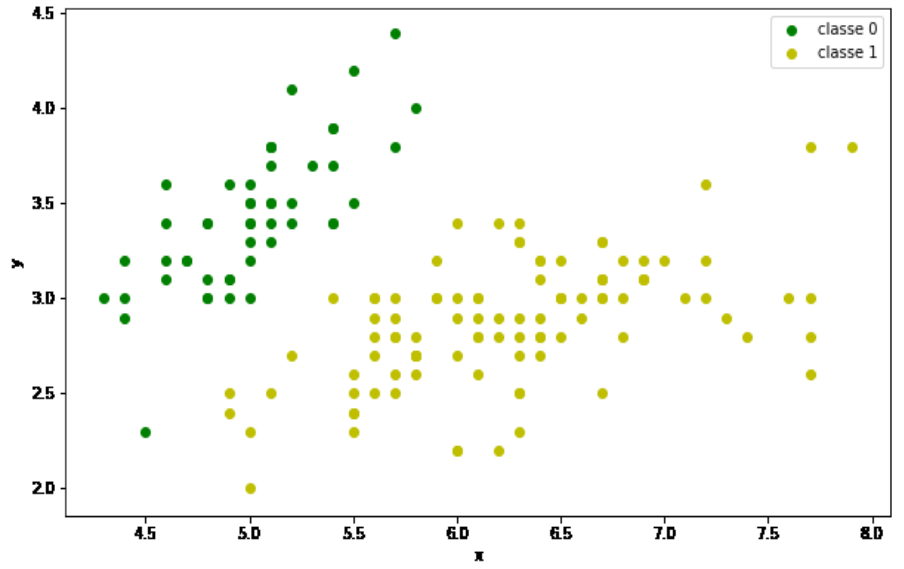
65

la régression Logistique implémenté dans sklearn

```
plt.figure(figsize=(10,6)) # Taille de la figure
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='g')
# En Vert les fleurs ayant l'étiquette 0
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='y')
# en Jaune les fleurs ayant l'étiquette 1
plt.legend(labels = ("classe 0", "classe 1"));
plt.xlabel("x")
plt.ylabel("y")
```

66

la régression Logistique implémenté dans sklearn



67

la régression Logistique implémenté dans sklearn

```
model = LogisticRegression(C=1e20)#tester avec 0.01
x_train, x_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=0)
#la fonction train_test_split mélange les données avant de faire la répartition
#si on fixe random_state sur une valeur précise, on exécute sur deux machines différentes
#les le mélange sera fait de la même façon sur les deux machines et on obtient les mêmes
model.fit(x_train, y_train)                                                    test_set
score=cross_val_score(model,x_train, y_train, cv=5,scoring='accuracy')
score1=model.score(x_test, y_test)
score2=model.score(x_train, y_train)
print("score sur test-set ="+str(score1*100)+'%')
print("score sur train-set ="+str(score2*100)+'%')
theta0=model.intercept_;  theta1=model.coef_[0][0]; theta2=model.coef_[0][1]
Iries_To_Predict = [    [5.5, 2.5],    [7, 3],    [3,2],    [5,3]]
# demande de prédiction
print(model.predict(Iries_To_Predict))
```

```
score sur test-set =100.0%
score sur train-set =100.0%
[1 1 0 0]
```

68

la régression Logistique implémenté dans sklearn

```
score=cross_val_score(model,x_train, y_train, cv=5,scoring='accuracy')
print(score)
print(score.mean())
def fct_reg_logistic(x1,x2):
    z=(theta0)+theta1*x1+(theta2*x2)
    return(z)
print([fct_reg_logistic(e[0],e[1]) for e in Iries_To_Predict])
```

```
[1.         0.95833333 1.         1.         1.        ]
0.9916666666666668
[array([24.57979467]), array([60.629162]), array([-45.97305733]), array([-8.37780735])]
```

69

la régression Logistique implémenté pas à pas

La probabilité de prédiction de la classe positive est déterminée par:

$$h(\theta, x) = \frac{1}{1 + e^{-\theta.x}}$$

La fonction de coût à minimiser :

$$j(\theta) = \frac{1}{m}\sum_{1}^{m}[(0 - y^i) * \log(h(x^i)) - (1 - y^i) * \log(1 - h(x^i))]$$

Le gradient est: $$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h(\theta, x^{(i)}) - y^{(i)}).x_j^{(i)}$$

70

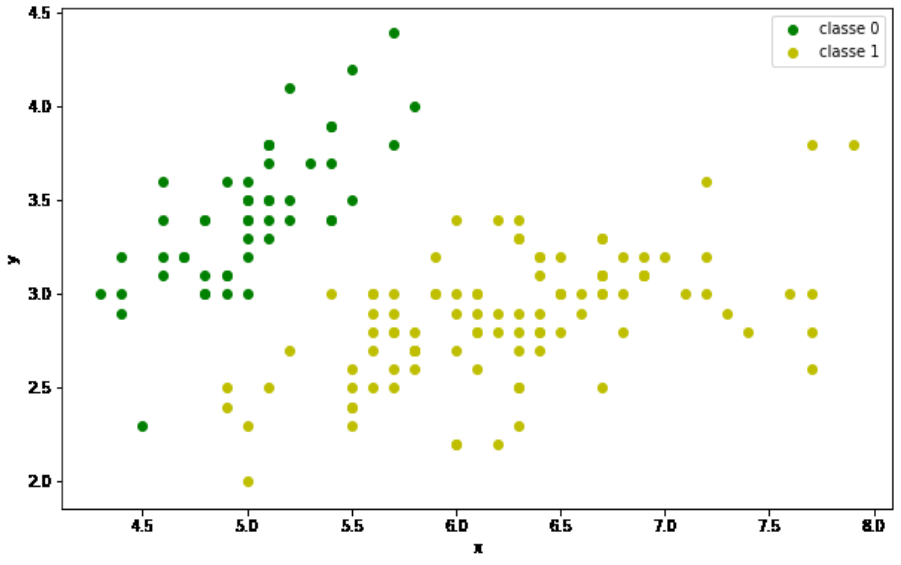
la régression Logistique implémenté pas à pas

```
import  numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
x= iris.data[:, :2] # avoir un problème de classification binaire.
y = []
for e in iris.target:
    if e==0:y+=[0]
    else:y+=[1]
y=np.array(y)
```

```
plt.figure(figsize=(10,6)) # Taille de la figure
plt.scatter(x[y == 0][:, 0], x[y == 0][:, 1], color='g', label='0')
# En Vert les fleurs ayant l'étiquette 0
plt.scatter(x[y == 1][:, 0], x[y == 1][:, 1], color='y', label='1')
# en Jaune les fleurs ayant l'étiquette 1
plt.legend(labels = ("classe 0", "classe 1"));
plt.xlabel("x")
plt.ylabel("y")
```

71

la régression Logistique implémenté pas à pas



72

la régression Logistique implémenté pas à pas

$$h(\theta,x)=\frac{1}{1+e^{-\theta.x}}$$

$$j(\theta)=\frac{1}{m}\sum_{1}^{m}[(0-y^{i})*\log(h(x^{i}))-(1-y^{i})*\log(1-h(x^{i}))]$$

```
X=np.concatenate((np.ones((y.shape[0],1)),x),axis=1)
theta = np.zeros(X.shape[1])
def h(theta,x):
    return 1/(1+np.exp(-np.dot(theta.T,x)))

def Erreurs_j_theta(x,y,theta):
    m=x.shape[0]
    s=0
    for i in range(m):
        s+=y[i]*np.log(h(theta,x[i]))+(1-y[i])*np.log(1-h(theta,x[i]))
    return(-s/m)
```

73

la régression Logistique implémenté pas à pas

Le gradient est: $$\frac{\partial J(\theta)}{\partial \theta_j}=\frac{1}{m}\sum_{i=1}^{m}(h(\theta,x^{(i)})-y^{(i)}).x_j^{(i)}$$

```
def gradient_D_j_theta(x,y,theta,j):
    m=x.shape[0]
    s=0
    for i in range(m):
        s+=(h(theta,x[i])-y[i])*x[i][j]
    return(s/m)
```

74

la régression Logistique implémenté pas à pas

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h(\theta, x^{(i)}) - y^{(i)}).x_j^{(i)}$$

```
def gradient_descent(X, y, theta, learning_rate, n_iterations):
    # création d'un tableau de stockage pour enregistrer l'évolution des erreurs
    historique_des_erreurs = np.zeros(n_iterations)
    for i in range(0, n_iterations):
        H=np.array([h(theta,x) for x in X])
        Hy=[abs(H[u]-y[u]) for u in range(X.shape[0])]
        theta[0] = theta[0] - (learning_rate/X.shape[0]) * sum(Hy)
        for j in range(X.shape[1]):
            theta[j] = theta[j] - learning_rate*gradient_D_j_theta(X, y, theta,j)
        historique_des_erreurs[i] = Erreurs_j_theta(X,y,theta)
    return theta, historique_des_erreurs
```

75

la régression Logistique implémenté pas à pas

```
# Example de test :
n_iterations = 10000
learning_rate = 0.01
theta_final, historique_des_erreurs=gradient_descent(X, y,theta, learning_rate, n_iterations)
print(theta_final) # thetaune fois que la machine a  été entrainée
def h_avec_ajout_biais_a_x(theta,x):
    XX=np.concatenate((np.array([1]),x),axis=0)
    return 1/(1+np.exp(-np.dot(theta.T,XX)))
```

```
[-12.42617591   4.4185545   -3.63355506]
```

76

la régression Logistique implémenté pas à pas

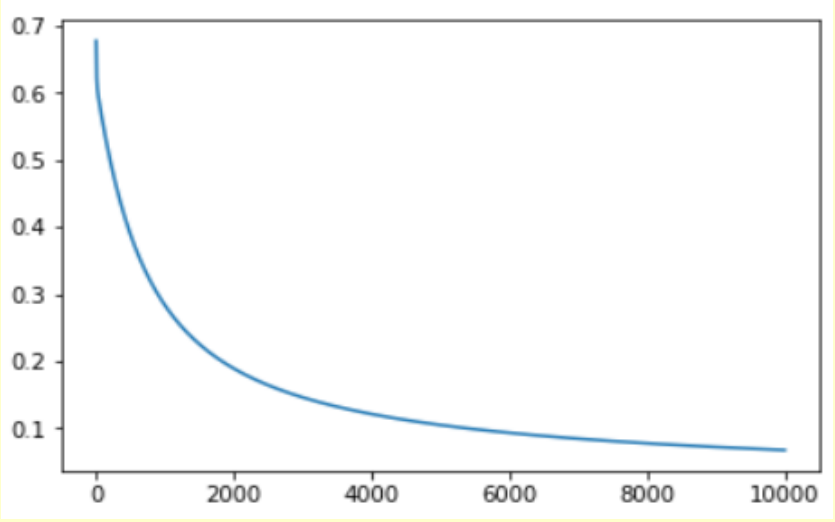
```
#calcul du taux de réussite sur l'ensemble d'apprentissage:
cpt=0
for i in range(y.shape[0]):
    t=h_avec_ajout_biais_a_x(theta_final,x[i])
    if t<0.5:
        target=0
    else:
        target=1
    if target==y[i]:
        cpt+=1
    else:
        print('Problème au point:',x[i])
print((cpt/y.shape[0])*100)
plt.plot(range(n_iterations),historique_des_erreurs)
plt.show()
```

```
100.0
```

77

la régression Logistique implémenté pas à pas

l'évolution des erreurs au cours des itérations



78

Représentation des données textuelles

Il existe principalement deux méthodes qui permettent de transformer et d'extraire de l'information grâce à la tokenization. On parle de représentation sparse des données qui sont réalisés grâce à deux méthodes : bags-of-words & TF-IDF.

bags-of-words(Sacs-de-mots):

```
1  from sklearn.feature_extraction.text import CountVectorizer
2  texte = ["La vie est douce","La vie est tranquille, est belle, est douce"]
3  vect = CountVectorizer()
4  T= vect.fit_transform(texte)
5  dictionnaire_des_mots=vect.vocabulary_
6  print("dictionnaire_des_mots :", dictionnaire_des_mots)
7  liste_des_mots=list(dictionnaire_des_mots.keys())
8  print("liste_des_mots :", liste_des_mots)
9  Matrice_sparse_correspondante=T.toarray()
10 print("Matrice_sparse_correspondante:\n",Matrice_sparse_correspondante)
```

```
dictionnaire_des_mots : {'la': 3, 'vie': 5, 'est': 2, 'douce': 1, 'tranquille':
4, 'belle': 0}
liste_des_mots : ['la', 'vie', 'est', 'douce', 'tranquille', 'belle']
Matrice_sparse_correspondante:
 [[0 1 1 1 0 1]
 [1 1 3 1 1 1]]
```

Représentation des données textuelles

Matrice sparse binaire: CountVectorizer(binary=True)

```
1  from sklearn.feature_extraction.text import CountVectorizer
2  texte = ["La vie est douce","La vie est tranquille et est belle"]
3  vect = CountVectorizer()
4  T= vect.fit_transform(texte)
5  dictionnaire_des_mots=vect.vocabulary_
6  print("dictionnaire_des_mots :", dictionnaire_des_mots)
7  liste_des_mots=list(dictionnaire_des_mots.keys())
8  print("liste_des_mots :", liste_des_mots)
9  Matrice_sparse_correspondante=T.toarray()
10 print("Matrice_sparse_correspondante:\n",Matrice_sparse_correspondante)
```

```
dictionnaire_des_mots : {'la': 4, 'vie': 6, 'est': 2, 'douce': 1, 'tranquille':
5, 'et': 3, 'belle': 0}
liste_des_mots : ['la', 'vie', 'est', 'douce', 'tranquille', 'et', 'belle']
Matrice_sparse_correspondante:
 [[0 1 1 0 1 0 1]
 [1 0 2 1 1 1 1]]
```

Représentation des données textuelles

TF-IDF (term frequency-inverse document frequency)

Les distances basées sur des tokens

- Une mesure qui est largement employée dans le domaine de la recherche d'informations, semble convenable ici. Il s'agit du TF/IDF .
- La fréquence de terme, TF, dans un document donné montre l'importance de ce terme dans le document en question.
- La fréquence inverse de document, IDF, est une mesure de l'importance générale du terme dans l'ensemble des documents.
- Les mesures de TF et TF/IDF sont définies comme suit :

Soit D un corpus des documents et t un terme à considérer:

le nombre d'occurrences du terme t dans un document

le nombre des termes dans ce document

$$TF = \frac{n(t)}{N} \text{ et } TF/_{IDF} = TF \times \log\left(\frac{|D|}{d(t)}\right)$$

dénote le nombre des documents dans le corpus D

le nombre des documents qui contiennent au moins une fois le terme t.

Représentation des données textuelles

```
import string
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
def netoyage(corpus_ensemble_documents):
    for i in range(len(corpus_ensemble_documents)):
        corpus_ensemble_documents[i]=corpus_ensemble_documents[i].lower()
    for i in range(len(corpus_ensemble_documents)):
        for c in string.punctuation:
            x=corpus_ensemble_documents[i].replace(c,' ')
            corpus_ensemble_documents[i]=x
    stopwords_anglais=stopwords.words('english')   #ou french
    for i in range(len(corpus_ensemble_documents)):
        L=corpus_ensemble_documents[i].split()
        for mot in L:
            if mot in stopwords_anglais:
                while mot in L:
                    L.remove(mot)
        corpus_ensemble_documents[i]=" ".join(L)
    return(corpus_ensemble_documents)
```

82

Représentation des données textuelles

```
from numpy import *
def TF(terme,corpus,numero_document):
    x=corpus[numero_document].count(terme)
    y=len(corpus[numero_document].split())
    return x/y

def IDF(terme,corpus,numero_document):
    D=len(corpus)
    d_t=0
    for document in corpus:
        if terme in document:
            d_t+=1
    TF_val=TF(terme,corpus,numero_document)
    return TF_val*log(1+(D/d_t))

def cles_correspondante_a_valeur(valeur,dictionnaire):
    for cle in dictionnaire.keys():
        if dictionnaire[cle]==valeur:
            return cle
```

83

Représentation des données textuelles

```
def matrice_sparse(dictionnaire,corpus_ensemble_documents):
    M=numpy.zeros((len(corpus_ensemble_documents),len(dictionnaire.values())))
    for i in range(len(corpus_ensemble_documents)):
        for j in dictionnaire.values():
            x=cles_correspondante_a_valeur(j,dictionnaire)
            M[i,j]=IDF(x,corpus_ensemble_documents,i)
    return M
```

```
def affiche(M):
    (n,p)=M.shape
    for i in range(n):
        for j in range(p):
            M[i,j]=round(M[i,j],2)
    print(M)
```

84

Représentation des données textuelles

```
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
texte = ["La vie est douce","La vie est tranquille, est belle, est douce",
         "le corona-virus est méchant"]
texte=netoyage(texte)
vect = TfidfVectorizer()
T= vect.fit_transform(texte)
dictionnaire_des_mots=vect.vocabulary_
print("dictionnaire_des_mots :", dictionnaire_des_mots)
liste_des_mots=list(dictionnaire_des_mots.keys())
print("liste_des_mots :", liste_des_mots)
Matrice_sparse_correspondante=T.toarray()
print("Matrice_sparse_methode predefinie:\n")
affiche(Matrice_sparse_correspondante)
#on donne un poid important aux tokens qui apparaissent souvent dans un
#document en particulier mais pas dans tous les documents du corpus
print("Matrice_sparse obtenue par notre methode:\n")
affiche(matrice_sparse(dictionnaire_des_mots,texte))
```

```
dictionnaire_des_mots : {'vie': 5, 'douce': 2, 'tranquille': 4, 'belle': 0, 'corona': 1, 'virus': 6,
'méchant': 3}
liste_des_mots : ['vie', 'douce', 'tranquille', 'belle', 'corona', 'virus', 'méchant']
Matrice_sparse_methode predefinie:

[[0.   0.   0.71 0.   0.   0.71 0.  ]
 [0.56 0.   0.43 0.   0.56 0.43 0.  ]
 [0.   0.58 0.   0.58 0.   0.   0.58]]
Matrice_sparse obtenue par notre methode:

[[0.   0.   0.46 0.   0.   0.46 0.  ]
 [0.35 0.   0.23 0.   0.35 0.23 0.  ]
 [0.   0.46 0.   0.46 0.   0.   0.46]]
```

85

Application de la régression Logistique sur les Spams

```
import numpy as np
import pandas
spams = pandas.read_table("D:\\SMSSpamCollection.txt",sep="\t",header=0)
spamsTrain, spamsTest = train_test_split(spams,train_size=0.7,random_state=1)
from sklearn.feature_extraction.text import CountVectorizer
parseur = CountVectorizer()
XTrain = parseur.fit_transform(spamsTrain['message'])
mdtTrain = XTrain.toarray()
from sklearn.linear_model import LogisticRegression
modelFirst= LogisticRegression()
#essayer cette version, pourquoi ça ne marche pas ?
#modelFirst.fit(spamsTrain['message'],spamsTrain['classe'])
modelFirst.fit(mdtTrain,spamsTrain['classe'])
score1=modelFirst.score(mdtTrain, spamsTrain['classe'])
print("score sur data train:"+str(score1))
mdtTest = parseur.transform(spamsTest['message'])
#predTest = modelFirst.predict(mdtTest)
score2=modelFirst.score(mdtTest, spamsTest['classe'])
print("score sur data test:"+str(score2))
```

```
score sur data train:0.9976923076923077
score sur data test:0.9838516746411483
```

Sans binary=True

86

Application de la régression Logistique sur les Spams

```
import numpy as np
import pandas
spams = pandas.read_table("D:\\SMSSpamCollection.txt",sep="\t",header=0)
spamsTrain, spamsTest = train_test_split(spams,train_size=0.7,random_state=1)
from sklearn.feature_extraction.text import CountVectorizer
parseur = CountVectorizer(binary=True)
XTrain = parseur.fit_transform(spamsTrain['message'])
mdtTrain = XTrain.toarray()
from sklearn.linear_model import LogisticRegression
modelFirst= LogisticRegression()
#essayer cette version, pourquoi ça ne marche pas ?
#modelFirst.fit(spamsTrain['message'],spamsTrain['classe'])
modelFirst.fit(mdtTrain,spamsTrain['classe'])
score1=modelFirst.score(mdtTrain, spamsTrain['classe'])
print("score sur data train:"+str(score1))
mdtTest = parseur.transform(spamsTest['message'])
#predTest = modelFirst.predict(mdtTest)
score2=modelFirst.score(mdtTest, spamsTest['classe'])
print("score sur data test:"+str(score2))
```

```
score sur data train:0.9976923076923077
score sur data test:0.9838516746411483
```

Avec binary=True

Application de la régression Logistique sur les Spams

```
from sklearn import metrics
parseurBis = CountVectorizer(stop_words='english')
XTrainBis = parseurBis.fit_transform(spamsTrain['message'])
mdtTrainBis = XTrainBis.toarray()
modelBis = LogisticRegression()
modelBis.fit(mdtTrainBis,spamsTrain['classe'])
mdtTestBis = parseurBis.transform(spamsTest['message'])
score3=modelBis.score(mdtTrainBis, spamsTrain['classe'])
print("score sur data train:"+str(score3))
score4=modelBis.score(mdtTestBis, spamsTest['classe'])
print("score sur data test:"+str(score4))
```

```
score sur data train:0.9956410256410256
score sur data test:0.9760765550239234
```

Sans stop_words

Application de la régression Logistique sur les Spams

```
1  from sklearn.feature_extraction.text import TfidfVectorizer
2  from sklearn import metrics
3  parseur3 = TfidfVectorizer()
4  XTrain3 = parseur3.fit_transform(spamsTrain['message'])
5  mdtTrain3 = XTrain3.toarray()
6  model3 = LogisticRegression()
7  model3.fit(mdtTrain3,spamsTrain['classe'])
8  mdtTest3 = parseur3.transform(spamsTest['message'])
9  score5=model3.score(mdtTrain3, spamsTrain['classe'])
10 print("score sur data train:"+str(score5))
11 score6=model3.score(mdtTest3, spamsTest['classe'])
12 print("score sur data test:"+str(score6))
```

```
score sur data train:0.9743589743589743
score sur data test:0.9712918660287081
```

Avec TfidfVectorizer