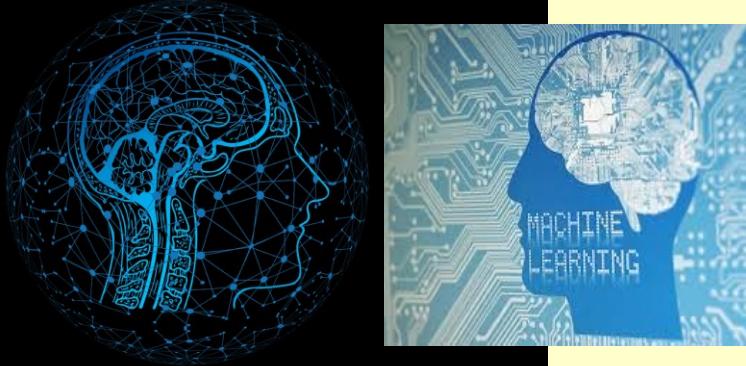


Machine Learning & Text Mining

Année universitaire : 2020-2021



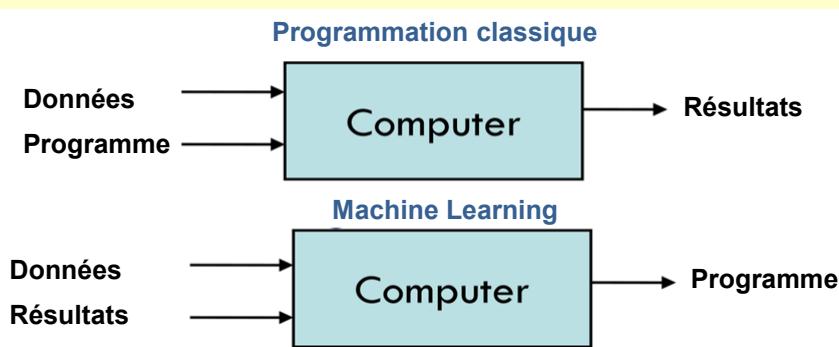
Pr. Abdelali ZAKRANI

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

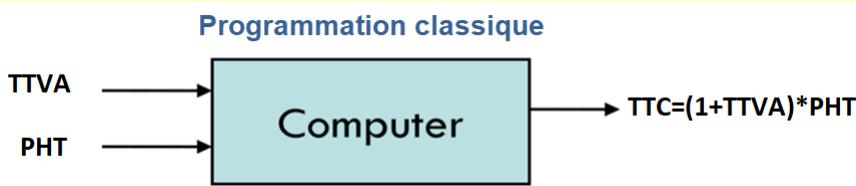
Introduction à l'apprentissage automatique

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:

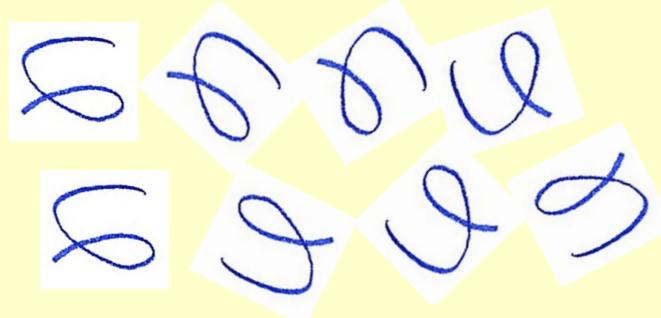


3

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 reconnaître 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.

5

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=IwAR0sdwCFrxJdMg2d75jAlgZ1EF_JH21ouRfr98v2hx2cEHpzYch0R9QSTvM

Le 21 avril 2020, Des chercheurs du MIT (Massachusetts 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.

6

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.

9

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 connaît 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

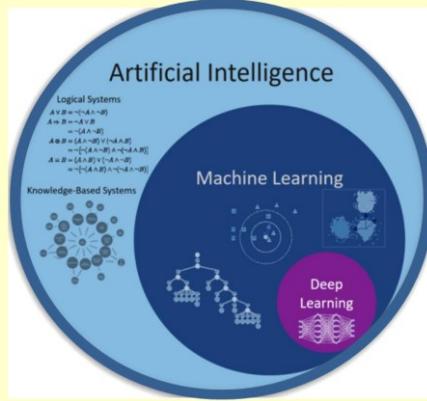
<https://archive.ics.uci.edu/ml/index.php>

<https://www.kaggle.com/datasets>

10

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

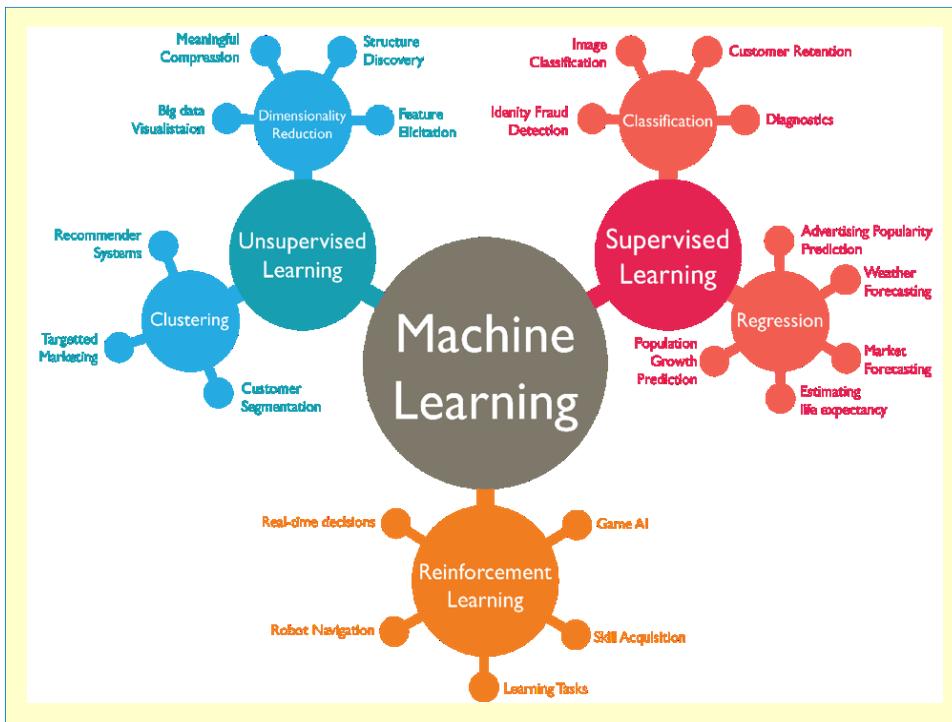
Le **Machine Learning** ou **Apprentissage automatique** est une branche de l'intelligence artificielle qui concerne la conception, l'analyse, le développement et l'implémentation de modèles, permettant à une machine d'apprendre à partir des données par un processus systématique afin de remplir une tâche.



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)



Canard



Canard



Pas canard



Pas canard



Modèle
prédictif

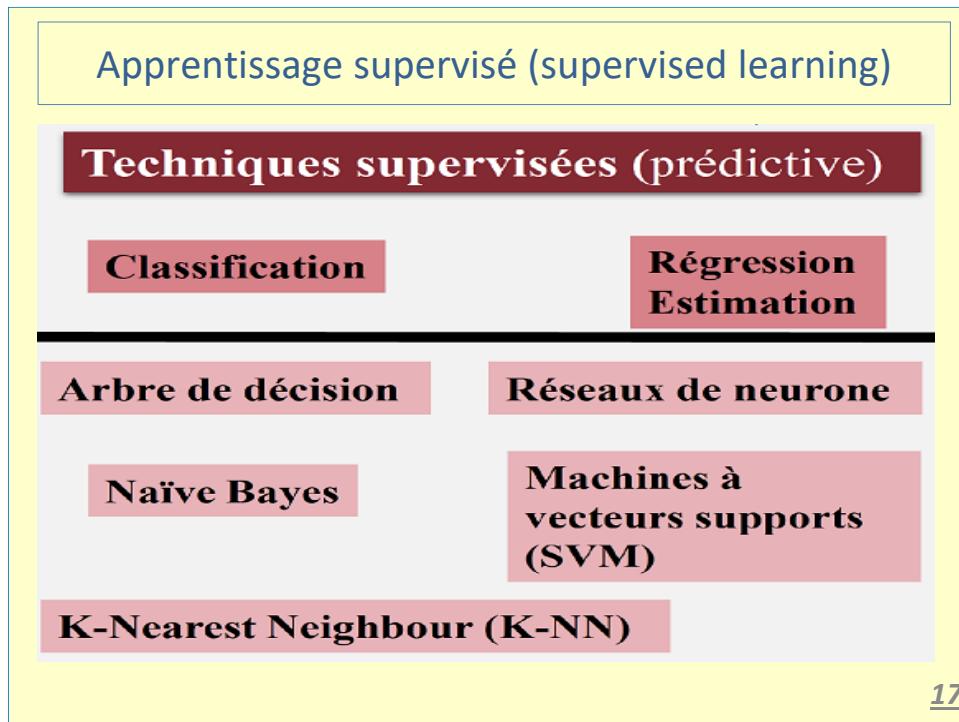


Modèle
prédictif

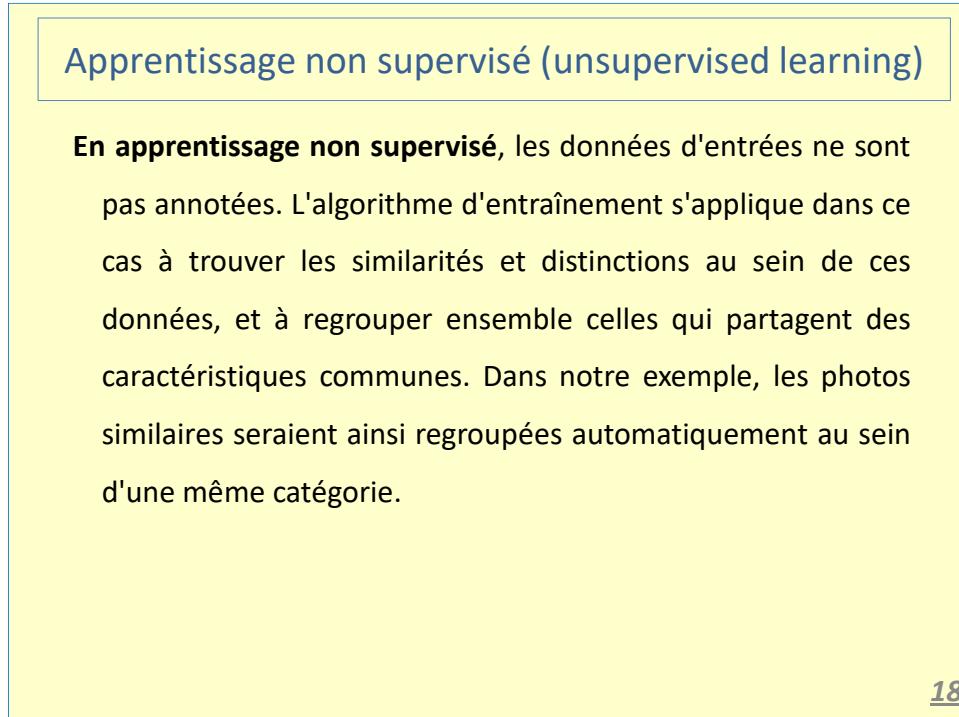


Canard

16

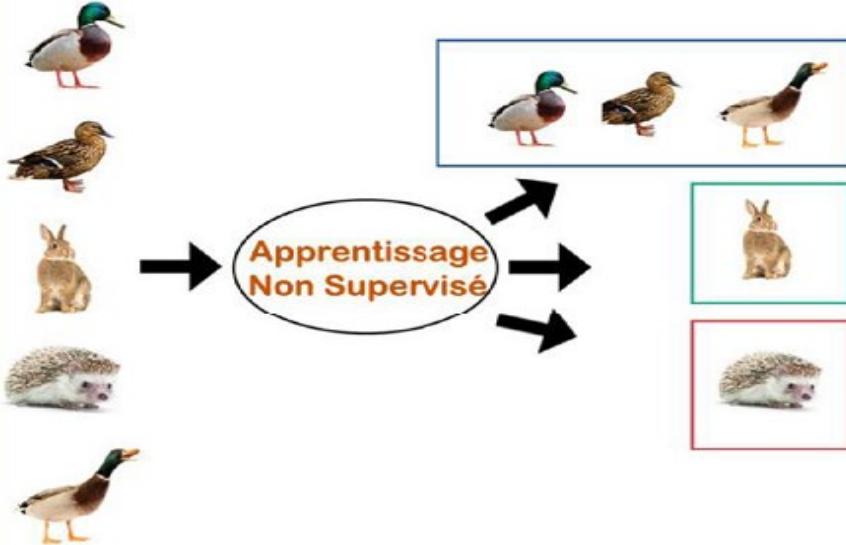


17



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.

21

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.

22

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

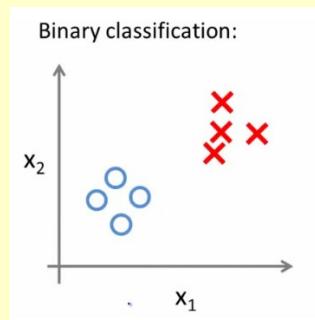
Les 7 étapes de l'apprentissage automatique

- 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 $Y \in \{0,1\}$



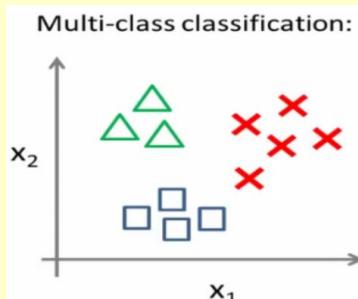
25

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).

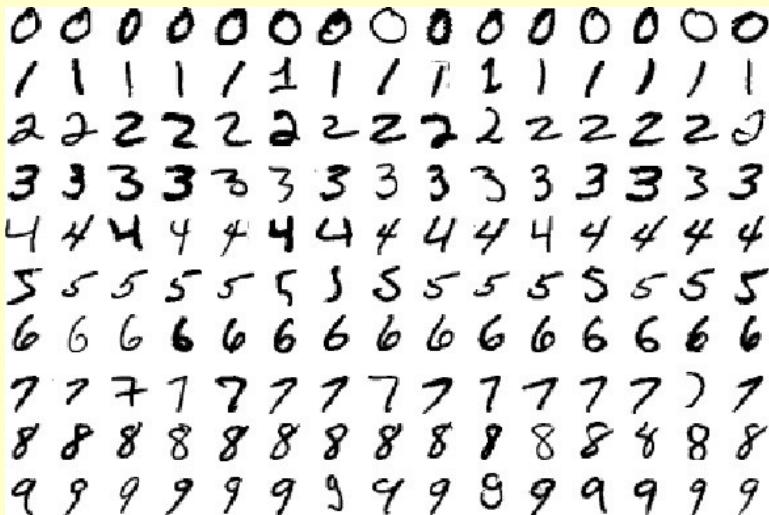
Note : Pour l'exemple de la tumeur, on peut attribuer arbitrairement la classe (étiquette) 1 pour dire qu'il s'agit d'une tumeur maligne et la valeur 0 pour les tumeurs bénignes.

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



0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9

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

```

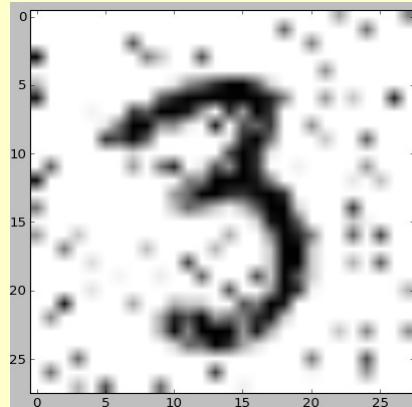
p=clf.predict(xtest)
count=0
for i in range(0,21000):
    count+=1 if p[i]==actual_label[i] else 0
print("Accuracy=", (count/21000)*100)
d=xtest[5]
Nombre_de_pixels_errones=100
for i in range(Nombre_de_pixels_errones):
    position=np.random.randint(0,784,1)[0]
    bruit=np.random.randint(-200,200,1)[0]
    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:

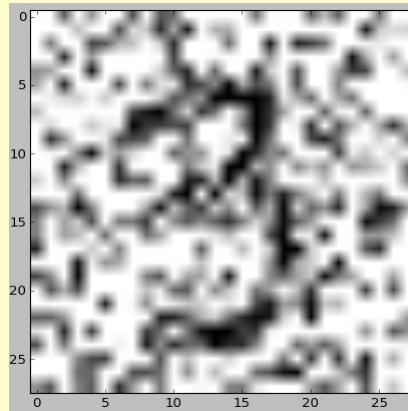


L'instruction `print(clf.predict([d]))` a donné la valeur 3 donc une reconnaissance réussie

[31](#)

Application de Machine Learning sur la reconnaissance des nombres manuscrits

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

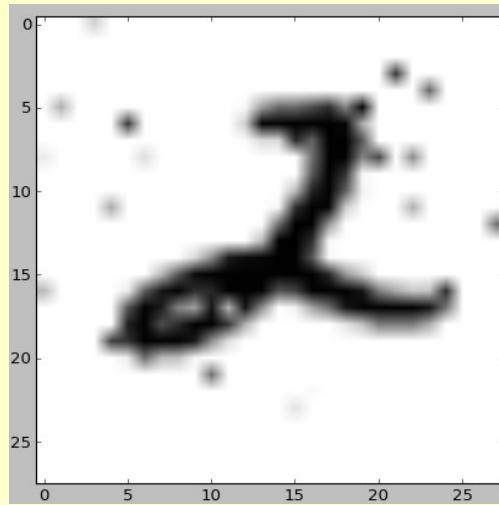


L'instruction `print(clf.predict([d]))` a donné la valeur 5 donc une reconnaissance non réussie à cause du bruit très fort.

[32](#)

Application de Machine Learning sur la reconnaissance des nombres manuscrits

Le programme donné en haut a pu reconnaître 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

35

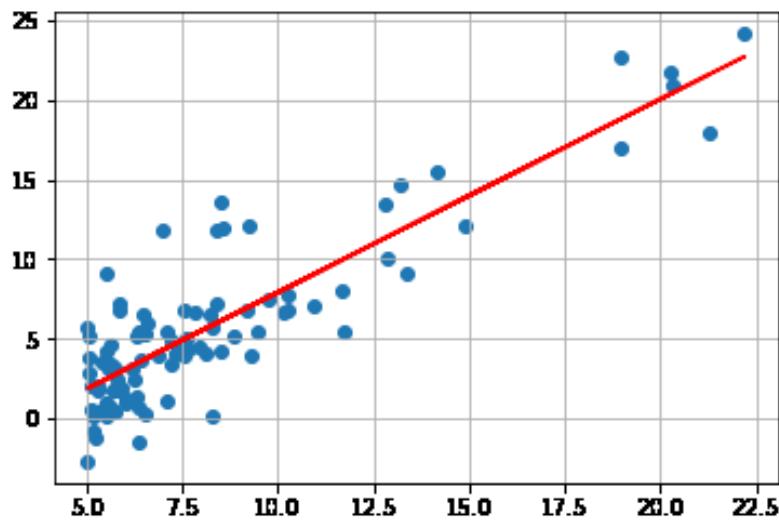
Régression linéaire

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.

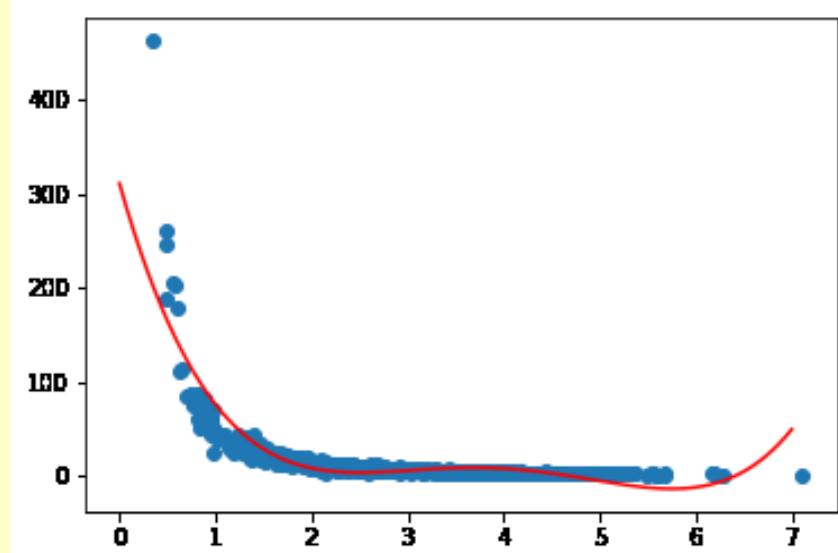
36

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()

```

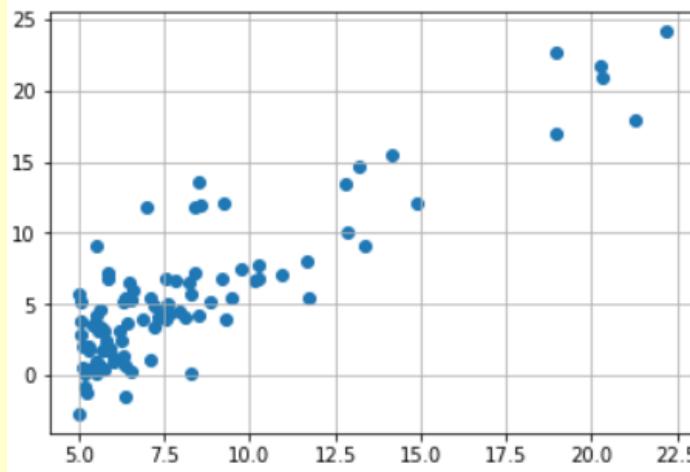
39

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 :

$$H(x) = \text{intercept} + \text{slope} * x$$

- 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
```

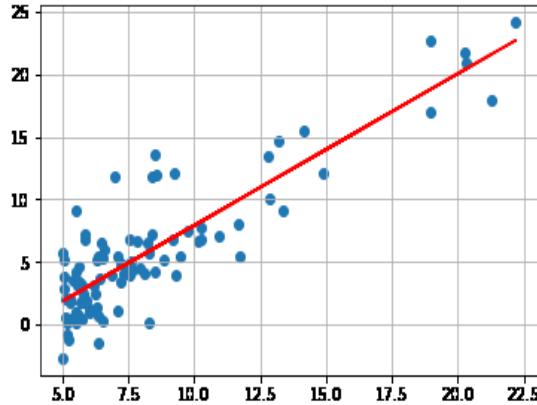
```
SL.rvalue:
```

```
0.8721572919685905
```

```
r= X.corr(Y)
```

```
print(r)
```

```
0.8721572919685905
```



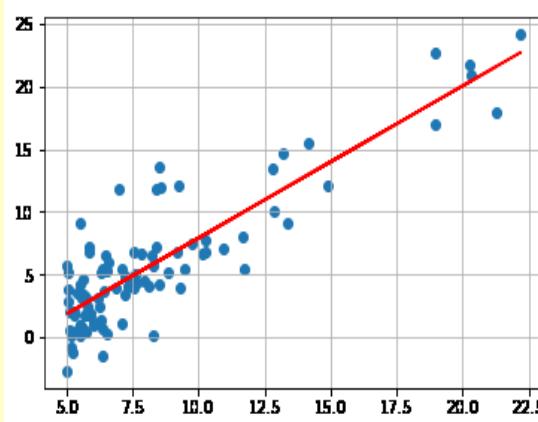
Régression linéaire

- 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 $H(x=22.5)$:

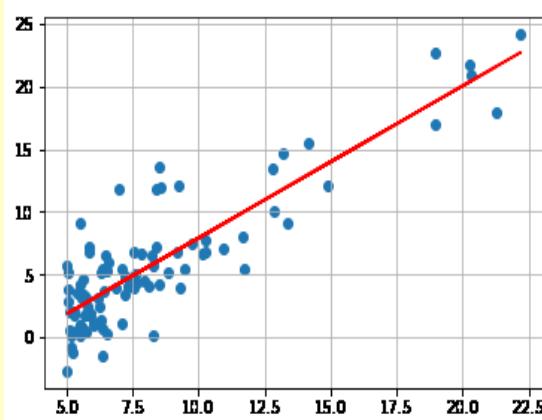
```
predict(22.5) → 23.093309207513975
```



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

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

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 : $f(x) = X \cdot \theta$
- la fonction Cout : $J(\theta) = \frac{1}{2m} \sum (X \cdot \theta - Y)^2$
- le gradient : $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} X^T \cdot (X \cdot \theta - Y)$
- 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 $J(\theta)$

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 erreurs
    historique_des_erreurs = np.zeros(n_iterations)
    for i in range(0, n_iterations):
        theta = theta - learning_rate * gradient(X, y, theta) # mise à 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 à X
X = np.hstack((x, np.ones(x.shape)))
# création d'un vecteur paramètre 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 à jour du theta
        historique_des_erreurs[i] = erreur_somme_des_distances(X, y, theta)
    return theta, historique_des_erreurs

```

48

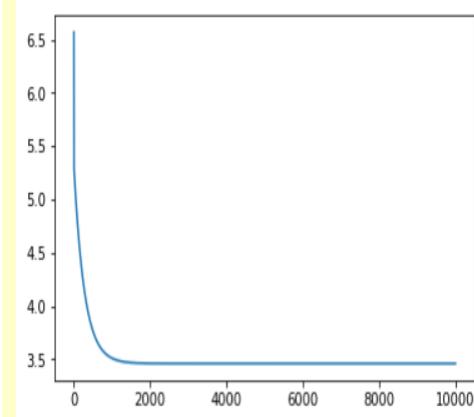
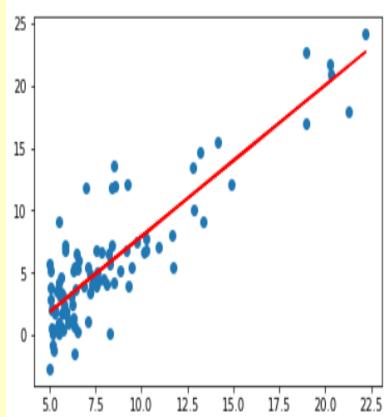
Régression linéaire avec l'algorithme de descente de gradient

```
# 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 entraînée
# création d'un vecteur prédictions qui contient les prédictions de notre modèle final
predictions = f(X, theta_final)
# Affiche les résultats de prédictions (en rouge) par rapport à 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
-----	-----

$x_0 = 5$

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

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

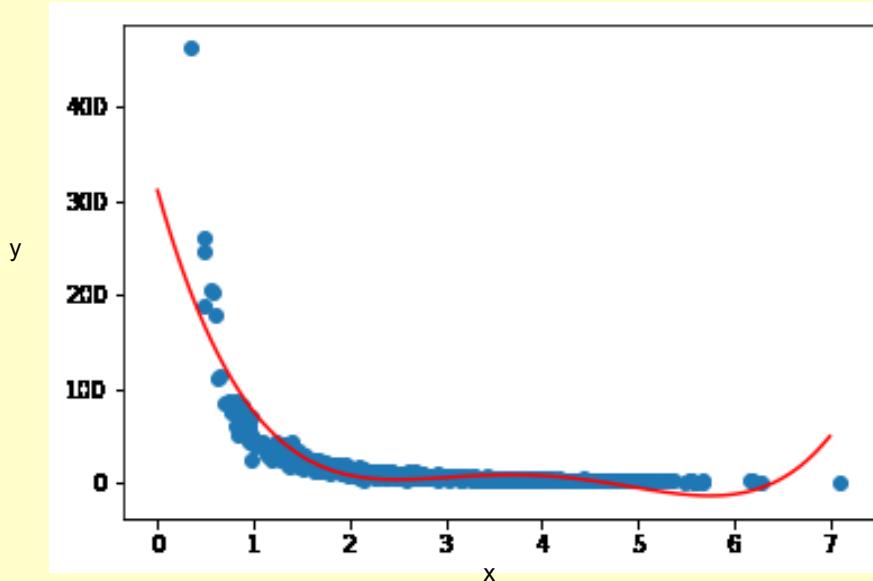
Minimum au point x: f(x)=0
X=0.5

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

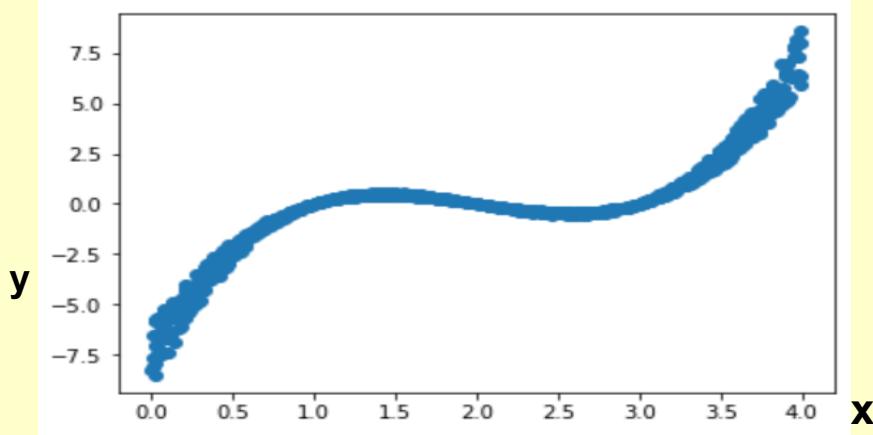
$x_{t+1} = x_t - \eta \times \nabla f(x_t)$

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 $P: P(X) \approx y$

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

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

```

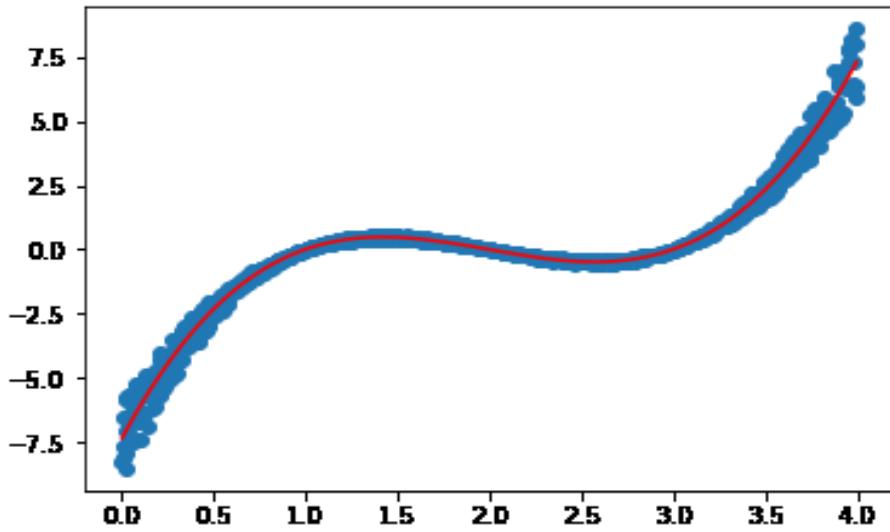
import numpy as np
import matplotlib.pyplot as plt
import pandas
data=pandas.read_csv("d:\\dataset_polynomial3.csv").as_matrix()
x=data[1:1001,0]
y=data[1:1001,1]
degre_polynome=5
p= np.poly1d(np.polyfit(x, y,degre_polynome))
LPol=list(p)
afficher_pol(LPol)
les_x= np.linspace(min(x), max(x), 100)
plt.scatter(x,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

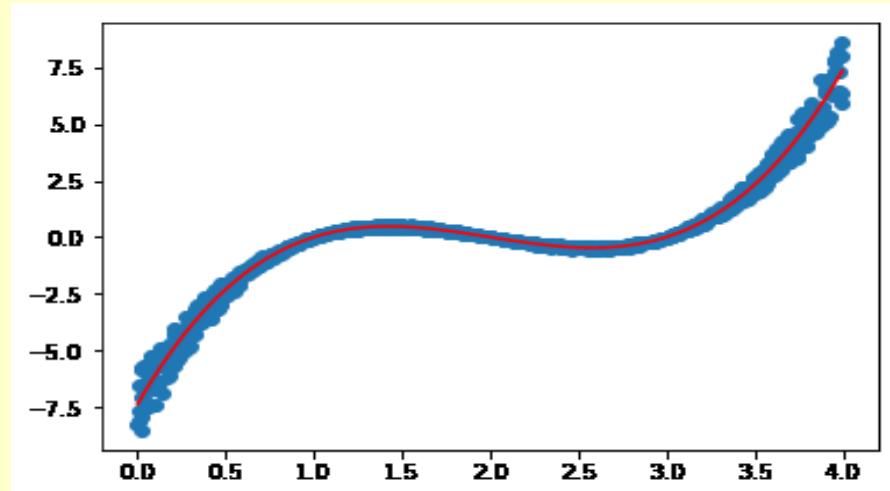
Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible

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



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

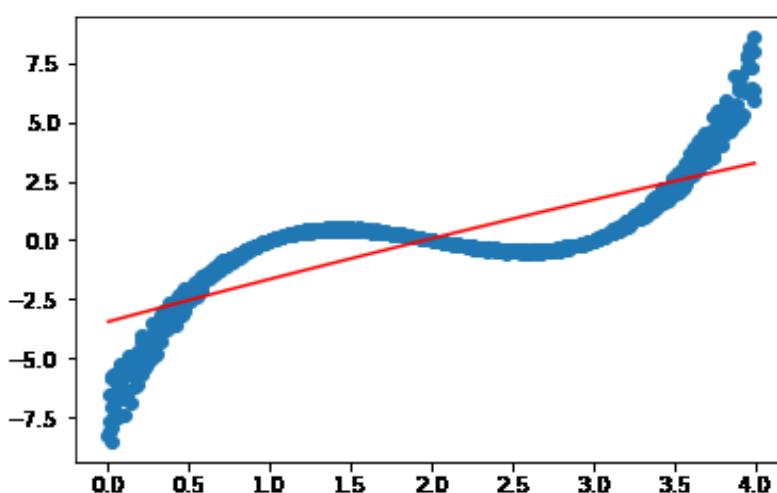
Polynôme de degré 3:
 $P(x) = (-7.38 \cdot X^0) + (13.53 \cdot X^1) + (-7.38 \cdot X^2) + (1.23 \cdot X^3)$



58

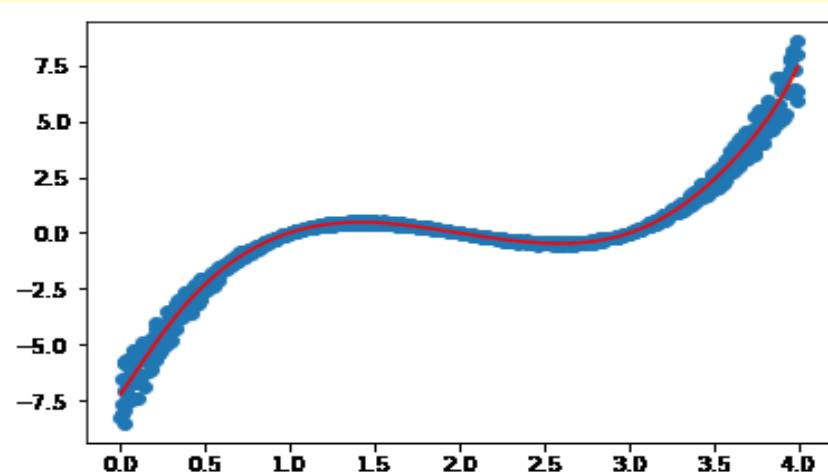
Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible

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

59

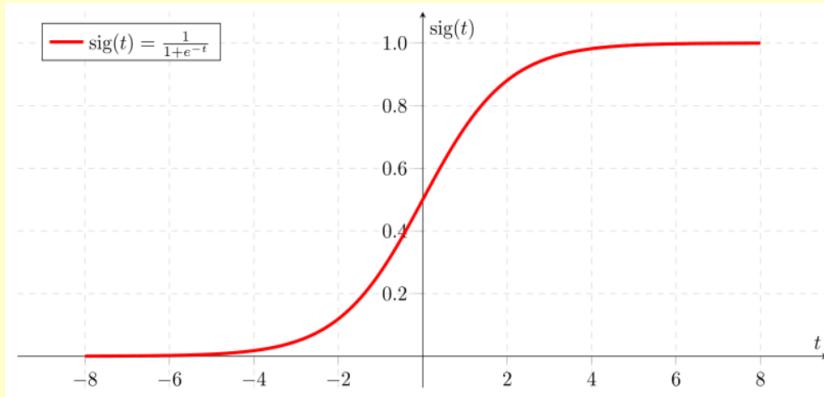
Régression polynomiale:
Relation non linéaire entre des variables prédictives et la variable cible

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

60

Sigmoid Function : La fonction pour la régression Logistique

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



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

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 maline ?

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

62

Sigmoid Function : La fonction pour la régression Logistique

- La fonction hypothèse:

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

- $X^{(i)}$: une observation (du *Training Set* ou du *Test Set*), cette variable est un vecteur contenant:

$$[x_1, x_2, x_3, \dots, x_n]$$

- 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: $[\theta_0, \theta_1, \theta_2, \theta_3, \dots, \theta_n]$ de sorte que :

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

$$\text{Probabilité}(X \in \text{Classe "1"}) = \text{Sigmoide}(\theta \cdot X)$$

$$= \frac{1}{1 + \exp(-S(X))} = \frac{1}{1 + \exp(-\theta \cdot X)}$$

Le résultat obtenu par la fonction **sigmoid** ($\theta \cdot X$) 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-etiquetage 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

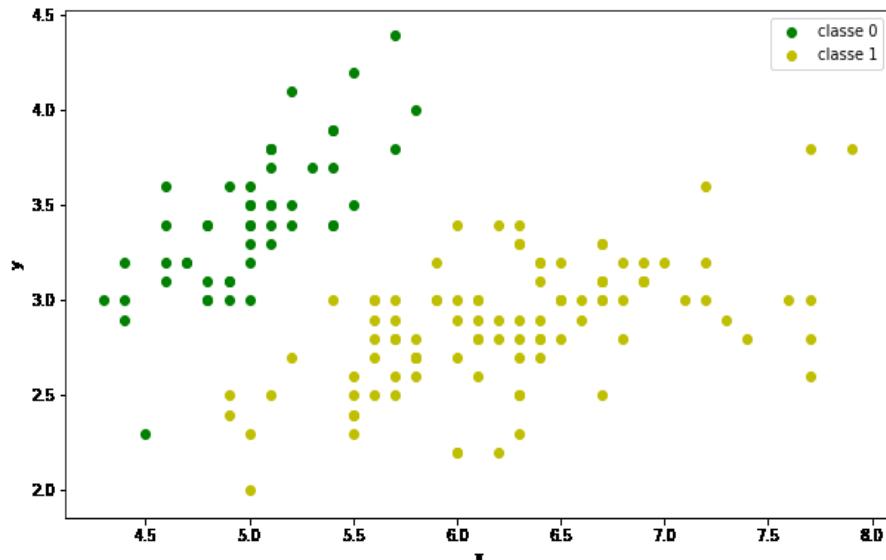
```

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



57

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 :

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

Le gradient est:

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

70

la régression Logistique implémenté pas à pas

```

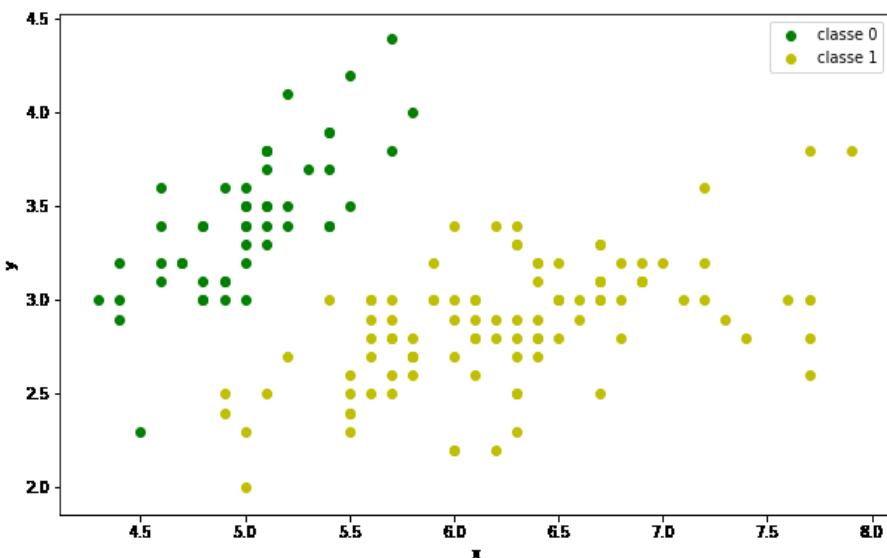
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
x= iris.data[:, :2] # avoir un problème de classification binaire.
y = []
for e in iris.target:
    if e==0:y+=[0]
    else:y+=[1]
y=np.array(y)

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

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

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

```
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:

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

```
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

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

```
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

```
# Exemple 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 entraînée
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

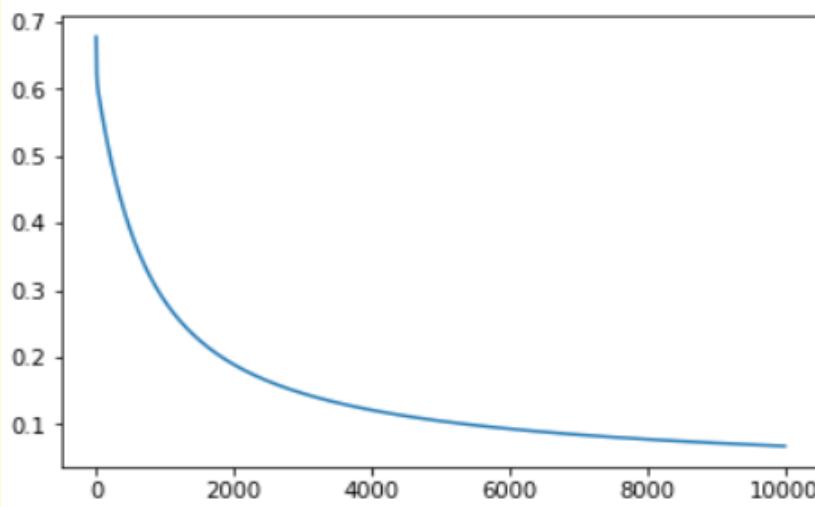
```
#calcul du taux de réussite sur l'ensemble d'apprentissage:
cpt=0
for i in range(y.shape[0]):
    t=h_avec_ajout_biais_a_x(theta_final,x[i])
    if t<0.5:
        target=0
    else:
        target=1
    if target==y[i]:
        cpt+=1
    else:
        print('Problème au point:',x[i])
print((cpt/y.shape[0])*100)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

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

le nombre d'occurrences du terme t dans un document

le nombre des termes dans ce document

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 nettoyage(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)
```

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=nettoyage(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 poids 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      Avec binary=True
score sur data test:0.9838516746411483

```

87

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      Sans stop_words
score sur data test:0.9760765550239234

```

88

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

89

Application de la régression Logistique sur la reconnaissance des chiffres manuscrits

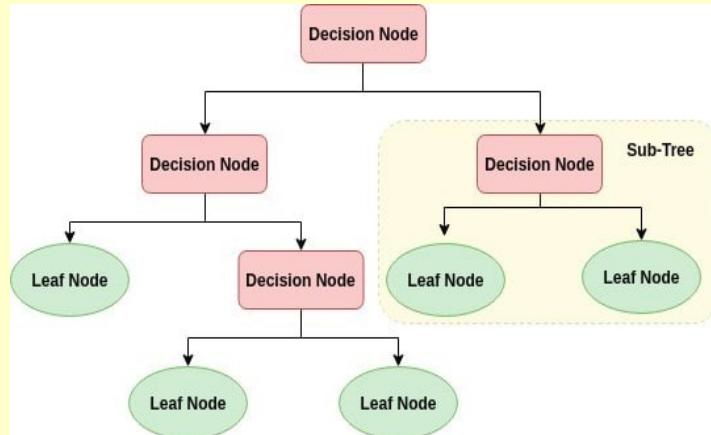
Refaire le TP 1 et remplacer le modèle par LogisticRegression

from sklearn.linear_model import LogisticRegression

90

Les arbres de décision

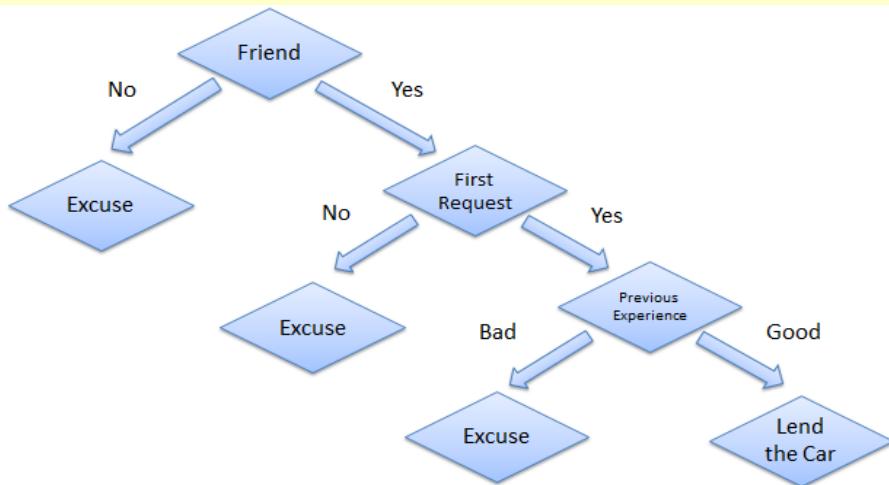
L'arbre de décision est un algorithme qui se base sur **un modèle d'arbre** pour définir la décision finale. Chaque noeud comporte une condition (une question), et les branchements sont en fonction de cette condition. Plus on descend dans l'arbre, plus on cumule les conditions.



91

Les arbres de décision

Considérez un scénario dans lequel une personne vous demande de lui prêter votre voiture pour une journée, et vous devez décider de lui prêter ou non ta voiture.



92

Les arbres de décision

L'utilisation des arbres de décision pour l'analyse prédictive présente plusieurs avantages:

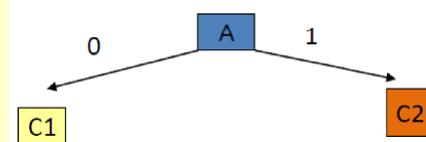
- Les arbres de décision peuvent être utilisés pour prédire des valeurs continues et discrètes, c'est-à-dire qu'ils fonctionnent bien pour les tâches de régression et de classification.
- Ils nécessitent relativement moins d'efforts pour former l'algorithme.
- Ils peuvent être utilisés pour classer des données séparables de manière non linéaire.
- Ils sont rapides et efficaces par rapport à KNN et à d'autres algorithmes de classification (revoir TP1-TP2).

93

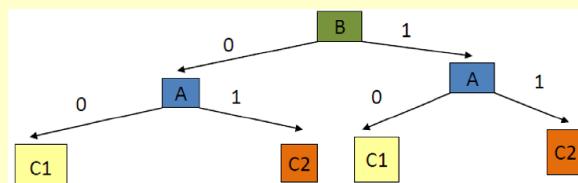
Choix de l'attribut de partitionnement

Partitionnement selon l'attribut A

A	B	Classe
0	1	C1
0	0	C1
1	1	C2
1	0	C2



Partitionnement selon l'attribut B



94

Choix de l'attribut de partitionnement

- Un arbre de décision représente **la suite** (\neq de ensemble) de questions à poser pour classer un nouveau exemple.
- Dans l'exemple précédent c'est l'attribut A qui apporte plus d'information à la classification, il est mieux que l'attribut B.
- **L'objectif est de minimiser le nombre de questions à poser.**
- **Question:** comment quantifier l'information apportée par chaque attribut ?

95

Les algorithmes de construction d'un arbre de décision:

- ID3
- CART
- C4.5
- OC1
- SLIQ
- SPRINT

96

Les algorithmes de construction d'un arbre de décision:

L'entropy

- Une formule pour calculer l'homogénéité d'un échantillon.
- Un échantillon complètement homogène a une entropie de 0.
- La formule pour l'entropie est:

$$\text{Entropie}(S) = -1 * \sum_{c \in \text{Classes}} \text{portion}(c) * \log_2(\text{portion}(c))$$

Avec S est l'ensemble des instances (les lignes du data_train...)

Choix de l'attribut de partitionnement

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

S: l'ensemble des instances contient les lignes L1,L2,L3 et L4.

Classes={C1,C2}

$$\text{Entropie}(S) = -1 * \sum_{c \in \text{Classes}(S)} \text{portion}(c) * \log_2(\text{portion}(c))$$

$$\text{Entropie}(S) = -1 * (0.5 * \log_2(0.5) + 0.5 * \log_2(0.5))$$

```
from math import *
>>> -1*(0.5*log2(0.5)+0.5*log2(0.5))
>>>1.0
```

Choix de l'attribut de partitionnement

```

1 from numpy import *
2 import pandas
3 df = pandas.DataFrame([[0,1,'C1'],[0,0,'C1'],
4 [1,1,'C2'],[1,0,'C2']],columns=['A', 'B', 'Classes'])
5 df=df.to_numpy()

```

```

1 def Entropie(S):
2     classes_distinctes=set(S[:, -1])
3     classes=list(S[:, -1])
4     s=0
5     for c in classes_distinctes:
6         p=classes.count(c)/len(classes)
7         s+=p*log2(p)
8     return(-1.0*s)
9 print(Entropie(df))

```

1.0

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

T15

99

Les algorithmes de construction d'un arbre de décision: calcul du Gain d'information

- Il faut sélectionner l'attribut **avec le plus grand gain d'information**.
- Si on fait un partitionnement de l'ensemble d'instances **S** selon l'attribut **Attribut** nous obtenons des sous ensembles d'instances **S_i** avec i compris entre 1 et le nombre k de valeurs distinctes de l'attribut **Attribut**:

$$S = \bigcup_{i=1}^k$$

- Il permet d'avoir l'attribut qui crée les branches les plus homogènes.
- Le gain d'information est basé sur la diminution de l'entropie après la division d'un ensemble de données par rapport à un attribut.
- Le Gain d'information est donné par la formule:**

$$Gain(S, \text{Attribut}) = Entropie(S) - \sum_{i=1}^k p_i * Entropie(S_i)$$

Avec pi est la portion du sous ensemble d'instances **S_i** par rapport à l'ensemble d'instances **S**:

$$pi = \frac{\text{Nombre de lignes de } S_i}{\text{Nombre de lignes de } S}$$

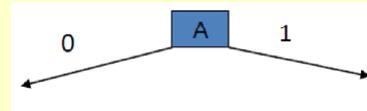
100

Choix de l'attribut de partitionnement

S: l'ensemble des instances
contient les lignes L1,L2,L3 et L4.
Classes={C1,C2}

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

Partitionnement selon l'attribut A



S1 contient
{L1==C1
L2==C1}

S2 contient
{L3==C2
L4==C2}

$$E(S1) = -1 * (\text{portion}(C1) * \log_2(\text{portion}(C1)) + \text{portion}(C2) * \log_2(\text{portion}(C2))) \\ = -1 * (1 * \log_2(1)) = 0$$

$$\text{Gain}(S, A) = 1 - (\text{portion}(S1) * E(S1) + \text{portion}(S2) * E(S2)) \\ = 1 - ((2/4) * E(S1) + (2/4) * E(S2)) \\ = 1 - ((2/4) * (0) + (2/4) * (0))$$

Gain(S,A)= 1

101

Choix de l'attribut de partitionnement

```

1 def Gain_d_information(S,numero_colon_attribut):
2     classes=list(set(S[:, -1]))
3     valeurs_attribut=list(set(S[:, numero_colon_attribut]))
4     Si=[[ ] for i in range(len(valeurs_attribut))]
5     for e in S:
6         e=list(e)
7         val_attribut_pour_e=e[numero_colon_attribut]
8         numero_sous_ensemble=valeurs_attribut.index(val_attribut_pour_e)
9         Si[numero_sous_ensemble].append(e)
10    Si=array(Si) ; som=0
11    for sous_ensemble in Si:
12        som+=(len(sous_ensemble)/len(S))*Entropie(sous_ensemble)
13    return(Entropie(S)-som)

1 print(Gain_d_information(df,0))
1.0

```

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

T15

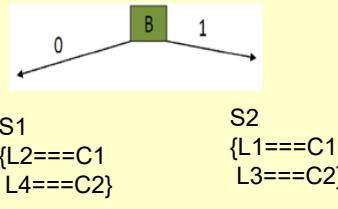
102

Choix de l'attribut de partitionnement

S: l'ensemble des instances
contient les lignes L1,L2,L3 et L4.
Classes={C1,C2}

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

Partitionnement selon l'attribut B



$$\begin{aligned} E(S1) &= -1 * (\text{portion}(C1) * \log_2(\text{portion}(C1)) + \text{portion}(C2) * \log_2(\text{portion}(C2))) \\ &= -1 * (0.5 * \log_2(0.5) + 0.5 * \log_2(0.5)) = 1.0 \end{aligned}$$

$$\begin{aligned} E(S2) &= -1 * (\text{portion}(C1) * \log_2(\text{portion}(C1)) + \text{portion}(C2) * \log_2(\text{portion}(C2))) \\ &= -1 * (0.5 * \log_2(0.5) + 0.5 * \log_2(0.5)) = 1.0 \end{aligned}$$

$$\begin{aligned} \text{Gain}(S, B) &= 1 - (\text{portion}(S1) * E(S1) + \text{portion}(S2) * E(S2)) \\ &= 1 - ((2/4) * E(S1) + (2/4) * E(S2)) = 1 - ((2/4) * (1.0) + (2/4) * (1.0)) = 0 \end{aligned}$$

Gain(S,B)=0 < Gain(S,A)=1 donc on doit faire le partitionnement selon A qui donne le plus grand gain.

103

Choix de l'attribut de partitionnement

```

1 def Gain_d_information(S, numero_colon_attribut):
2     classes=list(set(S[:, -1]))
3     valeurs_attribut=list(set(S[:, numero_colon_attribut]))
4     Si=[[ ] for i in range(len(valeurs_attribut))]
5     for e in S:
6         e=list(e)
7         val_attribut_pour_e=e[numero_colon_attribut]
8         numero_sous_ensemble=valeurs_attribut.index(val_attribut_pour_e)
9         Si[numero_sous_ensemble].append(e)
10    Si=array(Si) ; som=0
11    for sous_ensemble in Si:
12        som+=(len(sous_ensemble)/len(S))*Entropie(sous_ensemble)
13    return(Entropie(S)-som)
  
```

```

1 print(Gain_d_information(df,1))
  
```

0.0

T15

104

Choix de l'attribut de partitionnement

	A	B	C
1	Douleur	Inanime	Infarctus
2	poitrine	oui	oui
3	ailleurs	oui	oui
4	poitrine	non	oui
5	poitrine	oui	oui
6	ailleurs	non	oui
7	poitrine	non	non
8	ailleurs	non	non
9	poitrine	oui	non
10	ailleurs	non	non
11	ailleurs	non	non

La répartition sera faite alors avec l'attribut 1 (colonne infarctus) qui donne le plus grand gain d'information

```

1 from numpy import *
2 import pandas
3 fich=pandas.read_csv('D:\\\\donneesAD1.csv',sep=';')
4 Data_train=fich.iloc[:,:]
5 Data_train=Data_train.to_numpy()
6 print(Gain_d_information(Data_train,0))
7 print(Gain_d_information(Data_train,1))

```

0.02904940554533142
0.12451124978365313

T16

105

Choix de l'attribut de partitionnement: Indice de GINI

- Si on fait un partitionnement de l'ensemble d'instances **S** selon l'attribut **Attribut** nous obtenons des sous ensembles d'instances **S_i** avec i compris entre 1 et le nombre k de valeurs distinctes de l'attribut **Attribut**:

$$S = \bigcup_{i=1}^k$$

- L'indice de GINI** qui mesure le désordre est donné par la formule:

$$GINI(S) = 1 - \sum_{c \in \text{Classes}} (\text{portion}(c))^2$$

- On prend l'attribut qui donne la plus petite mesure de désordre:**

$$\text{Mesure_de_Désordre}(S, \text{Attribut}) = \sum_{i=1}^k p_i * GINI(S_i)$$

Avec pi est la portion du sous ensemble d'instances **S_i** par rapport à l'ensemble d'instances **S**:

$$p_i = \frac{\text{Nombre de lignes de } S_i}{\text{Nombre de lignes de } S}$$

106

Choix de l'attribut de partitionnement: Indice de GINI

S: l'ensemble des instances
contient les lignes L1,L2,L3 et L4.
Classes={C1,C2}

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

$$GINI(S) = 1 - \sum_{i=1}^k (p_i)^2$$

$$\begin{aligned}
 GINI(S) &= 1 - ((\text{portion}(C1))^2 + (\text{portion}(C2))^2) \\
 &= 1 - ((0.5)^2 + (0.5)^2) \\
 &= 0.5
 \end{aligned}$$

107

Choix de l'attribut de partitionnement: Indice de GINI

```

1 def GINI(S):
2     clonne_des_classes=list(S[:, -1])
3     classes=list(set(S[:, -1]))
4     k=len(classes) ; s=0
5     for i in range(k):
6         pi=clonne_des_classes.count(classes[i])/len(S[:, -1])
7         s+=pi**2
8     return 1-s

```

```

1 from numpy import *
2 import pandas
3 df = pandas.DataFrame([[0,1,'C1'],[0,0,'C1'],
4 [1,1,'C2'],[1,0,'C2']],columns=['A', 'B', 'Classes'])
5 df=df.to_numpy()
6 print(GINI(df))

```

0.5

T17

108

Choix de l'attribut de partitionnement: Indice de GINI

S: l'ensemble des instances
contient les lignes L1,L2,L3 et L4.
Classes={C1,C2}

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

Partitionnement selon l'attribut A

S1 contient {L1==C1, L2==C1} **S2** contient {L3==C2, L4==C2}

$$\text{GINI}(S_1) = 1 - ((\text{portion}(C1))^2 \cdot (\text{portion}(C1))) = 0$$

$$\text{GINI}(S_2) = 1 - ((\text{portion}(C2))^2 \cdot (\text{portion}(C2))) = 0$$

$$\text{GINI}(S) = 1 - \sum_{i=1}^k (p_i)^2$$

109

Choix de l'attribut de partitionnement: Indice de GINI

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

Partitionnement selon l'attribut A

S1 contient {L1==C1, L2==C1} **S2** contient {L3==C2, L4==C2}

$$\text{GINI}(S_1) = 1 - ((\text{portion}(C1))^2 \cdot (\text{portion}(C1))) = 0$$

$$\text{GINI}(S_2) = 1 - ((\text{portion}(C2))^2 \cdot (\text{portion}(C2))) = 0$$

$$\text{Mesure_de_Désordre}(S, \text{Attribut}) = \sum_{i=1}^k p_i * \text{GINI}(S_i)$$

$$\text{Mesure_de_desordre}(S, A) = 0.5 * 0 + 0.5 * 0 = 0$$

110

Choix de l'attribut de partitionnement: Indice de GINI

```

1 def Mesure_desordre(S,numero_colon_attribut):
2     classes=list(set(S[:, -1]))
3     valeurs_attribut=list(set(S[:, numero_colon_attribut]))
4     Si=[[] for i in range(len(valeurs_attribut))]
5     for e in S:
6         val_attribut_pour_e=e[numero_colon_attribut]
7         numero_sous_ensemble=valeurs_attribut.index(val_attribut_pour_e)
8         Si[numero_sous_ensemble].append(e)
9     Si=array(Si)
10    S=0
11    for sous_ensemble in Si:
12        S+=(len(sous_ensemble)/len(S))*GINI(sous_ensemble)
13    return(S)

1 from numpy import *
2 import pandas
3 df = pandas.DataFrame([[0,1,'C1'],[0,0,'C1'],
4 [1,1,'C2'],[1,0,'C2']],columns=['A', 'B', 'Classes'])
5 df=df.to_numpy()
6 A=Mesure_desordre(df,0);print(A)

```

0.0

T17

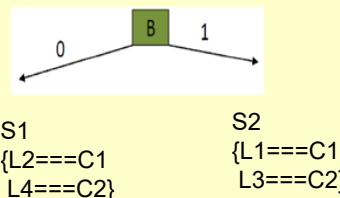
[111](#)

Choix de l'attribut de partitionnement: Indice de GINI

S: l'ensemble des instances
contient les lignes L1,L2,L3 et L4.
Classes={C1,C2}

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

Partitionnement selon l'attribut B



$$GINI(S) = 1 - \sum_{i=1}^k (p_i)^2$$

$$\begin{aligned} GINI(S1) &= 1 - ((\text{portion}(C1) * (\text{portion}(C1)) + (\text{portion}(C2) * (\text{portion}(C2)))) = 1 - (0.5^2 + 0.5^2) = 0.5 \\ GINI(S2) &= 1 - ((\text{portion}(C1) * (\text{portion}(C1)) + (\text{portion}(C2) * (\text{portion}(C2)))) = 1 - (0.5^2 + 0.5^2) = 0.5 \end{aligned}$$

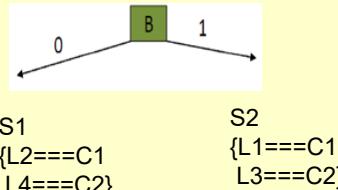
[112](#)

Choix de l'attribut de partitionnement: Indice de GINI

S: l'ensemble des instances
contient les lignes L1,L2,L3 et L4.
Classes={C1,C2}

Numéro de ligne	A	B	Classe
L1	0	1	C1
L2	0	0	C1
L3	1	1	C2
L4	1	0	C2

Partitionnement selon l'attribut B



$$\text{GINI}(S_1) = 1 - ((\text{portion}(C1)^2 \cdot \text{portion}(C1) + (\text{portion}(C2))^2 \cdot \text{portion}(C2))) = 1 - (0.5^2 + 0.5^2) = 0.5$$

$$\text{GINI}(S_2) = 1 - ((\text{portion}(C1))^2 \cdot \text{portion}(C1) + (\text{portion}(C2))^2 \cdot \text{portion}(C2)) = 1 - (0.5^2 + 0.5^2) = 0.5$$

$$\text{Mesure_de_Désordre}(S, \text{Attribut}) = \sum_{i=1}^k p_i * \text{GINI}(S_i)$$

$$\text{Mesure_de_desordre}(S, B) = 0.5 * 0.5 + 0.5 * 0.5 = 0.5$$

Mesure_de_desordre(S,B) > Mesure_de_desordre(S,A) donc on doit faire le partitionnement selon A qui donne la plus petite mesure de désordre.

113

Choix de l'attribut de partitionnement: Indice de GINI

```

1 def Mesure_desordre(S,numero_colon_attribut):
2     classes=list(set(S[:, -1]))
3     valeurs_attribut=list(set(S[:, numero_colon_attribut]))
4     Si=[[ ] for i in range(len(valeurs_attribut))]
5     for e in S:
6         val_attribut_pour_e=e[numero_colon_attribut]
7         numero_sous_ensemble=valeurs_attribut.index(val_attribut_pour_e)
8         Si[numero_sous_ensemble].append(e)
9     Si=array(Si)
10    s=0
11    for sous_ensemble in Si:
12        s+=(len(sous_ensemble)/len(S))*GINI(sous_ensemble)
13    return(s)
  
```

```

1 from numpy import *
2 import pandas
3 df = pandas.DataFrame([[0,1,'C1'],[0,0,'C1'],
4 [1,1,'C2'],[1,0,'C2']],columns=['A', 'B', 'Classes'])
5 df=df.to_numpy()
6 A=Mesure_desordre(df,1);print(A)
  
```

0.5

T17

114

Choix de l'attribut de partitionnement: **Indice de GINI**

	A	B	C
1	Douleur	Inanime	Infarctus
2	poitrine	oui	oui
3	ailleurs	oui	oui
4	poitrine	non	oui
5	poitrine	oui	oui
6	ailleurs	non	oui
7	poitrine	non	non
8	ailleurs	non	non
9	poitrine	oui	non
10	ailleurs	non	non
11	ailleurs	non	non

La répartition sera faite alors avec l'attribut 1 (colonne infarctus) qui donne la plus petite mesure de désordre

```

1 from numpy import *
2 import pandas
3 fich=pandas.read_csv('D:\\donneesAD1.csv',sep=';')
4 Data_train=fich.iloc[:, :]
5 Data_train=Data_train.to_numpy()
6 print(Mesure_desordre(Data_train,0))
7 print(Mesure_desordre(Data_train,1))

```

0.48
0.41666666666666667

T18

[115](#)

Choix de l'attribut de partitionnement: **Indice de GINI**

```

1 from numpy import *
2 import pandas
3 f = pandas.read_csv('D:\\balance-scale1.data')
4 print(f.shape[0])
5 Data_train=f.iloc[0:8,:]
6 Data_train=Data_train.to_numpy()
7
8 def GINI(S):
9     S=array(S)
10    clonne_des_classes=list(S[:, -1])
11    classes=list(set(S[:, -1]))
12    k=len(classes)
13    s=0
14    for i in range(k):
15        pi=clonne_des_classes.count(classes[i])/len(S[:, -1])
16        s+=pi**2
17    return 1-s

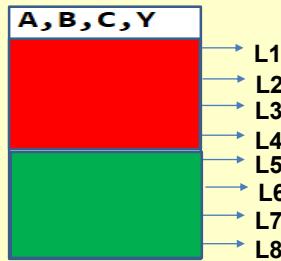
```

T20

[116](#)

Choix de l'attribut de partitionnement: Indice de GINI

- Dans les exemples précédents nous avons considéré des attributs catégoriels: Douleur(poitrine, ailleurs), inanime(oui, non)
- Attributs continues (age, poids, taux de réussite,)
- Ici nous considérons que les attributs A,B et C sont continus



Si on prend un attribut Note_du_Bac (avec deux chiffres après la virgule), si on a un data set qui contient cet attribut. Combien de sous ensemble obtenez vous si vous faites un regroupement selon cet attribut ?

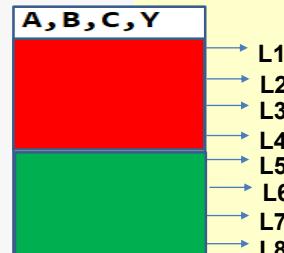
Exemple: Note_du_Bac Note_du_Bac=[8,9.2,9.6,11.5,13,14,15,17,16.6,18,19]
Gain d'information si on utilise la formule précédente ? Est-il maximal ?

[117](#)

Choix de l'attribut de partitionnement: Indice de GINI

```

22 def Mesure_desordre(S,numero_colon_attribut):
23     Bonne_mesure_desordre+=inf
24     Bonne_valeur_de_repartition=0
25     classes=list(set(S[:, -1]))
26     valeurs_attribut=list(set(S[:, numero_colon_attribut]))
27     valeurs_attribut.sort()
28     Gauche=[]; Droite=[]
29     for i in range(len(valeurs_attribut)-1):#le dernier est le max !
30         v=valeurs_attribut[i]; Gauche.clear(); Droite.clear()
31         for e in S:
32             val_attribut_pour_e=e[numero_colon_attribut]
33             if val_attribut_pour_e<=v:
34                 Gauche.append(e)
35             else:
36                 Droite.append(e)
37         pGau=len(Gauche)/len(S)
38         pDr=len(Droite)/len(S)
39         mesure=(pGau)*GINI(Gauche)+(pDr)*GINI(Droite)
40         if mesure<Bonne_mesure_desordre:
41             Bonne_mesure_desordre=mesure
42             Bonne_valeur_de_repartition=v
43     return(Bonne_mesure_desordre,Bonne_valeur_de_repartition)
    
```



Exemple: numero_colonne=1(B) et valeur=4

Gauche=[L1,L2,L3,L5,L8] et Droite=[L4,L6,L7]

[118](#)

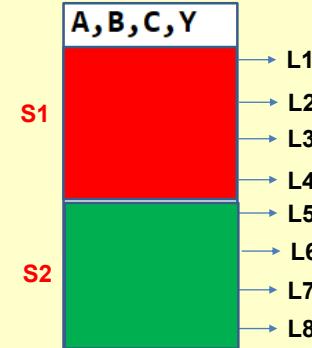
Choix de l'attribut de partitionnement: **Indice de GINI**

```

1 print(Mesure_desordre(Data_train,0))
2 print(Mesure_desordre(Data_train,1))
3 print(Mesure_desordre(Data_train,2))

(0.1875, 1)
(0.2083333333333326, 1)
(0.35714285714285715, 2)

```



T20

[119](#)

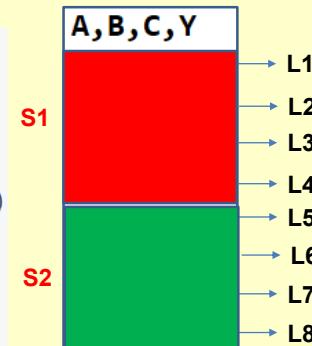
Choix de l'attribut de partitionnement: **Indice de GINI**

```

42 def meilleur_attribut_meilleure_valeur(S):
43     meilleur_attribut=0
44     meilleure_val=0
45     meilleure_mesure=+inf
46     for num_attribut in range(len(S[0])-1):
47         mesure,valeur=Mesure_desordre(S,num_attribut)
48         if mesure<meilleure_mesure:
49             meilleure_val=valeur
50             meilleure_mesure=mesure
51             meilleur_attribut=num_attribut
52     return meilleur_attribut,meilleure_val
53 print(meilleur_attribut_meilleure_valeur(Data_train))

(0, 1)

```



T20

[120](#)

Matrice de confusion

Définition: Matrice de confusion

Soit un Data_test qui contient m échantillons (m exemples) dont les classes sont "Positif" ou "Négatif". Avec un modèle de ML on classe ces m échantillons. Ces échantillons seront réparties en 4 groupes:

- Le premier groupe contient les V_p échantillons + (positifs) qui sont classés +
- Le deuxième contient les F_p échantillons + qui sont classés -
- Le troisième groupe contient les V_n échantillons - qui sont classés -
- Le quatrième groupe contient les F_n échantillons - qui sont classés +

Matrice de confusion :

	Classé +	Classé -
Exemple +	V_p	F_n
Exemple -	F_p	V_n

$$\text{Taux d'erreur} = 1 - \text{taux de réussite(accuracy)} = \frac{F_p + F_n}{F_p + F_n + V_p + V_n}$$

121

Matrice de confusion

Exemple: Matrice de confusion

Supposons dans l'exemple de détection des SMS spam (faux sms) des SMS ham (vrai SMS). Le classificateur a été testé par exemple sur 200 SMS.

		Classe estimée - (par le classificateur)	
		Ham	Spam
Classe réelle - (selon le destinataire humain des sms)	Ham	95 (vrais positifs)	5 (faux négatifs)
	Spam	3 (faux positifs)	97 (vrais négatifs)

$$\text{Taux d'erreur} = 1 - \text{taux de réussite(accuracy)} = \text{accuracy} = \frac{\text{rouge} + \text{bleu}}{\text{rouge} + \text{vert} + \text{bleu} + \text{orange}}$$

122

Les arbres de décision

```

1 import numpy as np
2 import pandas as pd
3 from sklearn.metrics import confusion_matrix
4 from sklearn.tree import DecisionTreeClassifier
5 from sklearn.metrics import *

[[1 1 2 'oui']
 [1 4 6 'oui']
 [1 1 3 'oui']
 [1 5 7 'non']
 [2 2 3 'non']
 [2 6 5 'non']
 [2 8 9 'non']
 [2 3 3 'non']
 [2 6 9 'non']
 [2 8 9 'non']
 [1 8 7 'oui']
 [1 9 6 'oui']
 [1 0 4 'oui']]
```

```

1 data = pd.read_csv('D:\\balance-scale1.data').as_matrix()
2 # Printing the dataset shape
3 print ("Dataset Length: ", len(data))
4 print ("Dataset Shape: ", data.shape)
5 print(data)
```

T19

Les arbres de décision

```

1 X = data[:8,0:3]; Y = data[:8,3] ; X_test=data[8:,0:3]; y_test=data[8:,3]
2 clf_gini = DecisionTreeClassifier(criterion = "gini")
3 clf_gini.fit(X, Y)
4 clf_entropy = DecisionTreeClassifier(criterion = "entropy")
5 clf_entropy.fit(X, Y)
6 print("Results Using Gini Index:")
7 y_pred_gini = clf_gini.predict(X_test)
8 print("Confusion Matrix: ",confusion_matrix(y_test, y_pred_gini))
9 print(X_test,y_test,y_pred_gini)
10 print ("Accuracy : ",accuracy_score(y_test,y_pred_gini)*100)
11 print("Results Using Entropy:")
12 y_pred_entropy = clf_entropy.predict(X_test)
13 print("Confusion Matrix: ",confusion_matrix(y_test, y_pred_entropy))
14 print ("Accuracy : ",accuracy_score(y_test,y_pred_entropy)*100)
```

T19

124

Les arbres de décision

Results Using Gini Index:

Confusion Matrix:

	prédite non	prédite oui
Les non	[2]	0]
Les oui	[1]	2]]

Classes correctes données par l'expert ['non' 'non' 'oui' 'oui' 'oui']

Classes prédites par AD-Gini ['non' 'non' 'non' 'oui' 'oui']

Accuracy : 80.0

Results Using Entropy:

Confusion Matrix: [[2 0]
[2 1]]

Accuracy : 60.0

Le nombre d'exemples qui sont bien classés est égal à la trace de la matrice de confusion: trace(MC)=somme de la diagonale (MC)

T19

[125](#)

Les arbres de décision

```

1 from sklearn.externals.six import StringIO
2 from IPython.display import Image
3 from sklearn.tree import export_graphviz
4 import pydotplus

```

```

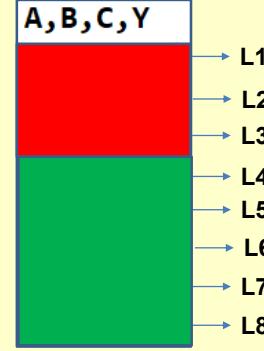
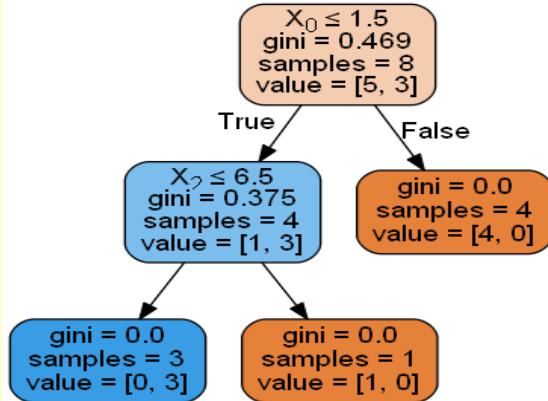
1 dot_data = StringIO()
2 export_graphviz(clf_gini, out_file=dot_data,
3                  filled=True, rounded=True,
4                  special_characters=True)
5 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
6
7 svg_graph = graph.create_svg()
8 svg = open('gini_tree.svg', 'wb')
9 svg.write(svg_graph)
10 svg.close()
11 Image(graph.create_png())

```

T19

[126](#)

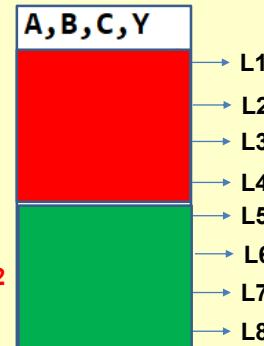
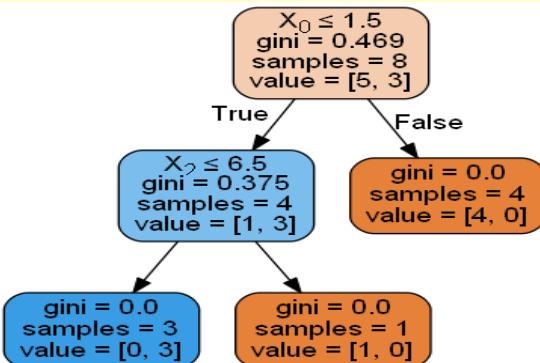
Les arbres de décision



Le data train $S=\{L1, L2, L3, L4, L5, L6, L7, L8\}$ contient 8 échantillons (samples=8)
 5 échantillons ont la classe "non" et 3 échantillons ont la classe "oui" (value=[5,3])

127

Les arbres de décision



La condition $X₀ \leq 1.5$ (ici $X₀$ = colonne A) permet de diviser l'ensemble $S=\{L1, L2, L3, L4, L5, L6, L7, L8\}$ en deux ensembles de cardinal 4:

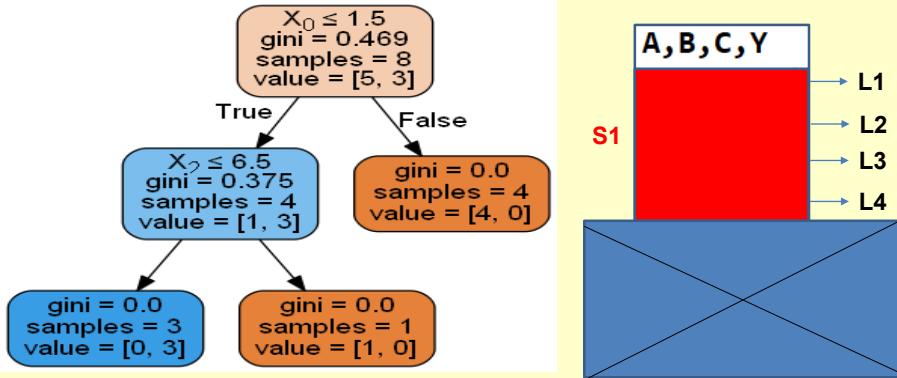
$S1=\{L1, L2, L3, L4\}$ et $S2=\{L5, L6, L7, L8\}$

Les échantillons de $S1$ sont ceux qui vérifient la condition:

$X₀ \leq 1.5$ (le mot True), dans $S1$: 3 échantillons ont la classe "oui" et 1 échantillon a la classe "non", donc l'ensemble $S1$ n'est pas homogène !

128

Les arbres de décision



La condition $X_2 \leq 6.5$ (ici X_2 = colonne C) permet de diviser l'ensemble $S1=\{L1, L2, L3, L4\}$ en deux ensembles :

$S3=\{L1, L2, L3\}$ et $S4=\{L4\}$

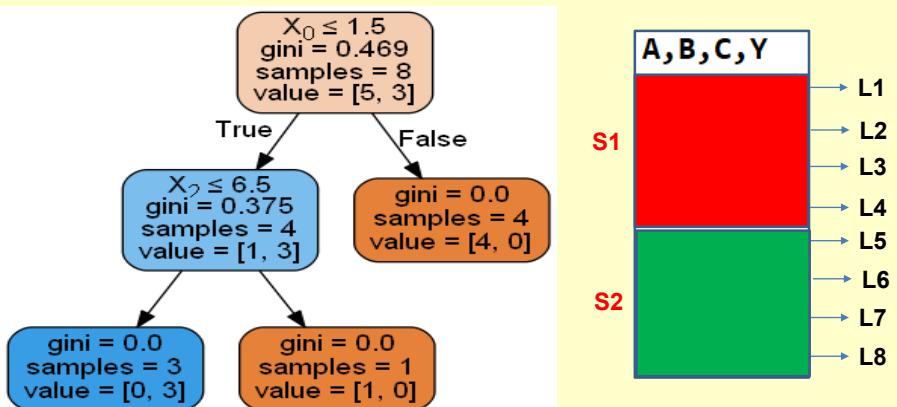
Les échantillons de $S3$ sont ceux qui vérifient la condition: $X_2 \leq 6.5$

dans $S3$, tous les échantillons ont la classe "oui" et $S3$ est homogène

Les échantillons de $S4$ sont ceux qui ne vérifient pas la condition: $X_2 \leq 6.5$ dans $S4$, tous les échantillons ont la classe "non" et $S4$ est homogène

129

Les arbres de décision



La condition $X_0 \leq 1.5$ (ici X_0 = colonne A) permet de diviser l'ensemble $S=\{L1, L2, L3, L4, L5, L6, L7, L8\}$ en deux ensembles de cardinal 4:

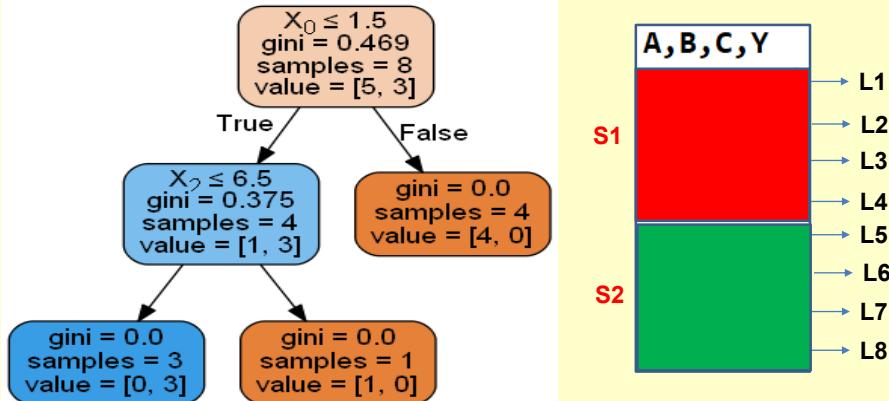
$S1=\{L1, L2, L3, L4\}$ et $S2=\{L5, L6, L7, L8\}$

Les échantillons de $S2$ sont ceux qui ne vérifient pas la condition:

$X_0 \leq 1.5$ (le mot False) et ils sont tous dans la classe "non"

130

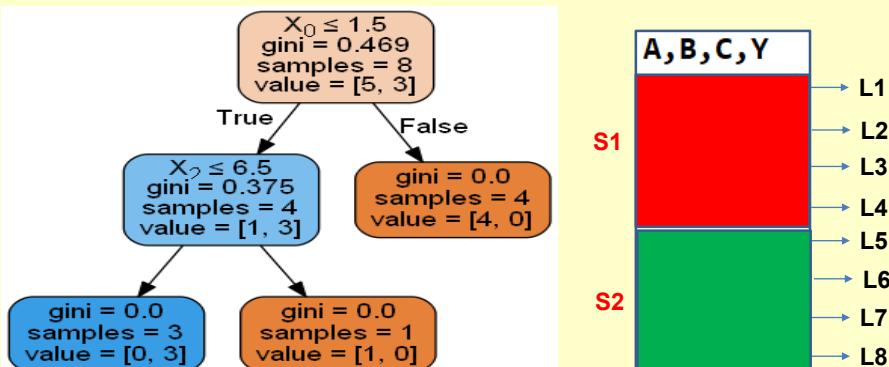
Les arbres de décision



Les échantillons de S2 sont ceux qui ne vérifient pas la condition: $X_0 \leq 1.5$ (le mot False) et ils sont tous dans la classe "non" donc S2 est homogène,

131

Les arbres de décision



```

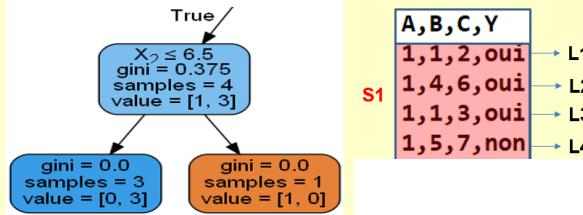
1 fich=pandas.read_csv('D:\\balance-scale1.data')
2 Data_train=fich.iloc[:8,:]
3 Data_train=Data_train.to_numpy()
4 print(GINI(Data_train))
5 print(meilleur_attribut_meilleure_valeur(Data_train))
  
```

0.46875
(0, 1)

Tp 21

132

Les arbres de décision



```

1 Data_train=array([[1, 1, 2, 'oui'],[1, 4, 6, 'oui'],
2 [1, 1, 3, 'oui'],[1, 5, 7, 'non']])
3 print(GINI(Data_train), end=' ')
4 print(Mesure_desordre(Data_train,0), end=' ')
5 print(Mesure_desordre(Data_train,1), end=' ')
6 print(Mesure_desordre(Data_train,2), end=' ')
7 print('--->'+str(meilleur_attribut_meilleure_valeur(Data_train)))

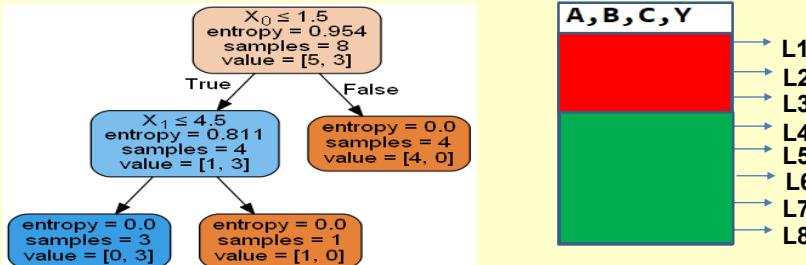
```

0.375 (inf, 0) (0.0, '4') (0.0, '6') --->(1, '4')

Tp 21

133

Les arbres de décision



```

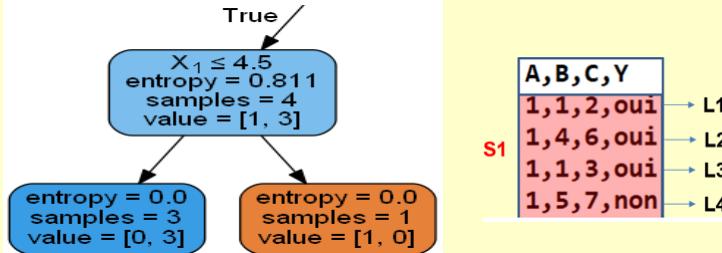
1 from numpy import *
2 import pandas
3 fich=pandas.read_csv('D:\\balance-scale1.data')
4 Data_train=fich.iloc[:8,:]
5 Data_train=Data_train.to_numpy()
6 print(Entropie(Data_train))
7 print(Mesure_Gain_d_information_continue(Data_train,0), end=' ')
8 print(Mesure_Gain_d_information_continue(Data_train,1), end=' ')
9 print(Mesure_Gain_d_information_continue(Data_train,2), end=' ')

```

Tp 22

0.954434002924965
(0.5487949406953987, 1) (0.46691718668869947, 1) (0.20443400292496505, 6)

Les arbres de décision



```

1 Data_train=array([[1, 1, 2, 'oui'],[1, 4, 6, 'oui'],
2 [1, 1, 3, 'oui'],[1, 5, 7, 'non']])
3 print(Entropie(Data_train))
4 print(Mesure_Gain_d_information_continue(Data_train,0), end=' ')
5 print(Mesure_Gain_d_information_continue(Data_train,1), end=' ')
6 print(Mesure_Gain_d_information_continue(Data_train,2), end=' ')
7

```

0.8112781244591328
(0, 0) (0.8112781244591328, '4') (0.8112781244591328, '6')

Tp 22

135

Les arbres de décision

```

1 def transformer_colText_Num(Data,num):
2     if(type(Data[0][num])==str):
3         Valeurs=list(set(Data[:,num]))
4         for i in range(len(Data[:,num])):
5             Data[i][num]=Valeurs.index(Data[i][num])

```



```

1 data = pd.read_csv('D:\\\\donneesAD.csv',sep=';').as_matrix()
2 print(data)
3 for i in range(data.shape[1]-1):
4     transformer_colText_Num(data,i)
5 print(data)

```

	Douleur	Age	Inanime	Infarctus
poitrine	45	oui	oui	[[0 45 1 'oui']]
ailleurs	25	oui	oui	[2 25 1 'oui']]
poitrine	35	non	oui	[0 35 0 'oui']]
Tete	70	oui	oui	[1 70 1 'oui']]
ailleurs	34	non	oui	[2 34 0 'oui']]
poitrine	60	non	non	[0 60 0 'non']]
ailleurs	67	non	non	[2 67 0 'non']]
Tete	52	oui	non	[1 52 1 'non']]
ailleurs	58	non	non	[2 58 0 'non']]
ailleurs	34	non	non	[2 34 0 'non']]

Tp 23

136

Les arbres de décision

```

1 X = data[:8,0:3]; Y = data[:8,3] ; X_test=data[8:,0:3]; y_test=data[8:,3]
2 clf_gini = DecisionTreeClassifier(criterion = "gini")
3 clf_gini.fit(X, Y)
4 print("Results Using Gini Index:")
5 y_pred_gini = clf_gini.predict(X_test)
6 print("Confusion Matrix: ",
7      confusion_matrix(y_test, y_pred_gini))
8 print ("Accuracy : ",
9      accuracy_score(y_test,y_pred_gini)*100)

Results Using Gini Index:
Confusion Matrix: [[1 1]
 [0 0]]
Accuracy : 50.0

```

```

1 from sklearn.externals.six import StringIO |
2 from IPython.display import Image
3 from sklearn.tree import export_graphviz
4 import pydotplus

```

Tp 23

[137](#)

Les arbres de décision

```

1 from sklearn.externals.six import StringIO |
2 from IPython.display import Image
3 from sklearn.tree import export_graphviz
4 import pydotplus

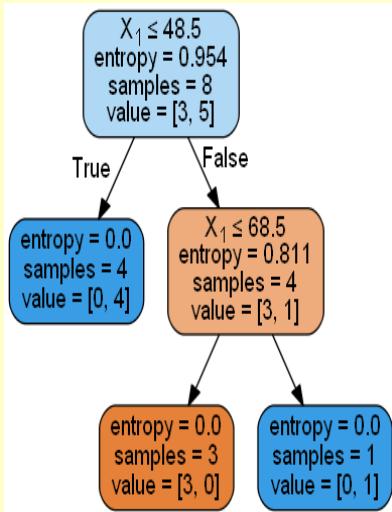
1 dot_data = StringIO()
2 export_graphviz(clf_gini, out_file=dot_data,
3                  filled=True, rounded=True,
4                  special_characters=True)
5 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
6
7 svg_graph = graph.create_svg()
8 svg = open('gini_tree.svg', 'wb')
9 svg.write(svg_graph)
10 svg.close()
11
12 Image(graph.create_png())

```

Tp 23

[138](#)

Les arbres de décision



```

[['poitrine' 45 'oui' 'oui'] —— [1 45 1 'oui']
['ailleurs' 25 'oui' 'oui'] —— [2 25 1 'oui']
['poitrine' 35 'non' 'oui'] —— [1 35 0 'oui']
['Tete' 70 'oui' 'oui'] —— [0 70 1 'oui']
['ailleurs' 34 'non' 'oui'] —— [2 34 0 'oui']
['poitrine' 60 'non' 'non'] —— [1 60 0 'non']
['ailleurs' 67 'non' 'non'] —— [2 67 0 'non']
['Tete' 52 'oui' 'non'] ] —— [0 52 1 'non']]
  
```

Tp 23

139

Les arbres de décision: Credit Scoring

Le processus d'acceptation d'un crédit à la consommation doit être très rapide en particulier lorsque les clients ont besoin d'un crédit en magasin au moment de leur achat (exemple : paiement en trois fois sans frais, etc.).

Depuis très longtemps, les banques et les organismes de crédit ont développé des méthodes automatiques et instantanées de notation de la solvabilité des clients et de leur capacité à rembourser le crédit. Ces outils sont très utiles pour une acceptation rapide d'octroi de crédit, mais également pour déterminer le montant de l'enveloppe d'autorisation de crédit ou pour effectuer le suivi du risque des clients et des portefeuilles de prêts.

Le score de crédit est une fonction qui attribue une valeur de qualité de crédit à un client ou un prêt en fonction de variables explicatives telles que le ratio d'endettement de l'emprunteur, son comportement de compte ou tout autre grandeur qui est corrélée au défaut de l'emprunteur.

140

Les arbres de décision

```

df = pd.read_csv("D:\\CreditScoring.csv")
print(df.isnull().sum())
#remplir les cases vides (NaN) par la moyenne
df.fillna(value=df.mean(), inplace=True)
print("++++++ 1 ++++++")
print(df.isnull().sum())
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# removing the features BAD, JOB, REASON
x_basic = df.drop(columns=["BAD", "JOB", "REASON"]); y = df["BAD"]
print("++++++ 2 ++++++")
print(x_basic.isnull().sum())
x_basic_tr,x_basic_te,y_tr,y_te = train_test_split(\ 
    x_basic,y,test_size=.33,random_state=1)
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(x_basic_tr,y_tr)
y_pre = model.predict(x_basic_te)
a= accuracy_score(y_te,y_pre)
print("accuracy score : ",a)

```

Tp 24

141

Les arbres de décision

	++++ 1 +++++		++++++ 2 ++++++
BAD	0	BAD	0
LOAN	0	LOAN	0
MORTDUE	518	MORTDUE	0
VALUE	112	VALUE	0
REASON	252	REASON	252
JOB	279	JOB	279
YOJ	515	YOJ	0
DEROG	708	DEROG	0
DELINQ	580	DELINQ	0
CLAGE	308	CLAGE	0
NINQ	510	NINQ	0
CLNO	222	CLNO	0
DEBTINC	1267	DEBTINC	0
			dtype: int64
			accuracy score : 0.8647686832740213

Tp 24

142

Les arbres de décision

```

1 X_test=np.array(x_basic_te); y_test=np.array(y_te)
2 print(X_test[127])
3 print(model.predict([X_test[127]]),end=' '); print(y_test[127])
[1.57000000e+04 8.77540000e+04 1.03045000e+05 2.10000000e+01
 0.00000000e+00 0.00000000e+00 1.47450177e+02 4.00000000e+00
 2.30000000e+01 2.91817426e+01]
[0] 0

1 print(X_test[0])
2 print(model.predict([X_test[0]]),end=' '); print(y_test[0])
[2.16000000e+04 1.54991000e+05 1.01776049e+05 8.92226814e+00
 2.00000000e+00 4.00000000e+00 1.65600830e+02 0.00000000e+00
 4.30000000e+01 3.80149173e+01]
[1] 1

```

143

Les algorithmes de construction d'un arbre de décision:

ID3: Iterative Dichotomiser 3

- Construit un arbre de décision de façon récursive en choisissant l'attribut qui maxime le gain d'information selon l'entropie de Shannon.
- ID3 se base sur les attributs et les classes, il cherche l'attribut le plus pertinent pour que l'arbre soit le plus court possible.
- Construit l'arbre de haut en bas, sans retour en arrière.
- Le Gain d'information est utilisé pour sélectionner l'attribut le plus utile pour la classification.

Les algorithmes de construction d'un arbre de décision:

Algorithme ID3 : Principe

- Déterminer un attribut A à placer en racine de l'arbre
 - A = Meilleur Attribut (Exemples)
- Pour chaque valeur de A, créer une branche étiquetée avec cette valeur ➔ un nouveau nœud fils de la racine
- Classer les exemples dans les nœuds fils en considérant tous les attributs excepté celui qui vient d'être mis à la racine
- Si tous les exemples d'un nœud fils sont homogènes, affecter leur classe au nœud ➔ Une feuille de l'arbre
- sinon recommencer à partir de ce nœud

Les arbres de décision pour la régression

Pour la régression avec les arbres de décision, scikit-learn offre la classe `DecisionTreeRegressor`. Comme pour la classification, la méthode `fit(...)` prend en entrée le paramètre X (attributs des observations). Attention : les y ne sont pas des étiquettes de classes mais des valeurs réelles.

```

1 from sklearn import tree
2 X = [[0, 0], [2, 2]]
3 y = [0.5, 2.5]
4 clf = tree.DecisionTreeRegressor()
5 clf = clf.fit(X, y)
6 clf.predict([[1, 1]])

array([0.5])

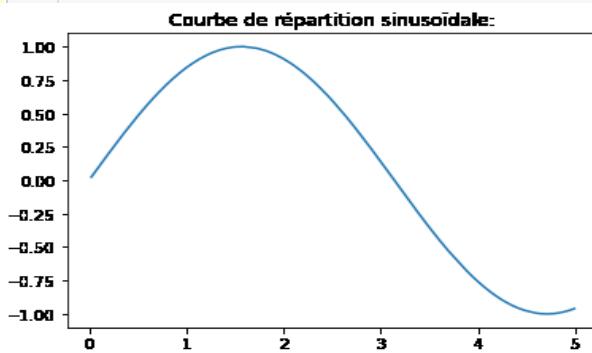
```

Les arbres de décision pour la régression

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.tree import DecisionTreeRegressor
4 # Créer les données d'apprentissage
5 np.random.seed(0)
6 X = np.sort(5 * np.random.rand(200, 1), axis=0)
7 y = np.sin(X)
8 plt.plot(X, y)
9 plt.title("Courbe de répartition sinusoïdale:")

```



Tp 25

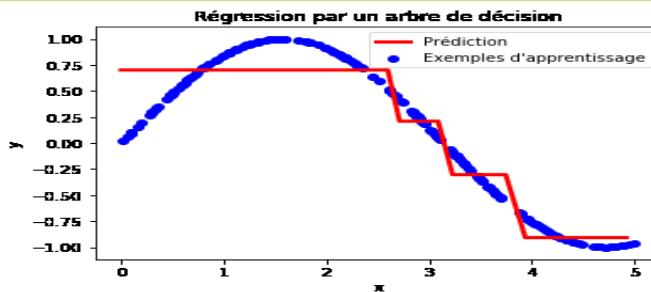
147

Les arbres de décision pour la régression

```

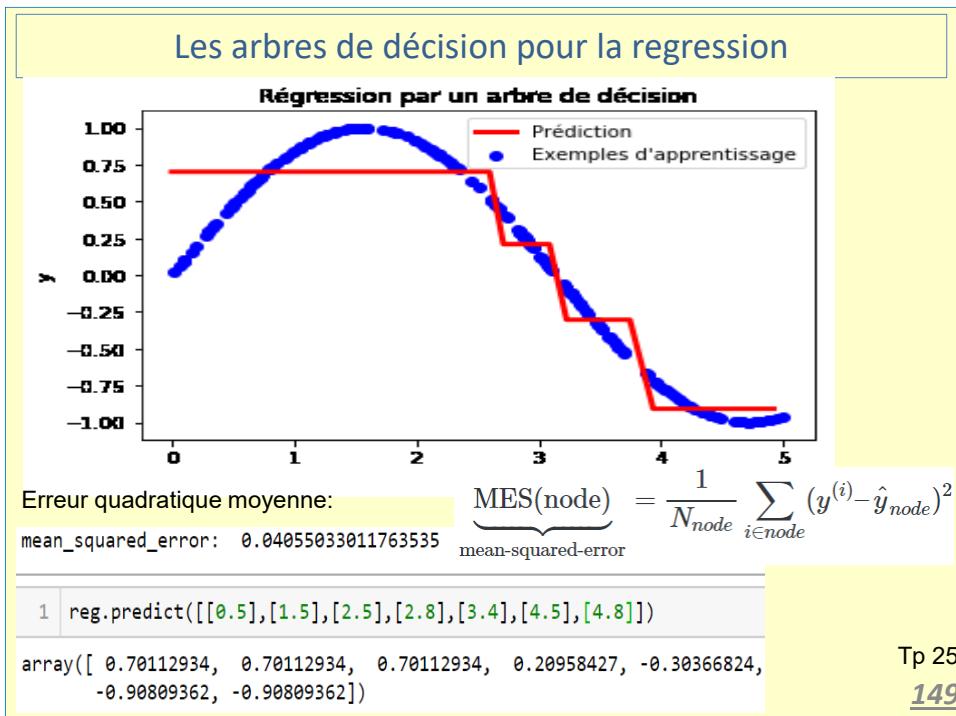
from sklearn.metrics import mean_squared_error
reg = DecisionTreeRegressor(max_depth=2)
reg.fit(X, y)
X_test = np.sort(5 * np.random.rand(80, 1), axis=0)
y_predicted = reg.predict(X_test)
y_test = np.sin(X_test)
plt.figure()
plt.scatter(X, y, color="b", label="Exemples d'apprentissage")
plt.plot(X_test, y_predicted, color="r", label="Prédiction", linewidth=3)
plt.xlabel("x"); plt.ylabel("y")
plt.title("Régression par un arbre de décision")
plt.legend(); plt.show()
a=mean_squared_error(y_test, y_predicted); print("mean_squared_error: ",a)

```



Tp 25

148



- ### Les arbres de décision pour la régression
- Les arbres de décision ont plusieurs **avantages** qui les rendent intéressants dans des contextes où il est utile de comprendre la séquence de décisions prise par le modèle :
 - Ils sont simples à comprendre et à visualiser.
 - Le coût d'utilisation des arbres est logarithmique.
 - Ils sont capables d'utiliser des données catégorielles et numériques.
 - Ils sont capables de traiter des problèmes multi-classe.
 - Modèle en boîte blanche : le résultat est facile à conceptualiser et à visualiser.
- 150

Les arbres de décision pour la régression

- Les arbres de décision ont plusieurs **avantages** qui les rendent intéressants dans des contextes où il est utile de comprendre la séquence de décisions prise par le modèle :
- Ils sont simples à comprendre et à visualiser.
- Le coût d'utilisation des arbres est logarithmique.
- Ils sont capables d'utiliser des données catégorielles et numériques.
- Ils sont capables de traiter des problèmes multi-classe.
- Modèle en boîte blanche : le résultat est facile à conceptualiser et à visualiser.

Sur-apprentissage : parfois les arbres générés sont trop complexes et généralisent mal. Choisir des bonnes valeurs pour les paramètres profondeur maximale (**max_depth**) et nombre minimal d'exemples par feuille (**min_samples_leaf**) permet d'éviter ce problème.

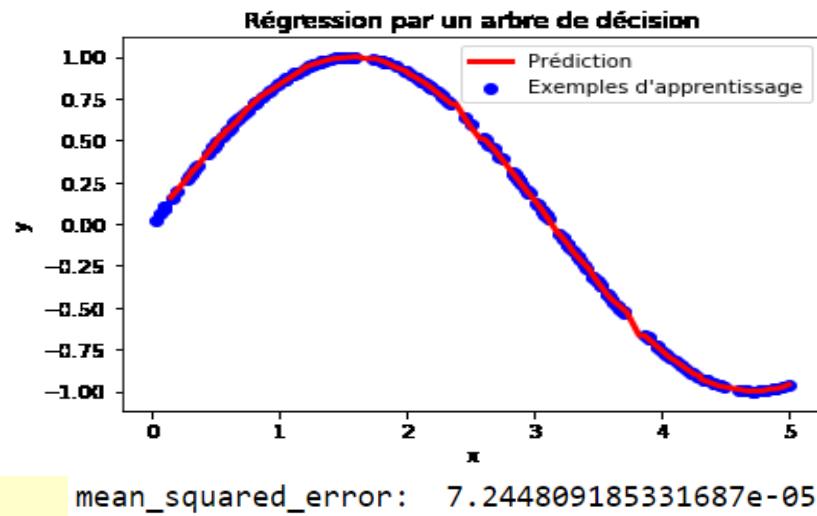
151

Les arbres de décision pour la régression

```
reg = DecisionTreeRegressor(max_depth=200)
reg.fit(X, y)
X_test = np.sort(5 * np.random.rand(80, 1), axis=0)
y_predicted = reg.predict(X_test)
y_test = np.sin(X_test)
plt.figure()
plt.scatter(X, y, color="b", label="Exemples d'apprentissage")
plt.plot(X_test, y_predicted, color="r", label="Prédiction", linewidth=3)
plt.xlabel("x"); plt.ylabel("y")
plt.title("Régression par un arbre de décision")
plt.legend(); plt.show()
a=mean_squared_error(y_test, y_predicted); print("mean_squared_error: ",a)
```

152

Les arbres de décision pour la régression



[153](#)

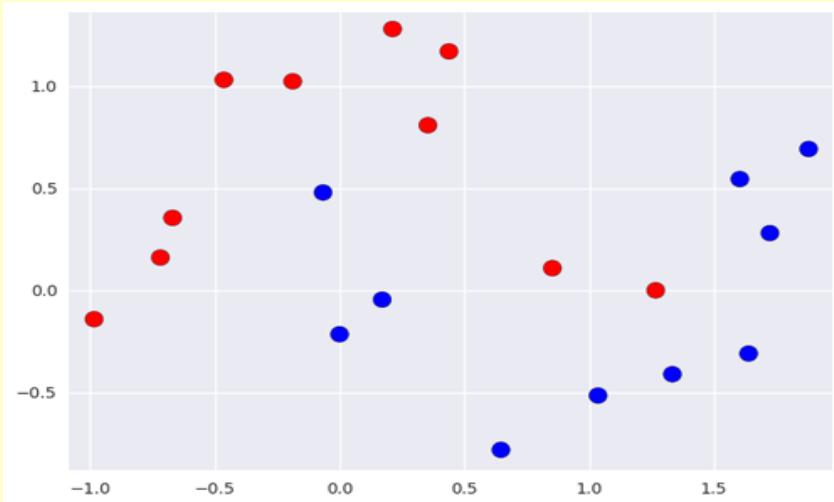
KNN: k Nearest Neighbors

C'est un algorithme qui peut servir autant pour la classification que la régression. Il est surnommé « nearest neighbors » (K plus proches voisins en français) car le principe de ce modèle consiste en effet à choisir les **k** données les plus proches du point étudié afin d'en prédire sa valeur (sa classe).

- K-NN est une méthode d'apprentissage **supervisé**, qui fonctionne sous le principe de : **“Dis moi qui sont tes voisins, je te dirais qui tu es...”**.
- Avant d'utiliser un vrai jeu de données, on va prendre un petit exemple afin de comprendre le fonctionnement de l'algorithme KNN: un jeu de données d'entraînement, avec deux classes, rouge et bleu. L'*input* est donc *bidimensionnel* ici, et la *target* est la couleur à classer.

[154](#)

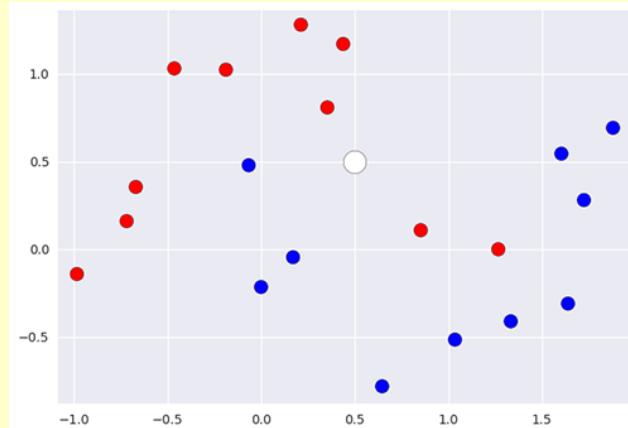
KNN: k Nearest Neighbors



155

KNN: k Nearest Neighbors

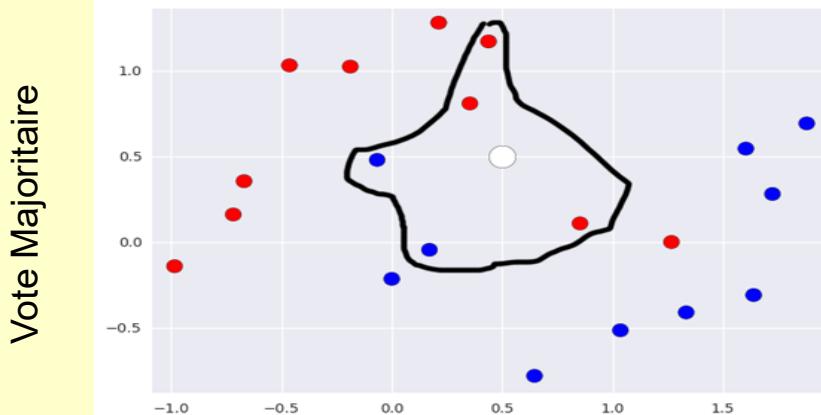
Si on a une nouvelle entrée dont on veut prédire la classe, comment pourrait-on faire ?



156

KNN: k Nearest Neighbors

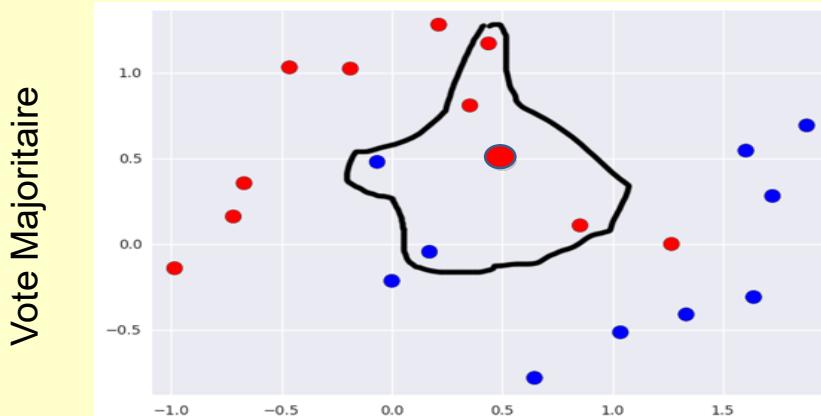
on va simplement regarder les **k** voisins les plus proches de ce point et regarder quelle classe constitue la majorité de ces points afin d'en déduire la classe du nouveau point. Par exemple ici, si on utilise le 5-NN, on peut prédire que la nouvelle donnée appartient à la classe **rouge** puisqu'elle a 3 rouges et 2 bleus dans son entourage.



157

KNN: k Nearest Neighbors

on va simplement regarder les **k** voisins les plus proches de ce point et regarder quelle classe constitue la majorité de ces points afin d'en déduire la classe du nouveau point. Par exemple ici, si on utilise le 5-NN, on peut prédire que la nouvelle donnée appartient à la classe **rouge** puisqu'elle a 3 rouges et 2 bleus dans son entourage.



158

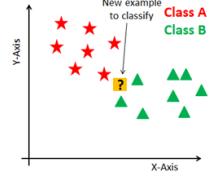
KNN: k Nearest Neighbors

1. Initialiser la nouvelle donnée

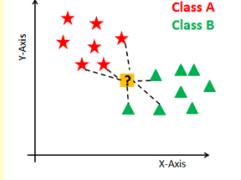
1. Calculer la distance

2. Trouver les plus proches voisins

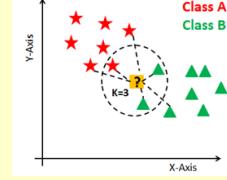
3. Voter pour les étiquettes



2. Calculer la Distance



3. Définir les k plus proches voisins & Voter pour les étiquettes

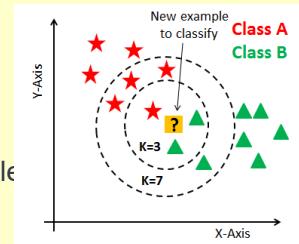


- **Distance Euclidienne :**
- **Distance de Manhattan :**
- **Distance de Minkowski :**

159

KNN: k Nearest Neighbors

- K est un **hyperparamètre** ou une variable de contrôle pour le modèle de prédiction.
- Chaque jeu de données a ses propres exigences.
- K doit être un nombre **impair** pour simplifier le vote majoritaire.
- Générer le modèle sur différentes valeurs de k et vérifier leurs performances.



160

KNN: k Nearest Neighbors

Les étapes de KNN:

- 1) Choisir la distance, la mesure de similarité entre les échantillons(instances)
- 2) Choisir (déterminer) la valeur de K.
- 3) Calculer la distance entre la nouvelle entrée et toutes les données de la base d'apprentissage.
- 4) Déterminer les classes des K plus proches voisins
- 5) Donner la prédiction pour la nouvelle entrée.

Avantages	Inconvénients
<ul style="list-style-type: none"> • Apprentissage rapide • Ne nécessite pas une généralisation de modèle • Peut être utile en cas de données non linéaires. • Méthode facile à comprendre . 	<ul style="list-style-type: none"> • Prédiction Lente et coûteuse en temps et en mémoire, car il faut stocker tous les exemples et il faut les revoir de nouveau pour chaque prédiction. • Ne convient pas pour les grandes dimensions.

161

KNN: L'algorithme

- 1.Charger les données
- 2.Initialiser **k** au nombre de plus proches voisins choisi
3. Pour chaque exemple dans les données:
 - 3.1 Calculer la distance entre notre requête et l'observation itérative actuelle de la boucle depuis les données.
 - 3.2 Ajouter la distance et l'indice de l'observation concernée à une collection ordonnée de données
4. Trier cette collection ordonnée contenant distances et indices de la plus petite distance à la plus grande (dans l'ordre croissant).
5. Sélectionner les **k** premières entrées de la collection de données triées (équivalent aux k plus proches voisins)
6. Obtenir les étiquettes des **k** entrées sélectionnées
7. Si **régression**, retourner la moyenne des **k** étiquettes
8. Si **classification**, retourner le mode (valeur la plus fréquente/comune)

162

KNN: k Nearest Neighbors

```

1 X = [[0], [1], [2], [3]]
2 y1 = [0, 0, 1, 1]
3 from sklearn.neighbors import KNeighborsClassifier
4 #valerur de k pour knn et choix de la distance
5 neigh = KNeighborsClassifier(n_neighbors=3, metric="euclidean")
6 neigh.fit(X, y1)
7 print('Knn predefinie-classification:',neigh.predict([[2.1]]))
8 from sklearn.neighbors import KNeighborsRegressor
9 knn_regression = KNeighborsRegressor(3)
10 y2 = [15.3, 8.6, 6.5, 4.29]
11 knn_regression.fit(X,y2)
12 print('Knn predefinie-regression:',knn_regression.predict([[2.1]]))
```

Knn predefinie-classification: [1]
 Knn predefinie-regression: [6.46333333]

Tp 26

KNN: k Nearest Neighbors

Rappel sur enumerate:

```

Mots = ['Spring', 'Summer', 'Fall', 'Winter']
print(list(enumerate(Mots)))
#[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
print(list(enumerate(Mots, start=1)))
#[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

Rappels sur Counter:

```

>>> from collections import Counter
>>> L=[3, 1, 3, 1, 3, 1, 1, 1, 2, 2, 2, 3, 2, 2, 2]
>>> Counter(L)
Counter({2: 6, 1: 5, 3: 4})
>>> Counter(L).most_common(1)
[(2, 6)]
>>> Counter(L).most_common(2)
[(2, 6), (1, 5)]
```

Tp 27

KNN: k Nearest Neighbors

```

1 from collections import Counter
2 import math
3 def mean(labels):
4     """Calcul de la moyenne des k étiquettes"""
5     return sum(labels) / len(labels)
6 def mode(labels):
7     """Calcul du mode des k étiquettes"""
8     return Counter(labels).most_common(1)[0][0]
9 def euclidean_distance(point1, point2):
10    """Calcul de la distance euclidienne"""
11    sum_squared_distance = 0
12    for i in range(len(point1)):
13        sum_squared_distance += math.pow(point1[i]-point2[i], 2)
14    return math.sqrt(sum_squared_distance)

```

Tp 27

KNN: k Nearest Neighbors

```

16 def knn(data, query, k, distance_utilisee, choix_knn_regress_ou_classif):
17     """Algorithme des k plus proches voisins KNN"""
18     distances_et_positions = []
19     for index, example in enumerate(data):
20         distance = distance_utilisee(example[:-1], query)
21         distances_et_positions.append((distance, index))
22     #ici une liste de tuples sera triée selon les distances
23     sorted_distances_et_positions = sorted(distances_et_positions)
24     k_proches_distances_and_indices = sorted_distances_et_positions[:k]
25     k_proches_classes = []
26     for dist_indice in k_proches_distances_and_indices:
27         num=dist_indice[1]; k_proches_classes+=[data[num][1]]
28     # 7. Si régression (choix_knn_regress_ou_classif = mean),
29     # retourner la moyenne des k étiquettes
30     # 8. Si classification, retourner le mode des k étiquettes
31     return k_proches_distances_and_indices ,\
32     choix_knn_regress_ou_classif(k_proches_classes),k_proches_classes

```

Tp 27

KNN: k Nearest Neighbors

```

1 # Données-régression Colonne 0:taille (en cm)Colonne 1: poids (en kg)
2 from numpy import *
3 taille_poids =array([[ 167.66, 112.99],[ 182.38, 136.49],
4 [ 176.97, 153.03],[ 173.96, 142.34],[ 172.86, 144.3 ],
5 [ 175.19, 123.3 ],[ 177.99, 141.49],[ 178.53, 136.46],
6 [ 173.15, 112.37],[ 169.55, 127.45]])
7 taille_en_cm=[161]
8 k=3; distance_utilisee=euclidean_distance; choix_knn=mean
9 R= knn(taille_poids, taille_en_cm, k, distance_utilisee, choix_knn)
10 knn_dist_et_indices, prediction_de_poids, knn_labels=R
11 print('Knn implementé pour regression:')
12 print(knn_dist_et_indices,'\\n',prediction_de_poids,'\\n',knn_labels)
13 from sklearn import neighbors
14 knn_regression = neighbors.KNeighborsRegressor(k)
15 X=[[e] for e in taille_poids[:,0]]; Y=[e for e in taille_poids[:,1]]
16 knn_regression.fit(X,Y)
17 print('Knn predefinie-regression:\\n',knn_regression.predict([taille_en_cm]))

```

Knn implementé pour regression:
[(6.65999999999997, 0), (8.55000000000011, 9), (11.86000000000014, 4)]
128.24666666666667
[112.99, 127.45, 144.3]
Knn predefinie-regression:
[128.246666667]

Tp 27

KNN: k Nearest Neighbors

```

1 # Données de Classification, Colonne 0: age Colonne 1: aime jeux?
2 age_aimer_jeux_electroniques = array([[22,1],[23,1],[21,1],[18,1],[19,1],
3 [25,0],[27,0],[29,0],[31,0],[45, 0]])
4 #age 33 ans aime-t-elle les jeux électroniques
5 age = [33]; k=5; distance_utilisee=euclidean_distance; choix_knn=mode
6 R= knn(age_aimer_jeux_electroniques, age, k, distance_utilisee, choix_knn)
7 knn_dist_et_indices, prediction_de_aimer_jeux, knn_labels=R
8 print('résultat knn implémenté pour la classification:')
9 print(knn_dist_et_indices,'\\n',prediction_de_aimer_jeux,'\\n', knn_labels)
10 from sklearn.neighbors import KNeighborsClassifier
11 neigh = KNeighborsClassifier(n_neighbors=k,metric="euclidean")
12 X=[[e] for e in age_aimer_jeux_electroniques[:,0]];
13 Y=[e for e in age_aimer_jeux_electroniques[:,1]]
14 neigh.fit(X,Y)
15 print('résultat knn prédefinie pour la classification:')
16 print(neigh.predict([age]))

```

résultat knn implémenté pour la classification:
[(2.0, 8), (4.0, 7), (6.0, 6), (8.0, 5), (10.0, 1)]
0
[0, 0, 0, 0, 1]
résultat knn prédefinie pour la classification:
[0]

Tp 27

KNN: k Nearest Neighbors

```

2 from sklearn.neighbors import KNeighborsClassifier
3 from sklearn.model_selection import GridSearchCV,train_test_split
4 data=pandas.read_csv("d:\\images_chiffres_codees_niveau_de_gris.csv")
5 data=np.array(data); x=data[0:1000,1:]; y=data[0:1000,0];
6 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,
7 random_state=0)
8 model=KNeighborsClassifier(n_neighbors=5,metric='manhattan')
9 model.fit(x_train,y_train); score=model.score(x_test, y_test)
10 print("score sur test-set par k=5 --> "+str(score*100)+"%")
11 distances=['euclidean','manhattan'] ; valeurs_de_k=np.arange(1,8)
12 parametres_grid={"n_neighbors":valeurs_de_k, 'metric':distances}
13 model=KNeighborsClassifier()
14 grid=GridSearchCV(model,parametres_grid, cv=3,scoring='accuracy')
15 grid.fit(x_train,y_train); print("best parameters:",grid.best_params_)
16 model=grid.best_estimator_ ; model.fit(x_train,y_train)
17 score=model.score(x_test, y_test)
18 print("score sur test-set-paramètres automatiques="+str(score*100)+"%")

```

score sur test-set par k=5 --> 85.0%
 best parameters: {'metric': 'euclidean', 'n_neighbors': 5}
 score sur test-set-paramètres automatiques=88.0%

Tp 28

KNN: k Nearest Neighbors

Classification de chats et de chiens à partir de photos par KNN



Tp 29

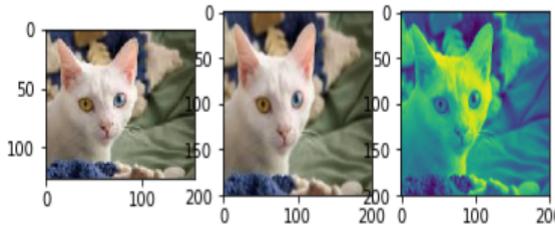
Taux de Classification de chats et de chiens non satisfaisant ! Solution ?

KNN: k Nearest Neighbors

```

1 import numpy as np ; import pandas as pd
2 from skimage import io; cat1 = io.imread('D:\\cat1.jpg')
3 from skimage.transform import resize; cat1_ = resize(cat1, (200,200,3))
4 from skimage.color import rgb2gray; cat1_gs = rgb2gray(cat1_)
5 import matplotlib.pyplot as plt; fig=plt.figure()
6 columns = 3; rows = 1; fig.add_subplot(rows, columns, 1);plt.imshow(cat1)
7 fig.add_subplot(rows, columns, 2); plt.imshow(cat1_)
8 fig.add_subplot(rows, columns, 3); plt.imshow(cat1_gs); plt.show()

```



Tp 29

KNN: k Nearest Neighbors

```

1 # L'ensemble des données d'apprentissage contient 4000 photos de cat et 4000 de dog
2 # L'ensemble des données de test contient 1000 photos de cat et 1000 de dog
3 #(indice commence de 4001 pour les images de test)
4 x_train = []; y_train = []
5 for i in range(1,2001):
6     cat = rgb2gray(resize(io.imread('D:/cat-and-dog/training_set/+'\
7         'training_set/cats/cat.{}.jpg'.format(i)), (200,200)))
8     x_train.append(cat);      y_train.append(0)
9 for i in range(1,2001):
10    dog = rgb2gray(resize(io.imread('D:/cat-and-dog/training_set/+'\
11        'training_set/dogs/dog.{}.jpg'.format(i)), (200,200)))
12    x_train.append(dog);      y_train.append(1)
13 x_train, y_train = np.asarray(x_train), np.asarray(y_train)
14 print('x_train shape: ', x_train.shape, 'y_train shape: ', y_train.shape)

```

x_train shape: (4000, 200, 200) y_train shape: (4000,)

Tp 29

KNN: k Nearest Neighbors

```

1 x_test = []
2 y_test = []
3 for i in range(4001,5001):
4     cat = rgb2gray(resize(io.imread('D:/cat-and-dog/test_set/test_set/+'\ 
5         'cats/cat.{}.jpg'.format(i)), (200,200)))
6     x_test.append(cat);      y_test.append(0)
7 for i in range(4001,5001):
8     dog = rgb2gray(resize(io.imread('D:/cat-and-dog/test_set/test_set/+'\ 
9         'dogs/dog.{}.jpg'.format(i)), (200,200)))
10    x_test.append(dog) ;   y_test.append(1)
11 x_test, y_test = np.asarray(x_test), np.asarray(y_test)
12 print('x_test shape: ', x_test.shape, 'y_test shape: ', y_test.shape)

```

x_test shape: (2000, 200, 200) y_test shape: (2000,)

Tp 29

KNN: k Nearest Neighbors

```

1 from numpy import *
2 distances=[0.5,8,1.2,4,5.6,2,1.25,3.01,1.05,0.36]
3 k=3
4 L=np.argsort(distances)
5 print('Les positions des '+str(k)+' valeurs minimales:',L[:k])
6 Y1=array([0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1])
7 counts1 = np.bincount(Y1); print('np.bincount(Y1): ', counts1)
8 print('np.argmax(counts1)=',np.argmax(counts1))
9 Y2=array([0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1])
10 counts2 = np.bincount(Y2); print('np.bincount(Y2): ',counts2)
11 print('np.argmax(counts2)=',np.argmax(counts2))

```

Les positions des 3 valeurs minimales: [9 0 8]
 np.bincount(Y1): [11 4]
 np.argmax(counts1)= 0
 np.bincount(Y2): [3 12]
 np.argmax(counts2)= 1

Tp 29

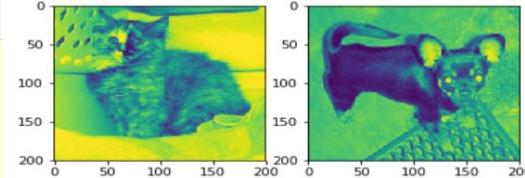
KNN: k Nearest Neighbors

```

def predict(X, k):
    distances=[]
    for i in range(0, len(x_train)):
        distances+= [np.sum(np.abs(x_train[i] - X))]
    min_indexs = np.argsort(distances)[:k]
    y_ = y_train[min_indexs]; counts = np.bincount(y_)
    if np.argmax(counts)==0: return('cat')
    else: return ('dog')

numeros_imagenes_a_preditore=[4,1089]#914#1129
fig=plt.figure(); predictions=[]
columns =2; rows = 1; i=1;
for num in numeros_imagenes_a_preditore:
    predictions+=[predict(x_test[num],3)]
    fig.add_subplot(rows, columns, i); plt.imshow(x_test[num]); i+=1
plt.show()
print(predictions)

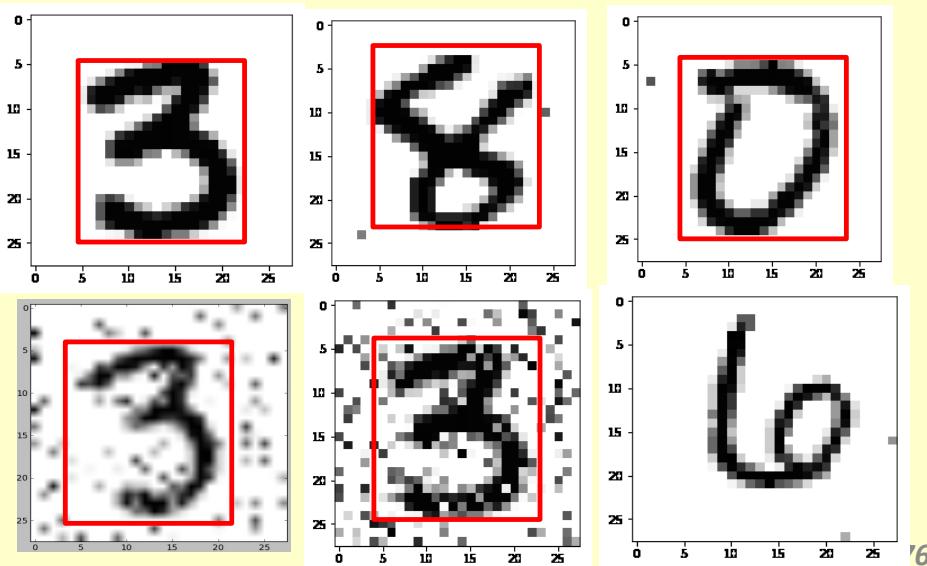
```



['cat', 'dog']

Tp 29

feature selection



feature selection

La sélection de caractéristique (ou sélection d'attribut ou de variable) est un processus utilisé en apprentissage automatique et en traitement de données. Il consiste, étant donné des données dans un espace de grande dimension, à trouver un sous-ensemble de variables pertinentes. C'est-à-dire que l'on cherche à minimiser la perte d'information venant de la suppression de toutes les autres variables. C'est une méthode de réduction de la dimensionnalité.

La sélection des features a un impact énorme sur les performances du modèle ML. Les features que vous utilisez pour former vos modèles d'apprentissage automatique ont une énorme influence sur les performances que vous pouvez obtenir.

177

feature selection

- Des caractéristiques non pertinentes ou partiellement pertinentes peuvent avoir un impact négatif sur les performances du modèle.
- La sélection des fonctionnalités et le nettoyage des données doivent être la première et la plus importante étape de la conception de votre modèle.
- La sélection des features est le processus dans lequel vous sélectionnez automatiquement ou manuellement les fonctionnalités qui contribuent le plus à votre variable de prédiction ou à la sortie qui vous intéresse.
- La présence de fonctionnalités non pertinentes dans vos données peut réduire la précision des modèles et faire en sorte que votre modèle apprenne en fonction de fonctionnalités non pertinentes.

178

feature selection

Quels sont les avantages d'effectuer une sélection de fonctionnalités avant de modéliser vos données?

- Réduit le sur-ajustement: moins de données redondantes signifie moins d'opportunité de prendre des décisions basées sur le bruit.
- Améliore la précision: moins de données trompeuses signifie que la précision de la modélisation s'améliore.
- Réduit le temps de formation: moins de points de données réduisent la complexité des algorithmes et les algorithmes s'entraînent plus rapidement.

179

feature selection

```

1 from numpy import *
2 X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
3 from sklearn.feature_selection import VarianceThreshold
4 Filtre= VarianceThreshold(threshold=0.2)
5 Donnees_filtrees=Filtre.fit_transform(X)
6 print(array(X))
7 print(Donnees_filtrees)
8 print(var(X, axis=0))

```

```

[[0 0 1]      [[0 1]
 [0 1 0]      [1 0]
 [1 0 0]      [0 0]      [0.13888889 0.22222222 0.25      ]
 [0 1 1]      [1 1]
 [0 1 0]      [1 0]
 [0 1 1]]     [1 1]

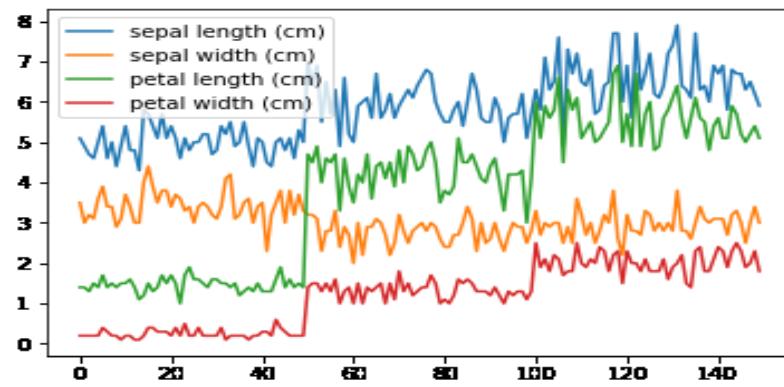
```

Tp 30

180

feature selection

```
from sklearn.feature_selection import VarianceThreshold
from sklearn.datasets import load_iris
iris = load_iris (); X=iris.data ; Y=iris.target
import matplotlib.pyplot as plt
plt.plot(X); plt.legend(iris.feature_names); plt.show()
```

181

feature selection

```
1 X.var(axis=0)
```

array([0.68112222, 0.18871289, 3.09550267, 0.57713289])

```
1 Flitre= VarianceThreshold(threshold=0.18872)
2 Donnees_filtrees=Flitre.fit_transform(X)
3 print(Donnees_filtrees)
```

```
[[5.1 1.4 0.2]
 [4.9 1.4 0.2]
 [4.7 1.3 0.2]
 [4.6 1.5 0.2]
 [5.  1.4 0.2]
 [5.4 1.7 0.4]
 [4.6 1.4 0.3]
 [5.  1.5 0.2]]
```

Tp 30

182

feature selection

```

1 Flitre.get_support()

array([ True, False,  True,  True])

1 %config IPCompleter.greedy=True

1 Attributs_filtres=array(iris.feature_names)[Flitre.get_support()]
2 print(Attributs_filtres)

['sepal length (cm)' 'petal length (cm)' 'petal width (cm)']

```

Tp 30

[183](#)

feature selection

```

1 from sklearn.feature_selection import chi2, SelectKBest

1 chi2(X,Y)

(array([ 10.81782088,   3.7107283 ,  116.31261309,   67.0483602 ]),
 array([4.47651499e-03,  1.56395980e-01,  5.53397228e-26,  2.75824965e-15]))

1 Filtre2=SelectKBest(chi2,k=1)
2 Donnees_filtrees2=Filtre2.fit_transform(X,Y)
3 print(Filtre2.get_support())
4 Filtre3=SelectKBest(chi2,k=3)
5 Donnees_filtrees3=Filtre3.fit_transform(X,Y)
6 print(Filtre3.get_support())

[False False  True False]
[ True False  True  True]

```

Tp 30

[184](#)

feature selection

```

1 from sklearn.feature_selection import SelectFromModel
2 from sklearn.neighbors import KNeighborsClassifier
3 model1=KNeighborsClassifier()
4 from sklearn.tree import DecisionTreeClassifier
5 model2 = DecisionTreeClassifier()
6 from sklearn.linear_model import LogisticRegression
7 model3=LogisticRegression(C=10e+10)
8 #'mean' median valeur précise
9 selector=SelectFromModel(model2,threshold='mean')
10 selector.fit_transform(X,Y); print(selector.get_support())
11 selector=SelectFromModel(model3,threshold='median')
12 selector.fit_transform(X,Y) ; print(selector.get_support())

```

[False False True True]
 [False False True True]

Tp 30

[185](#)

feature selection

```

from sklearn.feature_selection import VarianceThreshold
def trouver_donnees_importantes(x_train,TR):
    Flitre= VarianceThreshold(threshold=TR)
    Donnees_filtrees=Flitre.fit_transform(x_train)
    FS=list(Flitre.get_support()) ; #print(FS)
    X=[]
    for i in range(0,len(FS)):
        if FS[i]==True:
            X+=[i]
    return tuple(X)

```

Tp 31

[186](#)

feature selection

```

import numpy as np ; import pandas; import matplotlib.pyplot as plt
from sklearn.metrics import *
import numpy as np; import pandas ; from sklearn.metrics import *
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(C=10e+10)
data=pandas.read_csv("d:\\images_chiffres_codees_niveau_de_gris.csv")
data=np.array(data); print(data.shape)
X_train, X_test, y_train, y_test = train_test_split(
    data[:,1:], data[:,0], test_size=0.20, random_state=0)
model.fit(X_train,y_train)
score=model.score(X_test, y_test); print ("Accuracy avant: ",score*100, '%')
Attributs_importants=trouver_donnees_importantes(X_train,200);
x=X_train[:,Attributs_importants]; label=data[0:2000,0];
print(x.shape); model.fit(x,y_train); xtest=X_test[:,Attributs_importants];
score=model.score(xtest, y_test); print ("Accuracy après: ",score*100, '%')

(2559, 785)
Accuracy avant:  85.15625 %
(2047, 495)
Accuracy après:  85.546875 %

```

Tp 31

[187](#)

feature selection

```

1 from sklearn.feature_selection import chi2, SelectKBest
2 def trouver_donnees_importantes(X,Y,K_best):
3     selector=SelectKBest(chi2,k=K_best)
4     Donnees_filtrees=selector.fit_transform(X,Y)
5     FS=list(selector.get_support()); #print(FS)
6     X=[]
7     for i in range(0,len(FS)):
8         if FS[i]==True:
9             X+=[i]
10    return tuple(X)

```

Tp 32

[188](#)

feature selection

```

import numpy as np ; import pandas; import matplotlib.pyplot as plt
from sklearn.metrics import *
import numpy as np; import pandas ; from sklearn.metrics import *
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(C=10e+10)
data=pandas.read_csv("d:\\images_chiffres_codees_niveau_de_gris.csv")
data=np.array(data); print(data.shape)
X_train, X_test, y_train, y_test = train_test_split(
    data[:,1:], data[:,0], test_size=0.20, random_state=0)
model.fit(X_train,y_train)
score=model.score(X_test, y_test); print ("Accuracy avant: ",score*100,'%')
K=int(X_train.shape[1]*0.75)
Attributs_importantes=trouver_donnees_importantes(X_train,y_train,K);
x=X_train[:,Attributs_importantes]; label=data[0:2000,0];
print(x.shape); model.fit(x,y_train); xtest=X_test[:,Attributs_importantes];
score=model.score(xtest, y_test); print ("Accuracy après: ",score*100,'%')

```

Tp 32

[189](#)

feature selection

- SelectFromModel entraîne un estimateur (Arbre de décision, Régression logistique,) puis sélectionne les variables (features) les plus importantes pour cet estimateur.
- SelectFromModel est compatible avec les estimateurs qui développent une fonction paramétrée (le vecteur theta pour la régression logistique par exemple) qui possèdent l'attribut .coef_
- SelectFromModel est incompatible avec l'estimateur KNN puisque il ne développe pas une fonction paramétrée

[190](#)

feature selection

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectFromModel
from numpy import *
def trouver_donnees_importantes(X_d,Y_d,TR):
    model=DecisionTreeClassifier()
    selector=SelectFromModel(model,threshold=TR)
    Donnees_filtrees=selector.fit_transform(X_d,Y_d)
    FS=list(selector.get_support()) ; #print(FS)
    X=[]
    for i in range(0,len(FS)):
        if FS[i]==True:
            X+=[i]
    return tuple(X)

```

Tp 33

[191](#)

feature selection

```

1 import numpy as np; import pandas ; from sklearn.metrics import *
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.model_selection import train_test_split
4 model= DecisionTreeClassifier()
5 data=pandas.read_csv("d:\\images_chiffres_codees_niveau_de_gris.csv")
6 data=np.array(data); print(data.shape)
7 X_train, X_test, y_train, y_test = train_test_split(\n     data[:,1:], data[:,0], test_size=0.20, random_state=0)
8 model.fit(X_train,y_train)
9 score=model.score(X_test, y_test); print ("Accuracy avant: ",score*100,'%')
10 Attributs_importants=trouver_donnees_importantes(X_train,y_train,0.00001);
11 x=X_train[:,Attributs_importants]; label=data[0:2000,0];
12 print(x.shape); model.fit(x,y_train); xtest=X_test[:,Attributs_importants];
13 score=model.score(xtest, y_test); print ("Accuracy après: ",score*100,'%')

```

(2559, 785)
Accuracy avant: 69.7265625 %
(2047, 215)
Accuracy après: 71.2890625 %

Tp 33

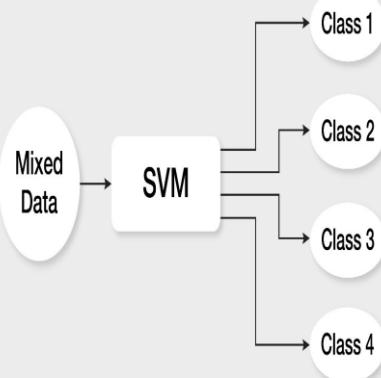
[192](#)

SVM : Support Vector Machine

Support Vector Machine (SVM) est un modèle linéaire pour les problèmes de classification et de régression. Les SVM sont appelées les séparateurs à vaste marge.

Il peut résoudre des problèmes linéaires et non linéaires et fonctionne bien pour de nombreux problèmes pratiques.

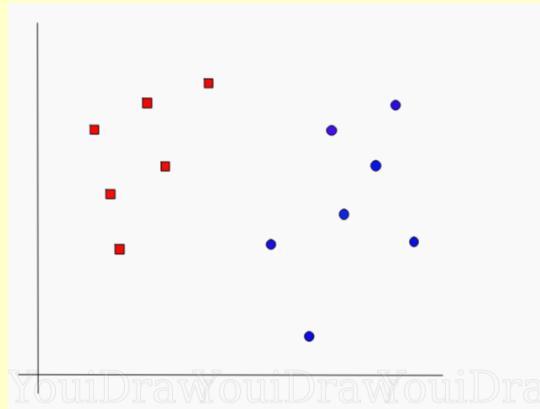
L'idée de SVM est simple: l'algorithme crée une ligne ou un hyperplan qui sépare les données en classes



193

SVM : Support Vector Machine

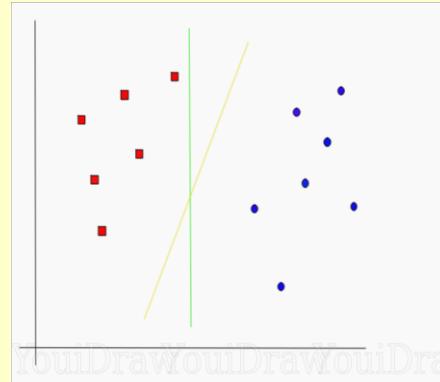
Problème de classification



Trouver une ligne / un hyperplan idéal qui sépare cet ensemble de données en catégories rouge et bleue. En fait, nous avons une infinité de lignes qui peuvent séparer ces deux classes. Alors, comment SVM trouve-t-il l'idéal ?

SVM : Support Vector Machine

Comment SVM trouve-t-il l'idéal ?



Il est visuellement assez intuitif dans ce cas que la ligne jaune soit mieux adapté pour la classification.

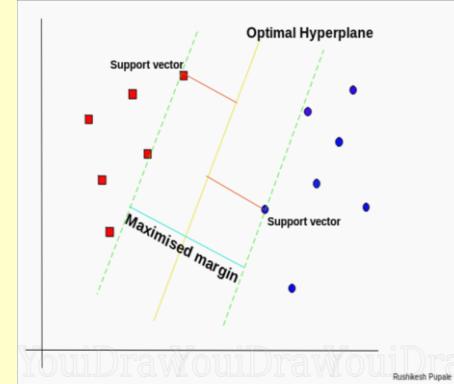
SVM : Support Vector Machine

Comment SVM trouve-t-il l'idéal ?

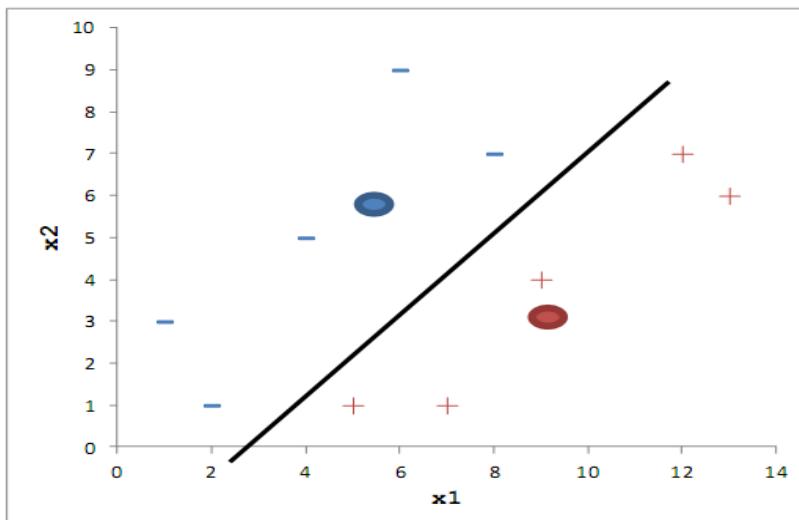
Selon l'algorithme SVM, nous trouvons les points les plus proches de la ligne à partir des deux classes.

Ces points sont appelés vecteurs de support. nous calculons la distance entre la ligne et les vecteurs de support. Cette distance s'appelle la marge.

L'objectif est de maximiser la marge. L'hyperplan pour lequel la marge est maximale est l'hyperplan optimal.



SVM : Support Vector Machine



197

SVM : Support Vector Machine

les données d'apprentissage sont linéairement séparables, c'est à dire qu'il existe un hyperplan qui sépare les données sans erreur.

Soit x un vecteur qui représente un échantillon des données d'apprentissage:

$$f(x) = x^T \cdot \beta + \beta_0 = x_1 \cdot \beta_1 + x_2 \cdot \beta_2 + \dots + x_p \cdot \beta_p + \beta_0$$

L'objectif est de trouver le vecteur β tel que:

$f(x) = 0$, on est sur la frontière

$f(x) > 0$, on classe « + »

$f(x) < 0$, on classe « - »

$f(x) = \pm 1$ on est sur les droites délimitant des vecteurs de support

SVM : Support Vector Machine

Hyperplan optimal doit maximiser la distance entre la frontière de séparation et les points de chaque classe qui lui sont les plus proche.

Distance d'un échantillon quelconque x avec la frontière:

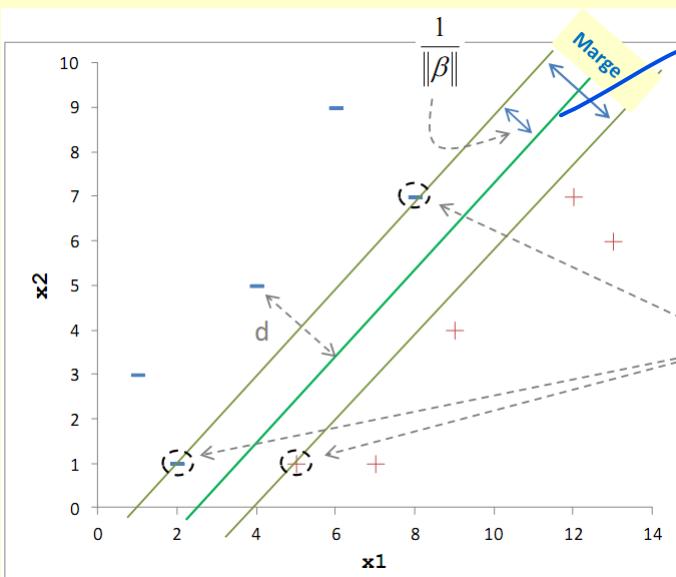
$$d = \frac{|x^T \beta + \beta_0|}{\|\beta\|} \quad \text{avec} \quad \|\beta\| = \sqrt{\beta_1^2 + \dots + \beta_p^2}$$

La marge maximale est égale à

$$\delta = \frac{2}{\|\beta\|}$$

199

SVM : Support Vector Machine



200

SVM : Support Vector Machine

Maximiser la marge revient à minimiser la norme du vecteur de paramètres β

$$\max \frac{2}{\|\beta\|} \Leftrightarrow \min \|\beta\|$$

Les contraintes indiquent que tous les points sont du bon côté, au pire ils sont sur la droite définissant les vecteurs de support.

$$\min_{\beta, \beta_0} \|\beta\| \text{ sous contrainte :}$$

$$y_i \times f(x_i) \geq 1, i = 1, \dots, n$$

201

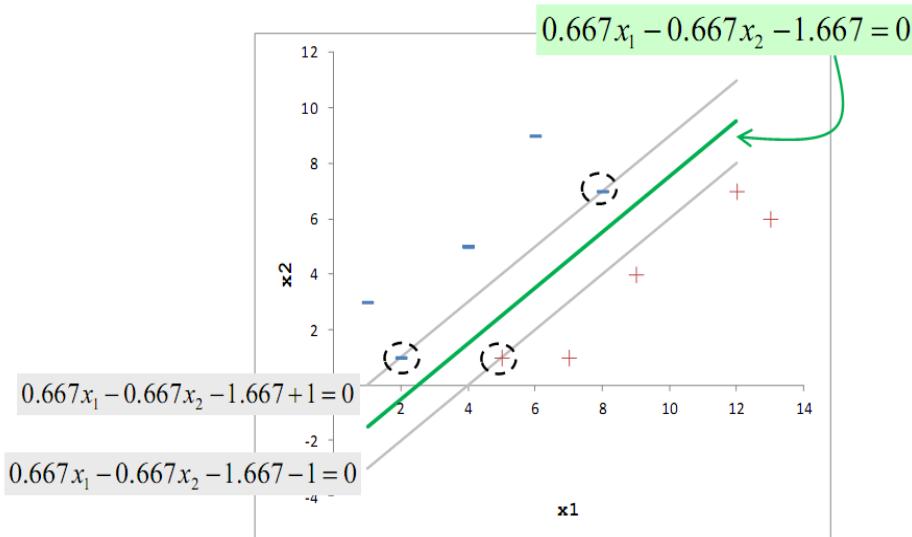
SVM : Support Vector Machine

	beta.1 0.667	beta.2 -0.667	beta.0 -1.667	f(x)	f(x)*y
n°	x1	x2	y		
1	1	3	-1	-3	3
2	2	1	-1	-1	1
3	4	5	-1	-2.333333333	2.333333333
4	6	9	-1	-3.666666667	3.666666667
5	8	7	-1	-1	1
6	5	1	1	1	1
7	7	1	1	2.333333333	2.333333333
8	9	4	1	1.666666667	1.666666667
9	12	7	1	1.666666667	1.666666667
10	13	6	1	3	3

Tp 34

202

SVM : Support Vector Machine



203

SVM : Support Vector Machine

Règle de classement pour un individu i^* basé sur les coefficients estimés $\hat{\beta}_j$

$$f(x_{i^*}) \begin{cases} \geq 0 \Rightarrow \hat{y}_{i^*} = 1 \\ < 0 \Rightarrow \hat{y}_{i^*} = -1 \end{cases}$$

beta.1	beta.2	beta.0	f(x)	prediction
0.667	-0.667	-1.667		
1	3	-1	-3	-1
2	1	-1	-1	-1
4	5	-1	-2.3333	-1
6	9	-1	-3.6667	-1
8	7	-1	-1	-1
5	1	1	1	1
7	1	1	2.33333	1
9	4	1	16.6667	1
12	7	1	16.6667	1
13	6	1	3	1

Tp 34 204

SVM : Support Vector Machine

```

1 import pandas as pd; from numpy import *
2 import matplotlib.pyplot as plt
3 from sklearn.metrics import confusion_matrix
4 donnees= pd.read_csv("D:\\pointsSVM.csv",sep=',')
5 donnees=array(donnees); x=donnees[:,0:2]; y=donnees[:,2];
6 axes = plt.axes(); axes.grid()
7 plt.scatter(x[y ==+1][:, 0], x[y ==+1][:, 1], color='r')
8 plt.scatter(x[y ==-1][:, 0], x[y == -1][:, 1], color='b')
9 def f(X,Beta):    return Beta[0]+sum(X.T*Beta[1:])
10 def classer(X,Beta):   return int(sign(f(X,Beta)))
11 from sklearn.svm import LinearSVC, SVC
12 lsvm = LinearSVC(C=10000); X_train=donnees[:,0:2]; y_train=donnees[:,2]
13 lsvm.fit(X_train, y_train); score = lsvm.score(X_train, y_train)
14 print(score); Beta=list(lsvm.intercept_)+list(lsvm.coef_[0]); print(Beta)
15 plt.show()

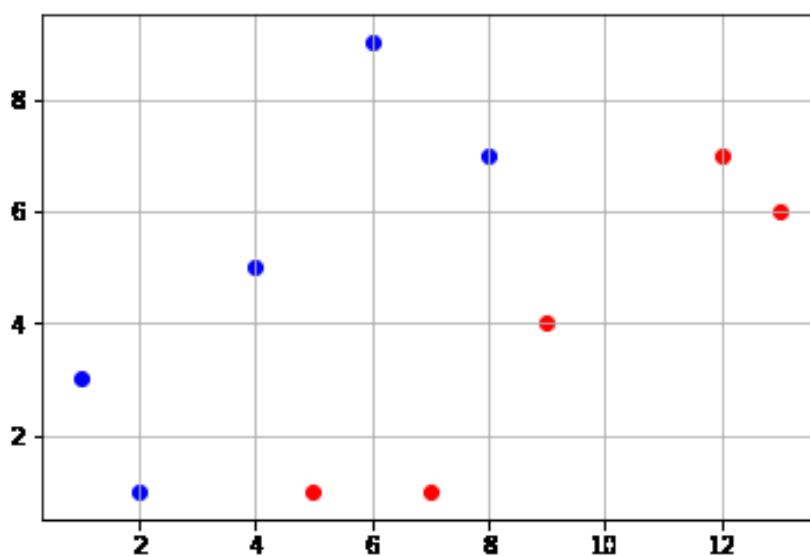
```

1.0

[-1.555381952653455, 0.6666117470826681, -0.7777181372481087]

Tp 34 [205](#)

SVM : Support Vector Machine

[206](#)

SVM : Support Vector Machine

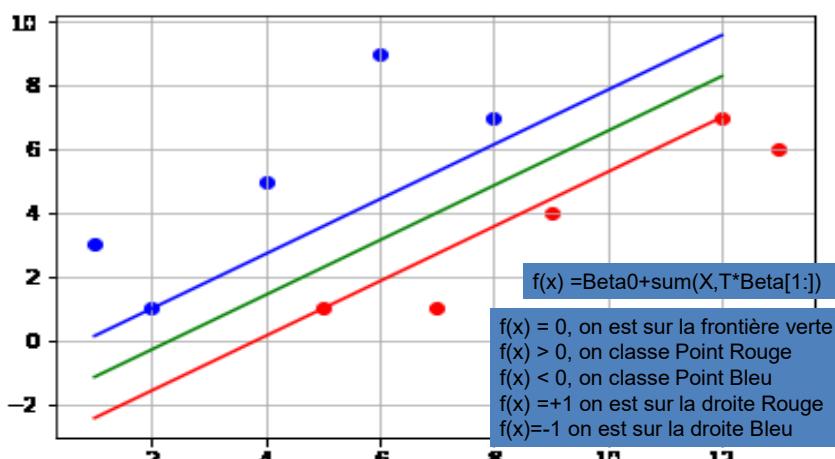
```

1 axes = plt.axes(); axes.grid()
2 plt.scatter(x[y ==+1][:, 0], x[y ==+1][:, 1], color='r')
3 plt.scatter(x[y ==-1][:, 0], x[y == -1][:, 1], color='b')
4 Bleu=[]; Rouge=[]; milieu=[]
5 for abscisseX in range(1,13):
6     y=-1.0*(Beta[0]+Beta[1]*abscisseX+1)/Beta[2]; Bleu+=[y]
7     y=-1.0*(Beta[0]+Beta[1]*abscisseX-1)/Beta[2]; Rouge+=[y]
8     y=-1.0*(Beta[0]+Beta[1]*abscisseX)/Beta[2]; milieu+=[y]
9 plt.plot(arange(1,13),Bleu,color='b')
10 plt.plot(arange(1,13),Rouge,color='r')
11 plt.plot(arange(1,13),milieu,color='g')
12 plt.show()
13 print(Beta)
14 print('[5,1]==>',f(array([5,1]),Beta));print('[2,1]',f(array([2,1]),Beta))
15 print('[6,9]==>',f(array([6,9]),Beta));print('[12,2]',f(array([12,1]),Beta))

```

Tp 34 [207](#)

SVM : Support Vector Machine



```

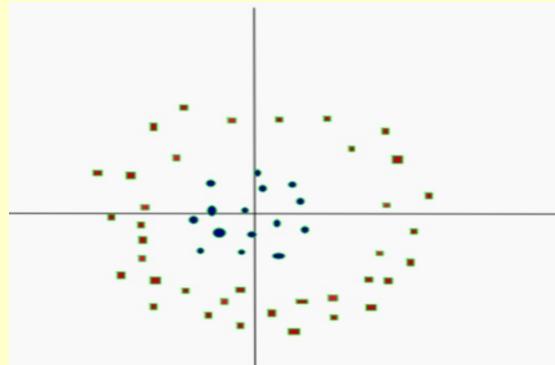
[-1.5553667185086004, 0.6666016592201605, -0.7776942389624746]
[5,1]==> 0.9999473386297271
[2,1] -0.9998576390307541
[6,9]==> -4.55500491384991
[12,2] 5.666158953170851

```

Tp 35 [208](#)

SVM : Support Vector Machine

Cas non séparable



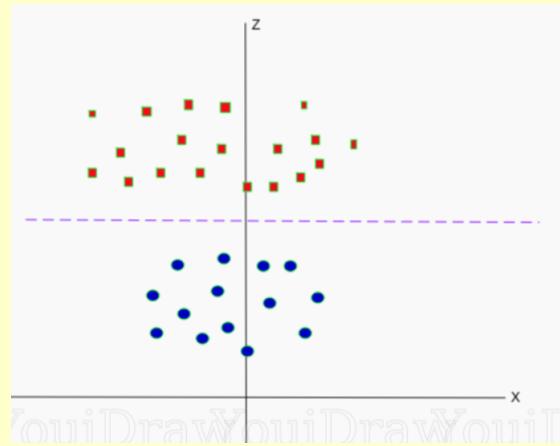
Considérons un ensemble de données peu complexe.

Nous ne pouvons pas tracer une ligne droite permettant de classer ces données.

Ces données peuvent être converties en données séparables linéairement dans une dimension supérieure.

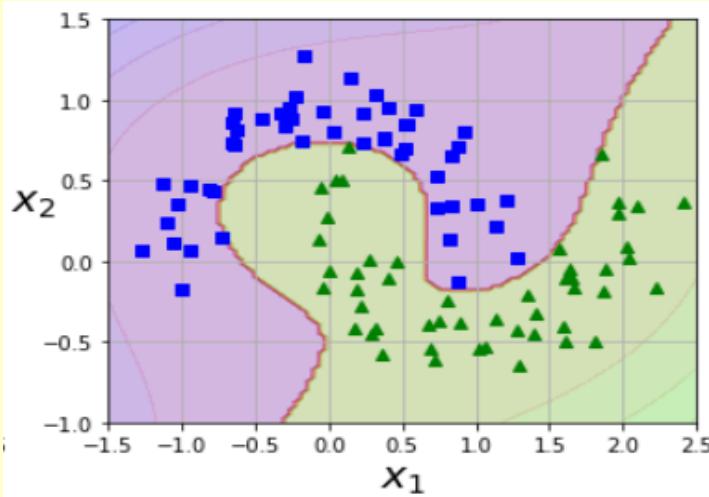
SVM : Support Vector Machine

Cas non séparable



La coordonnée z est le carré de la distance du point à l'origine.

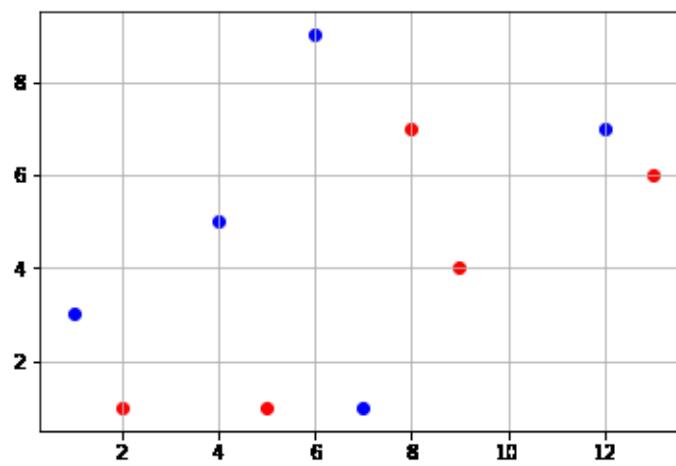
SVM : Support Vector Machine



LinearSVC peut-il trouver une droite de séparation ?
Non !

211

SVM : Support Vector Machine

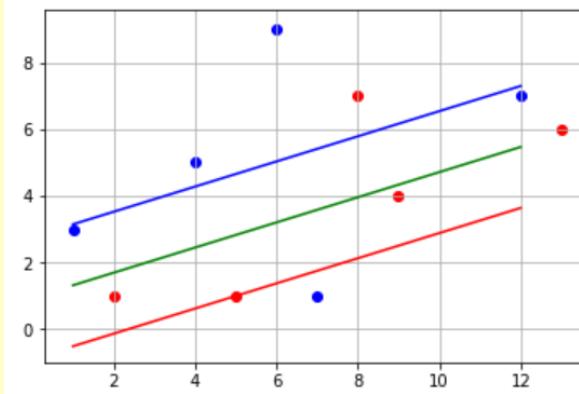


LinearSVC peut-il trouver une droite de séparation ?
Non !

Tp 35 212

SVM : Support Vector Machine

Avec LinearSVC(C=1000) dans le programme du Tp précédent on obtient un taux de 70% sur les données d'entraînement !



[0.5139860893703938, 0.20624078753074881, -0.5465213011016463]

LinearSVC peut-il trouver une droite de séparation ? Non !

Tp 35 [213](#)

SVM : Support Vector Machine

```

1 import pandas as pd; from numpy import *
2 import matplotlib.pyplot as plt
3 donnees= pd.read_csv("D:\\pointsSVM2.csv",sep=';')
4 donnees=array(donnees); x=donnees[:,0:2]; y=donnees[:,2];
5 axes = plt.axes(); axes.grid()
6 plt.scatter(x[y ==+1][:, 0], x[y ==+1][:, 1], color='r')
7 plt.scatter(x[y ==-1][:, 0], x[y == -1][:, 1], color='b')
8 def f(X,Beta):    return Beta[0]+sum(X.T*Beta[1:])
9 def classer(X,Beta):    return int(sign(f(X,Beta)))
10 from sklearn.svm import LinearSVC, SVC,NuSVC
11 X_tr=donnees[:,0:2]; y_tr=donnees[:,2]
12 lsvm= SVC(gamma='auto');lsvm.fit(X_tr, y_tr);
13 score = lsvm.score(X_tr, y_tr); print(score);
14 lsNuSVC= NuSVC(gamma='auto');lsNuSVC.fit(X_tr, y_tr);
15 score = lsNuSVC.score(X_tr, y_tr); print(score);
16 lsvm= LinearSVC(C=1000);lsvm.fit(X_tr, y_tr);
17 score = lsvm.score(X_tr, y_tr); print(score);
18 Beta=list(lsvm.intercept_)+list(lsvm.coef_[0]); print(Beta)
19 plt.show()
```

1.0

1.0

0.7

[0.6133585001959307, 0.366932764644063, -0.39293597967278737]

Tp 35

[214](#)

SVM : Support Vector Machine

Le SVM module (SVC, NuSVC, etc) prend en charge les différents noyaux, tandis que LinearSVC prend en charge uniquement un noyau linéaire.

SVC(kernel = 'linear') est "équivalent" à: LinearSVC()

LinearSVC prend uniquement en charge un noyau linéaire, est plus rapide et peut beaucoup mieux.

SVC et nuSVC sont mathématiquement équivalents. La principale différence est que SVC utilise le paramètre C tandis que nuSVC utilise le paramètre nu.

215

SVM : Support Vector Machine

Plusieurs méthodes ont été proposées pour étendre le schéma ci-dessus au cas où on a M classes sont à séparer ($M \geq 3$).

Les deux les plus connues sont:

❖ **One vs all:** consiste à construire M classifieurs binaires en attribuant le label 1 aux échantillons de l'une des classes et le label -1 à toutes les autres. Problème n'est plus équilibré, par exemple avec $M=10$, on utilise seulement 10 % d'exemples positifs pour 90 % d'exemples négatifs.

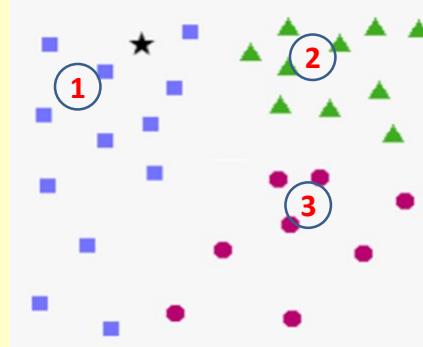
❖ **One vs one:** consiste à construire $M(M-1)/2$ classifieurs binaires en confrontant chaque couple de 2 classes différents.

SVM : Support Vector Machine

Problématique

Supposons que je veuille entraîner un SVM à distinguer trois classes C1, C2 et C3.
L'étoile noire est-elle de genre C1, C2 ou C3?

D'après la figure, notre nouvel échantillon (l'étoile) portera l'étiquette 1



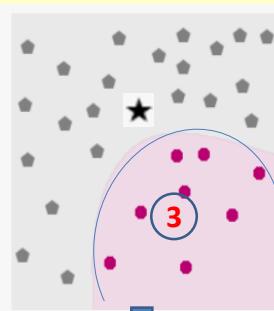
217

SVM : Support Vector Machine

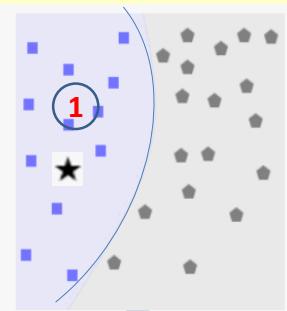
One vs all



**Est-ce que c'est
un « 2 » ➔ Non**



**Est-ce que c'est
un « 3 » ➔ Non**

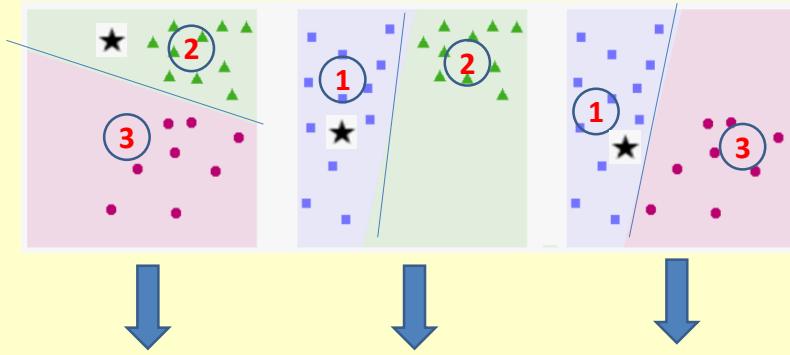


**Est-ce que c'est
un « 1 » ➔ oui**

218

SVM : Support Vector Machine

One vs one



Vote sur 2

Vote sur 1

Vote sur 1

Par votes majoritaires: le nouvel échantillon sera classé avec l'étiquette 1.

219

SVM : Support Vector Machine

• Régularisation /C

C désigne un paramètre de pénalité.

- ✓ Avec une valeur C petite, on obtient un hyperplan de faible marge
- ✓ Avec une valeur C supérieure, on obtient un hyperplan de valeur supérieure.

220

SVM : Support Vector Machine

- Gamma

- ✓ Avec une valeur petite, Gamma créera un ajustement lâche du jeu de données d'apprentissage. Une valeur faible de gamma ne prend en compte que les points proches pour le calcul d'un plan séparé.
- ✓ Avec une valeur gamma élevée permettra au modèle de s'adapter de manière plus appropriée. La valeur élevée de gamma prend en compte tous les points de données pour calculer la ligne de séparation finale.

221

SVM (Sentiment analysis case study)

```

1 with open("D:\\reviews.txt") as f:
2     reviews = f.read().split("\n")
3 with open("D:\\labels.txt") as f:
4     labels = f.read().split("\n")
5 reviews_tokens = [review.split() for review in reviews]
6 from sklearn.preprocessing import MultiLabelBinarizer
7 MLB = MultiLabelBinarizer()
8 MLB.fit(reviews_tokens)
9 from sklearn.model_selection import train_test_split
10 X_train, X_test, y_train, y_test = train_test_split(
11     reviews_tokens, labels, test_size=0.25, random_state=None)
12 from sklearn.svm import LinearSVC
13 lsvm = LinearSVC(); lsvm.fit(MLB.transform(X_train), y_train)
14 score = lsvm.score(MLB.transform(X_test), y_test); print(score*100, '%')

```

85.44 %

Tp 36

Ensemble learning

Méthodes d'agrégation

Les méthodes ensemblistes (ou d'agrégation) pour les algorithmes d'apprentissage statistique (en anglais : *ensemble learning*) sont basées sur l'idée de combiner les prédictions de plusieurs prédicteurs (ou classifieurs) pour une meilleure généralisation et pour compenser les défauts éventuels de prédicteurs individuels.

En général, on distingue deux familles de méthodes de ce type :

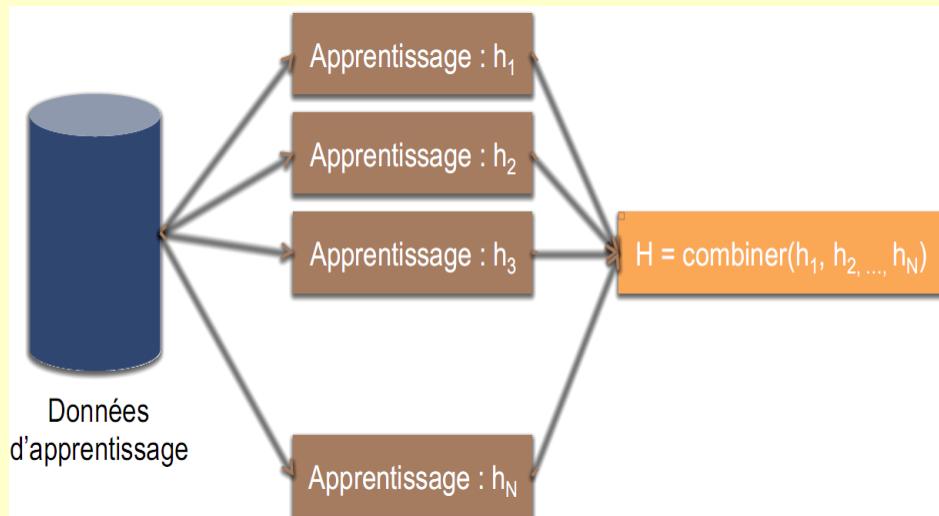
1. Méthodes adaptatives (*boosting*) où les paramètres sont itérativement adaptés pour produire un meilleur mélange.

2. Méthodes par moyennage (*bagging*, forêts aléatoires) où le principe est de faire la moyenne de plusieurs prédictions en espérant un meilleur résultat suite à la réduction de variance de l'estimateur moyenne.

223

Ensemble learning

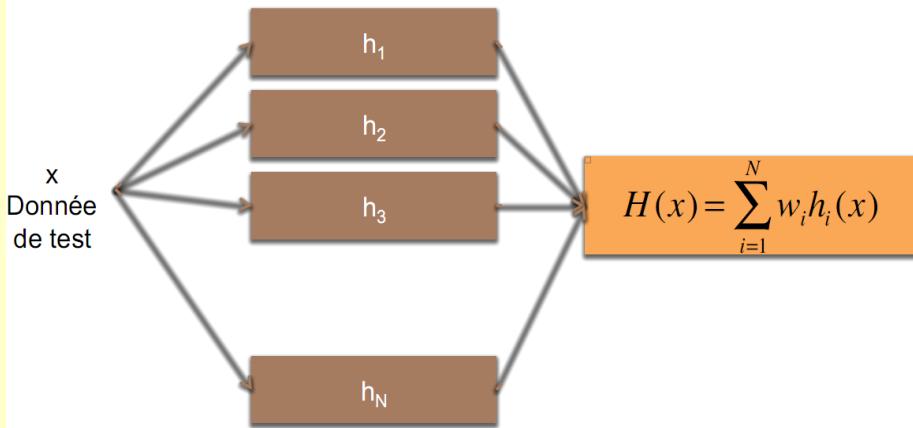
Principe général : apprentissage



224

Ensemble learning

Principe général : décision



- Comment générer les prédicteurs h_i ?
- Comment les combiner ?

225

Ensemble learning

LE BOOSTING: Le principe du *boosting* est d'évaluer une séquence de classificateurs faibles (*weak learners*) sur plusieurs versions légèrement modifiées des données d'apprentissage. Les décisions obtenues sont alors combinées par une somme pondérée pour obtenir le modèle final.

Exemple simple: Quel est le meilleur séparateur linéaire ?



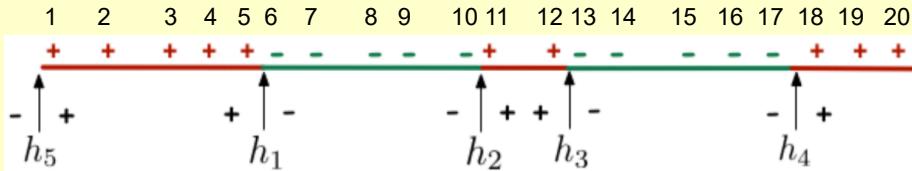
Taux d'erreur du prédicteur h_1 : $5/20 = 0,25 \rightarrow 25\%$ d'erreurs

- Peut-on le combiner avec d'autres classificateurs linéaires ?
- Par exemple :

$$H(x) = \text{signe} \left(\sum_{i=1}^l \alpha_i h_i(x) \right)$$

226

Ensemble learning



$$H(x) = \text{sign}(0.549 h_1(x) + 0.347 h_2(x) + 0.310 h_3(x) + 0.406 h_4(x) + 0.503 h_5(x))$$

```

def h1(x):
    if x<=5.5: return +1
    else: return -1
def h2(x):
    if x<=10.5: return -1
    else: return +1
def h3(x):
    if x<=12.5: return +1
    else: return -1
def h4(x):
    if x<=17.5: return -1
    else: return +1
def h5(x):
    if x<=0.5: return -1
    else: return +1

def H(x):
    y=0.549*h1(x)+0.347*h2(x)+0.310*h3(x)+0.406*h4(x)+0.503*h5(x)
    if y<=0: return -1
    else: return +1
  
```

Tp 37

Ensemble learning

```

16 def H(x):
17     y=0.549*h1(x)+0.347*h2(x)+0.310*h3(x)+0.406*h4(x)+0.503*h5(x)
18     if y<=0: return -1
19     else: return +1
20 Y_souhaite=[+1]*5*[-1]+2*[+1]+5*[-1]+3*[+1]
21 print(Y_souhaite)
22 Y_predite=[]
23 for i in range(1,21):
24     Y_predite+=[H(i)]
25 print(Y_predite)
26 from sklearn.metrics import *
27 print("Confusion Matrix: ")
28 print(confusion_matrix(Y_souhaite, Y_predite))
29 print ("Accuracy : ", accuracy_score(Y_souhaite,Y_predite)*100, '%')

[1, 1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1]
[1, 1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1]
Confusion Matrix:
[[10  0]
 [ 0 10]]
Accuracy : 100.0 %
  
```

Tp 37

Comment obtenir ce type de combinaison ? Par boosting !

228

Ensemble learning

Boosting : le principe

- Comment générer des classificateurs faibles décorrélés
 - En apprenant itérativement les classificateurs
 - En modifiant l'échantillon d'apprentissage à chaque itération
 - L'importance des exemples bien classés diminue
 - L'importance des exemples mal classés augmente
- Comment combiner les classificateurs :
 - Somme pondérée des décisions en fonction des performances des classificateurs
 - Il existe de très nombreux algorithmes de boosting.
 - Le plus connu : AdaBoost (Adaptive Boosting)

229

Ensemble learning

L'algorithme AdaBoost

- 1) $N \leftarrow$ le nombre d'échantillons d'apprentissage
- 2) Poids $\leftarrow [1/N, 1/N, \dots, 1/N]$ (N fois)
- 3) Vecteur_des_coefficients_ALPHA \leftarrow vecteur vide
- 4) Pour chaque classifieur C :
 - a) erreurs $\leftarrow [0]^*N$
 - b) pour $i=1$ à N : si le i ème échantillon est mal classé par C alors $\text{erreurs}[i] \leftarrow 1$
 - c) e \leftarrow somme des poids qui correspondent à des échantillons mal classés
 - d) $\alpha \leftarrow 0.5 * \log((1-e)/e)$
 - e) w $\leftarrow [0,0,0,\dots,0]$ (N fois)
 - f) pour $i=1$ à N :
 - si $\text{erreurs}[i] = 1$ alors $w[i] \leftarrow \text{Poids}[i] * \exp(\alpha)$
 - sinon: $w[i] \leftarrow \text{Poids}[i] * \exp(-\alpha)$
 - g) Poids $\leftarrow w / \text{somme}(w) #$
 - h) Ajouter le coefficient α au vecteur Vecteur_des_coefficients_ALPHA

230

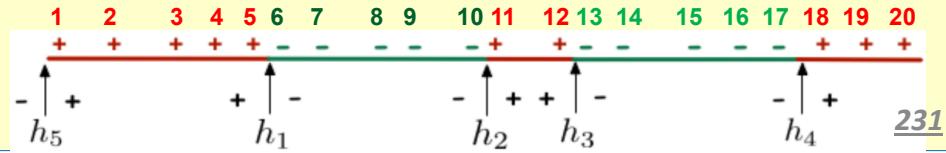
Ensemble learning

```

def h1(x):
    if x<=5.5:return +1
    else:return -1
def h2(x):
    if x<=10.5:return -1
    else:return +1
def h3(x):
    if x<=12.5:return +1
    else:return -1
def h4(x):
    if x<=17.5:return -1
    else:return +1
def h5(x):
    if x<=0.5:return -1
    else:return +1
from numpy import *
Y_souhaite=5*[+1]+5*[-1]+2*[+1]+5*[-1]+3*[+1]
dn=[]
for i in range(1,21):
    dn.append([i,Y_souhaite[i-1]])
training_set=array(dn)
N = len(training_set)
weights=ones(N)/N
Liste_de_Classifieurs=[]
ALPHA=[]

```

Tp 38



Ensemble learning

```

def ajouter_Classifieur(classifieur):
    global weights
    errors = array([t[1]!=classifieur(t[0]) for t in training_set])
    e = (errors*weights).sum()
    alpha = 0.5 * log((1-e)/e)
    print('e=% .2f a=% .2f'%(e, alpha))
    w = zeros(N)
    for i in range(N):
        if errors[i] == 1: w[i] = weights[i] * exp(alpha)
        else: w[i] = weights[i] * exp(-alpha)
    weights = w / w.sum()
    Liste_de_Classifieurs.append(classifieur)
    ALPHA.append(alpha)

```

Tp 38

232

Ensemble learning

```

41 ajouter_Classifieur(lambda x:h1(x)); ajouter_Classifieur(lambda x:h2(x))
42 ajouter_Classifieur(lambda x:h3(x)); ajouter_Classifieur(lambda x:h4(x))
43 ajouter_Classifieur(lambda x:h5(x)); print(ALPHA)
44 def evaluate():
45     NR = len(Liste_de_Classifieurs); s=0;
46     for (x,l) in training_set:
47         hx = [ALPHA[i]*Liste_de_Classifieurs[i](x) for i in range(NR)]
48         if sign(l) == sign(sum(hx)):s+=1
49     return 'taux de reussite='+str(s/len(training_set)*100)+'%'
50 evaluate()

```

[0.5493061443340549, 0.3465735902799726, 0.3095196042031118, 0.405465108108164
6, 0.502760932801049]

'taux de reussite=100.0%'

Tp 38

Ensemble learning

Boosting Avec scikit-learn

C'est la classe `AdaBoostClassifier` qui implémente cet algorithme. Les paramètres les plus importants sont :

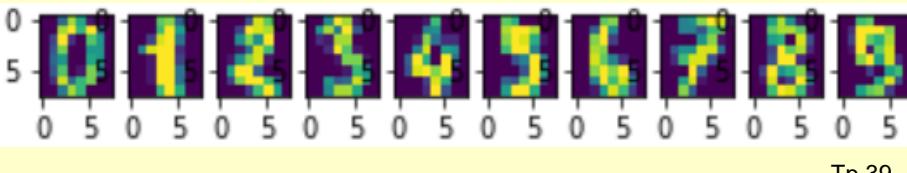
- `n_estimators` : integer, optional (default=10). Le nombre de classifieurs faibles.
- `learning_rate` : contrôle la vitesse de changement des poids par itération.
- `base_estimator` : (default=DecisionTreeClassifier) le classifieur faible utilisé.

234

Ensemble learning

Dans les Tps suivants nous allons utiliser la base de données `digits`, qui contient 10 classes (images des chiffres en écriture manuscrite). Il y a 1797 éléments, chaque élément a 64 attributs (8 pixels par 8).

```
from sklearn.datasets import load_digits
digits = load_digits()
# Affichage des 10 premières images
import matplotlib.pyplot as plt
fig = plt.figure()
for i, digit in enumerate(digits.images[:10]):
    fig.add_subplot(1,10,i+1); plt.imshow(digit)
plt.show()
```



Tp 39

[235](#)

Ensemble learning

```
1 from sklearn import tree
2 from sklearn.model_selection import train_test_split
3 from sklearn.datasets import load_digits
4 from sklearn.ensemble import AdaBoostClassifier
5 digits = load_digits(); X, y = digits.data, digits.target
6 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.80)
7 un_seul_arbre=tree.DecisionTreeClassifier(max_depth=5)
8 un_seul_arbre.fit(X_train, y_train)
9 accuracy = un_seul_arbre.score(X_test, y_test)
10 print('accuracy un seul arbre:',accuracy)
11 # AdaBoost basé sur 200 arbres de décision
12 clf = AdaBoostClassifier(base_estimator=tree.DecisionTreeClassifier(\n
13                         max_depth=5), n_estimators=200, learning_rate=2)
14 clf.fit(X_train, y_train); accuracy = clf.score(X_test, y_test)
15 print('accuracy Boosting 200 arbres:',accuracy)
```

accuracy un seul arbre: 0.6884561891515995

accuracy Boosting 200 arbres: 0.9325452016689847

Tp 39

Ensemble learning

```

from numpy import *
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error
from sklearn import datasets
from sklearn.ensemble import AdaBoostRegressor
boston = datasets.load_boston()
X, y = shuffle(boston.data, boston.target)
offset = int(0.7*len(X))
X_train, y_train = X[:offset], y[:offset]
X_test, y_test = X[offset:], y[offset:]
regressor = AdaBoostRegressor(n_estimators=5)
regressor.fit(X_train, y_train)
x = [11.95, 0.0, 18.100, 0, 0.6590, 5.6090, 90.00,\n     1.385, 24, 680.0, 20.20, 332.09, 12.13]
y = regressor.predict([x])                                         Tp 39
print("Prediction for " + str(x) + " = " + str(y))
print("Erreur quadratique moyenne MSE = "+\
      str(mean_squared_error(y_test,regressor.predict(X_test))))
```

Prediction for [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.2, 332.09, 12.13] = [20.62608696]
 Erreur quadratique moyenne MSE = 14.157643166109976

Ensemble learning

Le jeu de données Boston comprend le prix des maisons dans les différentes zones de Boston. L'objectif sera de prédire le prix des maisons grâce aux différentes informations présentes dans le jeu de données.

- CRIM per capita crime rate by town',
- ZN proportion of residential land zoned for lots over 25,000 sq.ft.',
- INDUS proportion of non-retail business acres per town',
- CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)',
- NOX nitric oxides concentration (parts per 10 million)',
- RM average number of rooms per dwelling',
- AGE proportion of owner-occupied units built prior to 1940',
- DIS weighted distances to five Boston employment centres',
- RAD index of accessibility to radial highways',
- TAX full-value property-tax rate per \$10,000',
- PTRATIO pupil-teacher ratio by town',
- B $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town',
- LSTAT % lower status of the population',
- MEDV Median value of owner-occupied homes in \$1000's"

Ensemble learning

Bagging

Les méthodes de type *bagging* construisent plusieurs instances d'un estimateur, calculées sur des échantillons aléatoires tirés de la base d'apprentissage (et éventuellement une partie des attributs, également sélectionnés de façon aléatoire), et ensuite combine les prédictions individuelles en réalisant leur moyenne pour réduire la variance de l'estimateur. Leur avantage principal réside dans le fait qu'ils construisent une version améliorée de l'algorithme de base, sans demander de modification de cet algorithme. Le prix à payer est un coût de calcul plus élevé. **Les méthodes *bagging* fonctionnent très bien avec des prédicteurs « forts ». Par contre, les méthodes *boosting* peuvent être adaptées à des prédicteurs faibles (*weak learners*).**

239

Ensemble learning

Dans Scikit-learn, les méthodes de *bagging* sont implémentées via la classe `BaggingClassifier` et `BaggingRegressor`. Les constructeurs prennent en paramètres un estimateur de base et la stratégie de sélection des points et attributs :

- `base_estimator` : optionnel (default=None). Si None alors l'estimateur est un arbre de décision.
- `max_samples` : la taille de l'échantillon aléatoire tiré de la base d'apprentissage.
- `max_features` : le nombre d'attributs tirés aléatoirement.
- `bootstrap` : boolean, optionnel (default=True). Tirage des points avec remise ou non.
- `bootstrap_features` : boolean, optionnel (default=False). Tirage des attributs avec remise ou non.
- `oob_score` : boolean. Estimer ou non l'erreur de généralisation OOB. 240

Ensemble learning

Pour ce TP, nous allons utiliser comme classifieur de base un arbre de décision `DecisionTreeClassifier`. Ce classifieur nous permet d'établir des performances de référence (c'est un ensemble à 1 modèle).

```

1 import numpy as np; from sklearn import tree
2 from sklearn.ensemble import BaggingClassifier
3 X, y = digits.data, digits.target
4 clf = tree.DecisionTreeClassifier()
5 clf.fit(X, y); accuracy = clf.score(X,y) ; print(accuracy)
6
7 from sklearn.model_selection import train_test_split
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.90)
9
10 clf = tree.DecisionTreeClassifier(); clf.fit(X_train, y_train)
11 Z=clf.predict(X_test); accuracy = clf.score(X_test,y_test);print(accuracy)

```

1.0
0.6860321384425216

Tp 40

Ensemble learning

Question :

Calculer le score moyen sur 100 tirages pour la séparation apprentissage/test.

Correction :

```

1 N = 100
2 accuracies = []
3 for i in range(N):
4     X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5)
5     clf = tree.DecisionTreeClassifier()
6     clf.fit(X_train, y_train)
7     Z = clf.predict(X_test)
8     accuracies.append(clf.score(X_test,y_test))
9     #print(accuracies[i])
10 print(np.mean(accuracies))

```

0.8276974416017798

Tp 40

Ensemble learning

Pour comparer, construisons maintenant un classifieur *bagging* sur nos données, basé sur **200** DecisionTreeClassifier :

```

1 clf = BaggingClassifier(tree.DecisionTreeClassifier(),
2                         max_samples=0.5, max_features=0.5, n_estimators=200)
3 #L'apprentissage et l'évaluation de cet ensemble se font de la façon habituelle :
4 clf.fit(X_train, y_train)
5 Z = clf.predict(X_test)
6 accuracy=clf.score(X_test,y_test)
7 print(accuracy)

```

0.9610678531701891

On obtient un meilleur classifieur (accuracy \sim 96.10%)

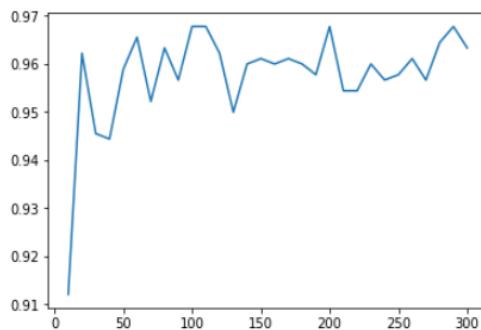
Tp 40

Ensemble learning

```

1 N = 30 ; accuracy = []
2 for i in range(N):
3     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
4     clf = BaggingClassifier(tree.DecisionTreeClassifier(),
5                             max_samples=0.5, max_features=0.5, n_estimators=(i+1)*10)
6     clf.fit(X_train, y_train) ; Z = clf.predict(X_test)
7     accuracy.append(clf.score(X_test,y_test))
8 import matplotlib.pyplot as plt
9 plt.plot([10*(i+1) for i in range(N)], accuracy)
10 plt.show()

```



Le taux d'erreur diminue avec n_estimators, mais à partir d'une valeur (ici=25) il se stabilise, donc on ne gagne rien à partir de cette valeur de n_estimators.

Tp 40

Ensemble learning

Recherche automatique de max_samples et max_features: [GridSearchCV](#)

```

1 digits = load_digits()
2 X, y = digits.data, digits.target
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
4
5 from sklearn.model_selection import GridSearchCV
6 pgrid = {"max_samples": [0.2, 0.4, 0.5, 0.55, 0.6, 0.8],
7           "max_features": [0.2, 0.4, 0.5, 0.55, 0.6, 0.8]}
8 grid_search = GridSearchCV(BaggingClassifier(tree.DecisionTreeClassifier()),
9                           param_grid=pgrid, cv=5)
10 grid_search.fit(X_train, y_train)
11 print("Meilleur score : " + str(grid_search.best_estimator_.score(X_test, y_test)))
12 print("max_samples: " + str(grid_search.best_estimator_.max_samples))
13 print("max_features: " + str(grid_search.best_estimator_.max_features))

```

Meilleur score : 0.9276974416017798

max_samples: 0.8

max_features: 0.6

Tp 40

Ensemble learning

Forêts aléatoires

L'algorithme des forêts aléatoires propose une optimisation des arbres de décision. Il utilise le même principe que le *bagging*, mais avec une étape supplémentaire de randomisation dans la sélection des attributs des nœuds dans le but de réduire la variance de l'estimateur obtenu. Les deux objets Python qui implémentent les forêts aléatoires sont [RandomForestClassifier](#) et [RandomForestRegressor](#). Les paramètres les plus importants sont :

- [`n_estimators`](#) : integer, optional (default=10). Le nombre d'arbres.
- [`max_features`](#) : le nombre d'attributs à considérer à chaque split.
- [`max_samples`](#) : la taille de l'échantillon aléatoire tiré de la base d'apprentissage.
- [`min_samples_leaf`](#) : le nombre minimal d'éléments dans un nœud feuille.
- [`oob_score`](#) : boolean. Estimer ou non l'erreur de généralisation.

Par la suite nous allons refaire la classification sur la base Digits en utilisant un classifieur [RandomForestClassifier](#). On sépare les données en gardant 10% pour l'apprentissage et 90% pour le test.

246

Ensemble learning

```

1 from sklearn.datasets import load_digits
2 digits=load_digits(); X,y=digits.data, digits.target
3 from sklearn.model_selection import train_test_split
4 X_train, X_test, y_train, y_test =\
    train_test_split(X, y, test_size=0.90)
5 from sklearn.ensemble import RandomForestClassifier
6 clf = RandomForestClassifier(n_estimators=200)
7 clf.fit(X_train, y_train)
8 y_pred = clf.predict(X_test)
9 accuracy = clf.score(X_test,y_test)
10 print(accuracy)
11

```

0.9159456118665018

Tp 41

Ensemble learning

Question : Comment la valeur de la variable accuracy se compare avec le cas *bagging* qui utilise le même nombre d'arbres (200 dans notre cas) ?

```

1 from sklearn import tree
2 from sklearn.ensemble import BaggingClassifier
3 clf = tree.DecisionTreeClassifier()
4 clf.fit(X_train, y_train)
5 Z = clf.predict(X_test)
6 print("Arbre de décision : " + str(clf.score(X_test,y_test)))
7 clf = BaggingClassifier(tree.DecisionTreeClassifier(),\
8     max_samples=0.5, max_features=0.5, n_estimators=200)
9 clf.fit(X_train, y_train) ; Z = clf.predict(X_test)
10 print("Bagging (200 arbres) : " + str(clf.score(X_test,y_test)))
11 clf = RandomForestClassifier(n_estimators=200)
12 clf.fit(X_train, y_train); Z = clf.predict(X_test)
13 print("Forêt aléatoire (200 arbres) : " + str(clf.score(X_test,y_test)))

```

Arbre de décision : 0.6711990111248455

Bagging (200 arbres) : 0.9066749072929543

Forêt aléatoire (200 arbres) : 0.9177997527812114

Ce qui confirme notre attente : les forêts aléatoires produisent un classifieur un peu meilleur sur cette base de données. Tp 41

Ensemble learning

Question :

Construire la variance de la valeur accuracy sur 100 tirages pour la séparation apprentissage/test. Que pouvons-nous conclure en comparant avec le *bagging* ?

Correction :

```

1 N = 10
2 bagging_accuracies = []
3 for i in range(N):
4     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
5     clf = BaggingClassifier(tree.DecisionTreeClassifier(), n_estimators=30)
6     clf.fit(X_train, y_train)
7     bagging_accuracies.append(clf.score(X_test,y_test))
8 print(np.mean(bagging_accuracies), np.std(bagging_accuracies))
9 rf_accuracies = []
10 for i in range(N):
11     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
12     clf = RandomForestClassifier(n_estimators=30)
13     clf.fit(X_train, y_train)
14     rf_accuracies.append(clf.score(X_test,y_test))
15 print(np.mean(rf_accuracies), np.std(rf_accuracies))
```

Tp 41

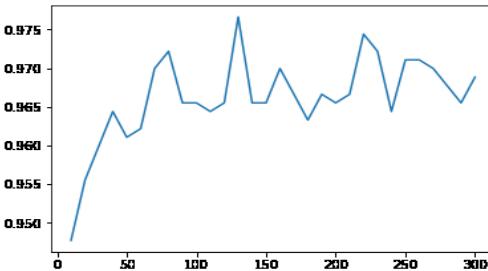
0.9397107897664071 0.010289059915325636 Nous avons un plus faible taux
0.9581757508342603 0.004138170241988469 moyen d'erreurs, variance plus faible.

Ensemble learning

Question : Construire le graphique accuracy vs n_estimators. Que constatez-vous ? A partir de quelle valeur on n'améliore plus ?

```

import matplotlib.pyplot as plt
N = 30; accuracy = []
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
for i in range(N):
    clf = RandomForestClassifier(n_estimators=(i+1)*10)
    clf.fit(X_train, y_train)
    accuracy.append(clf.score(X_test,y_test))
plt.plot([10*(i+1) for i in range(30)], accuracy); plt.show()
```



A partir de n_estimators = 25 (ou 75) le résultat commence à osciller autour de la moyenne donc on ne gagne plus rien en augmentant la valeur.

Tp 41